Red Hat Process Automation Manager 7.11

Red Hat Process Automation Manager でのデシジョンサービスの開発

Last Updated: 2023-05-19

Red Hat Process Automation Manager 7.11 Red Hat Process Automation Manager でのデシジョンサービスの開発

法律上の通知

概要

本ガイドでは、DMN (Decision Model and Notation) モデル、Droools ルール言語 (DRL) ファイル、ガイド付きデシジョンテーブルなどのデシジョンオーサリングアセットを使用して、Red Hat Process Automation Manager でデシジョンサービスを開発する方法を説明します。

目次

はじめに

ビジネスデシジョンの開発者は、Red Hat Process Automation Manager で DMN (Decision Model and Notation) モデル、Drools ルール言語 (DRL) ルール、ガイド付きデシジョンテーブルなどのルールオーサリングアセットを使用してデシジョンサービスを開発できます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みにより、これらの変更は今後の複数のリリースに対して段階的に実施されます。詳細は、**弊社の CTO である Chris Wright のメッセージ**を参照してください。

パート I. DMN モデルを使用したデシジョンサービスの作成

ビジネスアナリストやルール作成者は、DMN (Decision Model and Notation) を使用して、デシジョンサービスを視覚的にモデル化できます。DMN デシジョンモデルの意思決定要件は、1 つ以上の意思決定要件ダイアグラム (DRD) で記述された意思決定要件グラフ (DRG) により決まります。DMN モデルでは、DRD は DRG 全体の一部またはすべてを表します。DRD は、デシジョンテーブルなど、DMN ボックス式で定義されたロジックを使用した各デシジョンノードで、開始から終了までビジネスデシジョンを追跡します。

Red Hat Process Automation Manager は、適合レベル 3 で DMN 1.2 モデルの設計およびランタイムをサポートし、適合レベル 3 で DMN 1.1 および 1.3 モデルはランタイムのみサポートします。DMN モデルは、Business Central で直接作成するか、VSCode の Red Hat Process Automation Manager DMN モデラーを使用して作成するか、既存の DMN モデルを Red Hat Process Automation Manager プロジェクトにインポートしてデプロイおよび実行することができます。Business Central にインポートした DMN 1.1 および 1.3 モデル (DMN 1.3 機能は含まれません) は、DMN デザイナーで開き、保存時に DMN 1.2 モデルに変換されます。

DMN に関する詳細は、Object Management Group (OMG) の Decision Model and Notation specification を参照してください。

DMN デシジョンサービスの例を使用した段階的なチュートリアルは、**デシジョンサービスのスタートガイド** を参照してください。

第1章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表1.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第2章 RED HAT PROCESS AUTOMATION MANAGER の BPMN モデラーおよび DMN モデラー

Red Hat Process Automation Manager は、グラフィカルモデラーを使用して Business Process Model and Notation (BPMN) プロセスモデルと、Decision Model and Notation (DMN) デシジョンモデルを設計するのに使用できる次の拡張機能またはアプリケーションを提供します。

- **Business Central**: 関連する埋め込みデザイナーで、BPMN モデル、DMN モデル、およびテストシナリオファイルを表示および設計できます。
  Business Central を使用するには、Business Central を含む開発環境を設定してビジネスルールおよびプロセスを作成し、KIE Server を作成して、作成したビジネスルールとプロセスを実行およびテストします。

- **Red Hat Process Automation Manager VSCode 拡張** Visual Studio Code (VSCode) で BPMN モデル、DMN モデル、およびテストシナリオファイルを表示して、作成できるようにします。VSCode 拡張機能には VSCode 1.46.0 以降が必要です。
  Red Hat Process Automation Manager VSCode 拡張機能をインストールするには、VSCode で **Extensions** メニューオプションを選択して、**Red Hat Business Automation Bundle**拡張を検索し、インストールします。

- **スタンドアロン BPMN および DMN エディター**: Web アプリケーションに組み込まれた BPMN モデルおよび DMN モデルを表示して、作成できます。必要なファイルをダウンロードするには、NPM レジストリー から NPM アーティファクトを使用するか、**https://<YOUR_PAGE>/dmn/index.js** (DMN スタンドアロンのエディターライブラリーの場合)、または **https://<YOUR_PAGE>/bpmn/index.js** (BPMN スタンドアロンエディターライブラリーの場合) で JavaScript ファイルを直接ダウンロードします。

2.1. RED HAT PROCESS AUTOMATION MANAGER VSCODE 拡張機能バンドルのインストール

Red Hat Process Automation Manager は、**Red Hat Business Automation Bundle**VSCode 拡張機能を提供します。これにより、Decision Model and Notation (DMN) デシジョンモデル、Business Process Model and Notation (BPMN) 2.0 ビジネスプロセス、およびテストシナリオを VSCode で直接作成できます。VSCode は、新しいビジネスアプリケーションを開発するために推奨される統合開発環境 (IDE) です。Red Hat Process Automation Manager は、必要に応じて DMN サポートまたは BPMN サポートに VSCode 拡張機能である **DMN Editor** および **BPMN Editor** をそれぞれ提供します。

重要

VSCode のエディターは、Business Central のエディターと部分的に互換性があり、VSCode では複数の Business Central 機能がサポートされていません。

前提条件

- VSCode の最新の安定版がインストールされている。

手順

1. VSCode IDE で **Extensions** メニューオプションを選択し、DMN、BPMN、およびテストシナリオファイルのサポートに対して **Red Hat Business Automation Bundle**を検索します。
   DMN ファイルまたは BPMN ファイルだけをサポートする場合は、**DMN Editor** または **BPMN Editor** 拡張機能をそれぞれ検索することもできます。

2. **Red Hat Business Automation Bundle**拡張機能が VSCode に表示されたら、これを選択して **Install** をクリックします。

3. VSCode エディターの動作を最適化するには、拡張機能のインストールの完了後に、VSCode のインスタンスをリロードするか、閉じて再起動します。

VSCode 拡張バンドルをインストールした後、VSCode で開くか作成するすべての **.dmn** ファイル、**.bpmn** ファイル、または **.bpmn2** ファイルがグラフィカルモデルとして自動的に表示されます。さらに、開くまたは作成する **.scesim** ファイルが、ビジネスデシジョンの機能をテストするテーブルテストシナリオモデルとして自動的に表示されます。

DMN、BPMN、またはテストシナリオモデラーが DMN、BPMN、またはテストシナリオファイルの XML ソースのみを開き、エラーメッセージが表示される場合は、報告されたエラーおよびモデルファイルを確認して、すべての要素が正しく定義されていることを確認します。

注記

新しい DMN モデルまたは BPMN モデルの場合は、Web ブラウザーで **dmn.new** または **bpmn.new** を入力して、オンラインモデラーで DMN モデルまたは BPMN モデルを設計することもできます。モデルの作成が終了したら、オンラインモデラーページで **Download** をクリックして、DMN ファイルまたは BPMN ファイルを VSCode の Red Hat Process Automation Manager プロジェクトにインポートできます。

2.2. RED HAT PROCESS AUTOMATION MANAGER スタンドアロンのエディターの設定

Red Hat Process Automation Manager は、自己完結型のライブラリーに分散されたスタンドアロンのエディターを提供し、エディターごとにオールインワンの JavaScript ファイルを提供します。JavaScript ファイルは、包括的な API を使用してエディターを設定および制御します。

スタンドアロンのエディターは、以下の 3 つの方法でインストールできます。

- 各 JavaScript ファイルを手動でダウンロード
- NPM パッケージの使用

手順

1. 以下の方法のいずれかを使用して、スタンドアロンのエディターをインストールします。
   各 JavaScript ファイルを手動でダウンロード: この方法の場合は、以下の手順に従います。

   a. JavaScript ファイルをダウンロードします。

   b. ダウンロードした Javascript ファイルをホスト型アプリケーションに追加します。

   c. 以下の **<script>** タグを HTML ページに追加します。

   DMN エディターの HTML ページのスクリプトタグ

   ```
   <script src="https://<YOUR_PAGE>/dmn/index.js"></script>
   ```

   BPMN エディターの HTML ページのスクリプトタグ

   ```
   <script src="https://<YOUR_PAGE>/bpmn/index.js"></script>
   ```

NPM パッケージの使用: この方法の場合は、以下の手順に従います。

a. NPM パッケージを **package.json** ファイルに追加します。

NPM パッケージの追加

```
npm install @redhat/kogito-tooling-kie-editors-standalone
```

b. 各エディターライブラリーを TypeScript ファイルにインポートします。

各エディターのインポート

```
import * as DmnEditor from "@redhat/kogito-tooling-kie-editors-standalone/dist/dmn"
import * as BpmnEditor from "@redhat/kogito-tooling-kie-editors-standalone/dist/bpmn"
```

2. スタンドアロンのエディターをインストールしたら、以下の例のように提供されたエディター API を使用して必要なエディターを開き、DMN エディターを開きます。API は、各エディターで同じものになります。

DMN スタンドアロンのエディターを開く

```
const editor = DmnEditor.open({
  container: document.getElementById("dmn-editor-container"),
  initialContent: Promise.resolve(""),
  readOnly: false,
  origin: "",
  resources: new Map([
    [
      "MyIncludedModel.dmn",
      {
        contentType: "text",
        content: Promise.resolve("")
      }
    ]
  ])
});
```

エディター API で以下のパラメーターを使用します。

表2.1 パラメーターの例

パラメーター	説明
container	エディターが追加される HTML 要素。

パラメーター	説明
initialContent	DMN モデルのコンテンツへの Promise。以下の例のように、このパラメーターは空にすることができます。 • **Promise.resolve("")** • **Promise.resolve("<DIAGRAM_CONTENT_DIRECTLY_HERE>")** • **fetch("MyDmnModel.dmn").then(content ⇒ content.text())**
readonly (任意)	エディターでの変更を許可します。コンテンツの編集を許可する場合は **false** (デフォルト)、エディターで読み取り専用モードの場合は **true** に設定します。
origin (任意)	リポジトリーの起点。デフォルト値は **window.location.origin** です。
resources (任意)	エディターのリソースのマッピング。たとえば、このパラメーターを使用して、BPMN エディターの DMN エディターまたは作業アイテム定義に含まれるモデルを提供します。マップの各エントリーには、リソース名と、**content-type** (**text** または **binary**) および **content** (**initialContent** パラメーターと同様) で設定されるオブジェクトが含まれています。

返されるオブジェクトには、エディターの操作に必要なメソッドが含まれます。

表2.2 返されたオブジェクトメソッド

メソッド	説明
getContent(): Promise<string>	エディターのコンテンツを含む promise を返します。
setContent(content: string): void	エディターの内容を設定します。
getPreview(): Promise<string>	現在のダイアグラムの SVG 文字列が含まれる promise を返します。
subscribeToContentChanges(callback: (isDirty: boolean) ⇒ void): (isDirty: boolean) ⇒ void	エディターでコンテンツを変更し、サブスクライブ解除に使用されるのと同じコールバックを返す際に呼び出されるコールバックを設定します。
unsubscribeToContentChanges(callback: (isDirty: boolean) ⇒ void): void	エディターでコンテンツが変更される際に渡されたコールバックのサブスクライブを解除します。

メソッド	説明
markAsSaved(): void	エディターの内容が保存されることを示すエディターの状態をリセットします。また、コンテンツの変更に関連するサブスクライブされたコールバックをアクティベートします。
undo(): void	エディターの最後の変更を元に戻します。また、コンテンツの変更に関連するサブスクライブされたコールバックをアクティベートします。
redo(): void	エディターで、最後に元に戻した変更をやり直します。また、コンテンツの変更に関連するサブスクライブされたコールバックをアクティベートします。
close(): void	エディターを終了します。
getElementPosition(selector: string): Promise<Rect>	要素をキャンバスまたはビデオコンポーネント内に置いた場合に、標準のクエリーセレクターを拡張する方法を提供します。**selector** パラメーターは、**Canvas:::MySquare**、**Video:::PresenterHand** などの **<PROVIDER>:::<SELECT>** 形式に従う必要があります。このメソッドは、要素の位置を表す **Rect** を返します。
envelopeApi: MessageBusClientApi<KogitoEditorEnvelopeApi>	これは高度なエディター API です。高度なエディター API の詳細は、MessageBusClientApi および KogitoEditorEnvelopeApi を参照してください。

第3章 MAVEN を使用した DMN モデルおよび BPMN モデルの作成および実行

Maven アーキタイプを使用して、Business Central ではなく Red Hat Process Automation Manager VSCode 拡張機能を使用して、VSCode で DMN モデルおよび BPMN モデルを開発できます。その後、必要に応じて、Business Central で、アーキタイプを Red Hat Process Automation Manager のデシジョンサービスおよびプロセスサービスに統合できます。DMN モデルおよび BPMN モデルを開発する方法は、Red Hat Process Automation Manager VSCode 拡張機能を使用して新規ビジネスアプリケーションを構築する場合に便利です。

手順

1. コマンドターミナルで、新しい Red Hat Process Automation Manager プロジェクトを保存するローカルディレクトリーに移動します。
2. 以下のコマンドを入力して、以下の Maven アーキタイプを使用して、定義したディレクトリーにプロジェクトを生成します。

   Maven アーキタイプを使用したプロジェクトの生成

   ```
   mvn archetype:generate \
       -DarchetypeGroupId=org.kie \
       -DarchetypeArtifactId=kie-kjar-archetype \
       -DarchetypeVersion=7.52.0.Final-redhat-00007
   ```

   このコマンドにより、必要な依存関係で Maven プロジェクトが生成され、ビジネスアプリケーションを構築するのに必要なディレクトリーとファイルが生成されます。プロジェクト開発時に Git バージョン制御システム (推奨) を設定して使用できます。

   同じディレクトリーに複数のプロジェクトを生成する場合は、直前のコマンドに **-DgroupId=<groupid> -DartifactId=<artifactId>** を追加して、生成されたビジネスアプリケーションの **artifactId** および **groupId** を指定できます。
3. VSCode IDE で **File** をクリックし、**Open Folder** を選択し、直前のコマンドを使用して生成されたディレクトリーに移動します。
4. 最初のアセットを作成する前に、ビジネスアプリケーションのパッケージ (例: **org.kie.businessapp**) を設定し、以下のパスにそれぞれのディレクトリーを作成します。
   - **PROJECT_HOME/src/main/java**
   - **PROJECT_HOME/src/main/resources**
   - **PROJECT_HOME/src/test/resources**

   たとえば、**org.kie.businessapp** パッケージの **PROJECT_HOME/src/main/java/org/kie/businessapp** を作成できます。
5. VSCode を使用して、ビジネスアプリケーションにアセットを作成します。以下の方法で、Red Hat Process Automation Manager VSCode 拡張機能がサポートするアセットを作成できます。
   - ビジネスプロセスを作成するには、**PROJECT_HOME/src/main/java/org/kie/businessapp** ディレクトリーに、**.bpmn** または **.bpmn2** の新規ファイルを作成します (例: **Process.bpmn**)。

- DMN モデルを作成するには、**PROJECT_HOME/src/main/java/org/kie/businessapp** ディレクトリーに、**.dmn** の新規ファイルを作成します (例: **AgeDecision.dmn**)。
- テストシナリオシミュレーションモデルを作成するには、**PROJECT_HOME/src/main/java/org/kie/businessapp** ディレクトリーに、**.scesim** の新規ファイルを作成します (例: **TestAgeScenario.scesim**)。

6. Maven アーキタイプでアセットを作成したら、コマンドラインで (**pom.xml** がある) プロジェクトのルートディレクトリーに移動し、以下のコマンドを実行してプロジェクトのナレッジ JAR (KJAR) を構築します。

```
mvn clean install
```

ビルドに失敗したら、コマンドラインのエラーメッセージに記載されている問題に対応し、ビルドに成功するまでプロジェクトの妥当性確認を行います。ただし、ビルドに成功すると、**PROJECT_HOME/target** ディレクトリーでビジネスアプリケーションのアーティファクトを確認できます。

注記

mvn clean install コマンドを使用して、開発中の主要な変更ごとにプロジェクトを検証します。

REST API を使用して実行中の KIE Server に、ビジネスアプリケーションの生成されたナレッジ JAR (KJAR) をデプロイできます。プロセスの REST API の使用方法は、KIE API を使用した Red Hat Process Automation Manager の操作 を参照してください。

第4章 DMN (DECISION MODEL AND NOTATION)

DMN (Decision Model and Notation) は、業務的意思決定を説明してモデル化するために、OMG (Object Management Group) が確立している規格です。DMN は XML スキーマを定義して、DMN モデルを DMN 準拠のプラットフォーム間や組織間で共有し、ビジネスアナリストやビジネスルール開発者が DMN デシジョンサービスの設計と実装で協力できるようにするものです。DMN 規格は、ビジネスプロセスを開発してモデル化する BPMN (Business Process Model and Notation) 規格と類似しており、一緒に使用できます。

DMN の背景およびアプリケーションの詳細は、OMG の Decision Model and Notation specification を参照してください。

4.1. DMN 適合レベル

DMN 仕様は、ソフトウェア実装における増分の適合レベルを 3 つ定義します。特定のレベルの準拠を主張する製品は、その前の適合レベルにも準拠する必要があります。たとえば、適合レベル 3 を実装するには、適合レベル 1 および 2 でサポートされるコンポーネントにも対応する必要があります。各適合レベルの公式な定義は OMG の Decision Model and Notation specification を参照してください。

以下の一覧では、3 つの DMN 適合レベルをまとめています。

適合レベル 1

DMN 適合レベル 1 の実装は、意思決定要件ダイアグラム (DRD)、デシジョンロジック、デシジョンテーブルをサポートしますが、デシジョンモデルは実行可能ではありません。式の定義には、自然言語、非体系化言語を含むすべての言語を使用できます。

適合レベル 2

DMN 適合レベル 2 の実装には、適合レベル 1 の要件のほかに、S-FEEL (Simplified Friendly Enough Expression Language) 式と、完全に実行可能なデシジョンモデルをサポートします。

適合レベル 3

DMN 適合レベル 3 の実装には、適合レベル 1 および 2 の要件のほかに、FEEL (Friendly Enough Expression Language) 式、ボックス式の完全セット、完全に実行可能なデシジョンモデルをサポートします。

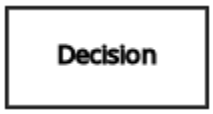
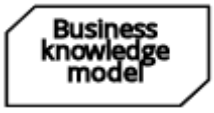
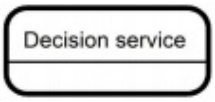
Red Hat Process Automation Manager は、適合レベル 3 で DMN 1.2 モデルの設計およびランタイムをサポートし、適合レベル 3 で DMN 1.1 および 1.3 モデルはランタイムのみサポートします。DMN モデルは、Business Central で直接作成するか、VSCode の Red Hat Process Automation Manager DMN モデラーを使用して作成するか、既存の DMN モデルを Red Hat Process Automation Manager プロジェクトにインポートしてデプロイおよび実行することができます。Business Central にインポートした DMN 1.1 および 1.3 モデル (DMN 1.3 機能は含まれません) は、DMN デザイナーで開き、保存時に DMN 1.2 モデルに変換されます。

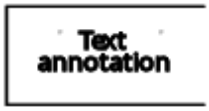
4.2. DMN 意思決定要件ダイアグラム (DRD) のコンポーネント

デシジョン要件ダイアグラム (DRD) は、DMN モデルを視覚的にしたものです。DMN モデルでは、DRD は意思決定要件グラフ (DRG) 全体の一部またはすべてを表します。DRD は、デシジョンノード、ビジネスナレッジモデル、ビジネスナレッジのソース、入力データ、およびデシジョンサービスを使用して、ビジネスデシジョンを追跡します。

以下の表では、DRD のコンポーネントについてまとめています。

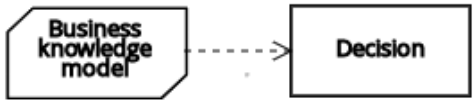
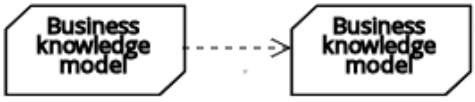
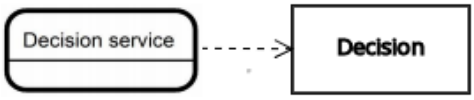
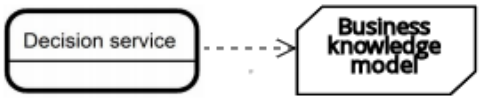
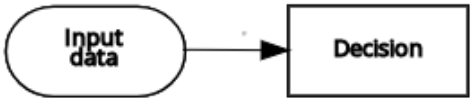
表4.1 DRD コンポーネント

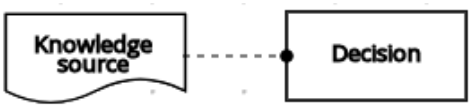
コンポーネント		説明	表記
要素	デシジョン	1つ以上の要素が定義したデシジョンロジックをもとに出力を決定するノード。	
	ビジネスナレッジモデル	1つまたは複数のデシジョン要素が含まれる再利用可能な関数。同じロジックですが、サブの入力または決定が異なるため、ビジネスナレッジモデルを使用してどの手順に従うかを決定します。	
	ナレッジソース	デシジョンまたはビジネスナレッジモデルを規定する外部の機関、ドキュメント、委員会またはポリシー。ナレッジソースは、実行可能なビジネスルールではなく、実際の要因への参照となります。	
	入力データ	デシジョンノードまたはビジネスナレッジモデルで使用する情報。入力データには通常、融資戦略で使用するローン申請データなど、ビジネスに関連するビジネスレベルのコンセプトまたはオブジェクトが含まれます。	
	デシジョンサービス	呼び出しのサービスとして公開される、再利用可能なデシジョンセットを含むトップレベルのデシジョン。デシジョンサービスは、外部アプリケーションまたは BPMN ビジネスプロセスから呼び出し可能です。	
要件コネクター	情報要件	情報を必要とする別のデシジョンノードへの入力データノードまたはデシジョンノードからの接続	
	ナレッジ要件	デシジョンロジックを呼び出す別のビジネスナレッジモデルまたはデシジョンノードへのビジネスナレッジモデルからの接続	
	認証局の要件	入力データノードまたはデシジョンノードから従属するナレッジソース、またはナレッジソースからデシジョンノード、ビジネスナレッジモデル、または別のナレッジソースへの接続	

コンポーネント	説明		表記
アーティファクト	テキストのアノテーション	入力データノード、デシジョンノード、ビジネスナレッジモデル、またはナレッジソースに関連する注釈	
	関連付け	入力データノード、デシジョンノード、ビジネスナレッジモデル、またはナレッジソースからテキストアノテーションへの接続	

以下の表では、DRD 要素間で使用可能なコネクターについてまとめています。

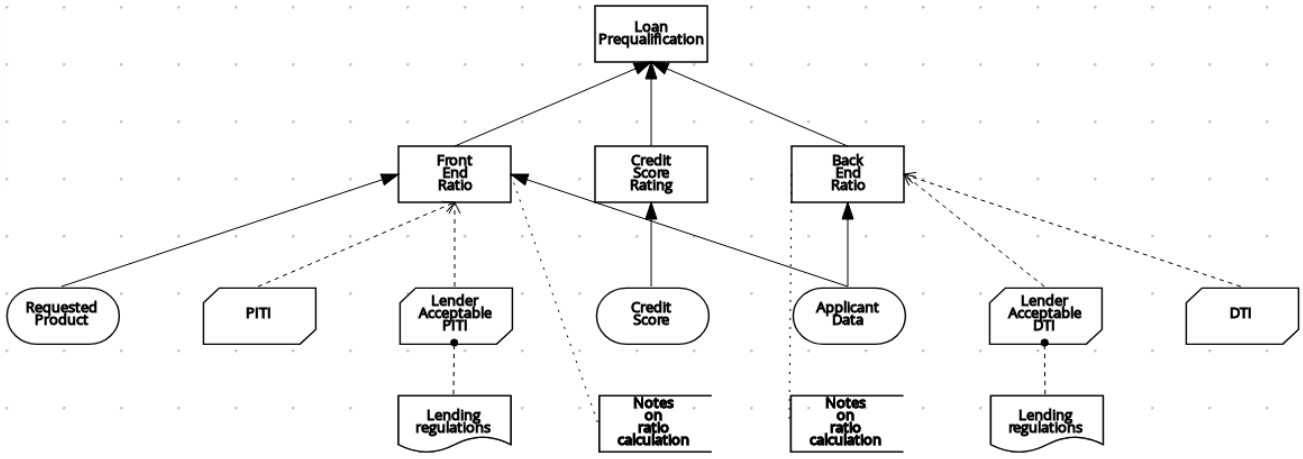
表4.2 DRD コネクタールール

接続元	接続先	接続の種類	例
デシジョン	デシジョン	情報要件	
ビジネスナレッジモデル	デシジョン	ナレッジ要件	
	ビジネスナレッジモデル		
デシジョンサービス	デシジョン	ナレッジ要件	
	ビジネスナレッジモデル		
入力データ	デシジョン	情報要件	

接続元	接続先	接続の種類	例
	ナレッジソース	認証局の要件	
ナレッジソース	デシジョン	認証局の要件	
	ビジネスナレッジモデル		
	ナレッジソース		
デシジョン	テキストのアノテーション	関連付け	
ビジネスナレッジモデル			
ナレッジソース			
入力データ			

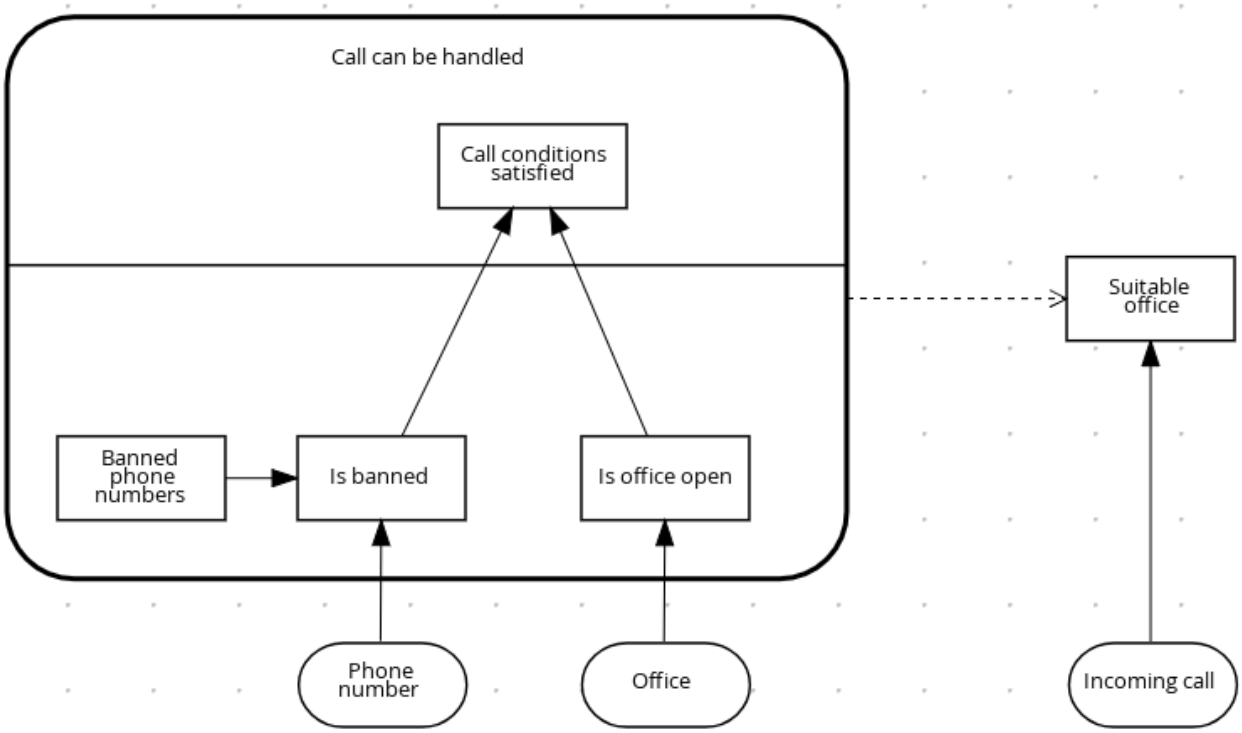
以下の DRD は、これらの DMN コンポーネントの実際の使用例です。

図4.1 DRD 例: ローンの事前審査



以下の DRD は、再利用可能なデシジョンサービスの一部となる DMN コンポーネントについて例示しています。

図4.2 DRD 例: デシジョンサービスとしての電話の対応



DMN デシジョンサービスノードでは、一番下のセグメントデシジョンノードはデシジョンサービス外からの入力データを組み込んで、デシジョンサービスノードにある一番上のセグメントの最終地点に行き着きます。デシジョンサービスから返される上位のデシジョンは、後続のデシジョンまたは DMN モデルのビジネスナレッジ要件に実装されます。他の DMN モデル内の DMN デシジョンサービスを再利用し、異なる入力データや外向け接続で、同じデシジョンロジックを適用します。

4.3. FEEL を使用したルール表現

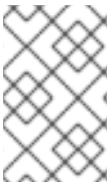
FEEL (Friendly Enough Expression Language) は、オブジェクトマネージメントグループ (OMG: Object Management Group) の DMN 仕様が定義する式言語です。FEEL 式は DMN モデルを使用して、意思決定のロジックを定義します。FEEL は、デシジョンモデル設定概念にセマンティクスを割り当てて、意思決定のモデル化および実行を容易にすることを目的としています。意思決定要件ダイアグラム (DRD) の FEEL 式は、デシジョンノードおよびビジネスナレッジモデルのボックス式のテーブルセルで使用されます。

DMN における FEEL の詳細は、OMG の Decision Model and Notation specification を参照してください。

4.3.1. FEEL のデータ型

FEEL (Friendly Enough Expression Language) では、以下のデータ型がサポートされます。

- 数値
- 文字列
- ブール値
- 日付
- 時間
- 日時
- 日時で指定する期間
- 年および月で指定する期間
- 関数
- コンテキスト
- 範囲 (または間隔)
- リスト



注記

DMN 仕様で変数を **function**、**context**、**range**、または **list** として宣言する明示的な方法はありませんが、Red Hat Process Automation Manager では、これらの種類の変数をサポートするように DMN 組み込み型が拡張されています。

次の一覧では、各データ型を説明します。

数値

数値は、FEEL では IEEE 754-2008 の 10 進法の 128 形式 (34 桁) に基づいています。内部的には、数値は Java の **MathContext DECIMAL128** を持つ **BigDecimals** として表されます。FEEL でサポートされる数値データ型は 1 つしかないため、整数と浮動小数点には同じ型が使用されます。FEEL では、小数点の記号にドット (.) が使用されます。**-INF**、**+INF**、または **NaN** はサポートされません。FEEL では、**null** を使用して、無効な数字を表します。

Red Hat Process Automation Manager では、DMN 仕様が拡張され、以下の数値表記法もサポートされます。

- **科学的記数法**: 接尾辞 **e<exp>** または **E<exp>** を付けて、科学的記数法を使用できます。たとえば、**1.2e3** は **1.2*10**3** と表記するのと同じですが、式ではなくリテラルを使用しています。
- **16 進数**: プリフィックス **0x** を付けて、16 進数を使用できます。たとえば、**0xff** は、10 進数の **255** と同じです。大文字および小文字いずれもサポートされます。たとえば、**0XFF** は **0xff** と同じです。

- **型の接尾辞:** 型の接尾辞として **f**、**F**、**d**、**D**、**l**、**L** を使用できます。この接尾辞は無視されます。

文字列

FEEL では、二重引用符で区切った文字が文字列として解釈されます。

例

```
"John Doe"
```

ブール値

FEEL は、3 値ブール論理を使用するため、ブール論理式には **true**、**false**、または **null** を使用できます。

日付

FEEL では日付リテラルがサポートされていませんが、組み込みの **date()** 関数を使用して日付の値を構築できます。時間文字列は、FEEL では XML Schema Part 2: Datatypes ドキュメントに定義されている形式に準拠します。形式は、**"YYYY-MM-DD"** で、**YYYY** は 4 桁の年数、**MM** は 2 桁の月数、**DD** は日数に置き換えます。
以下に例を示します。

```
date( "2017-06-23" )
```

日付オブジェクトには、真夜中を表す **"00:00:00"** と同一の時間があります。日付には、タイムゾーンがなく、ローカルであると見なされます。

時間

FEEL では時間リテラルがサポートされていませんが、組み込みの **time()** 関数を使用して時間の値を構築できます。時間文字列は、FEEL では XML Schema Part 2: Datatypes ドキュメントで定義されている形式に準拠します。形式は **"hh:mm:ss[.uuu][(+-)hh:mm]"** です。ここで、**hh** は時間 (**00** から **23**)、**mm** は分、**ss** は秒です。任意で、ミリ秒 (**uuu**) を定義でき、UTC 時間の正 (**+**) または負 (**-**) のオフセットを追加してタイムゾーンを定義できます。オフセットを使用する代わりに、**z** 文字を使用して UTC 時間を表すことができますが、これは **-00:00** のオフセットと同じです。オフセットが定義されていない場合には、時間はローカルとみなされます。
例 :

```
time( "04:25:12" )
time( "14:10:00+02:00" )
time( "22:35:40.345-05:00" )
time( "15:00:30z" )
```

オフセットまたはタイムゾーンを定義する時間値は、オフセットまたはタイムゾーンを定義しないローカル時間と比較できません。

日時

FEEL では日時リテラルがサポートされていませんが、組み込みの **date and time()** 関数を使用して値を構築できます。日時文字列は、FEEL では XML Schema Part 2: Datatypes ドキュメントに定義された形式に準拠します。形式は **"<date>T<time>"** です。**<date>** および **<time>** は規定の XML スキーマ形式に準拠し、**T** で結合されます。
例 :

```
date and time( "2017-10-22T23:59:00" )
```

```
date and time( "2017-06-13T14:10:00+02:00" )
date and time( "2017-02-05T22:35:40.345-05:00" )
date and time( "2017-06-13T15:00:30z" )
```

オフセットまたはタイムゾーンを定義する日時の値と、オフセットまたはタイムゾーンを定義しないローカルの日時の値を比較することはできません。

重要

DMN 仕様の実装が、XML スキーマでスペースをサポートしない場合は、キーワード **dateTime** を **date and time** の同義語として使用してください。

日時で指定する期間

FEEL では日と時間で指定する期間を表すリテラルがサポートされていませんが、組み込みの **duration()** 関数を使用して値を構築できます。FEEL では、XML Schema Part 2: Datatypes ドキュメントに定義されている形式に準拠しますが、日、時間、分、および秒にしか適用されません。月および年はサポートされません。
例 :

```
duration( "P1DT23H12M30S" )
duration( "P23D" )
duration( "PT12H" )
duration( "PT35M" )
```

重要

DMN 仕様の実装が、XML スキーマでスペースをサポートしない場合は、キーワード **dayTimeDuration** を **days and time duration** の同義語として使用してください。

年および月で指定する期間

FEEL では年と月で指定する期間リテラルがサポートされていませんが、組み込みの **duration()** 関数を使用して値を構築できます。FEEL では XML Schema Part 2: Datatypes ドキュメントに定義されている形式に準拠しますが、年と月にしか適用されません。日、時間、分、または秒はサポートされません。
例 :

```
duration( "P3Y5M" )
duration( "P2Y" )
duration( "P10M" )
duration( "P25M" )
```

重要

DMN 仕様の実装が、XML スキーマでスペースをサポートしない場合は、キーワード **yearMonthDuration** を **years and months duration** の同義語として使用してください。

関数

FEEL には、関数を作成するのに使用する **function** リテラル (または無名関数) があります。DMN 仕様で変数を **function** として宣言する明示的な方法が提供されていませんが、Red Hat Process Automation Manager では、関数の変数をサポートするように DMN 型が拡張されています。
以下に例を示します。

```
function(a, b) a + b
```

この例では、FEEL 式は、パラメーター **a** および **b** を追加して結果を返す関数を作成します。

コンテキスト

FEEL には、コンテキストを作成するのに使用する **context** リテラルがあります。**context** は、FEEL ではキーと値のペアのリストとなり、Java などの言語におけるマッピングに似ています。DMN 仕様で変数を **context** として宣言する明示的な方法が提供されていませんが、Red Hat Process Automation Manager では、コンテキストの変数をサポートするようにビルドイン DMN 型が拡張されています。
以下に例を示します。

```
{ x : 5, y : 3 }
```

この式では、チャート内の等位を示す 2 つのエントリー (**x** および **y**) を持つコンテキストが作成されます。

DMN 1.2 では、コンテキスト作成に、キーの一覧を属性として含めてアイテム定義を作成し、そのアイテム定義型を含めて変数を宣言する方法も使用できます。

Red Hat Process Automation Manager の DMN API では、**DMNContext** の DMN **ItemDefinition** 構造様式として、次の 2 つがサポートされます。

- ユーザー定義の Java タイプ: DMN の **ItemDefinition** で各コンポーネントのプロパティーとゲッターを定義する有効な JavaBeans オブジェクト。必要に応じて、無効な Java 識別子になるコンポーネント名を示すゲッターに対して **@FEELProperty** アノテーションを使用することもできます。
- **java.util.Map** インターフェイス: DMN の **ItemDefinition** でコンポーネント名に対応するキーで、適切なエンティティーを定義する必要があります。

範囲 (または間隔)

FEEL には、範囲または間隔を作成するのに使用する **range** リテラルがあります。FEEL の **range** は、下方境界および上方境界を定義する値で、開区間または閉区間のいずれかにできます。DMN 仕様には、変数を **range** と宣言する明示的な方法はありませんが、Red Hat Process Automation Manager では、範囲をサポートするようにビルドイン DMN 型が拡張されています。
範囲の構文は以下の形式で定義されます。

```
range          := interval_start endpoint '..' endpoint interval_end
interval_start := open_start | closed_start
open_start     := '(' | ']'
closed_start   := '['
interval_end   := open_end | closed_end
open_end       := ')' | '['
closed_end     := ']'
endpoint       := expression
```

エンドポイントの式は比較可能な値を返す必要があり、下方エンドポイントは上方エンドポイントよりも低くなる必要があります。

たとえば、以下のリテラル式は、**1** から **10** まで (いずれも閉区間) の間隔を定義します。

```
[ 1 .. 10 ]
```

以下のリテラル式は 1 時間から 12 時間までの間隔を定義します。下方境界は含まれます (閉区間) が、上方境界は含まれません (開区間)。

```
[ duration("PT1H") .. duration("PT12H") )
```

デシジョンテーブルの範囲を使用して値の範囲をテストしたり、単純なリテラル式で範囲を使用したりできます。たとえば、以下のリテラル式は、変数 **x** が **0** から **100** の間にある場合は **true** を返します。

```
x in [ 1 .. 100 ]
```

リスト

FEEL には、アイテムの一覧を作成するのに使用する **list** リテラルがあります。FEEL の **list** は、値のコンマ区切りの一覧を角カッコで囲んで表現できます。DMN 仕様には (別の式で使用する以外に) 変数を **list** と宣言する明示的な方法はありませんが、Red Hat Process Automation Manager では、リスト型の変数をサポートするようにビルドイン DMN 型が拡張されています。
以下に例を示します。

```
[ 2, 3, 4, 5 ]
```

FEEL のリストはすべて同じ型の要素を含み、変更できません。リストの要素はインデックスでアクセスでき、最初の要素が **1** になります。負のインデックスは、リストの末尾から数えた要素を表します。たとえば、**-1** は最後の要素にアクセスできることを示します。

たとえば、以下の式は、リスト **x** の 2 番目の要素を返します。

```
x[2]
```

以下の式は、リスト **x** の、最後から 2 番目の要素を返します。

```
x[-2]
```

一覧の要素は、**count** の関数でカウントすることもでき、この関数は、要素の一覧をパラメーターとして使用します。

たとえば、以下の式では **4** を返します。

```
count([ 2, 3, 4, 5 ])
```

4.3.2. FEEL の組み込み関数

他のプラットフォームやシステムとの相互運用性を促進するために、FEEL (Friendly Enough Expression Language) には組み込み関数のライブラリーが含まれています。組み込みの FEEL 機能は、DMN デシジョンサービスで関数を使用できるように、Drools の Decision Model and Notation (DMN) エンジンで実装されています。

以下のセクションでは、**NAME(PARAMETERS)** の形式で記載されている、FEEL の組み込み関数ごとに説明します。DMN における FEEL の詳細は、OMG の Decision Model and Notation specification を参照してください。

4.3.2.1. 変換関数

以下の関数は、異なるタイプの値同士での変換をサポートします。以下の例のように、これらの関数の一部は、特定の文字列形式を使用します。

- **date string**: **2020-06-01** など、XML Schema Part 2: Datatypes ドキュメントで定義されている形式に準拠します。
- **time string**: 以下のいずれかの形式に従います。
  - **23:59:00z** など、XML Schema Part 2: Datatypes ドキュメントに定義されている形式
  - **00:01:00@Etc/UTC** など、ISO 8601 で定義したローカルの時間の後に **@** および IANA タイムゾーンを続けた形式
- **date time string**: **2012-12-25T11:00:00Z** のように、**date string** の後に **T** と **time string** 列が続く形式に従います。
- **duration string**: **P1Y2M** などの XQuery 1.0 and XPath 2.0 Data Model に定義されている **days and time duration** および **years and months duration** の形式に従います

date(from) - date の使用

from を **date** 値に変換します。

表4.3 パラメーター

パラメーター	タイプ	形式
from	**string**	**date string**

例

```
date( "2012-12-25" ) - date( "2012-12-24" ) = duration( "P1D" )
```

date(from) - date and time の使用

値を **from** から **date** に変換し、時間のコンポーネントを null に設定します。

表4.4 パラメーター

パラメーター	タイプ
from	**date and time**

例

```
date(date and time( "2012-12-25T11:00:00Z" )) = date( "2012-12-25" )
```

date(year, month, day)

指定の year、month、および day の値から **date** を生成します。

表4.5 パラメーター

パラメーター	タイプ
year	**number**
month	**number**
day	**number**

例

```
date( 2012, 12, 25 ) = date( "2012-12-25" )
```

date and time(date, time)

指定した日付から **date and time** を生成して、時間のコンポーネントと指定の時間を無視します。

表4.6 パラメーター

パラメーター	タイプ
date	**date** または **date and time**
time	**time**

例

```
date and time ( "2012-12-24T23:59:00" ) = date and time(date( "2012-12-24" ), time( "23:59:00" ))
```

date and time(from)

指定の文字列から **date and time** を生成します。

表4.7 パラメーター

パラメーター	タイプ	形式
from	**string**	**date time string**

例

```
date and time( "2012-12-24T23:59:00" ) + duration( "PT1M" ) = date and time( "2012-12-25T00:00:00" )
```

time(from)

指定の文字列から **time** を生成します。

表4.8 パラメーター

パラメーター	タイプ	形式
from	**string**	**time string**

例

```
time( "23:59:00z" ) + duration( "PT2M" ) = time( "00:01:00@Etc/UTC" )
```

time(from)

指定のパラメーターから **time** を生成し、日付コンポーネントを無視します。

表4.9 パラメーター

パラメーター	タイプ
from	**time** または **date and time**

例

```
time(date and time( "2012-12-25T11:00:00Z" )) = time( "11:00:00Z" )
```

time(hour, minute, second, offset?)

指定した hour、minute、および second のコンポーネント値から **time** を生成します。

表4.10 パラメーター

パラメーター	タイプ
hour	**number**
minute	**number**
second	**number**
offset (任意)	**days and time duration** または null

例

```
time( "23:59:00z" ) = time(23, 59, 0, duration( "PT0H" ))
```

number(from, grouping separator, decimal separator)

指定の区切り文字を使用して **from** を **number** に変換します。

表4.11 パラメーター

パラメーター	タイプ
from	有効な数字を表す **string**
grouping separator	スペース ( )、コンマ (**,**)、ピリオド (**.**)、または null
decimal separator	**grouping separator** と同じタイプですが、同じ値は使用できません。

例

```
number( "1 000,0", " ", "," ) = number( "1,000.0", ",", "." )
```

string(from)

指定のパラメーターを文字列表現にします。

表4.12 パラメーター

パラメーター	タイプ
from	null 値以外の値

例

```
string( 1.1 ) = "1.1"
string( null ) = null
```

duration(from)

from を **days and time duration**、または **years and months duration** に変換します。

表4.13 パラメーター

パラメーター	タイプ	形式
from	**string**	**duration string**

例

```
date and time( "2012-12-24T23:59:00" ) - date and time( "2012-12-22T03:45:00" ) = duration(
"P2DT20H14M" )
duration( "P2Y2M" ) = duration( "P26M" )
```

years and months duration(from, to)

指定した 2 つのパラメーター間の **years and months duration** を計算します。

表4.14 パラメーター

パラメーター	タイプ
from	**date** または **date and time**
to	**date** または **date and time**

例

```
years and months duration( date( "2011-12-22" ), date( "2013-08-24" ) ) = duration( "P1Y8M" )
```

4.3.2.2. ブール値関数

以下の関数は、ブール値の操作をサポートします。

not(negand)

negand オペランドの論理否定を実行します。

表4.15 パラメーター

パラメーター	タイプ
negand	**boolean**

例

```
not( true ) = false
not( null ) = null
```

4.3.2.3. 文字列関数

以下の関数は、文字列の操作をサポートします。

注記

FEEL では、Unicode 文字はコードポイントを基にカウントされます。

substring(string, start position, length?)

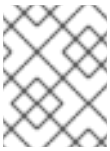
指定の長さの開始地点からサブ文字列を返します。最初の文字の位置値は **1** です。

表4.16 パラメーター

パラメーター	タイプ
string	**string**
start position	**number**
length (任意)	**number**

例

```
substring( "testing",3 ) = "sting"
substring( "testing",3,3 ) = "sti"
substring( "testing", -2, 1 ) = "n"
substring( "\U01F40Eab", 2 ) = "ab"
```



注記

FEEL では、文字リテラルの **"\U01F40Eab"** は　**ab** 文字列 (馬の記号の後に **a** と **b**) となります。

string length(string)

指定の文字列の長さを計算します。

表4.17 パラメーター

パラメーター	タイプ
string	**string**

例

```
string length( "tes" ) = 3
string length( "\U01F40Eab" ) = 3
```

upper case(string)

指定の文字列の大文字バージョンを生成します。

表4.18 パラメーター

パラメーター	タイプ
string	**string**

例

```
upper case( "aBc4" ) = "ABC4"
```

lower case(string)

指定の文字列の小文字バージョンを生成します。

表4.19 パラメーター

パラメーター	タイプ
string	**string**

例

```
lower case( "aBc4" ) = "abc4"
```

substring before(string, match)

一致した値の前にあるサブ文字列を計算します。

表4.20 パラメーター

パラメーター	タイプ
string	**string**
match	**string**

例

```
substring before( "testing", "ing" ) = "test"
substring before( "testing", "xyz" ) = ""
```

substring after(string, match)

一致した値の後にあるサブ文字列を計算します。

表4.21 パラメーター

パラメーター	タイプ
string	**string**
match	**string**

例

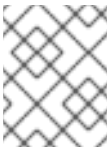
```
substring after( "testing", "test" ) = "ing"
substring after( "", "a" ) = ""
```

replace(input, pattern, replacement, flags?)

正規表現の置換を計算します。

表4.22 パラメーター

パラメーター	タイプ
input	**string**
pattern	**string**
replacement	**string**
flags (任意)	**string**



注記

この関数は、XQuery 1.0 and XPath 2.0 Functions and Operators で定義されている正規表現パラメーターを使用します。

例

```
replace( "abcd", "(ab)|(a)", "[1=$1][2=$2]" ) = "[1=ab][2=]cd"
```

contains(string, match)

文字列に一致部分が含まれる場合に **true** を返します。

表4.23 パラメーター

パラメーター	タイプ
string	**string**
match	**string**

例

```
contains( "testing", "to" ) = false
```

starts with(string, match)

指定の値と一致する文字列で開始する場合に、**true** を返します。

表4.24 パラメーター

パラメーター	タイプ
string	**string**
match	**string**

例

```
starts with( "testing", "te" ) = true
```

ends with(string, match)

指定の値と一致する文字列で終了する場合に、**true** を返します。

表4.25 パラメーター

パラメーター	タイプ
string	**string**
match	**string**

例

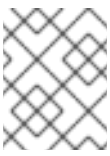
```
ends with( "testing", "g" ) = true
```

matches(input, pattern, flags?)

入力が正規表現と一致する場合に **true** を返します。

表4.26 パラメーター

パラメーター	タイプ
input	**string**
pattern	**string**
flags (任意)	**string**



注記

この関数は、XQuery 1.0 and XPath 2.0 Functions and Operators で定義されている正規表現パラメーターを使用します。

例

```
matches( "teeesting", "^te*sting" ) = true
```

split(string, delimiter)

元の文字列のリストを返し、区切り文字の正規表現パターンで分割します。

表4.27 パラメーター

パラメーター	タイプ
string	**string**
delimiter	正規表現パターンの **string**

注記

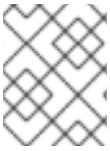
この関数は、XQuery 1.0 and XPath 2.0 Functions and Operators で定義されている正規表現パラメーターを使用します。

例

```
split( "John Doe", "\\s" ) = ["John", "Doe"]
split( "a;b;c;;", ";" ) = ["a","b","c","",""]
```

4.3.2.4. リスト関数

以下の関数は、リストの操作をサポートします。



注記

FEEL では、リストに含まれる最初の要素のインデックスは **1** となります。リストに含まれる最後の要素のインデックスは **-1** として特定できます。

list contains(list, element)

リストに対象の要素が含まれる場合には **true** を返します。

表4.28 パラメーター

パラメーター	タイプ
list	**list**
element	null を含むすべてのタイプ

例

```
list contains( [1,2,3], 2 ) = true
```

count(list)

リスト内の要素をカウントします。

表4.29 パラメーター

パラメーター	タイプ
list	**list**

例

```
count( [1,2,3] ) = 3
count( [] ) = 0
count( [1,[2,3]] ) = 2
```

min(list)

一覧の値と同じ最小の要素を返します。

表4.30 パラメーター

パラメーター	タイプ
list	**list**

別の署名

```
min( e1, e2, ..., eN )
```

例

```
min( [1,2,3] ) = 1
min( 1 ) = 1
min( [1] ) = 1
```

max(list)

一覧の値と同じ最大の要素を返します。

表4.31 パラメーター

パラメーター	タイプ
list	**list**

別の署名

```
max( e1, e2, ..., eN )
```

例

```
max( 1,2,3 ) = 3
max( [] ) = null
```

-

sum(list)

リスト内の数字の合計を返します。

表4.32 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
sum( n1, n2, ..., nN )
```

例

```
sum( [1,2,3] ) = 6
sum( 1,2,3 ) = 6
sum( 1 ) = 1
sum( [] ) = null
```

mean(list)

リスト内の要素の平均 (計算平均) を計算します。

表4.33 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
mean( n1, n2, ..., nN )
```

例

```
mean( [1,2,3] ) = 2
mean( 1,2,3 ) = 2
mean( 1 ) = 1
mean( [] ) = null
```

all(list)

リスト内の全要素が true の場合は **true** を返します。

表4.34 パラメーター

パラメーター	タイプ
list	**boolean** 要素の **list**

別の署名

```
all( b1, b2, ..., bN )
```

例

```
all( [false,null,true] ) = false
all( true ) = true
all( [true] ) = true
all( [] ) = true
all( 0 ) = null
```

any(list)

リスト内の要素が true の場合は **true** を返します。

表4.35 パラメーター

パラメーター	タイプ
list	**boolean** 要素の **list**

別の署名

```
any( b1, b2, ..., bN )
```

例

```
any( [false,null,true] ) = true
any( false ) = false
any( [] ) = false
any( 0 ) = null
```

sublist(list, start position, length?)

開始位置からサブリストを返します。ただし、length 要素に限定されます。

表4.36 パラメーター

パラメーター	タイプ
list	**list**
start position	**number**

パラメーター	タイプ
length (任意)	**number**

例

```
sublist( [4,5,6], 1, 2 ) = [4,5]
```

append(list, item)

アイテムに追加されるリストを作成します。

表4.37 パラメーター

パラメーター	タイプ
list	**list**
item	任意のタイプ

例

```
append( [1], 2, 3 ) = [1,2,3]
```

concatenate(list)

連結された一覧の結果で一覧を作成します。

表4.38 パラメーター

パラメーター	タイプ
list	**list**

例

```
concatenate( [1,2],[3] ) = [1,2,3]
```

insert before(list, position, newItem)

指定の位置に挿入された **newItem** でリストを作成します。

表4.39 パラメーター

パラメーター	タイプ
list	**list**
position	**number**

パラメーター	タイプ
newItem	任意のタイプ

例

```
insert before( [1,3],1,2 ) = [2,1,3]
```

remove(list, position)

指定の位置から除外された要素を削除して一覧を作成します。

表4.40 パラメーター

パラメーター	タイプ
list	**list**
position	**number**

例

```
remove( [1,2,3], 2 ) = [1,3]
```

reverse(list)

逆リストを返します。

表4.41 パラメーター

パラメーター	タイプ
list	**list**

例

```
reverse( [1,2,3] ) = [3,2,1]
```

index of(list, match)

要素に一致するインデックスを返します。

パラメーター

- **list** タイプの **list**
- 任意のタイプの **match**

表4.42 パラメーター

パラメーター	タイプ
list	**list**
match	任意のタイプ

例

```
index of( [1,2,3,2],2 ) = [2,4]
```

union(list)

複数のリストから全要素の一覧を返し、重複を除外します。

表4.43 パラメーター

パラメーター	タイプ
list	**list**

例

```
union( [1,2],[2,3] ) = [1,2,3]
```

distinct values(list)

単一リストから要素の一覧を返し、重複を除外します。

表4.44 パラメーター

パラメーター	タイプ
list	**list**

例

```
distinct values( [1,2,3,2,1] ) = [1,2,3]
```

flatten(list)

フラット化されたリストを返します。

表4.45 パラメーター

パラメーター	タイプ
list	**list**

例

```
flatten( [[1,2],[[3]], 4] ) = [1,2,3,4]
```

product(list)

リスト内の数字の積を返します。

表4.46 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
product( n1, n2, ..., nN )
```

例

```
product( [2, 3, 4] ) = 24
product( 2, 3, 4 ) = 24
```

median(list)

リストの数字の中央値を返します。要素の数が奇数の場合、結果は中央の要素になります。要素の数が偶数の場合、結果は中央にある 2 つの要素の平均になります。

表4.47 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
median( n1, n2, ..., nN )
```

例

```
median( 8, 2, 5, 3, 4 ) = 4
median( [6, 1, 2, 3] ) = 2.5
median( [ ] ) = null
```

stddev(list)

リストの数値の標準偏差を返します。

表4.48 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
stddev( n1, n2, ..., nN )
```

例

```
stddev( 2, 4, 7, 5 ) = 2.0816659994661327352822977069799 31
stddev( [47] ) = null
stddev( 47 ) = null
stddev( [ ] ) = null
```

mode(list)

リスト内の数字の最頻値を返します。複数の要素が返される場合、番号は昇順でソートされます。

表4.49 パラメーター

パラメーター	タイプ
list	**number** 要素の **list**

別の署名

```
mode( n1, n2, ..., nN )
```

例

```
mode( 6, 3, 9, 6, 6 ) = [6]
mode( [6, 1, 9, 6, 1] ) = [1, 6]
mode( [ ] ) = [ ]
```

4.3.2.5. 数値関数

以下の関数は、数値演算をサポートしています。

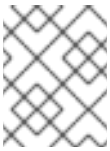
decimal(n, scale)

指定されたスケールの数値を返します。

表4.50 パラメーター

パラメーター	タイプ
n	**number**

パラメーター	タイプ
scale	範囲の **[–6111..6176]** の **number**



注記

この関数は、少数値を最も近い 10 進数の数値に丸める **FEEL:number** 定義と合致するように実装されます。

例

```
decimal( 1/3, 2 ) = .33
decimal( 1.5, 0 ) = 2
decimal( 2.5, 0 ) = 2
decimal( 1.035, 2 ) = 1.04
decimal( 1.045, 2 ) = 1.04
decimal( 1.055, 2 ) = 1.06
decimal( 1.065, 2 ) = 1.06
```

floor(n)

指定された数値以下の最大の整数を返します。

表4.51 パラメーター

パラメーター	タイプ
n	**number**

例

```
floor( 1.5 ) = 1
floor( -1.5 ) = -2
```

ceiling(n)

指定された数値以上の最小の整数を返します。

表4.52 パラメーター

パラメーター	タイプ
n	**number**

例

```
ceiling( 1.5 ) = 2
ceiling( -1.5 ) = -1
```

abs(n)

絶対値を返します。

表4.53 パラメーター

パラメーター	タイプ
n	**number**、**days and time duration**、または **years and months duration**

例

```
abs( 10 ) = 10
abs( -10 ) = 10
abs( @"PT5H" ) = @"PT5H"
abs( @"-PT5H" ) = @"PT5H"
```

modulo(dividend, divisor)

除数による被除数の除算の余りを返します。被除数または除数のいずれかが負の場合、結果は除数と同じ符号になります。

注記

この関数は、**modulo(dividend, divisor) = dividend - divisor*floor(dividend/divisor)** としても表されます。

表4.54 パラメーター

パラメーター	タイプ
dividend	**number**
divisor	**number**

例

```
modulo( 12, 5 ) = 2
modulo( -12,5 )= 3
modulo( 12,-5 )= -3
modulo( -12,-5 )= -2
modulo( 10.1, 4.5 )= 1.1
modulo( -10.1, 4.5 )= 3.4
modulo( 10.1, -4.5 )= -3.4
modulo( -10.1, -4.5 )= -1.1
```

sqrt(number)

指定された数値の平方根を返します。

表4.55 パラメーター

パラメーター	タイプ
n	**number**

例

```
sqrt( 16 ) = 4
```

log(number)

指定された数値の対数を返します。

表4.56 パラメーター

パラメーター	タイプ
n	**number**

例

```
decimal( log( 10 ), 2 ) = 2.30
```

exp(number)

オイラーの数値 **e** を、指定された数の累乗で返します。

表4.57 パラメーター

パラメーター	タイプ
n	**number**

例

```
decimal( exp( 5 ), 2 ) = 148.41
```

odd(number)

指定された数が奇数の場合は **true** を返します。

表4.58 パラメーター

パラメーター	タイプ
n	**number**

例

```
odd( 5 ) = true
odd( 2 ) = false
```

even(number)

指定された数が偶数の場合は **true** を返します。

表4.59 パラメーター

パラメーター	タイプ
n	**number**

例

```
even( 5 ) = false
even ( 2 ) = true
```

4.3.2.6. 日付および時刻の関数

以下の関数は、日付および時刻の演算をサポートしています。

is(value1, value2)

両方の値が FEEL セマンティックドメインの同じ要素である場合は、**true** を返します。

表4.60 パラメーター

パラメーター	タイプ
value1	任意のタイプ
value2	任意のタイプ

例

```
is( date( "2012-12-25" ), time( "23:00:50" ) ) = false
is( date( "2012-12-25" ), date( "2012-12-25" ) ) = true
is( time( "23:00:50z" ), time( "23:00:50" ) ) = false
```

4.3.2.7. リスト関数

次の関数は、単一のスカラー値とそのような値の範囲の間の関係を確立するための時間的順序付け操作をサポートします。これらの関数は、Health Level Seven (HL7) International の Clinical Quality Language (CQL) 1.4 syntax のコンポーネントに似ています。

before( )

要素 **A** が要素 **B** の前にあり、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **before(point1 point2)**

b. **before(point range)**

c. **before(range point)**

d. **before(range1,range2)**

true に評価するための要件

a. **point1 < point2**

b. **point < range.start or (point = range.start and not(range.start included))**

c. **range.end < point or (range.end = point and not(range.end included))**

d. **range1.end < range2.start or ((not(range1.end included) or not(range2.start included)) and range1.end = range2.start)**

例

```
before( 1, 10 ) = true
before( 10, 1 ) = false
before( 1, [1..10] ) = false
before( 1, (1..10] ) = true
before( 1, [5..10] ) = true
before( [1..10], 10 ) = false
before( [1..10), 10 ) = true
before( [1..10], 15 ) = true
before( [1..10], [15..20] ) = true
before( [1..10], [10..20] ) = false
before( [1..10), [10..20] ) = true
before( [1..10], (10..20] ) = true
```

after( )

要素 **A** が要素 **B** の後にあり、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **after(point1 point2)**

b. **after(point range)**

c. **after(range, point)**

d. **after(range1 range2)**

true に評価するための要件

a. **point1 > point2**

b. **point > range.end or (point = range.end and not(range.end included))**

c. **range.start > point or (range.start = point and not(range.start included))**

d. **range1.start > range2.end or ((not(range1.start included) or not(range2.end included)) and range1.start = range2.end)**

例

```
after( 10, 5 ) = true
after( 5, 10 ) = false
after( 12, [1..10] ) = true
after( 10, [1..10) ) = true
after( 10, [1..10] ) = false
after( [11..20], 12 ) = false
after( [11..20], 10 ) = true
after( (11..20], 11 ) = true
after( [11..20], 11 ) = false
after( [11..20], [1..10] ) = true
after( [1..10], [11..20] ) = false
after( [11..20], [1..11) ) = true
after( (11..20], [1..11] ) = true
```

meets( )

要素 **A** が要素 **B** を満たし、**true** に評価されるための関連要件も満たされた場合は **true** を返します。

署名

a. **meets(range1, range2)**

true に評価するための要件

a. **range1.end included and range2.start included and range1.end = range2.start**

例

```
meets( [1..5], [5..10] ) = true
meets( [1..5), [5..10] ) = false
meets( [1..5], (5..10] ) = false
meets( [1..5], [6..10] ) = false
```

met by( )

要素 **A** が要素 **B** によって満たされ、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **met by(range1, range2)**

true に評価するための要件

a. **range1.start included and range2.end included and range1.start = range2.end**

例

```
met by( [5..10], [1..5] ) = true
met by( [5..10], [1..5) ) = false
met by( (5..10], [1..5] ) = false
met by( [6..10], [1..5] ) = false
```

overlaps( )

要素 **A** が要素 **B** とオーバーラップし、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **overlaps(range1, range2)**

true に評価するための要件

a. **(range1.end > range2.start or (range1.end = range2.start and (range1.end included or range2.end included))) and (range1.start < range2.end or (range1.start = range2.end and range1.start included and range2.end included))**

例

```
overlaps( [1..5], [3..8] ) = true
overlaps( [3..8], [1..5] ) = true
overlaps( [1..8], [3..5] ) = true
overlaps( [3..5], [1..8] ) = true
overlaps( [1..5], [6..8] ) = false
overlaps( [6..8], [1..5] ) = false
overlaps( [1..5], [5..8] ) = true
overlaps( [1..5], (5..8] ) = false
overlaps( [1..5), [5..8] ) = false
overlaps( [1..5), (5..8] ) = false
overlaps( [5..8], [1..5] ) = true
overlaps( (5..8], [1..5] ) = false
overlaps( [5..8], [1..5) ) = false
overlaps( (5..8], [1..5) ) = false
```

overlaps before( )

要素 **A** が要素 **B** の前にオーバーラップし、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **overlaps before(range1 range2)**

true に評価するための要件

a. **(range1.start < range2.start or (range1.start = range2.start and range1.start included and range2.start included)) and (range1.end > range2.start or (range1.end = range2.start and range1.end included and range2.start included)) and (range1.end < range2.end or (range1.end = range2.end and (not(range1.end included) or range2.end included)))**

例

```
overlaps before( [1..5], [3..8] ) = true
overlaps before( [1..5], [6..8] ) = false
overlaps before( [1..5], [5..8] ) = true
overlaps before( [1..5], (5..8] ) = false
overlaps before( [1..5), [5..8] ) = false
overlaps before( [1..5), (1..5] ) = true
overlaps before( [1..5], (1..5] ) = true
overlaps before( [1..5), [1..5] ) = false
overlaps before( [1..5], [1..5] ) = false
```

overlaps after( )

要素 **A** が要素 **B** の後にオーバーラップする場合に、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **overlaps after(range1 range2)**

true に評価するための要件

a. **(range2.start < range1.start or (range2.start = range1.start and range2.start included and not(range1.start included))) and (range2.end > range1.start or (range2.end = range1.start and range2.end included and range1.start included)) and (range2.end < range1.end or (range2.end = range1.end and (not(range2.end included) or range1.end included)))**

例

```
overlaps after( [3..8], [1..5] )= true
overlaps after( [6..8], [1..5] )= false
overlaps after( [5..8], [1..5] )= true
overlaps after( (5..8], [1..5] )= false
overlaps after( [5..8], [1..5) )= false
overlaps after( (1..5], [1..5) )= true
overlaps after( (1..5], [1..5] )= true
overlaps after( [1..5], [1..5) )= false
overlaps after( [1..5], [1..5] )= false
overlaps after( (1..5), [1..5] )= false
overlaps after( (1..5], [1..6] )= false
overlaps after( (1..5], (1..5] )= false
overlaps after( (1..5], [2..5] )= false
```

finishes( )

要素 **A** が要素 **B** を終了し、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **finishes(point, range)**

b. **finishes(range1, range2)**

true に評価するための要件

a. **range.end included and range.end = point**

b. **range1.end included = range2.end included and range1.end = range2.end and (range1.start > range2.start or (range1.start = range2.start and (not(range1.start included) or range2.start included)))**

例

```
finishes( 10, [1..10] ) = true
finishes( 10, [1..10) ) = false
finishes( [5..10], [1..10] ) = true
finishes( [5..10), [1..10] ) = false
finishes( [5..10), [1..10) ) = true
finishes( [1..10], [1..10] ) = true
finishes( (1..10], [1..10] ) = true
```

finished by( )

要素 **A** が要素 **B** によって終了し、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **finished by(range, point)**

b. **finished by(range1 range2)**

true に評価するための要件

a. **range.end included and range.end = point**

b. **range1.end included = range2.end included and range1.end = range2.end and (range1.start < range2.start or (range1.start = range2.start and (range1.start included or not(range2.start included))))**

例

```
finished by( [1..10], 10 ) = true
finished by( [1..10), 10 ) = false
finished by( [1..10], [5..10] ) = true
finished by( [1..10], [5..10) ) = false
finished by( [1..10), [5..10) ) = true
finished by( [1..10], [1..10] ) = true
finished by( [1..10], (1..10] ) = true
```

includes( )

要素 **A** が要素 **B** を含み、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **includes(range, point)**

b. **includes(range1, range2)**

true に評価するための要件

a. **(range.start < point and range.end > point) or (range.start = point and range.start included) or (range.end = point and range.end included)**

b. **(range1.start < range2.start or (range1.start = range2.start and (range1.start included or not(range2.start included)))) and (range1.end > range2.end or (range1.end = range2.end and (range1.end included or not(range2.end included))))**

例

```
includes( [1..10], 5 ) = true
includes( [1..10], 12 ) = false
includes( [1..10], 1 ) = true
includes( [1..10], 10 ) = true
includes( (1..10], 1 ) = false
includes( [1..10), 10 ) = false
includes( [1..10], [4..6] ) = true
includes( [1..10], [1..5] ) = true
includes( (1..10], (1..5] ) = true
includes( [1..10], (1..10) ) = true
includes( [1..10), [5..10) ) = true
includes( [1..10], [1..10) ) = true
includes( [1..10], (1..10] ) = true
includes( [1..10], [1..10] ) = true
```

during( )

要素 **A** が要素 **B** の間にあり、 **true** に評価するための関連要件も満たされた場合は **true** を返します。

署名

a. **during(point, range)**

b. **during(range1 range2)**

true に評価するための要件

a. **(range.start < point and range.end > point) or (range.start = point and range.start included) or (range.end = point and range.end included)**

b. **(range2.start < range1.start or (range2.start = range1.start and (range2.start included or not(range1.start included)))) and (range2.end > range1.end or (range2.end = range1.end and (range2.end included or not(range1.end included))))**

例

```
during( 5, [1..10] ) = true
during( 12, [1..10] ) = false
during( 1, [1..10] ) = true
during( 10, [1..10] ) = true
during( 1, (1..10] ) = false
during( 10, [1..10) ) = false
during( [4..6], [1..10] ) = true
during( [1..5], [1..10] ) = true
```

```
during( (1..5], (1..10] ) = true
during( (1..10), [1..10] ) = true
during( [5..10), [1..10) ) = true
during( [1..10), [1..10] ) = true
during( (1..10], [1..10] ) = true
during( [1..10], [1..10] ) = true
```

starts( )

要素 **A** が要素 **B** を開始し、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **starts(point, range)**

b. **starts(range1, range2)**

true に評価するための要件

a. **range.start = point and range.start included**

b. **range1.start = range2.start and range1.start included = range2.start included and (range1.end < range2.end or (range1.end = range2.end and (not(range1.end included) or range2.end included)))**

例

```
starts( 1, [1..10] ) = true
starts( 1, (1..10] ) = false
starts( 2, [1..10] ) = false
starts( [1..5], [1..10] ) = true
starts( (1..5], (1..10] ) = true
starts( (1..5], [1..10] ) = false
starts( [1..5], (1..10] ) = false
starts( [1..10], [1..10] ) = true
starts( [1..10), [1..10] ) = true
starts( (1..10), (1..10) ) = true
```

started by( )

要素 **A** が要素 **B** によって開始し、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **started by(range, point)**

b. **started by(range1, range2)**

true に評価するための要件

a. **range.start = point and range.start included**

b. **range1.start = range2.start and range1.start included = range2.start included and (range2.end < range1.end or (range2.end = range1.end and (not(range2.end included) or range1.end included)))**

例

```
started by( [1..10], 1 ) = true
started by( (1..10], 1 ) = false
started by( [1..10], 2 ) = false
started by( [1..10], [1..5] ) = true
started by( (1..10], (1..5] ) = true
started by( [1..10], (1..5] ) = false
started by( (1..10], [1..5] ) = false
started by( [1..10], [1..10] ) = true
started by( [1..10], [1..10) ) = true
started by( (1..10), (1..10) ) = true
```

coincides( )

要素 **A** が要素 **B** に一致し、評価が **true** になるための関連要件も満たされた場合は **true** を返します。

署名

a. **coincides(point1, point2)**

b. **coincides(range1, range2)**

true に評価するための要件

a. **point1 = point2**

b. **range1.start = range2.start and range1.start included = range2.start included and range1.end = range2.end and range1.end included = range2.end included**

例

```
coincides( 5, 5 ) = true
coincides( 3, 4 ) = false
coincides( [1..5], [1..5] ) = true
coincides( (1..5), [1..5] ) = false
coincides( [1..5], [2..6] ) = false
```

4.3.2.8. 時間関数

以下の関数は、一般的な時間演算をサポートしています。

day of year(date)

その年のその日のグレゴリオ暦の数値を返します。

表4.61 パラメーター

パラメーター	タイプ
date	**date** または **date and time**

例

```
day of year( date(2019, 9, 17) ) = 260
```

day of week(date)

グレゴリオ暦の曜日 (**"Monday"**、**"Tuesday"**、**"Wednesday"**、**"Thursday"**、**"Friday"**、**"Saturday"**、または **"Sunday"**) を返します。

表4.62 パラメーター

パラメーター	タイプ
date	**date** または **date and time**

例

```
day of week( date(2019, 9, 17) ) = "Tuesday"
```

month of year(date)

グレゴリオ暦の月 (**"January"**、**"February"**、**"March"**、**"April"**、**"May"**、**"June"**、**"July"**、**"August"**、**"September"**、**"October"**、**"November"**、または **"December"**) を返します。

表4.63 パラメーター

パラメーター	タイプ
date	**date** または **date and time**

例

```
month of year( date(2019, 9, 17) ) = "September"
```

month of year(date)

ISO 8601 で定義されているグレゴリオ暦の週を返します。

表4.64 パラメーター

パラメーター	タイプ

パラメーター	タイプ
date	**date** または **date and time**

例

```
week of year( date(2019, 9, 17) ) = 38
week of year( date(2003, 12, 29) ) = 1
week of year( date(2004, 1, 4) ) = 1
week of year( date(2005, 1, 1) ) = 53
week of year( date(2005, 1, 3) ) = 1
week of year( date(2005, 1, 9) ) = 1
```

4.3.2.9. ソート関数

次の関数は、並べ替え操作をサポートしています。

sort(list, precedes)

同じ要素のリストを返しますが、ソート関数に従って順序付けされます。

表4.65 パラメーター

パラメーター	タイプ
list	**list**
precedes	**function**

例

```
sort( list: [3,1,4,5,2], precedes: function(x,y) x < y ) = [1,2,3,4,5]
```

4.3.2.10. コンテキスト関数

以下の関数は、コンテキスト操作をサポートします。

get value(m, key)

指定のエントリーキーのコンテキストから値を返します。

表4.66 パラメーター

パラメーター	タイプ
m	**context**
key	**string**

例

```
get value( {key1 : "value1"}, "key1" ) = "value1"
get value( {key1 : "value1"}, "unexistent-key" ) = null
```

get entries(m)

指定されたコンテキストのキーと値のペアの一覧を返します。

表4.67 パラメーター

パラメーター	タイプ
m	**context**

例

```
get entries( {key1 : "value1", key2 : "value2"} ) = [ { key : "key1", value : "value1" }, {key : "key2", value : "value2"} ]
```

4.3.3. FEEL の変数および関数名

従来の多くの式言語と異なり、FEEL (Friendly Enough Expression Language) は、変数および関数名でスペースと少数の特殊文字をサポートします。FEEL 名は **文字**、**?**、または _ の要素で始める必要があります。ユニコード文字も使用できます。変数名は、言語キーワード (**and**、**true**、**every** など) で開始することはできません。先頭以外には **(複数桁の) 数値**、空白文字、特殊文字 (**+**、**-**、/、*、'、**.** など) を使用できます。

たとえば、以下の名前はすべて有効な FEEL 名です。

- Age
- Birth Date
- Flight 234 pre-check procedure

FEEL の変数名および関数名には、いくつかの制約が適用されます。

曖昧性 (多義性)

名前の一部に、スペース、キーワード、およびその他の特殊文字を使用して FEEL に多義性を持たせることができます。この多義性は、式のコンテキストで左から右に名前を一致させて解決されます。パーサーは、変数名を、その範囲に一致する中で一番長い名前に解決します。必要に応じて、**( )** を使用して名前の多義性を排除できます。

名前で使用されるスペース

DMN 仕様では、FEEL 名におけるスペース使用を制限します。DMN 仕様によると、名前には複数のスペースを使用できますが、連続して使用することはできません。
言語を使いやすく、スペース使用に関するよくある誤りを回避するために、Red Hat Process Automation Manager では、スペースを連続して使用する制限が取り除かれています。Red Hat Process Automation Manager では、スペースを連続して使用する変数名がサポートされますが、スペースを連続して使用してもスペースの数は 1 つに正規化されます。Red Hat Process Automation Manager の変数参照では、**First Name** (スペース 1 つ) および **First Name** (スペース 2 つ) の両方が使用できます。

また、Red Hat Process Automation Manager では、Web ページ、タブ、改行でよく見られる分割できない空白文字などの使用を正規化します。Red Hat Process Automation Manager の FEEL エンジンの観点では、このような文字はすべて、処理される前に 1 つの空白文字に正規化されます。

キーワードの in

キーワードの **in** は、この言語の中で、唯一変数名に使用できないキーワードです。仕様では、変数名にキーワードを使用できますが、変数名に **in** を使用すると、**for**、**every**、**some** の各表現概念と矛盾します。

4.4. ボックス式の DMN デシジョンロジック

DMN のボックス式は、意思決定要件ダイアグラム (DRD) でデシジョンノードおよびビジネスナレッジモデルの基盤ロジックを定義するのに使用するテーブルです。ボックス式には他のボックス式が含まれる場合がありますが、トップレベルのボックス式は単一の DRD アーティファクトのデシジョンロジックに対応します。DRD は DMN デシジョンモデルのフローを表現し、ボックス式は個別ノードの実際のデシジョンロジックを定義します。DRD とボックス式は、完全で機能的な DMN デシジョンモデルを形成します。

以下は、DMN のボックス式の種類です。

- デシジョンテーブル
- リテラル式
- コンテキスト
- 関係
- 関数
- 呼び出し
- リスト

注記

Red Hat Process Automation Manager では、Business Central にボックスリスト式が含まれていませんが、FEEL **list** のデータ型が含まれているためボックスリテラル式で使用できます。Red Hat Process Automation Manager に含まれる **list** のデータ型およびその他の FEEL データ型については、「FEEL のデータ型」 を参照してください。

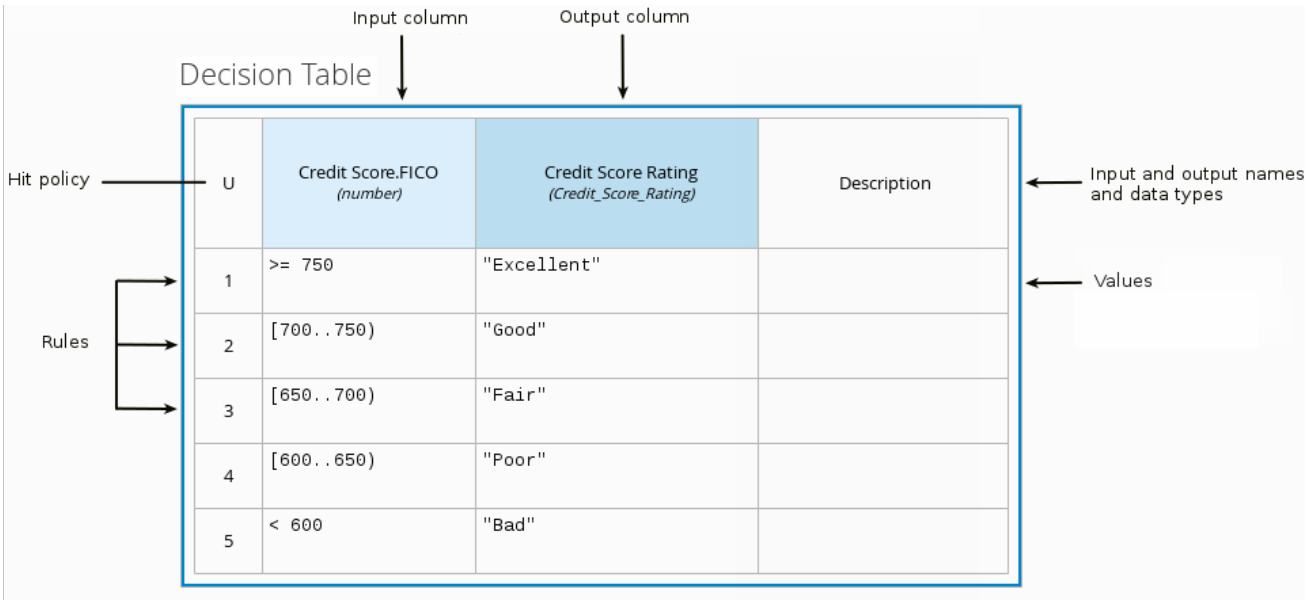
ボックス式で使用する Friendly Enough Expression Language (FEEL) 式はすべて、OMG の Decision Model and Notation specification に記載されている FEEL 構文の要件に準拠する必要があります。

4.4.1. DMN デシジョンテーブル

DMN のデシジョンテーブルは、1 つ以上のビジネスルールをテーブル形式で視覚的に表します。デシジョンテーブルを使用して、デシジョンモデルの特定の地点でこれらのルールを適用するデシジョンノードのルールを定義します。テーブルの各行はルール 1 つで設定されており、その特定行に対する条件 (入力) と結果 (出力) を定義する列が含まれます。各行の定義は、条件の値を使用して結果を取得できるほど正確です。入力と出力の値には、FEEL 式または定義済みのデータ型の値を指定できます。

たとえば、以下のデシジョンテーブルでは、ローン申請者のクレジットスコアの定義範囲に基づき、クレジットスコアを評価します。

図4.3 クレジットスコア評価のデシジョンテーブル



以下のデシジョンテーブルでは、申請者の借り入れ資格や Berueu Call Type に従い、申請者の融資戦略における次のステップを決定します。

図4.4 融資戦略のデシジョンテーブル

Strategy *(Decision Table)*

U	Eligibility *(string)*	BureauCallType *(string)*	Strategy *(tStrategy)*	Description
1	"INELIGIBLE"	-	"DECLINE"	Disregard BureauCallType when ineligible.
2	"ELIGIBLE"	"FULL","MINI"	"BUREAU"	
3	"ELIGIBLE"	"NONE"	"THROUGH"	

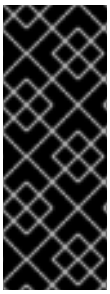
以下のデシジョンテーブルでは、ローン事前審査のデシジョンモデルで終端デシジョンノードとして、申請者のローン適正を決定します。

図4.5 ローン事前審査のデシジョンテーブル

Loan Pre-Qualification *(Decision Table)*

F	Credit Score Rating *(Credit_Score_Rating)*	Back End Ratio *(Back_End_Ratio)*	Front End Ratio *(Front_End_Ratio)*	Loan Pre-Qualification *(Loan_Qualification)*		Description
				Qualification *(string)*	Reason *(string)*	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

デシジョンテーブルは、ルールとデシジョンロジックのモデル化の方法として一般的で、多くの方法論 (DMN など) や実装フレームワーク (Drools など) で使用されます。



重要

Red Hat Process Automation Manager は DMN デシジョンテーブルおよび Drools ネイティブのデシジョンテーブルの両方をサポートしますが、アセットのタイプが異なると構文の要件も異なり、それぞれを置き換えて使用できません。Red Hat Process Automation Manager の Drools ネイティブのデシジョンテーブルに関する情報は、**スプレッドシートのデシジョンテーブルを使用したデシジョンサービスの作成** を参照してください。

4.4.1.1. DMN デシジョンテーブルのヒットポリシー

ヒットポリシーは、デシジョンテーブルにある複数のルールが指定の入力値と一致する場合に、どのように結果に到達するかを決定します。たとえば、デシジョンテーブルの中の 1 つのルールでは、軍人に価格の割引を適用し、別のルールでは学生に割引を適用する場合に、学生であり軍人である顧客には、デシジョンテーブルのヒットポリシーに割引を 1 つだけ適用するのか (**Unique**、**First**) または両方の割引を適用するのか (**Collect Sum**) 指定しておく必要があります。ヒットポリシーの 1 文字 (**U**、**F**、**C+**) をデシジョンテーブルの左上隅に指定します。

DMN では、以下のデシジョンテーブルのヒットポリシーがサポートされます。

- **Unique (U):** 一致するルールを 1 つだけ許可します。重複はエラーとなります。
- **Any (A):** 複数のルールが一致するのを許可しますが、出力は同じである必要があります。一致している複数のルールで出力が同じでないと、エラーが発生します。
- **Priority (P):** 複数のルールが一致し、結果が異なるのを許可します。出力値リストで最初に出力されるものが選択されます。
- **First (F):** ルールの順番に従い、最初に一致するのを使用します。
- **Collect (C+、C>、C<、C#):** 集約関数に基づいて、複数のルールから出力を集めます。
  - **Collect (C):** 任意のリストで値を集めます。
  - **Collect Sum (C+):** 集計したすべての値の合計を出力します。値は数値でなければなりません。

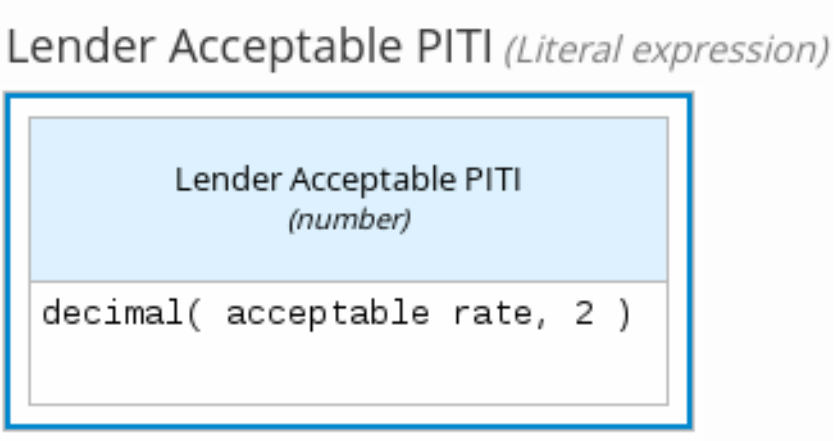
- **Collect Min (C<):** 一致する中で最小の値を出力します。結果の値は、数値、日付、またはテキスト (辞書的順序) など、比較可能な値である必要があります。
- **Collect Max (C>):** 一致する中で最高の値を出力します。結果の値は、数値、日付、またはテキスト (辞書的順序) など、比較可能な値である必要があります。
- **Collect Count (C#):** 一致するルールの数を出力します。

4.4.2. ボックスリテラル式

DMN のボックスリテラル式は、テーブルのセル内のテキストとして使用するリテラル FEEL 式で、通常ラベル付きの列およびデータタイプが割り当てられています。ボックスリテラル式を使用して、デシジョンの特定のノードに対して FEEL で直接、単純または複雑なノードロジックまたはデシジョンデータを定義できます。リテラル FEEL 式は OMG の Decision Model and Notation specification の FEEL 構文要件に準拠する必要があります。

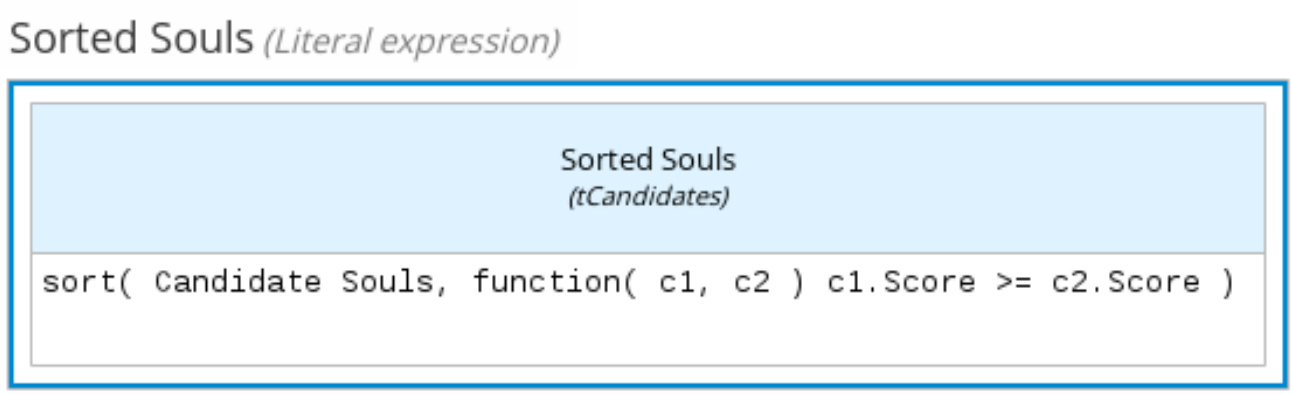
たとえば、以下のボックスリテラル式では、融資のデシジョンにおいて最低限許容できる PITI 計算 (元金 (Principal)、利子 (Interest)、税金 (Tax)、保険 (Insurance)) を定義します。ここでの **acceptable rate** は、DMN モデルで定義した変数です。

図4.6 PITI の最小値のボックスリテラル式



以下のボックスリテラル式は、年齢、場所、趣味などの基準のスコアをもとに、オンラインの出会い系アプリでデート相手の候補 (ソウルメイト) 一覧をソートします。

図4.7 オンラインでデート相手の候補者をマッチングするボックスリテラル式



4.4.3. ボックスコンテキスト式

DMN のボックスコンテキスト式は、結果の値が含まれる、値と変数名のセットです。名前と値のペア

はそれぞれ、コンテキストエントリーとなっています。コンテキスト式を使用して、デシジョンロジックでデータの定義を表現し、DMN デシジョンモデル内で任意のデシジョン要素の値を設定します。ボックスコンテキスト式の値は、データ型の値または FEEL 式を指定でき、デシジョンテーブル、リテラル式、または別のコンテキスト式など、どの型でもサブ式をネスト化させることができます。

たとえば、以下のボックスコンテキスト式では、定義したデータ型 (**tPassengerTable**, **tFlightNumberList**) をもとに、飛行機の再予約を行うデシジョンモデルで遅延客をソートする要素を定義します。

図4.8 航空機利用客のウェイティングリストのボックスコンテキスト式

Prioritized Waiting List *(Context)*

#	Prioritized Waiting List *(tPassengerTable)*	
1	Cancelled Flights *(tFlightNumberList)*	`Flight List[ Status = "cancelled" ].Flight Number`
2	Waiting List *(tPassengerTable)*	`Passenger List[ list contains( Cancelled Flights, Flight Number ) ]`
	<result>	`sort( Waiting List, Passenger Priority )`

以下のボックスコンテキスト式では、サブコンテキスト式が含まれるフロントエンドの割合計算として表現されている PITI (元金 (Principal)、利子 (Interest)、税金 (Tax)、保険 (Insurance)) をもとに、ローンの申請者が最小限必要とされるローンの支払いをしているかを決定する要素を定義します。

図4.9 フロントエンドクライアント PITI 割合のボックスコンテキスト式

Front End Ratio *(Context)*

#	Front End Ratio *(Front_End_Ratio)*		
1	Client PITI *(number)*	#	PITI
		1	pmt *(number)*	`(Requested Product.Amount*((Requested Product.Rate/100)/12))/ (1-(1/(1+(Requested Product.Rate/100)/12)**-Requested Product.Term))`
		2	tax *(number)*	`Applicant Data.Monthly.Tax`
		3	insurance *(number)*	`Applicant Data.Monthly.Insurance`
		4	income *(number)*	`Applicant Data.Monthly.Income`
	<result>	`if Client PITI <= Lender Acceptable PITI() then "Sufficient" else "Insufficient"`	

4.4.4. ボックスリレーション式

DMN のボックスリレーション式は、指定のエンティティーに関する情報 (行として記載) が含まれる従来のデータテーブルです。ボックスリレーションテーブルを使用して、特定のノードでのデシジョンで関連するエンティティーのデシジョンデータを定義します。ボックスリレーション式は、変数名と値を設定する点ではコンテキスト式に似ていますが、リレーション式には結果の値が含まれておらず、定義した変数を 1 つをもとに全変数値を列ごとにリストします。

たとえば、以下のボックスリレーション式は、従業員の勤務表デシジョンで従業員に関する情報を提供します。

図4.10 従業員の情報を含むボックスリレーション式

Employee Information *(Relation)*

#	Name *(string)*	Dept *(string)*	Salary *(number)*
1	"John"	"Sales"	100000
2	"Mary"	"Finances"	120000

4.4.5. ボックス関数式

DMN のボックス関数式は、リテラル FEEL 式、外部の JAVA または PMML 関数のネスト化されたコンテキスト式、あらゆる型のネスト化されたボックス式を含む、パラメーターを使用するボックス式です。デフォルトでは、全ビジネスナレッジモデルは、ボックス関数式として定義されます。ボックス関数式を使用して、デシジョンロジックで関数を呼び出し、全ビジネスナレッジモデルを定義します。

たとえば、以下のボックス関数式では、フライトの予約変更デシジョンモデルで、航空機の定員を決定します。

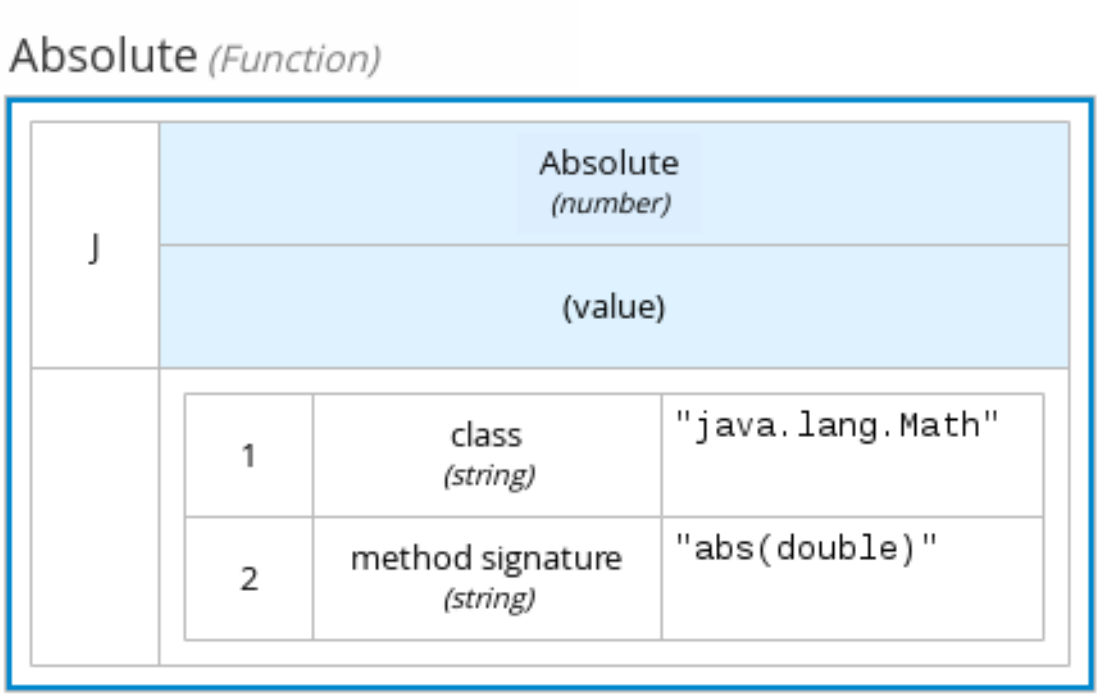
図4.11 フライトの定員に使用するボックス関数式

Flight Capacity *(Function)*

F	Flight Capacity *(boolean)*
	(flight, rebooked list)
	`flight.Capacity > count( rebooked list[ Flight Number = flight.Flight Number ] )`

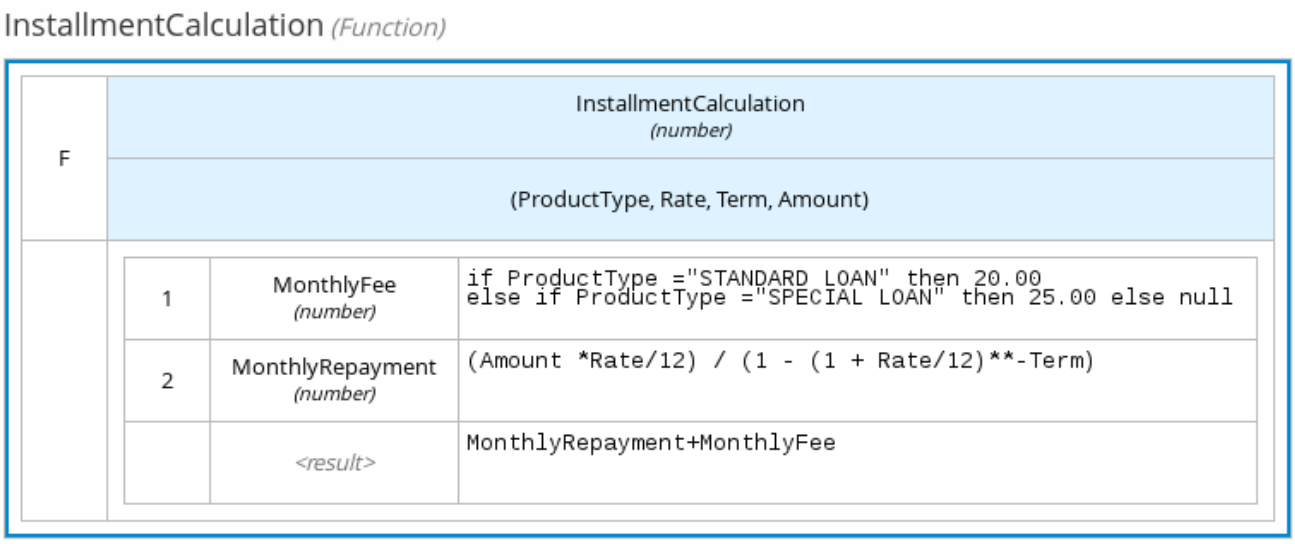
以下のボックス関数式には、デシジョンモデルの計算で絶対値を判断するコンテキスト式として使用する基本的な Java 関数が含まれています。

図4.12 絶対値のボックス関数式



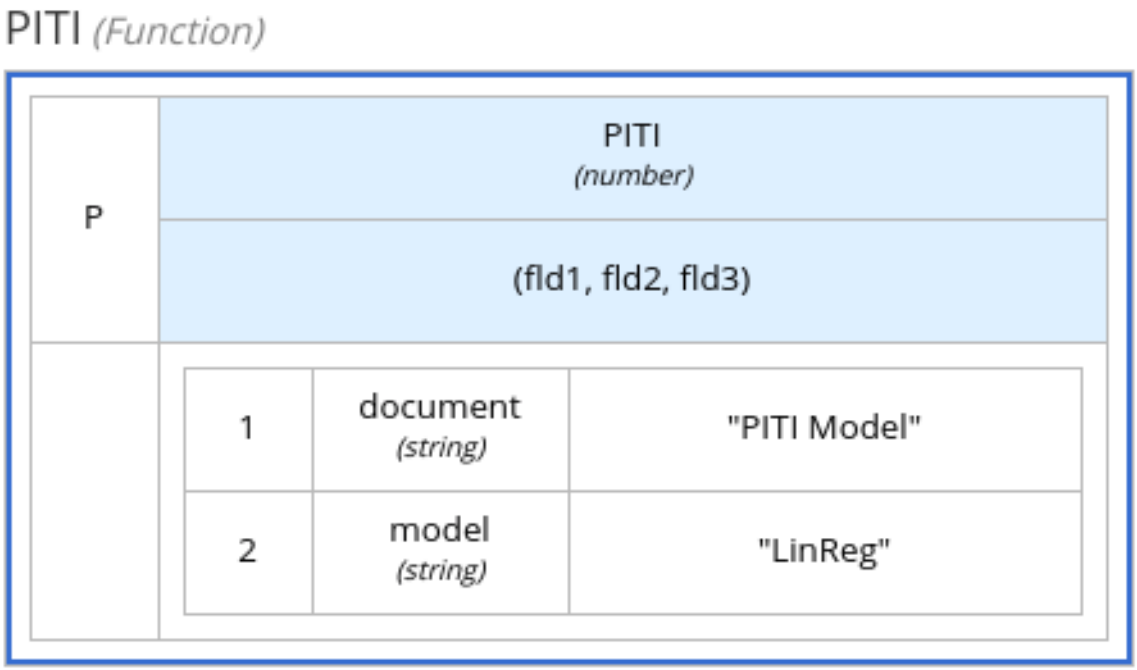
以下のボックス関数式では、ネスト化されたコンテキスト式として定義された関数値を使用し、融資のデシジョンのビジネスナレッジモデルとして、住宅ローンの月額を決定します。

図4.13 ビジネスナレッジモデルのローン計算で使用するボックス関数式



以下のボックス関数式は、DMN ファイルに含まれる PMML モデルを使用して、融資の意思決定において、最低許容可能な PITI (元金、利息、税金、保険) の計算を定義します。

図4.14 ビジネスナレッジモデルに PMML モデルが含まれるボックス関数式



4.4.6. ボックス呼び出し式

DMN のボックス呼び出し式は、ビジネスナレッジモデルを呼び出すボックス式です。ボックス呼び出し式には、呼び出すビジネスナレッジモデルの名前と、パラメーターバインディングのリストが含まれています。各バインディングは、1 行に 2 つのボックス式を入れることで表現します。左のボックスにはパラメーターの名前、右のボックスには呼び出したビジネスナレッジモデルを評価するパラメーターに割り当てられる値のバインディング式が含まれます。ボックス式を使用して、デシジョンモデルで定義されているビジネスナレッジモデルを特定のデシジョンノードで呼び出します。

たとえば、以下のボックス呼び出し式では、フライト予約変更のデシジョンモデルで終端デシジョンノードとして **Reassign Next Passenger** ビジネスナレッジモデルを呼び出します。

図4.15 フライトの乗客を再割り当てするボックス呼び出し式

Rebooked Passengers *(Invocation)*

#	Rebooked Passengers *(tPassengerTable)*	
	Reassign Next Passenger	
1	Waiting List *(tPassengerTable)*	`Prioritized Waiting List`
2	Reassigned Passengers List *(tPassengerTable)*	`[]`
3	Flights *(tFlightTable)*	`Flight List`

以下のボックス呼び出し式では、**InstallmentCalculation** ビジネスナレッジモデルを呼び出し、ローンを負担できるかどうか決定する前に、ローンの月額を計算します。

図4.16 必要な月額を判断するボックス呼び出し式

RequiredMonthlyInstallment *(Invocation)*

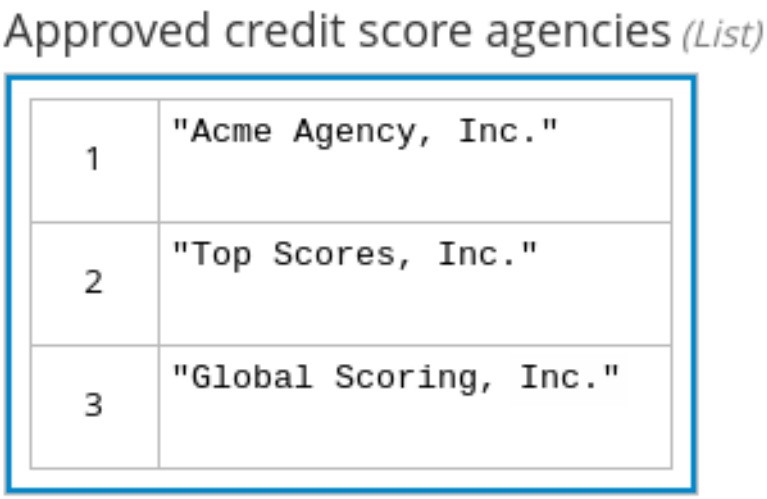
#	RequiredMonthlyInstallment *(number)*	
	InstallmentCalculation	
1	ProductType *(string)*	`RequestedProduct.ProductType`
2	Rate *(number)*	`RequestedProduct.Rate`
3	Term *(string)*	`RequestedProduct.Term`
4	Amount *(number)*	`RequestedProduct.Amount`

4.4.7. ボックスリスト式

DMN のボックスリスト式は、アイテムの FEEL 一覧を表します。ボックスリストを使用して、デシジョン内にある特定のノードの関連アイテムをリストで定義します。セル内のリストアイテムにリテラル FEEL 式を使用して、より複雑なリストを作成することもできます。

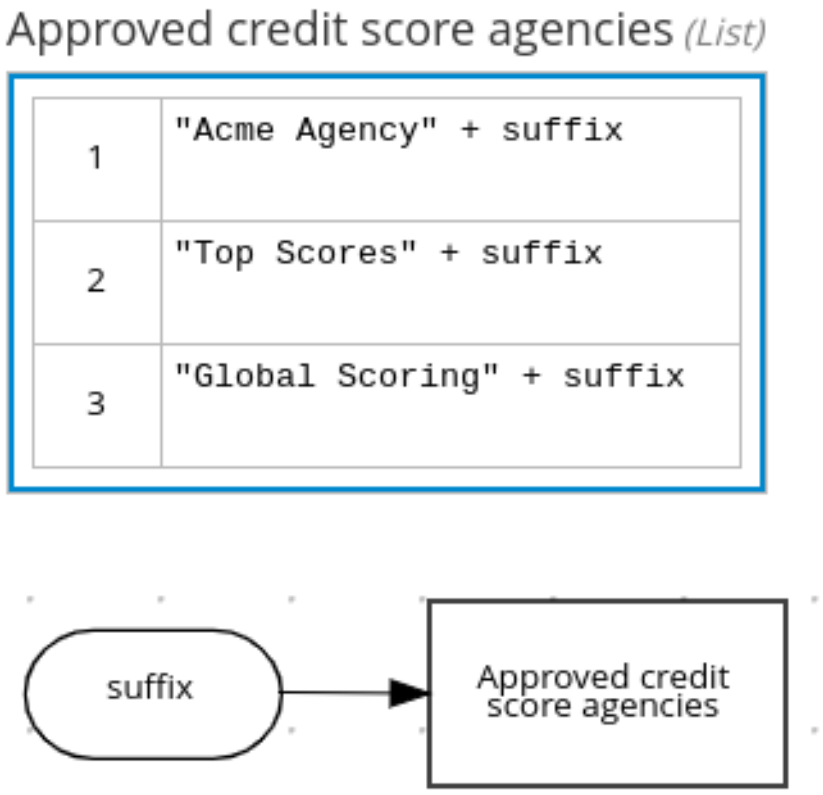
たとえば、次のボックスリスト式では、ローン申請のデシジョンサービスで、承認されたクレジットスコア機関を特定します。

図4.17 承認されたクレジットスコア機関のボックスリスト式



以下のボックスリスト式では、承認されたクレジットスコア機関も特定しますが、FEEL ロジックを使用して、DMN 入力ノードを基に機関のステータス (Inc.、LLC、SA、GA) を定義します。

図4.18 承認されたクレジットスコア機関のステータスに FEEL ロジックを使用したボックスリスト式



4.5. DMN モデルの例

以下は、入力データ、状況、企業のガイドラインをもとに、デシジョンモデルをどのように使用して決断に至るかを判断する実際の DMN モデル例です。以下のシナリオでは、サンディエゴからニューヨークへのフライトがキャンセルされ、欠航となってしまったフライトの航空会社は、このフライトの乗客に対して、別のフライトを手配する必要があります。

まずは、乗客を目的地に運ぶ最適な方法を決めるのに必要な情報を集めます。

入力データ

- フライトリスト
- 乗客リスト

決定

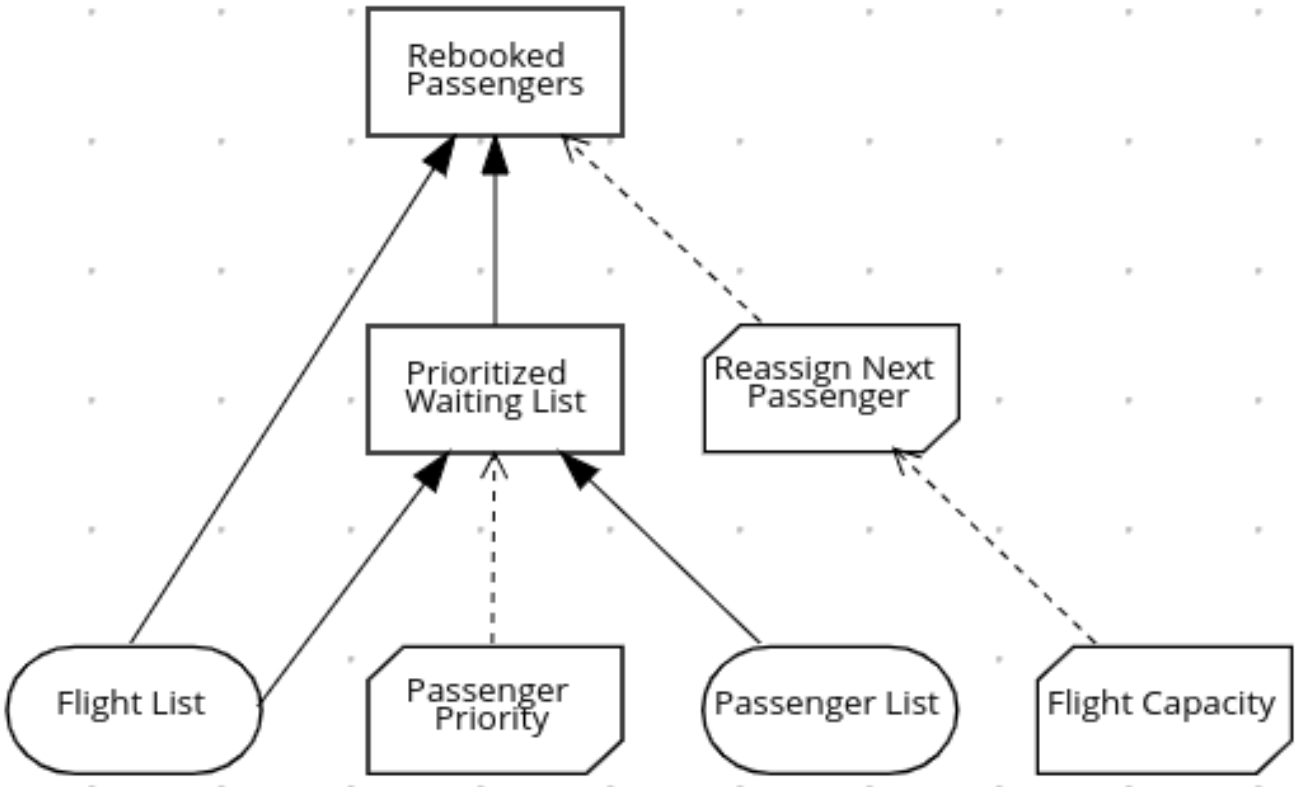
- 新しいフライトで席を確保する乗客の優先順位をつける
- 乗客に提示するフライトを決定する

ビジネスナレッジモデル

- 乗客の優先順位を決定する企業のプロセス
- 席に余裕があるフライト
- フライトをキャンセルされた乗客を再割り当てするのに最適な方法を決定する会社のルール

次に、航空会社は、DMN 仕様を使用して、以下の意思決定要件ダイアグラム (DRD) でそのデシジョンプロセスをモデル化し、予約変更の最適解を決める以下のダイアグラムを作成します。

図4.19 フライト予約変更の DRD



DRD では、フローチャートのように、プロセスの各要素に異なる形状を使用します。楕円形には必要な入力データが 2 つ、長方形にはモデルでのデシジョンポイントを含み、端が欠けた長方形 (ビジネスナレッジモデル) には、繰り返し呼び出せる再利用可能なロジックが含まれます。

DRD は、FEEL 式またはデータ型の値を使用して変数定義を提供するボックス式から各要素のロジックを引き出します。

ウィティングリストの優先順位を確立する以下のデシジョンなど、ボックス式には基本的なものもあります。

図4.20 ウェイティングリストの優先順位に関するボックスコンテキスト式のサンプル

Prioritized Waiting List *(Context)*

#	Prioritized Waiting List *(tPassengerTable)*	
1	Cancelled Flights *(tFlightNumberList)*	`Flight List[ Status = "cancelled" ].Flight Number`
2	Waiting List *(tPassengerTable)*	`Passenger List[ list contains( Cancelled Flights, Flight Number ) ]`
	<result>	`sort( Waiting List, Passenger Priority )`

ボックス式には、次の遅延客を再割り当てするための以下のビジネスナレッジモデルなど、詳細にわたる情報や計算が含まれ、さらに複雑なものもあります。

図4.21 乗客再割り当てのボックス関数式

Reassign Next Passenger *(Function)*

F	Reassign Next Passenger *(tPassengerTable)*		
	(Waiting List, Reassigned Passengers List, Flights)		
1	Next Passenger *(tPassenger)*	`Waiting List[1]`	
2	Original Flight *(tFlight)*	`Flights[ Flight Number = Next Passenger.Flight Number ][1]`	
3	Best Alternate Flight *(tFlight)*	`Flights[ From = Original Flight.From and To = Original Flight.To and Departure > Original Flight.Departure and Status = "scheduled" and Flight Capacity( item, Reassigned Passengers List ) ][1]`	
4	Reassigned Passenger *(tPassenger)*	1	Name *(string)*	`Next Passenger.Name`
		2	Status *(string)*	`Next Passenger.Status`
		3	Miles *(number)*	`Next Passenger.Miles`
		4	Flight Number *(string)*	`Best Alternate Flight.Flight Number`
			<result>	*Select expression*
5	Remaining Waiting List *(tPassengerTable)*	`remove( Waiting List, 1 )`	
6	Updated Reassigned Passengers List *(tPassengerTable)*	`append( Reassigned Passengers List, Reassigned Passenger )`	
	<result>	`if count( Remaining Waiting List ) > 0 then Reassign Next Passenger( Remaining Waiting List, Updated Reassigned Passengers List, Flights ) else Updated Reassigned Passengers List`	

以下は、このデシジョンモデルの DMN ソースファイルです。

```
<dmn:definitions xmlns="https://www.drools.org/kie-dmn/Flight-rebooking"
```

```
xmlns:dmn="http://www.omg.org/spec/DMN/20151101/dmn.xsd"
xmlns:feel="http://www.omg.org/spec/FEEL/20140401" id="_0019_flight_rebooking" name="0019-
flight-rebooking" namespace="https://www.drools.org/kie-dmn/Flight-rebooking">
  <dmn:itemDefinition id="_tFlight" name="tFlight">
    <dmn:itemComponent id="_tFlight_Flight" name="Flight Number">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_From" name="From">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_To" name="To">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_Dep" name="Departure">
      <dmn:typeRef>feel:dateTime</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_Arr" name="Arrival">
      <dmn:typeRef>feel:dateTime</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_Capacity" name="Capacity">
      <dmn:typeRef>feel:number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tFlight_Status" name="Status">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_tFlightTable" isCollection="true" name="tFlightTable">
    <dmn:typeRef>tFlight</dmn:typeRef>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_tPassenger" name="tPassenger">
    <dmn:itemComponent id="_tPassenger_Name" name="Name">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tPassenger_Status" name="Status">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tPassenger_Miles" name="Miles">
      <dmn:typeRef>feel:number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_tPassenger_Flight" name="Flight Number">
      <dmn:typeRef>feel:string</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_tPassengerTable" isCollection="true" name="tPassengerTable">
    <dmn:typeRef>tPassenger</dmn:typeRef>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_tFlightNumberList" isCollection="true" name="tFlightNumberList">
    <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemDefinition>
  <dmn:inputData id="i_Flight_List" name="Flight List">
    <dmn:variable name="Flight List" typeRef="tFlightTable"/>
  </dmn:inputData>
  <dmn:inputData id="i_Passenger_List" name="Passenger List">
    <dmn:variable name="Passenger List" typeRef="tPassengerTable"/>
  </dmn:inputData>
  <dmn:decision name="Prioritized Waiting List" id="d_PrioritizedWaitingList">
```

```
    <dmn:variable name="Prioritized Waiting List" typeRef="tPassengerTable"/>
    <dmn:informationRequirement>
      <dmn:requiredInput href="#i_Passenger_List"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement>
      <dmn:requiredInput href="#i_Flight_List"/>
    </dmn:informationRequirement>
    <dmn:knowledgeRequirement>
      <dmn:requiredKnowledge href="#b_PassengerPriority"/>
    </dmn:knowledgeRequirement>
    <dmn:context>
      <dmn:contextEntry>
        <dmn:variable name="Cancelled Flights" typeRef="tFlightNumberList"/>
        <dmn:literalExpression>
          <dmn:text>Flight List[ Status = "cancelled" ].Flight Number</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Waiting List" typeRef="tPassengerTable"/>
        <dmn:literalExpression>
          <dmn:text>Passenger List[ list contains( Cancelled Flights, Flight Number ) ]</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:literalExpression>
          <dmn:text>sort( Waiting List, passenger priority )</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
    </dmn:context>
  </dmn:decision>
  <dmn:decision name="Rebooked Passengers" id="d_RebookedPassengers">
    <dmn:variable name="Rebooked Passengers" typeRef="tPassengerTable"/>
    <dmn:informationRequirement>
      <dmn:requiredDecision href="#d_PrioritizedWaitingList"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement>
      <dmn:requiredInput href="#i_Flight_List"/>
    </dmn:informationRequirement>
    <dmn:knowledgeRequirement>
      <dmn:requiredKnowledge href="#b_ReassignNextPassenger"/>
    </dmn:knowledgeRequirement>
    <dmn:invocation>
      <dmn:literalExpression>
        <dmn:text>reassign next passenger</dmn:text>
      </dmn:literalExpression>
      <dmn:binding>
        <dmn:parameter name="Waiting List"/>
        <dmn:literalExpression>
          <dmn:text>Prioritized Waiting List</dmn:text>
        </dmn:literalExpression>
      </dmn:binding>
      <dmn:binding>
        <dmn:parameter name="Reassigned Passengers List"/>
        <dmn:literalExpression>
          <dmn:text>[]</dmn:text>
        </dmn:literalExpression>
```

```
    </dmn:binding>
    <dmn:binding>
      <dmn:parameter name="Flights"/>
      <dmn:literalExpression>
        <dmn:text>Flight List</dmn:text>
      </dmn:literalExpression>
    </dmn:binding>
  </dmn:invocation>
</dmn:decision>
<dmn:businessKnowledgeModel id="b_PassengerPriority" name="passenger priority">
  <dmn:encapsulatedLogic>
    <dmn:formalParameter name="Passenger1" typeRef="tPassenger"/>
    <dmn:formalParameter name="Passenger2" typeRef="tPassenger"/>
    <dmn:decisionTable hitPolicy="UNIQUE">
      <dmn:input id="b_Passenger_Priority_dt_i_P1_Status" label="Passenger1.Status">
        <dmn:inputExpression typeRef="feel:string">
          <dmn:text>Passenger1.Status</dmn:text>
        </dmn:inputExpression>
        <dmn:inputValues>
          <dmn:text>"gold", "silver", "bronze"</dmn:text>
        </dmn:inputValues>
      </dmn:input>
      <dmn:input id="b_Passenger_Priority_dt_i_P2_Status" label="Passenger2.Status">
        <dmn:inputExpression typeRef="feel:string">
          <dmn:text>Passenger2.Status</dmn:text>
        </dmn:inputExpression>
        <dmn:inputValues>
          <dmn:text>"gold", "silver", "bronze"</dmn:text>
        </dmn:inputValues>
      </dmn:input>
      <dmn:input id="b_Passenger_Priority_dt_i_P1_Miles" label="Passenger1.Miles">
        <dmn:inputExpression typeRef="feel:string">
          <dmn:text>Passenger1.Miles</dmn:text>
        </dmn:inputExpression>
      </dmn:input>
      <dmn:output id="b_Status_Priority_dt_o" label="Passenger1 has priority">
        <dmn:outputValues>
          <dmn:text>true, false</dmn:text>
        </dmn:outputValues>
        <dmn:defaultOutputEntry>
          <dmn:text>false</dmn:text>
        </dmn:defaultOutputEntry>
      </dmn:output>
      <dmn:rule id="b_Passenger_Priority_dt_r1">
        <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i1">
          <dmn:text>"gold"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i2">
          <dmn:text>"gold"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i3">
          <dmn:text>>= Passenger2.Miles</dmn:text>
        </dmn:inputEntry>
        <dmn:outputEntry id="b_Passenger_Priority_dt_r1_o1">
          <dmn:text>true</dmn:text>
        </dmn:outputEntry>
```

```
    </dmn:rule>
    <dmn:rule id="b_Passenger_Priority_dt_r2">
      <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i1">
        <dmn:text>"gold"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i2">
        <dmn:text>"silver","bronze"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i3">
        <dmn:text>-</dmn:text>
      </dmn:inputEntry>
      <dmn:outputEntry id="b_Passenger_Priority_dt_r2_o1">
        <dmn:text>true</dmn:text>
      </dmn:outputEntry>
    </dmn:rule>
    <dmn:rule id="b_Passenger_Priority_dt_r3">
      <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i1">
        <dmn:text>"silver"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i2">
        <dmn:text>"silver"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i3">
        <dmn:text>>= Passenger2.Miles</dmn:text>
      </dmn:inputEntry>
      <dmn:outputEntry id="b_Passenger_Priority_dt_r3_o1">
        <dmn:text>true</dmn:text>
      </dmn:outputEntry>
    </dmn:rule>
    <dmn:rule id="b_Passenger_Priority_dt_r4">
      <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i1">
        <dmn:text>"silver"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i2">
        <dmn:text>"bronze"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i3">
        <dmn:text>-</dmn:text>
      </dmn:inputEntry>
      <dmn:outputEntry id="b_Passenger_Priority_dt_r4_o1">
        <dmn:text>true</dmn:text>
      </dmn:outputEntry>
    </dmn:rule>
    <dmn:rule id="b_Passenger_Priority_dt_r5">
      <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i1">
        <dmn:text>"bronze"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i2">
        <dmn:text>"bronze"</dmn:text>
      </dmn:inputEntry>
      <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i3">
        <dmn:text>>= Passenger2.Miles</dmn:text>
      </dmn:inputEntry>
      <dmn:outputEntry id="b_Passenger_Priority_dt_r5_o1">
        <dmn:text>true</dmn:text>
      </dmn:outputEntry>
```

```
      </dmn:rule>
    </dmn:decisionTable>
  </dmn:encapsulatedLogic>
  <dmn:variable name="passenger priority" typeRef="feel:boolean"/>
 </dmn:businessKnowledgeModel>
 <dmn:businessKnowledgeModel id="b_ReassignNextPassenger" name="reassign next passenger">
  <dmn:encapsulatedLogic>
   <dmn:formalParameter name="Waiting List" typeRef="tPassengerTable"/>
   <dmn:formalParameter name="Reassigned Passengers List" typeRef="tPassengerTable"/>
   <dmn:formalParameter name="Flights" typeRef="tFlightTable"/>
   <dmn:context>
    <dmn:contextEntry>
     <dmn:variable name="Next Passenger" typeRef="tPassenger"/>
     <dmn:literalExpression>
      <dmn:text>Waiting List[1]</dmn:text>
     </dmn:literalExpression>
    </dmn:contextEntry>
    <dmn:contextEntry>
     <dmn:variable name="Original Flight" typeRef="tFlight"/>
     <dmn:literalExpression>
      <dmn:text>Flights[ Flight Number = Next Passenger.Flight Number ][1]</dmn:text>
     </dmn:literalExpression>
    </dmn:contextEntry>
    <dmn:contextEntry>
     <dmn:variable name="Best Alternate Flight" typeRef="tFlight"/>
     <dmn:literalExpression>
      <dmn:text>Flights[ From = Original Flight.From and To = Original Flight.To and Departure >
Original Flight.Departure and Status = "scheduled" and has capacity( item, Reassigned Passengers
List ) ][1]</dmn:text>
     </dmn:literalExpression>
    </dmn:contextEntry>
    <dmn:contextEntry>
     <dmn:variable name="Reassigned Passenger" typeRef="tPassenger"/>
     <dmn:context>
      <dmn:contextEntry>
       <dmn:variable name="Name" typeRef="feel:string"/>
       <dmn:literalExpression>
        <dmn:text>Next Passenger.Name</dmn:text>
       </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
       <dmn:variable name="Status" typeRef="feel:string"/>
       <dmn:literalExpression>
        <dmn:text>Next Passenger.Status</dmn:text>
       </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
       <dmn:variable name="Miles" typeRef="feel:number"/>
       <dmn:literalExpression>
        <dmn:text>Next Passenger.Miles</dmn:text>
       </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
       <dmn:variable name="Flight Number" typeRef="feel:string"/>
       <dmn:literalExpression>
        <dmn:text>Best Alternate Flight.Flight Number</dmn:text>
```

```
            </dmn:literalExpression>
          </dmn:contextEntry>
        </dmn:context>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Remaining Waiting List" typeRef="tPassengerTable"/>
        <dmn:literalExpression>
          <dmn:text>remove( Waiting List, 1 )</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Updated Reassigned Passengers List" typeRef="tPassengerTable"/>
        <dmn:literalExpression>
          <dmn:text>append( Reassigned Passengers List, Reassigned Passenger )</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:literalExpression>
          <dmn:text>if count( Remaining Waiting List ) > 0 then reassign next passenger( Remaining
Waiting List, Updated Reassigned Passengers List, Flights ) else Updated Reassigned Passengers
List</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
    </dmn:context>
  </dmn:encapsulatedLogic>
  <dmn:variable name="reassign next passenger" typeRef="tPassengerTable"/>
  <dmn:knowledgeRequirement>
    <dmn:requiredKnowledge href="#b_HasCapacity"/>
  </dmn:knowledgeRequirement>
 </dmn:businessKnowledgeModel>
 <dmn:businessKnowledgeModel id="b_HasCapacity" name="has capacity">
  <dmn:encapsulatedLogic>
    <dmn:formalParameter name="flight" typeRef="tFlight"/>
    <dmn:formalParameter name="rebooked list" typeRef="tPassengerTable"/>
    <dmn:literalExpression>
      <dmn:text>flight.Capacity > count( rebooked list[ Flight Number = flight.Flight Number ]
)</dmn:text>
    </dmn:literalExpression>
  </dmn:encapsulatedLogic>
  <dmn:variable name="has capacity" typeRef="feel:boolean"/>
 </dmn:businessKnowledgeModel>
</dmn:definitions>
```

第5章 RED HAT PROCESS AUTOMATION MANAGER における DMN サポート

Red Hat Process Automation Manager は、適合レベル 3 で DMN 1.2 モデルの設計およびランタイムをサポートし、適合レベル 3 で DMN 1.1 および 1.3 モデルはランタイムのみサポートします。DMN モデルは、お使いの Red Hat Process Automation Manager デシジョンサービスと複数の方法で統合できます。

- DMN デザイナーを使用して Business Central で直接 DMN モデルを設計します。
- Business Central で プロジェクトに DMN ファイルをインポートします (**Menu → Design → Projects → Import Asset**)。Business Central にインポートした DMN 1.1 および 1.3 モデル (DMN 1.3 機能は含まれません) は、DMN デザイナーで開き、保存時に DMN 1.2 モデルに変換されます。
- Business Central を使用せずにプロジェクトのナレッジ JAR (KJAR) ファイルの一部として DMN ファイルをパッケージ化します。

次の表は、Red Hat Process Automation Manager の各 DMN バージョンの設計とランタイムサポートをまとめたものです。

表5.1 Red Hat Process Automation Manager における DMN サポート

DMN バージョン	DMN エンジンのサポート	DMN モデラーのサポート	
	実行	開く	保存
DMN 1.1	✓	✓	✗
DMN 1.2	✓	✓	✓
DMN 1.3	✓	✓	✗

全 DMN 適合レベル 3 の要件に加え、Red Hat Process Automation Manager には FEEL および DMN モデルコンポーネントに機能拡張および修正が含まれており、Red Hat Process Automation Manager での DMN デシジョンサービスの実装体験を最適化します。DMN モデルは、プラットフォームの観点からすると、Red Hat Process Automation Manager プロジェクトに追加したり、DMN デシジョンサービスを起動するために KIE Server にデプロイしたりできるため、DRL ファイルやスプレッドシートのデシジョンテーブルなど、Red Hat Process Automation Manager の他のビジネスアセットとよく似ています。

Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイメント方法を使用して外部の DMN ファイルを追加する方法は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

5.1. RED HAT PROCESS AUTOMATION MANAGER における設定可能な DMN プロパティー

Red Hat Process Automation Manager には、クライアントアプリケーションの KIE Server で DMN モデルを実行する時に設定できる以下の DMN プロパティーが含まれます。これらのプロパティーは、KIE Server にプロジェクトをデプロイする際に、Red Hat Process Automation Manager プロジェクトの **kmodule.xml** ファイルを使用して設定できます。

org.kie.dmn.strictConformance

このプロパティーを有効にすると、一部の helper 関数や、DMN 1.1 にバックポートされた DMN 1.2 の機能強化など、DMN 規定以外に提供された拡張機能やプロファイルをデフォルトで無効にします。このプロパティーを使用して、DMN Technology Compatibility Kit (TCK) を実行するなど、純粋な DMN 機能だけをサポートするデシジョンエンジンを設定できます。
デフォルト値は **false** です。

```
-Dorg.kie.dmn.strictConformance=true
```

org.kie.dmn.runtime.typecheck

このプロパティーを有効にすると、DRD 要素の入力または出力として、DMN モデルに宣言した型に従う実際の値を検証できるようになります。このプロパティーを使用して、DMN モデルに提供されたデータ、または DMN モデルが生成したデータが、モデルに指定したものに準拠するかどうかを検証できます。
デフォルト値は **false** です。

```
-Dorg.kie.dmn.runtime.typecheck=true
```

org.kie.dmn.decisionservice.coercesingleton

このプロパティーは、デフォルトで、1 つの出力デシジョンを定義するデシジョンサービスの結果を、1 つの出力デシジョン値にします。このプロパティーを無効にすると、出力されたデシジョンを定義するデシジョンサービスの結果を、その意思決定のエントリーを 1 つ持つ **context** にします。このプロパティーを使用して、プロジェクト要件に従ってデシジョンサービスを調整できます。
デフォルト値: **true**

```
-Dorg.kie.dmn.decisionservice.coercesingleton=false
```

org.kie.dmn.profiles.$PROFILE_NAME

このプロパティーは、Java の完全修飾名で設定された場合に、起動時にデシジョンエンジンに DMN プロファイルを読み込みます。このプロパティーを使用して、DMN 規定とは異なるサポート機能、またはそれ以外のサポート機能を使用する事前定義した DMN プロファイルを実装できます。Signavio DMN モデラーを使用して DMN モデルを作成する場合は、このプロパティーを使用して Signavio DMN プロファイルからお使いの DMN デシジョンサービスに機能を実装します。

```
-Dorg.kie.dmn.profiles.signavio=org.kie.dmn.signavio.KieDMNSignavioProfile
```

org.kie.dmn.runtime.listeners.$LISTENER_NAME

Java の完全修飾名で値を指定すると、このプロパティーは起動時に DMN Runtime Listener をデシジョンエンジンに読み込み、登録します。DMN モデルの評価時に複数のイベントを通知するには、このプロパティーを使用して DMN リスナーを登録してください。
KIE Server にプロジェクトをデプロイする際にこのプロパティーを設定するには、プロジェクトの **kmodule.xml** ファイルでこのプロパティーを変更します。このアプローチは、リスナーがプロジェクトに固有であり、KIE Server でデプロイされたプロジェクトにのみその設定を適用する必要があ

る場合に役立ちます。

```
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
  <configuration>
    <property key="org.kie.dmn.runtime.listeners.mylistener" value="org.acme.MyDMNListener"/>
  </configuration>
</kmodule>
```

Red Hat Process Automation Manager 環境に対してこのプロパティーをグローバルに設定するには、コマンドターミナルまたはその他のグローバルアプリケーション設定メカニズムを使用してこのプロパティーを変更します。このアプローチは、デシジョンエンジンが Java アプリケーションの一部として埋め込まれた場合に便利です。

```
-Dorg.kie.dmn.runtime.listeners.mylistener=org.acme.MyDMNListener
```

org.kie.dmn.compiler.execmodel

このプロパティーが有効な場合には、ランタイムに実行可能なルールモデルに DMN デシジョンテーブルロジックをコンパイルできます。このプロパティーを使用して、DMN デシジョンテーブルのロジックをより効率的に評価できます。このプロパティーは、実行可能なモデルのコンパイルがプロジェクトのコンパイル時に実行されなかった場合に有用です。このプロパティーを有効にすると、デシジョンエンジンにより最初の評価時のコンパイル時間が増加してしまいますが、その後のコンパイルがより効率的になります。
デフォルト値は **false** です。

```
-Dorg.kie.dmn.compiler.execmodel=true
```

5.2. RED HAT PROCESS AUTOMATIONMANAGER で設定可能な DMN 検証

デフォルトでは、Red Hat Process Automation Manager プロジェクトの **pom.xml** ファイルの **kie-maven-plugin** コンポーネントは、以下の **<validateDMN>** 設定を使用して DMN モデルアセットの事前コンパイル検証を実行し、DMN デシジョンテーブルの静的分析を実行します。

- **VALIDATE_SCHEMA**: DMN モデルファイルは DMN 仕様の XSD スキーマに対して検証され、ファイルが有効な XML であり、この仕様に準拠することを確認します。
- **VALIDATE_MODEL**: DMN モデルに対して事前コンパイル分析が実行され、基本的なセマンティックが DMN 仕様と合致するようにします。
- **ANALYZE_DECISION_TABLE**: DMN デシジョンテーブルは、ギャップまたは重複に対して静的に分析され、デシジョンテーブルのセマンティックがベストプラクティスに従います。

以下の例のように、デフォルトの DMN 検証および DMN デシジョンテーブル分析動作を変更して、プロジェクトビルドで指定された検証のみを実行するか、以下の例に示すように、このデフォルト動作を完全に無効にできます。

DMN 検証およびデシジョンテーブル分析のデフォルト設定

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
```

```
  <configuration>

<validateDMN>VALIDATE_SCHEMA,VALIDATE_MODEL,ANALYZE_DECISION_TABLE</validateD
MN>
  </configuration>
</plugin>
```

DMN デシジョンテーブルの静的分析のみを実行する設定

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>ANALYZE_DECISION_TABLE</validateDMN>
  </configuration>
</plugin>
```

XSD スキーマ検証のみを実行する設定

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>VALIDATE_SCHEMA</validateDMN>
  </configuration>
</plugin>
```

DMN モデルの検証のみを実行する設定

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>VALIDATE_MODEL</validateDMN>
  </configuration>
</plugin>
```

すべての DMN 検証を無効化する設定

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>disable</validateDMN>
  </configuration>
</plugin>
```

注記

認識されない **<validateDMN>** 設定フラグを入力すると、すべての事前コンパイル検証は無効になり、Maven プラグインが関連するログメッセージを生成します。

第6章 BUSINESS CENTRAL での DMN モデルの作成および編集

Business Central の DMN デザイナーを使用すると、DMN 意思決定要件ダイアグラム (DRD) を設計し、完全で機能的な DMN 意思決定モデルの意思決定論理を定義できます。Red Hat Process Automation Manager は、適合レベル 3 の DMN 1.2 モデルに対する設計とランタイムの両方のサポートを提供し、FEEL と DMN モデルコンポーネントの機能拡張と修正が含まれており、Red Hat Process Automation Manager での DMN デシジョンサービスの実装が最適化されます。Red Hat Process Automation Manager では、適合レベル 3 の DMN 1.1 および 1.3 に対してランタイムのみをサポートしますが、Business Central にインポートした DMN 1.1 および 1.3 (DMN 1.3 機能は含まない) はすべて、DMN デザイナーで開かれ、保存時に DMN 1.2 モデルに変換されます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. Business Central プロジェクトで DMN ファイルを作成するか、インポートします。
   DMN ファイルを作成するには、**Add Asset → DMN** をクリックし、わかりやすい DMN モデル名を入力して、適切な **Package** を選択してから、**Ok** をクリックします。

   既存の DMN ファイルをインポートするには、**Import Asset** をクリックし、DMN モデル名を入力して、適切な **Package** を選択し、アップロードする DMN ファイルを選択してから **Ok** をクリックします。

   新しい DMN ファイルが **Project Explorer** の **DMN** パネルに表示され、DMN 意思決定要件ダイアグラム (DRD) のキャンバスが表示されます。

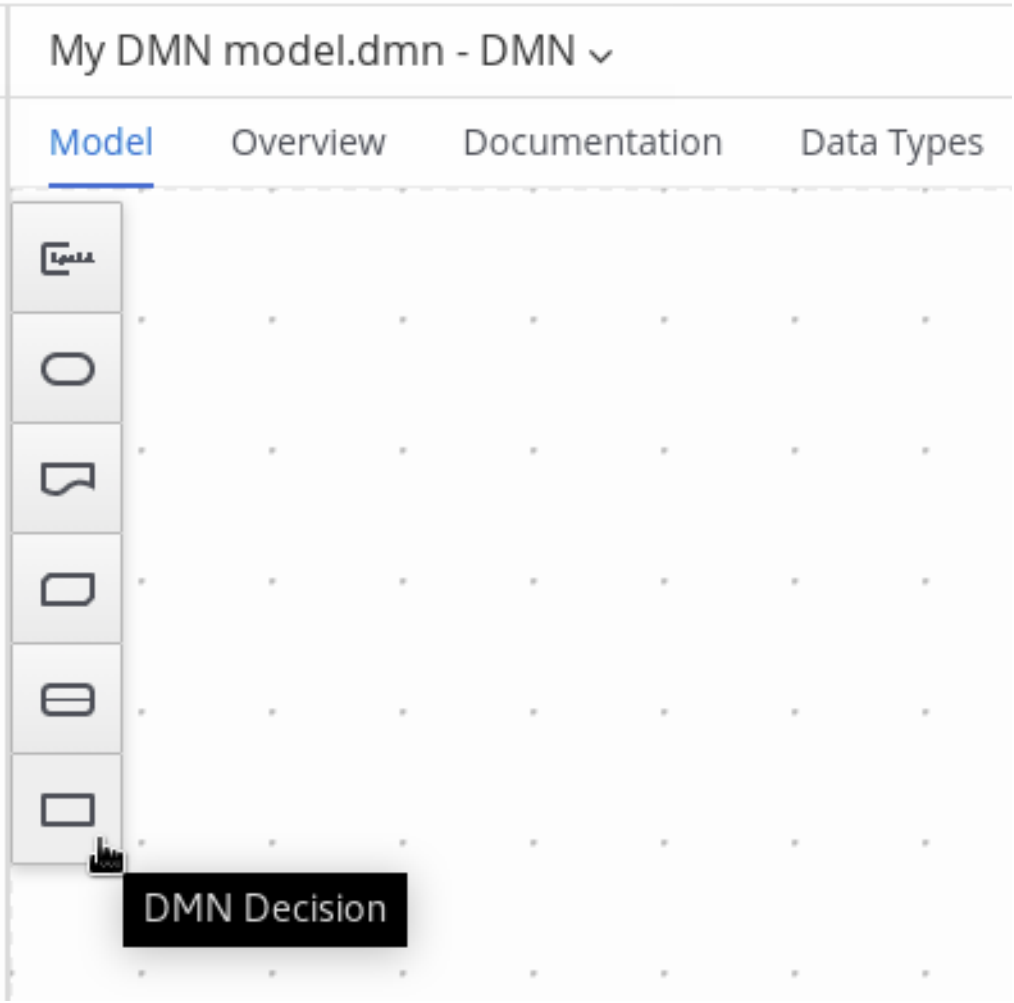
   

   注記

   レイアウトの情報が含まれていない DMN ファイルをインポートした場合は、インポートした意思決定要件ダイアグラム (DRD) は DMN デザイナーで自動的にフォーマットされます。DMN デザイナーで **Save** をクリックして、DRD レイアウトを保存します。

   インポートした DRD が自動的にフォーマットされていない場合、DMN デザイナーの右上のツールバーにある **Perform automatic layout** アイコンを選択して DRD をフォーマットしてください。
3. 左側のツールバーから DMN ノードの 1 つをクリックしてドラッグし、新規またはインポートした DMN 意思決定要件ダイアグラム (DRD) にコンポーネントを追加しはじめてください。

図6.1 DRD コンポーネントの追加



以下の DRD コンポーネントを利用できます。

- **デシジョン**: DMN ディジョンにこのノードを使用します。1 つ以上の要素が定義したデシジョンロジックをもとに出力を決定するノード。
- **ビジネスナレッジモデル**: 1 つまたは複数のデシジョン要素が含まれる再利用可能な関数には、このノードを使用します。同じロジックですが、サブの入力または決定が異なるため、ビジネスナレッジモデルを使用してどの手順に従うかを決定します。
- **ナレッジソース**: デシジョンまたはビジネスナレッジモデルを規定する外部の機関、ドキュメント、委員会またはポリシーにはこのノードを使用します。ナレッジソースは、実行可能なビジネスルールではなく、実際の要因への参照となります。
- **入力データ**: デシジョンノードまたはビジネスナレッジモデルで使用する情報にはこのノードを使用します。入力データには通常、融資戦略で使用するローン申請データなど、ビジネスに関連するビジネスレベルのコンセプトまたはオブジェクトが含まれます。
- **テキストの注釈**: 入力データノード、デシジョンノード、ビジネスナレッジモデル、またはナレッジソースに関連する注釈にはこのノードを使用します。
- **デシジョンサービス**: 呼び出し用にデシジョンサービスとして実装される再利用可能なデシジョンセットを含めるにはこのノードを使用します。デシジョンサービスは、他の DMN モデルで使用し、外部アプリケーションまたは BPMN ビジネスプロセスから呼び出しできます。

4. DMN デザイナーキャバスで、新規の DRD ノードをダブルクリックして情報ノード名を入力します。

5. ノードがデシジョンまたはビジネスナレッジモデルの場合は、ノードオプションを表示するノードを選択して **Edit** アイコンをクリックし、DMN ボックス式を開き、ノードのデシジョンロジックを定義します。

図6.2 新規デシジョンノードのボックス式の表示

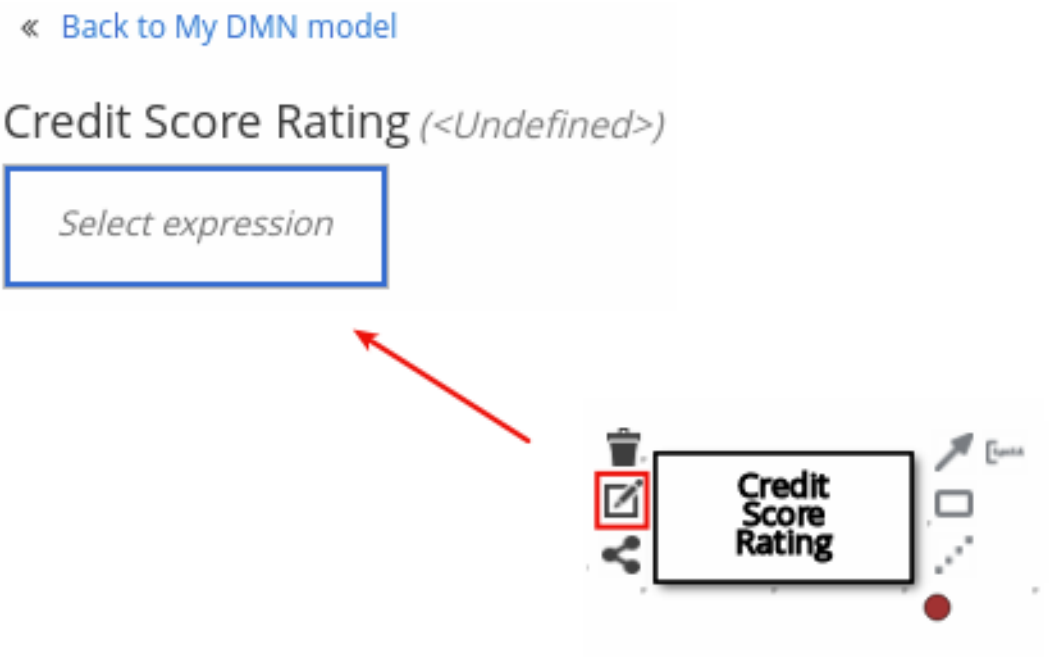
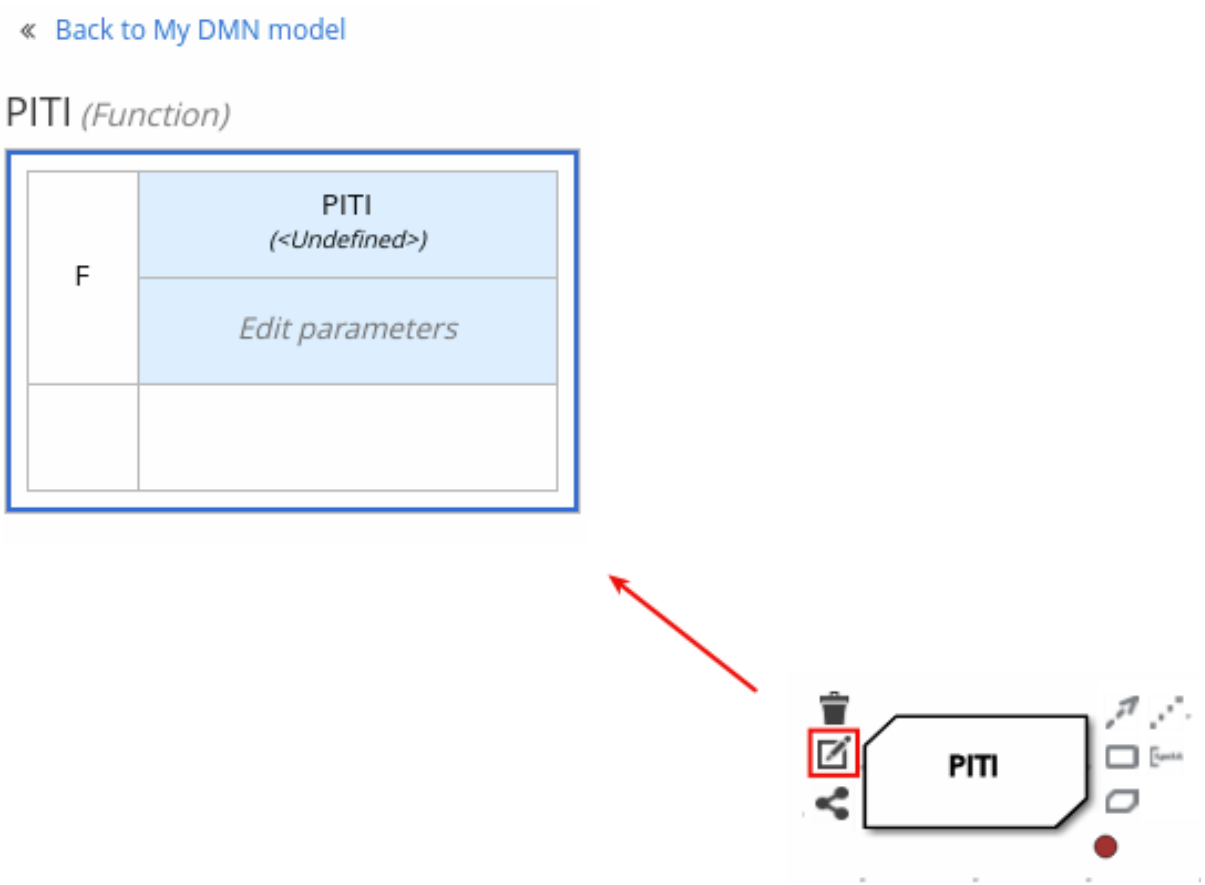


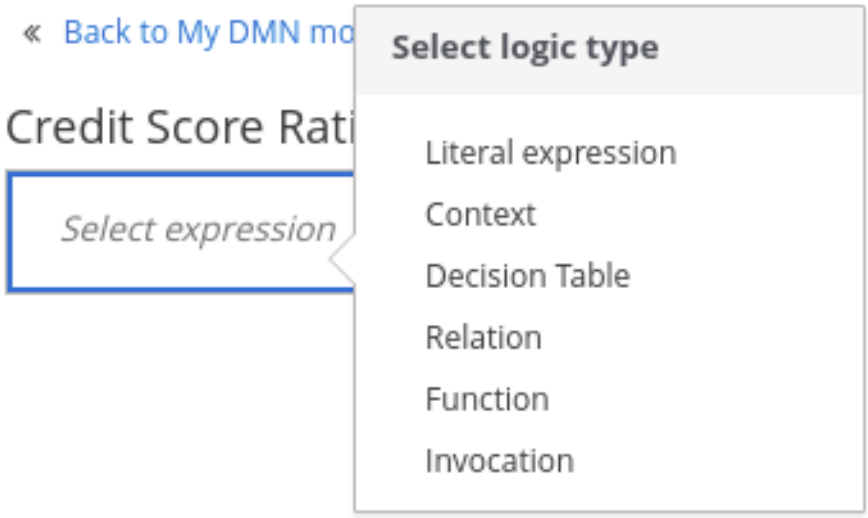
図6.3 新規ビジネスナレッジモデルのボックス式の表示



デフォルトでは、ビジネスナレッジモデルはすべて、リテラル FEEL 式、外部の JAVA または PMML 関数のネスト化されたコンテキスト式、またはあらゆる型のネスト化されたボックス式を含む、ボックス関数式として定義されます。

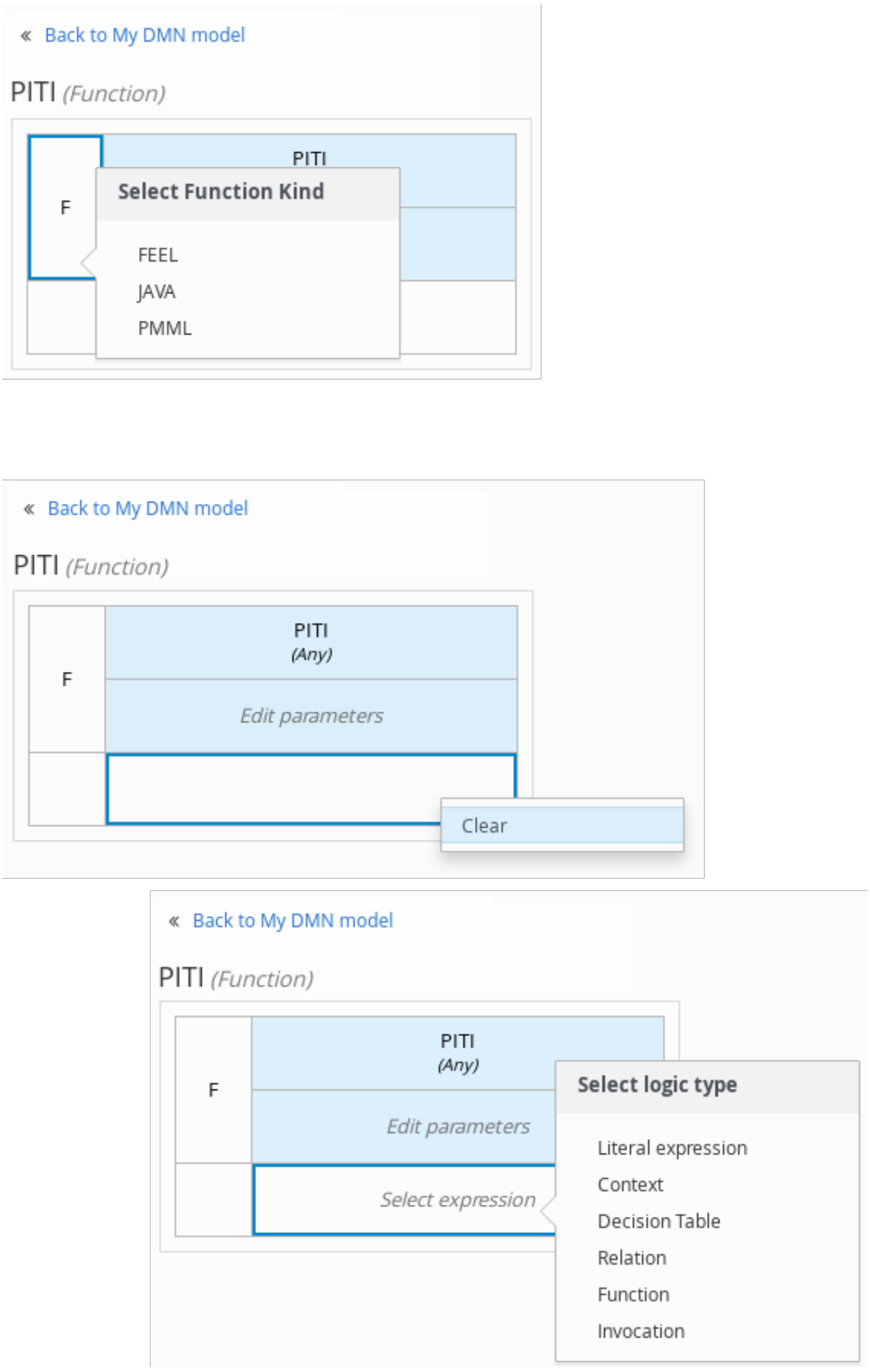
デシジョンノードの場合は、定義されていないテーブルをクリックし、ボックスリテラル式、ボックスコンテキスト式、デシジョンテーブル、またはその他の DMN ボックスコンテキスト式など、使用するボックス式のタイプを選択します。

図6.4 デシジョンノードの論理タイプの選択



ビジネスナレッジモデルの場合は、左上の関数セルをクリックして関数型を選択するか、関数値のセルを右クリックし、**Clear** を選択して、別の型のボックス式を選択します。

図6.5 ビジネスナレッジモデルの機能または他のロジックタイプの選択

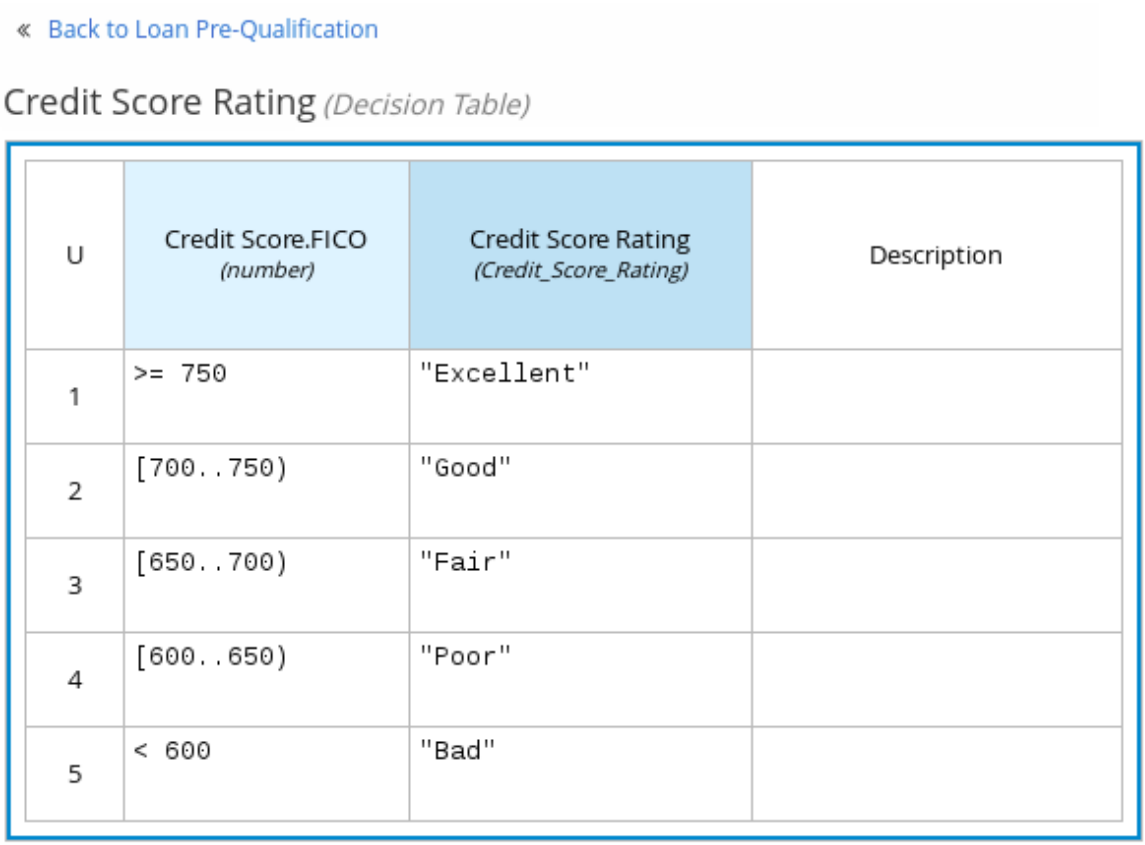


6. デザインノード (任意の式タイプ) またはビジネスナレッジモデル (関数式) のいずれかに対して

選択したボックス式デザイナーで、該当するテーブルセルをクリックして、デシジョンロジックに含めるテーブル名、変数データ型、変数名、値、関数パラメーター、バインディング、FEEL 式を定義します。
セルを右クリックして、テーブルの行および列の挿入または削除、テーブルのコンテンツの消去など、随時、追加のアクションを実行します。

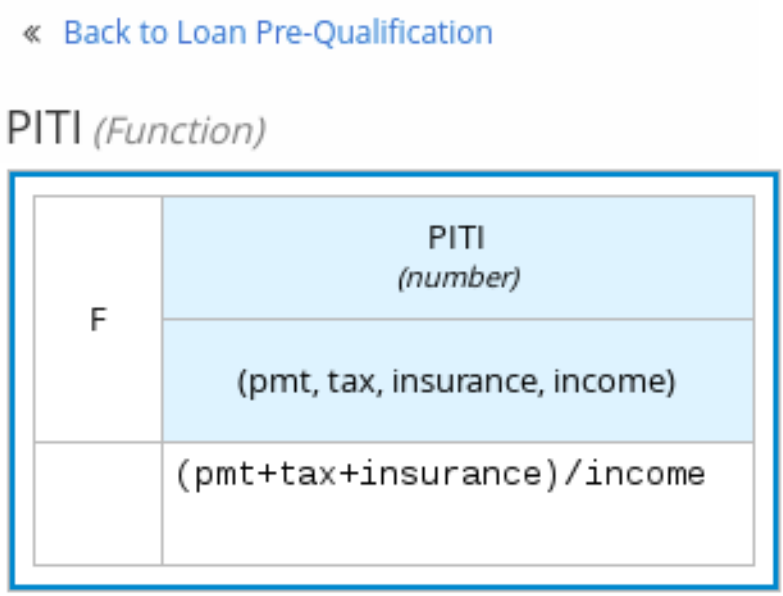
以下は、ローン申請者のクレジットスコアの定義範囲をもとに、クレジットスコアの評価を決定するデシジョンノードのデシジョンテーブルの一例です。

図6.6 クレジットスコア評価のデシジョンノードのデシジョンテーブル



以下は、元金、利子、税金、保険 (PITI) をもとに、リテラル式として住宅ローンの支払額を計算するビジネスナレッジモデルのボックス関数式の一例です。

図6.7 PITI 計算のビジネスナレッジモデルの関数



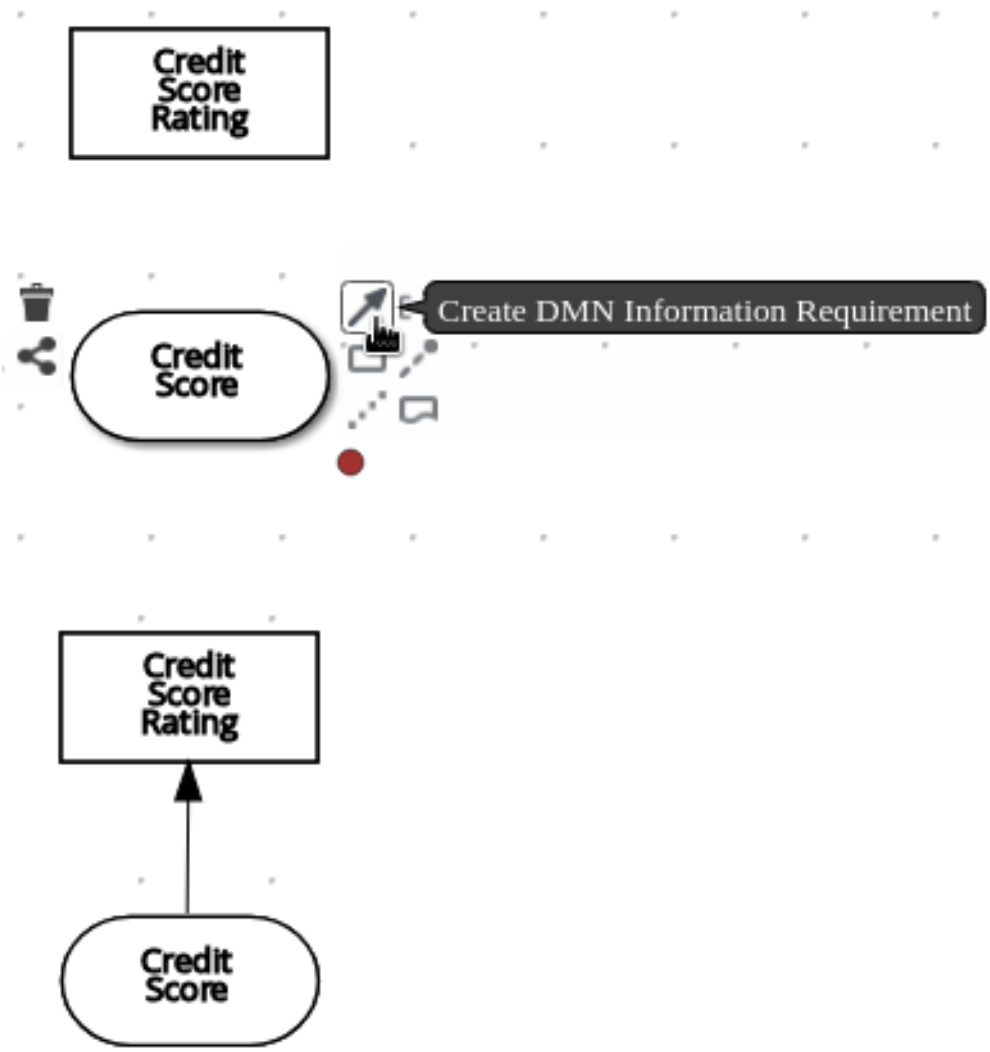
7. 選択したノードのデシジョンロジックを定義した後に、**Back to "<MODEL_NAME>"** をクリックして DRD ビューに戻ります。

8. 選択した DRD ノードについては、利用可能な接続オプションを使用して、DRD の次のノードを作成して接続するか、左のツールバーから DRD キャンバスに新規ノードをクリックしてドラッグします。
ノードタイプで、どの接続オプションがサポートされているかが決まります。たとえば、**入力データ** ノードは、アプリケーションの接続タイプを使用してデシジョンオード、ナレッジソース、またはテキストの注釈を接続できますが、**ナレッジソース** ノードは、どの DRD 要素にでも接続できます。**デシジョン** ノードは、別のデシジョンまたはテキスト注釈にだけ接続できます。

   以下の接続タイプは、ノードの種類に応じて利用できます。

   - **情報要件**: 入力データノードまたはデシジョンノードから、情報を必要とする別のデシジョンノードに移動するにはこの接続を使用します。
   - **ナレッジ要件**: ビジネスナレッジモデルからデシジョンロジックを呼び出す別のビジネスナレッジモデルまたはデシジョンノードに移動するにはこの接続を使用します。
   - **認証局の要件**: 入力データノードまたはデシジョンノードから従属するナレッジソース、またはナレッジソースからデシジョンノード、ビジネスナレッジモデルまたは別のナレッジソースに移動するにはこの接続を使用します。
   - **関連付け**: 入力データノード、デシジョンノード、ビジネスナレッジモデル、またはナレッジソースからテキストアノテーションに移動するにはこの接続を使用します。

図6.8 クレジットスコアの入力からクレジットスコア評価のデシジョンへの接続



9. 継続して、デシジョンモデルの残りの DRD コンポーネントを追加し、定義します。DMN デザイナーで定期的に **Save** をクリックして作業を保存します。

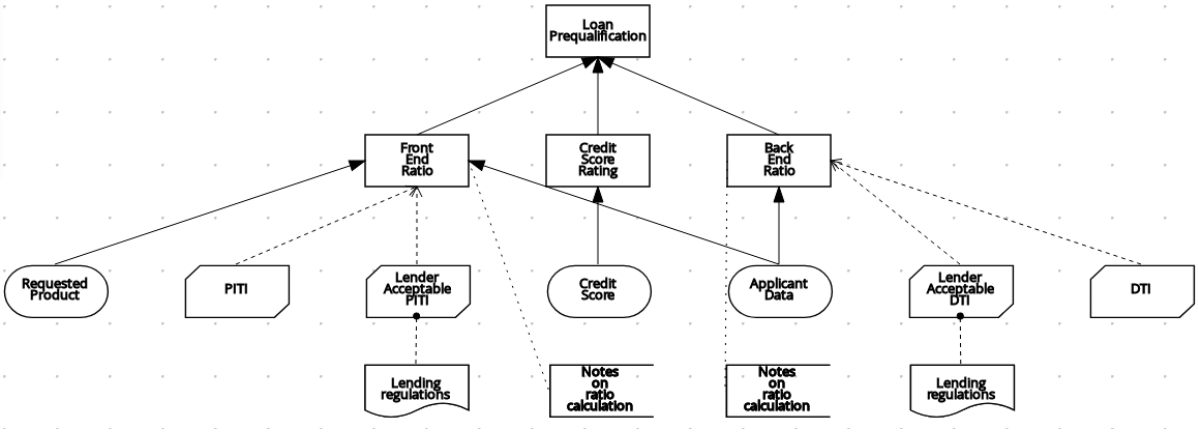
注記

DRD を定期的に保存すると、DMN デザイナーは DMN モデルを静的に検証し、モデルが完全に定義されるまでエラーメッセージを出力する可能性があります。DMN モデルをすべて定義し終えてもエラーが発生する場合は、特定の問題を随時トラブルシューティングしてください。

10. DRD の全コンポーネントを追加して定義した後に、**Save** をクリックし、完了した DRD を保存して検証します。
DRD レイアウトを調節するには、DMN デザイナーの右上のツールバーにある **Perform automatic layout** アイコンを選択してください。

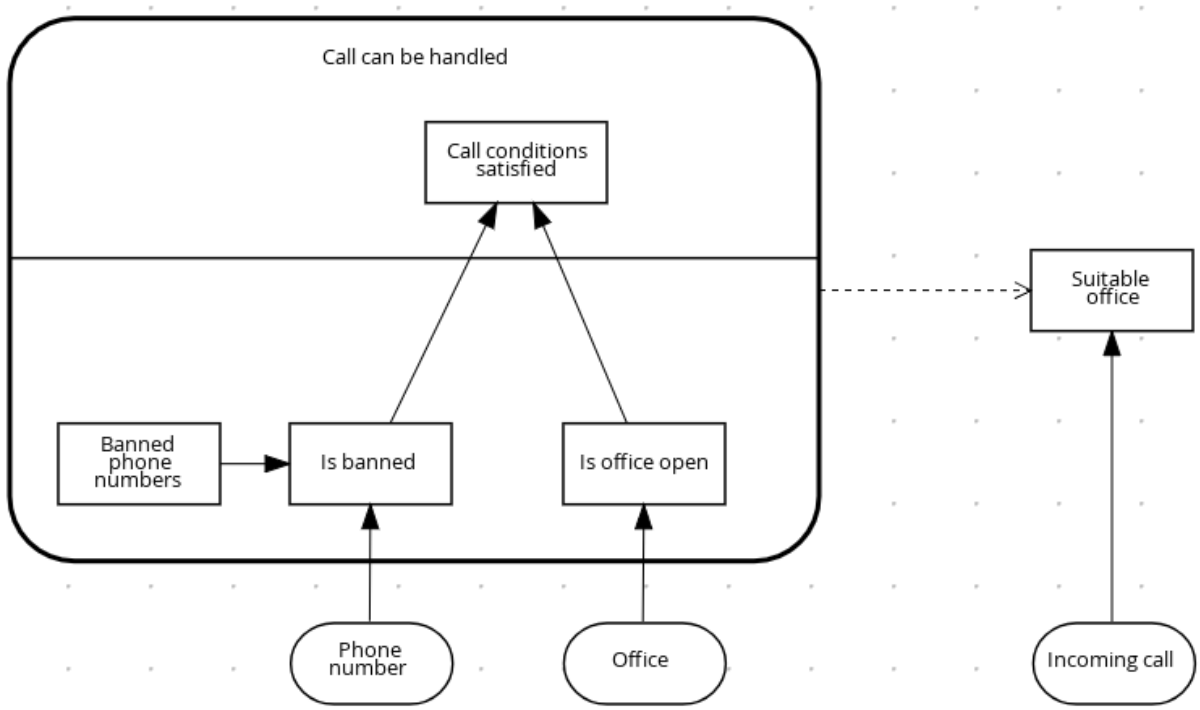
以下は、ローンの事前審査デシジョンモデルの DRD の一例です。

図6.9 ローンの事前審査の完全な DRD



以下は、再利用可能なデシジョンサービスを使用した電話対応デシジョンモデルの DRD 例です。

図6.10 デシジョンサービスを使用した電話対応の完全な DRD



DMN デシジョンサービスノードでは、一番下のセグメントデシジョンノードはデシジョンサービス外からの入力データを組み込んで、デシジョンサービスノードにある一番上のセグメントの最終地点に行き着きます。デシジョンサービスから返される上位のデシジョンは、後続のデシジョンまたは DMN モデルのビジネスナレッジ要件に実装されます。他の DMN モデル内の DMN デシジョンサービスを再利用し、異なる入力データや外向け接続で、同じデシジョンロジックを適用します。

6.1. BUSINESS CENTRAL でボックス式を使用した DMN デシジョンロジックの定義

DMN のボックス式は、意思決定要件ダイアグラム (DRD) でデシジョンノードおよびビジネスナレッジモデルの基盤ロジックを定義するのに使用するテーブルです。ボックス式には他のボックス式が含まれる場合がありますが、トップレベルのボックス式は単一の DRD アーティファクトのデシジョンロジッ

クに対応します。DRD は DMN デシジョンモデルのフローを表現し、ボックス式は個別ノードの実際のデシジョンロジックを定義します。DRD とボックス式は、完全で機能的な DMN デシジョンモデルを形成します。

Business Central で DMN デザイナーを使用して、同梱のボックス式で DRD コンポーネントのデシジョンロジックを定義できます。

前提条件

- Business Central で DMN ファイルを作成しているか、インポートしている。

手順

1. Business Central で **Menu → Design → Projects** に移動して、プロジェクト名をクリックし、変更する DMN ファイルを選択します。
2. DMN デザイナーのキャンバスで、定義するデシジョンノードまたはビジネスナレッジモデルノードを選択し、**Edit** アイコンをクリックして DMN ボックス式デザイナーを開きます。

図6.11 新規デシジョンノードのボックス式の表示

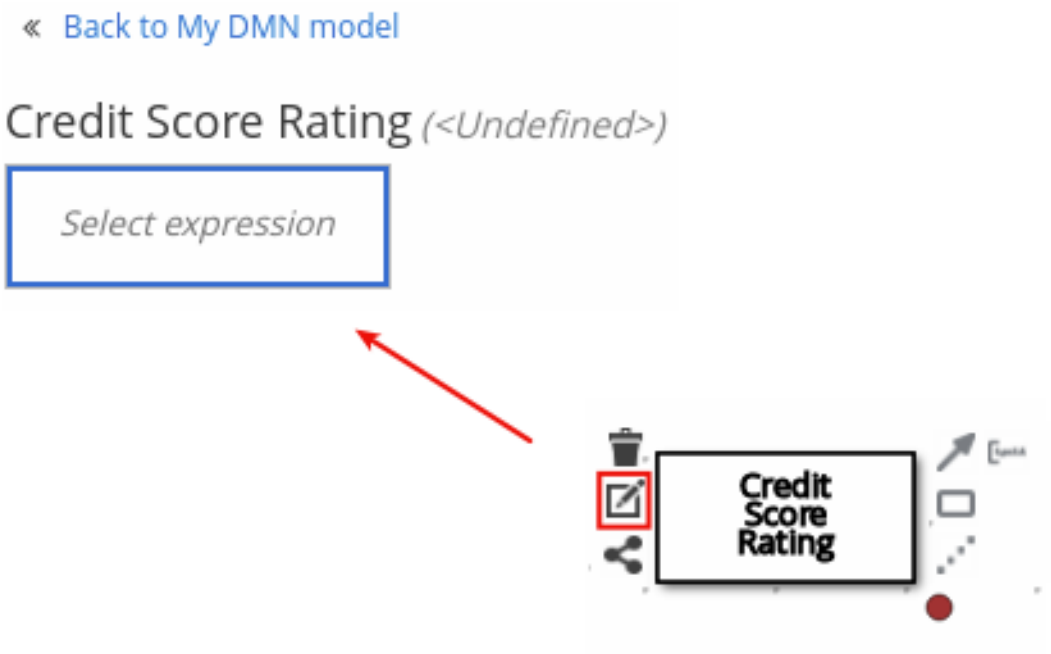
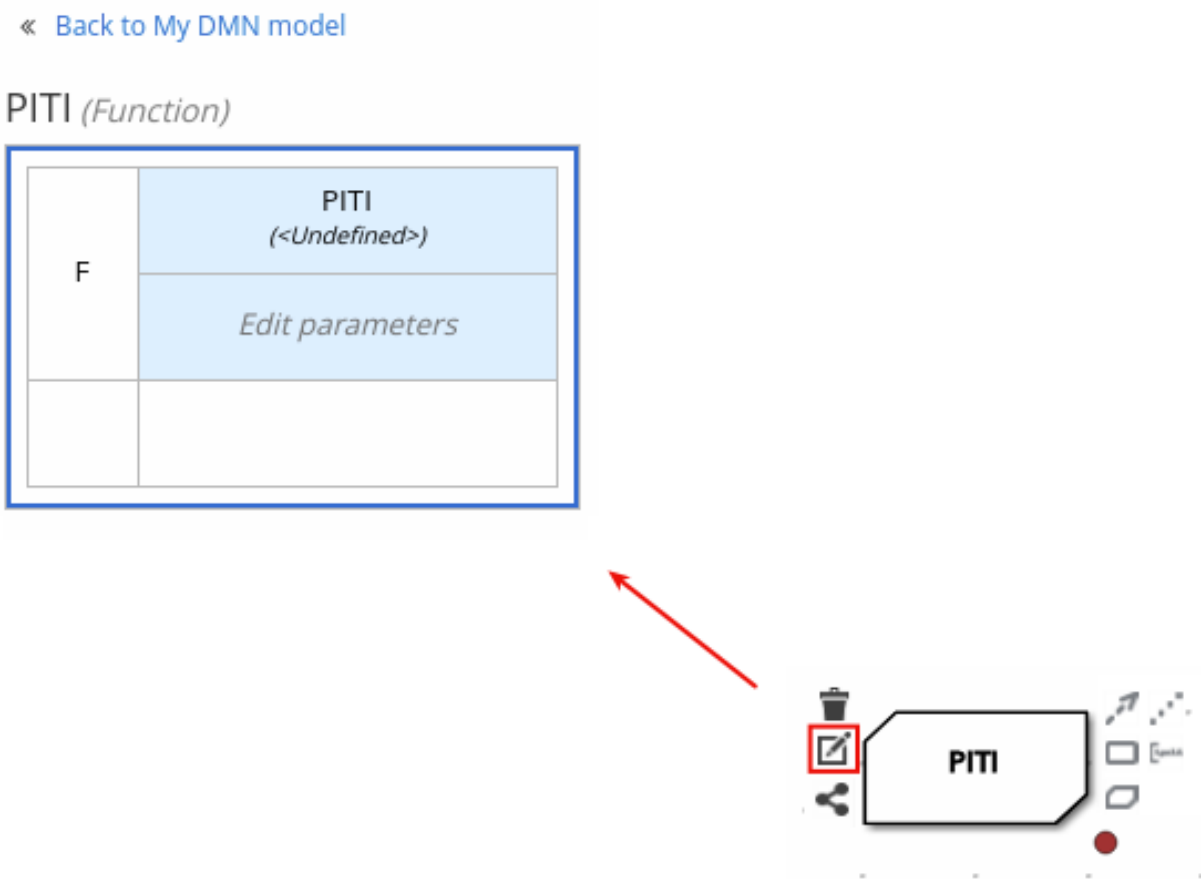


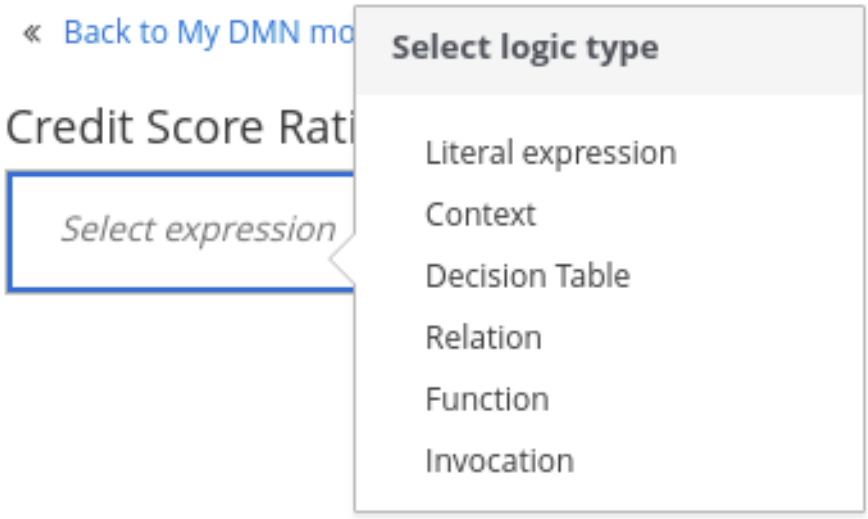
図6.12 新規ビジネスナレッジモデルのボックス式の表示



デフォルトでは、ビジネスナレッジモデルはすべて、リテラル FEEL 式、外部の JAVA または PMML 関数のネスト化されたコンテキスト式、またはあらゆる型のネスト化されたボックス式を含む、ボックス関数式として定義されます。

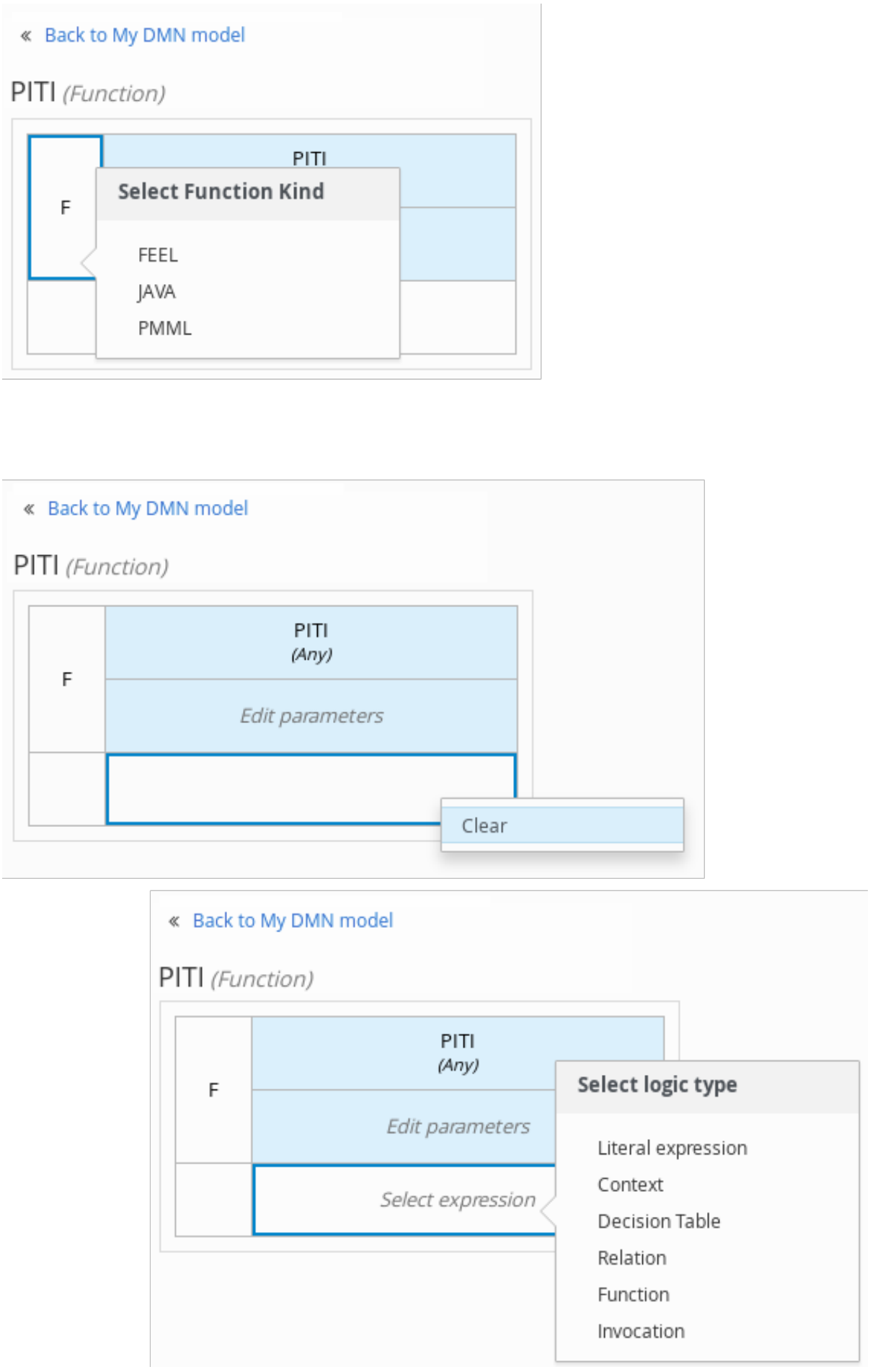
デシジョンノードの場合は、定義されていないテーブルをクリックし、ボックスリテラル式、ボックスコンテキスト式、デシジョンテーブル、またはその他の DMN ボックスコンテキスト式など、使用するボックス式のタイプを選択します。

図6.13 デシジョンノードの論理タイプの選択



ビジネスナレッジモデルノードの場合は、左上の関数セルをクリックして関数型を選択するか、関数値のセルを右クリックし、**Clear** を選択して、別の型のボックス式を選択します。

図6.14 ビジネスナレッジモデルの機能または他のロジックタイプの選択

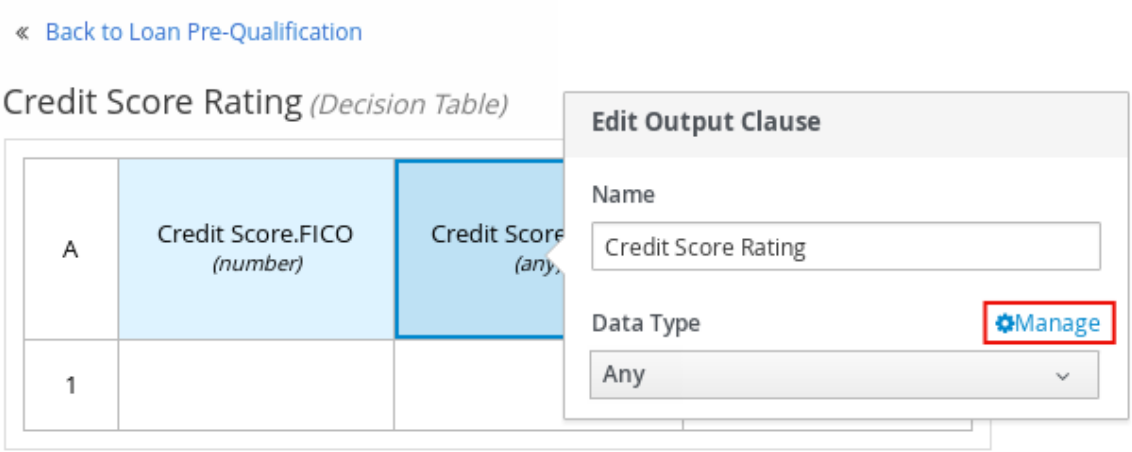


3. この例では、デシジョンノードを使用して、ボックス式タイプとして **デシジョンテーブル** を使用します。
DMN のデシジョンテーブルは、1 つ以上のルールをテーブル形式で視覚的に表します。テーブルの各行はルール 1 つで設定されており、その特定行に対する条件 (入力) と結果 (出力) を定義する列が含まれます。

4. 入力列のヘッダーをクリックして、入力条件の名前とデータ型を定義します。たとえば、入力列に **Credit Score.FICO** という名前で、**number** のデータ型を指定します。この列は、数字のクレジットスコア値または、各種ローン申請者を指定します。

5. 出力列ヘッダーをクリックして、出力値の名前とデータ型を定義します。たとえば、出力列に **Credit Score Rating** という名前を指定して、**Data Type** オプションの横で **Manage** をクリックし、**Data Types** ページに移動して、スコア評価を制約として、カスタムのデータ型を作成します。

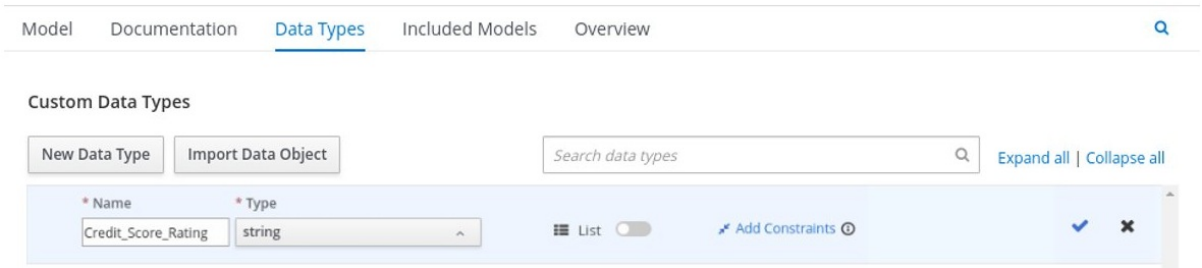
図6.15 列のヘッダー値のデータ型管理



6. **Data Types** ページで、**New Data Type** をクリックして新規データ型を追加するか、**Import Data Object** をクリックして、使用するプロジェクトから既存のデータオブジェクトを DMN データ型としてインポートします。
プロジェクトから DMN データ型としてデータオブジェクトをインポートしてから、そのオブジェクトを更新した場合は、DMN データ型としてデータオブジェクトを再インポートして DMN モデルに変更を適用する必要があります。

   この例では、**New Data Type** をクリックし、**Credit_Score_Rating** のデータ型を **string** で作成します。

図6.16 新しいデータ型の追加

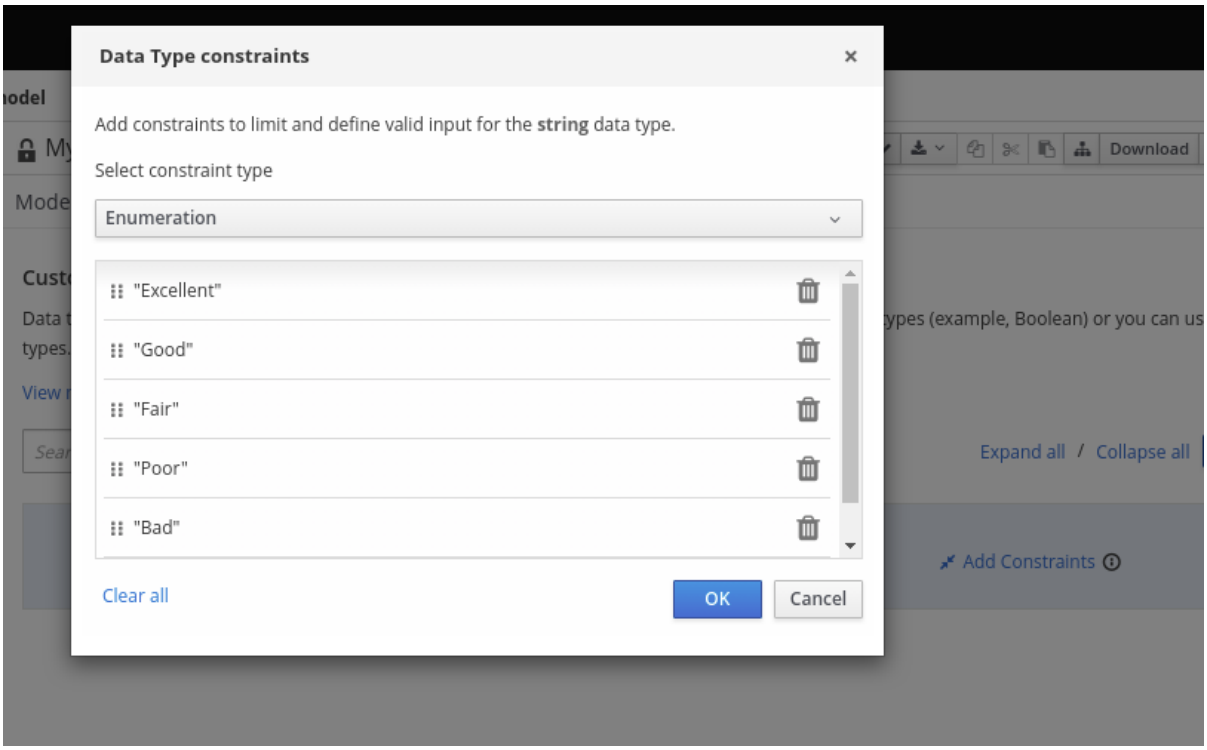


7. **Add Constraints** をクリックして、ドロップダウンオプションから **Enumeration** を選択し、以下の制約を追加します。

   - **"Excellent"**

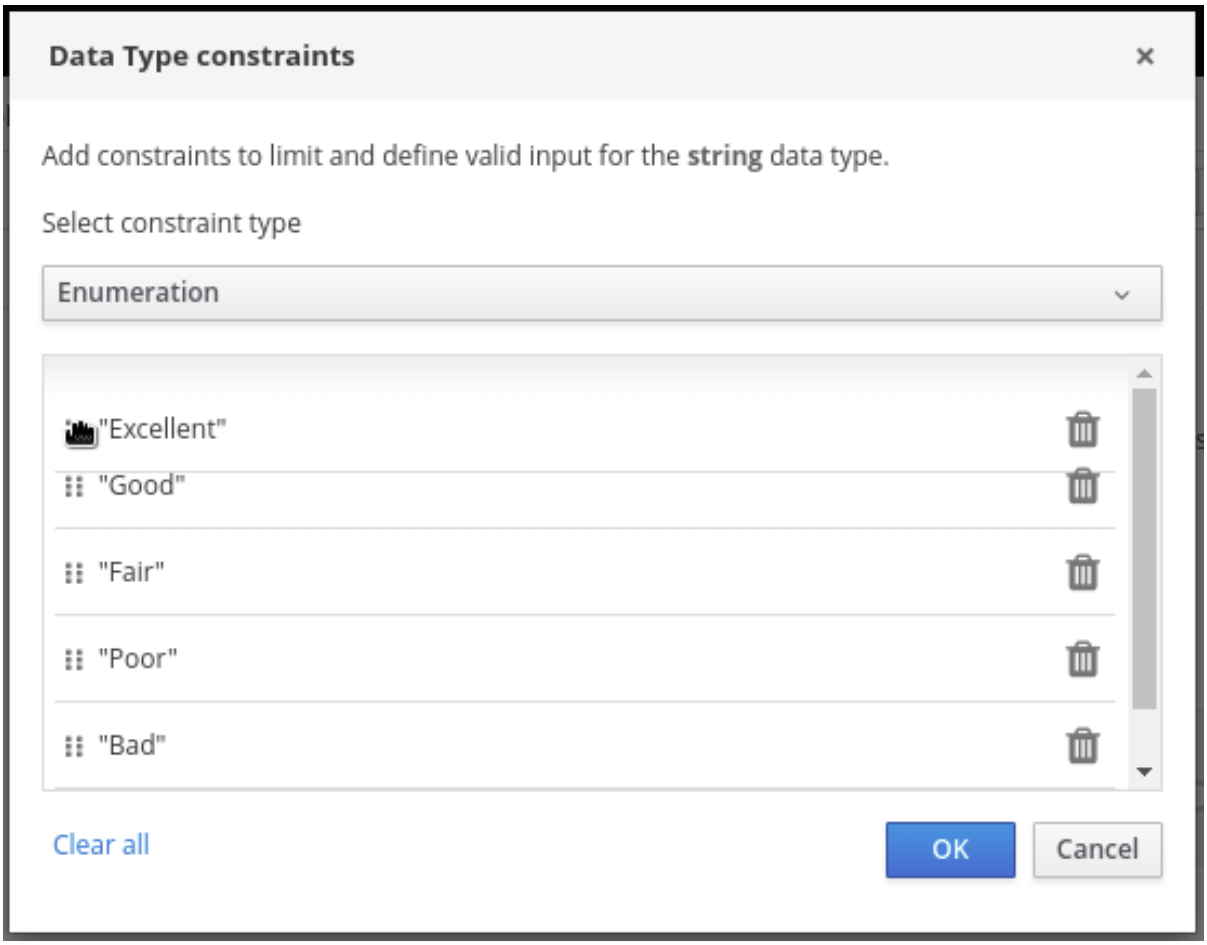
- **"Good"**
- **"Fair"**
- **"Poor"**
- **"Bad"**

図6.17 新しいデータ型への制約の追加



データ型の制約の順序を変更するには、必要に応じて、制約の行の左端をクリックしてその行をドラッグします。

図6.18 制約をドラッグして制約順序を変更



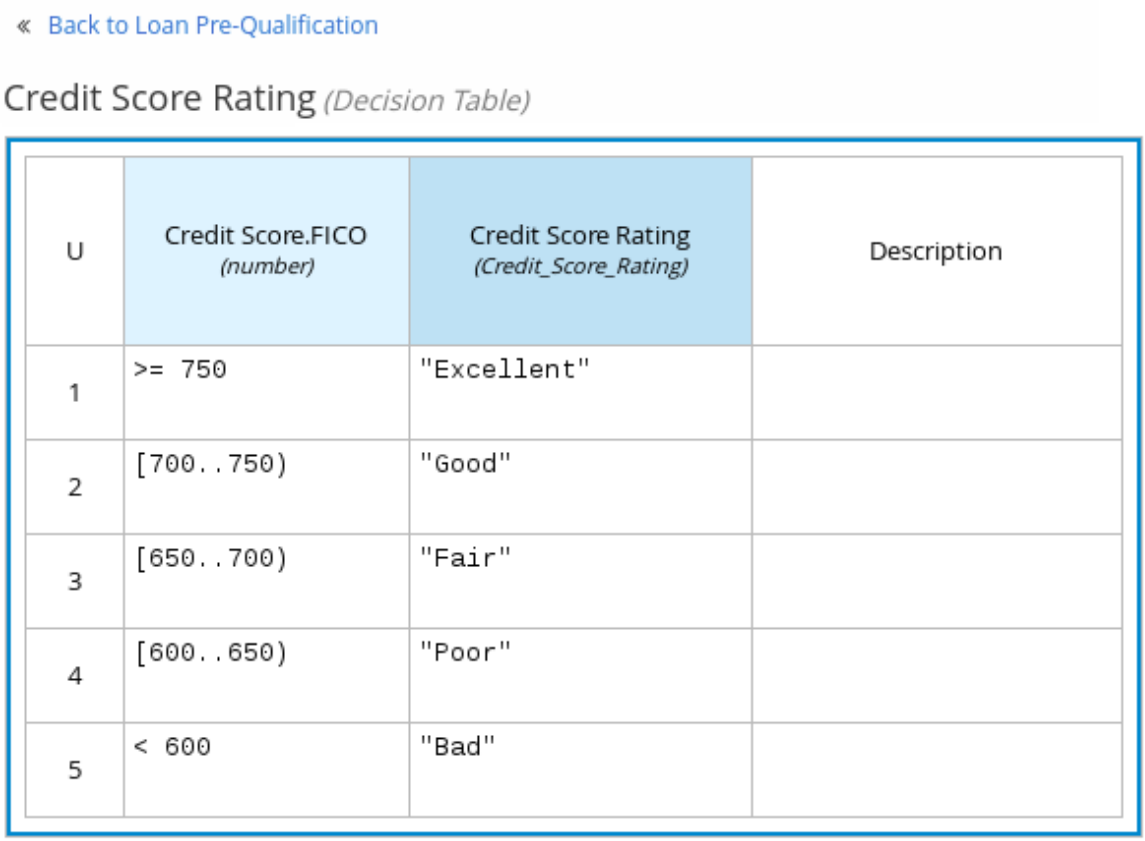
指定のデータ型の制約タイプと構文要件に関する情報は、Decision Model and Notation specification を参照してください。

8. **OK** をクリックして制約を保存し、データ型の右側のチェックマークをクリックしてデータ型を保存します。

9. **Credit Score Rating** デシジョンテーブルに戻り、**Credit Score Rating** 列ヘッダーをクリックして、保存したデータ型をこの新規カスタムデータ型に設定します。

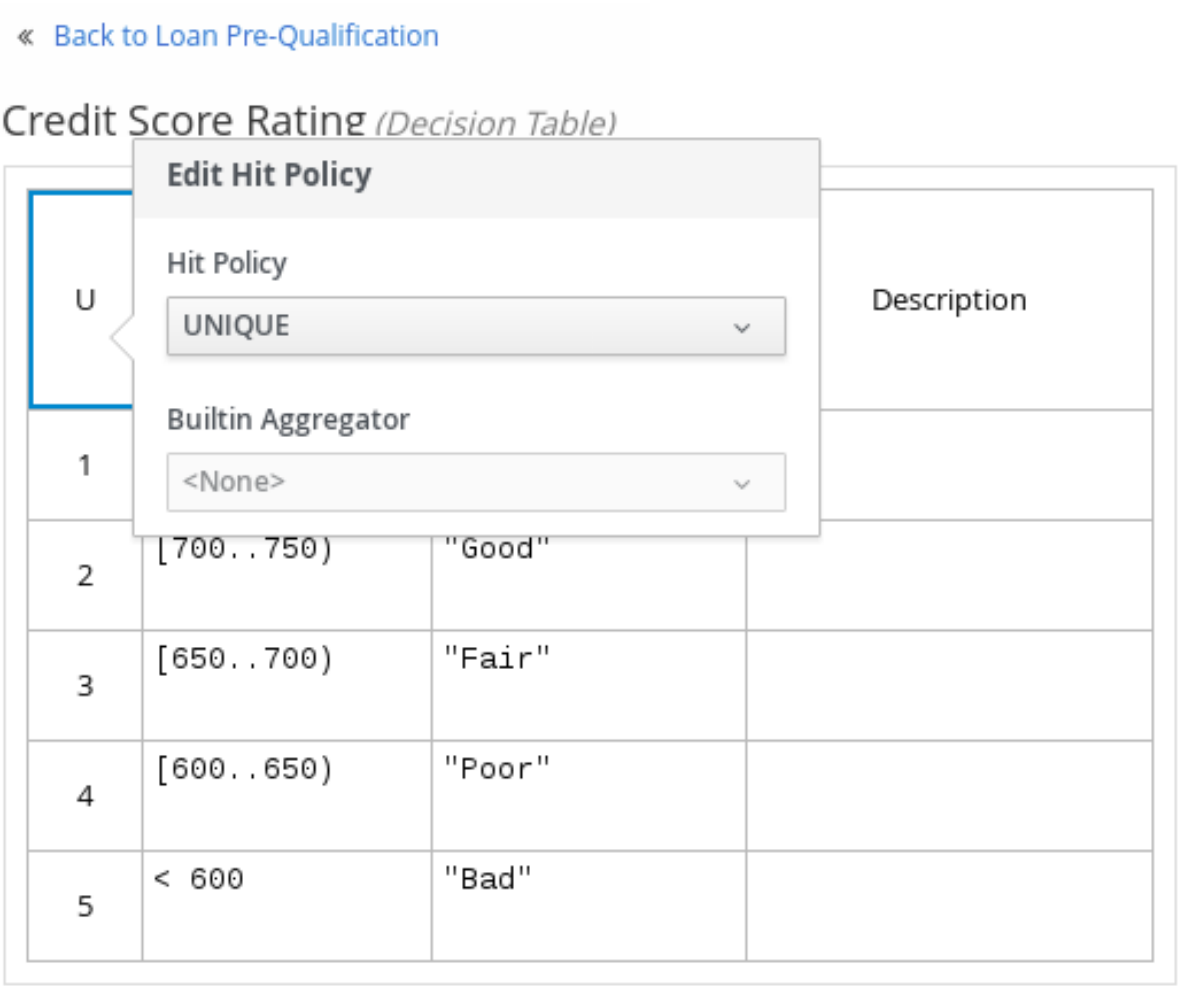
10. **Credit Score.FICO** の入力列を使用して、クレジットスコアの値またはクレジットの範囲を定義し、**Credit Score Rating** 列を使用して、**Credit_Score_Rating** のデータ型で定義した対応する評価の 1 つを指定します。
値のセルを右クリックして、行 (ルール) および列 (句) を挿入または削除します。

図6.19 クレジットスコア評価のデシジョンノードのデシジョンテーブル



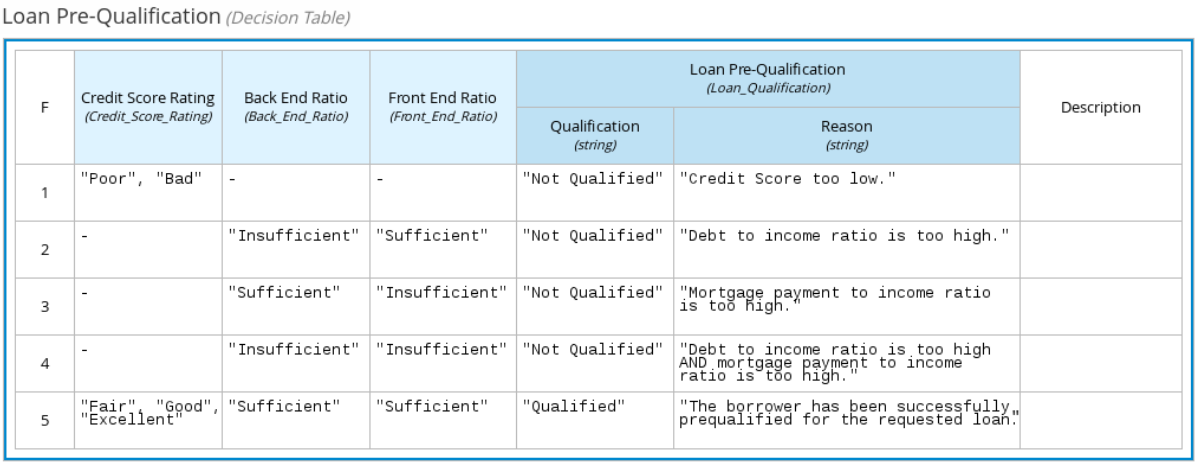
11. 全ルールを定義した後に、デシジョンテーブルの左上隅をクリックして、**ヒットポリシー** と **組み込みアグリゲーター** (COLLECT ヒットポリシーのみ) のルールを定義します。
ヒットポリシーは、デシジョンテーブルにある複数のルールが指定の入力値とマッチした場合にどのように結果に到達するのかを決定します。組み込みアグリゲーターは、COLLECT ヒットポリシーを使用する場合にどのようにルール値を累積するかを決定します。

図6.20 デシジョンテーブルヒットポリシーの定義



以下のデシジョンテーブルは、より複雑なデシジョンテーブルで、ローン事前審査のデシジョンモデルで終端デシジョンノードとして、申請者のローン適正を決定します。

図6.21 ローン事前審査のデシジョンテーブル



F	Credit Score Rating (Credit_Score_Rating)	Back End Ratio (Back_End_Ratio)	Front End Ratio (Front_End_Ratio)	Loan Pre-Qualification (Loan_Qualification)		Description
				Qualification (string)	Reason (string)	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

デシジョンテーブル以外のボックス式タイプの場合は、ボックス式のテーブルを移動して、デシジョンロジックの変数とパラメーターを定義するのと同様にこれらのガイドラインに従いますが、ボックス式のタイプの要件に従うようにしてください。ボックスリテラル式など、ボックス式の一部では列が1つのテーブルの場合や、関数、テキスト、呼び出し式などのボックス式は、他のタイプのボックス式がネスト化された、列が複数あるテーブルの場合もあります。

たとえば、以下のボックスコンテキスト式では、サブコンテキスト式が含まれるフロントエンドの割合計算として表現されている PITI (元金 (Principal)、利子 (Interest)、税金 (Tax)、保険 (Insurance)) をもとに、ローンの申請者が最小限必要とされるローンの支払いをしているかを決定するパラメーターを定義します。

図6.22 フロントエンドクライアント PITI 割合のボックスコンテキスト式

Front End Ratio *(Context)*

#	Front End Ratio *(Front_End_Ratio)*		
1	Client PITI *(number)*	#	PITI
		1	pmt *(number)*	`(Requested Product.Amount*((Requested Product.Rate/100)/12))/ (1-(1/(1+(Requested Product.Rate/100)/12)**-Requested Product.Term))`
		2	tax *(number)*	`Applicant Data.Monthly.Tax`
		3	insurance *(number)*	`Applicant Data.Monthly.Insurance`
		4	income *(number)*	`Applicant Data.Monthly.Income`
	<result>	`if Client PITI <= Lender Acceptable PITI() then "Sufficient" else "Insufficient"`	

以下のボックス関数式では、ネスト化されたコンテキスト式として定義された関数値を使用し、融資のデシジョンのビジネスナレッジモデルとして、住宅ローンの月額を決定します。

図6.23 ビジネスナレッジモデルのローン計算で使用するボックス関数式

InstallmentCalculation *(Function)*

F	InstallmentCalculation *(number)*		
	(ProductType, Rate, Term, Amount)		
	1	MonthlyFee *(number)*	`if ProductType ="STANDARD LOAN" then 20.00 else if ProductType ="SPECIAL LOAN" then 25.00 else null`
	2	MonthlyRepayment *(number)*	`(Amount *Rate/12) / (1 - (1 + Rate/12)**-Term)`
		<result>	`MonthlyRepayment+MonthlyFee`

各ボックス式のタイプの例および詳細は、「ボックス式の DMN デシジョンロジック」 を参照してください。

6.2. BUSINESS CENTRAL での DMN ボックス式のカスタムデータ型の作成

Business Central の DMN のボックス式では、データ型により、ボックス式の関連のテーブル、列、またはフィールド内で使用するデータの構造を決定します。デフォルトの DMN データ型 (文字列、数字、ブール値など) を使用するか、独自のデータ型を作成して、ボックス式の値に実装する新たなフィールドや制限を指定することもできます。

ボックス式のカスタムのデータ型として、単純なデータ型または構造化されたデータ型のいずれかを作成できます。

- **単純な** データ型では、名前とタイプの割当のみを指定できます。例: **Age (number)**
- **構造化された** データ型には、親データ型に関連する複数のフィールドが含まれます。例: **Name (string)**、**Age (number)**、**Email (string)** のフィールドが含まれる単一の型 **Person**

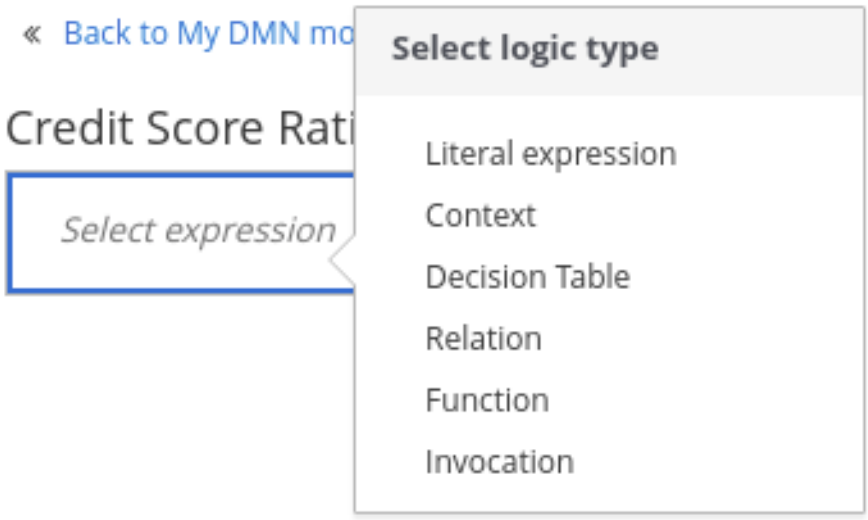
前提条件

- Business Central で DMN ファイルを作成しているか、インポートしている。

手順

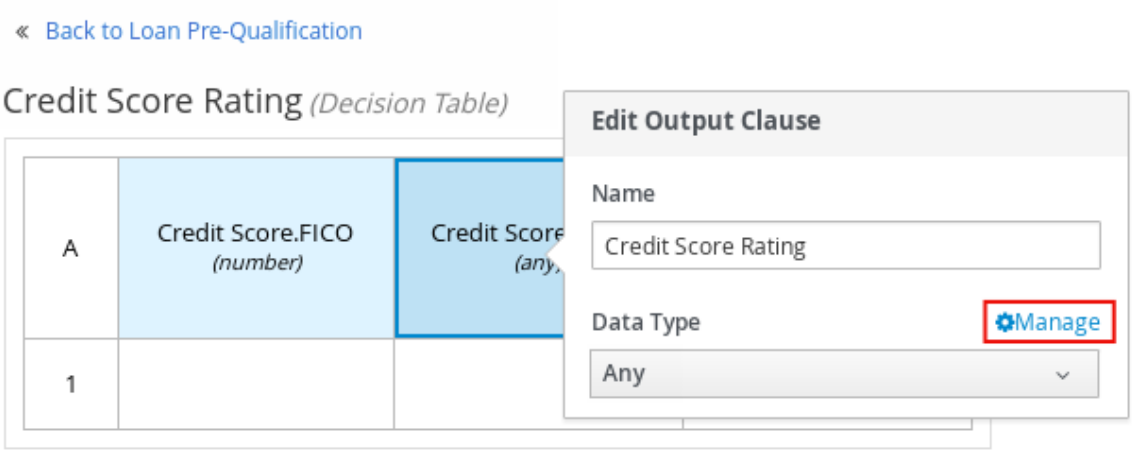
1. Business Central で **Menu → Design → Projects** に移動して、プロジェクト名をクリックし、変更する DMN ファイルを選択します。
2. DMN デザイナーのキャンバスで、データ型を定義するデシジョンノードまたはビジネスナレッジモデルを選択し、**Edit** アイコンをクリックして DMN ボックス式デザイナーを開きます。
3. ボックス式が定義されていないデシジョンノードの場合は、定義されていないテーブルをクリックし、ボックスリテラル式、ボックスコンテキスト式、デシジョンテーブル、またはその他の DMN ボックスコンテキスト式など、使用するボックス式のタイプを選択します。

   図6.24 デシジョンノードの論理タイプの選択



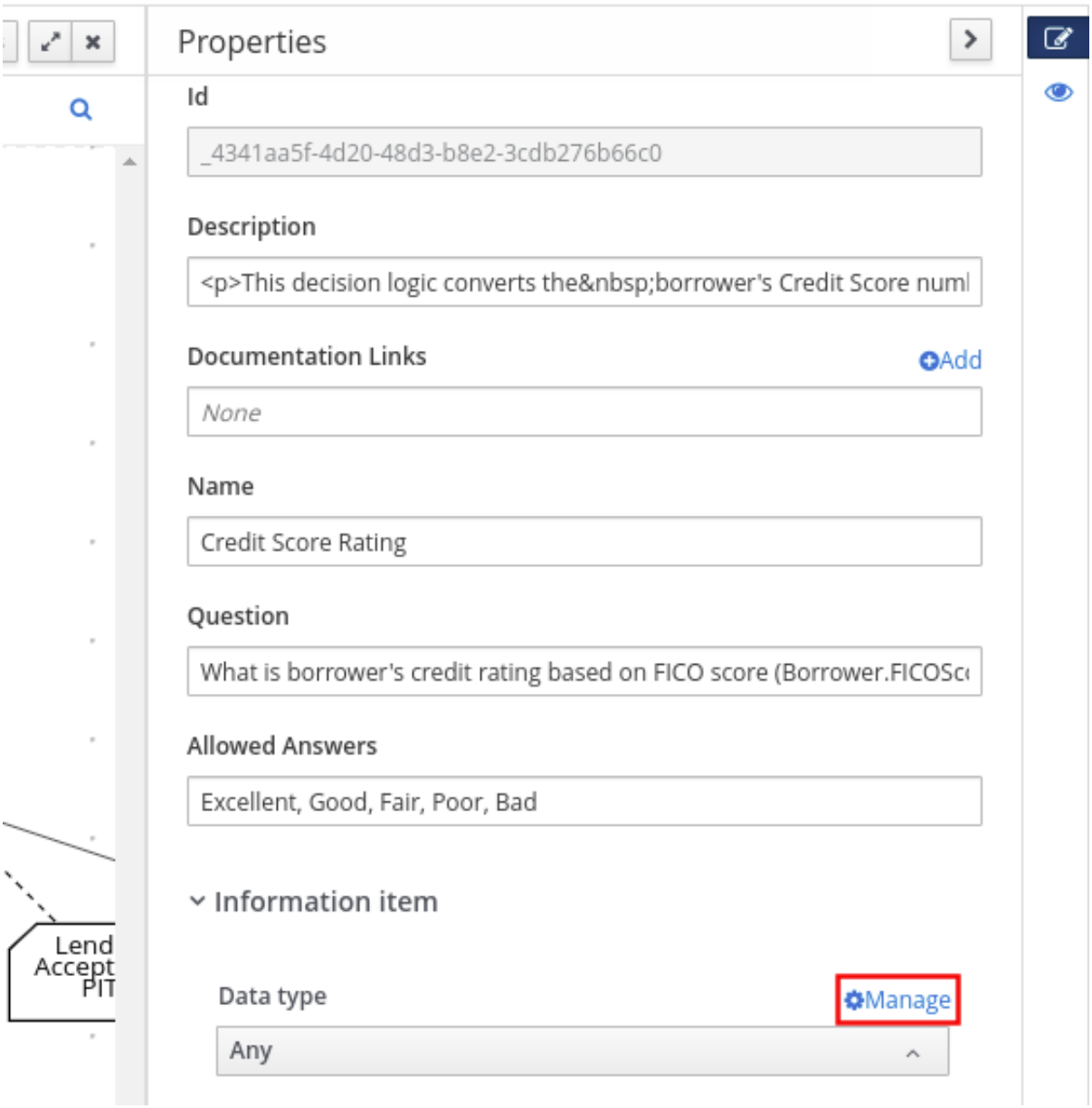
4. テーブルヘッダのセルまたは、(ボックス式のタイプに合わせて) データ型を定義するパラメーターフィールドをクリックし、**Manage** をクリックして、カスタムのデータ型を作成する **Data Types** ページに移動します。

図6.25 列のヘッダー値のデータ型管理



DMN デザイナーの右上隅の **Properties** アイコンを選択して、指定のデシジョンノードまたはビジネスナレッジモデルノードのカスタムデータ型を設定して管理することも可能です。

図6.26 意思決定要件 (DRD) プロパティーでのデータ型の管理



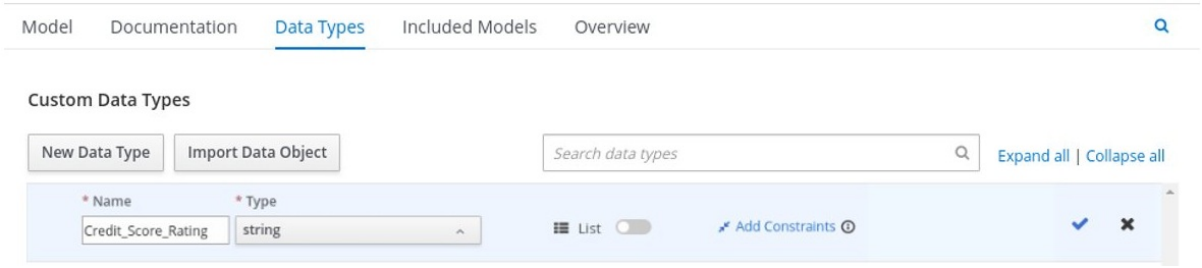
ボックス式で指定のセルに定義するデータ型により、ボックス式の関連のテーブル、列、フィールド内で使用するデータの構造を決定します。

この例では、DMN デシジョンテーブルの出力列 **クレジットスコア評価** は、申請者のクレジットスコアをもとにカスタムのクレジットスコア評価を定義します。

5. **Data Types** ページで、**New Data Type** をクリックして新規データ型を追加するか、**Import Data Object** をクリックして、使用するプロジェクトから既存のデータオブジェクトを DMN データ型としてインポートします。
プロジェクトから DMN データ型としてデータオブジェクトをインポートしてから、そのオブジェクトを更新した場合は、DMN データ型としてデータオブジェクトを再インポートして DMN モデルに変更を適用する必要があります。

   この例では、**New Data Type** をクリックし、**Credit_Score_Rating** のデータ型を **string** で作成します。

   図6.27 新しいデータ型の追加

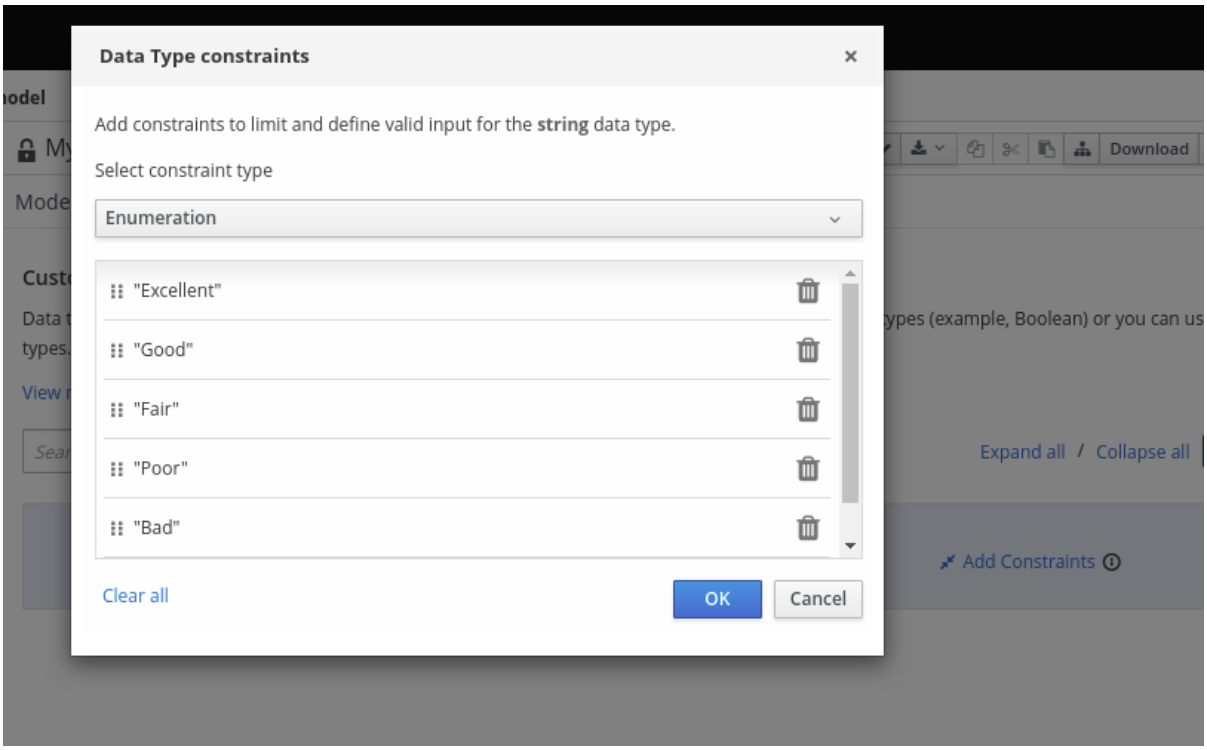


   データ型がアイテムの一覧を必要とする場合は、**List** 設定を有効にします。

6. **Add Constraints** をクリックして、ドロップダウンオプションから **Enumeration** を選択し、以下の制約を追加します。

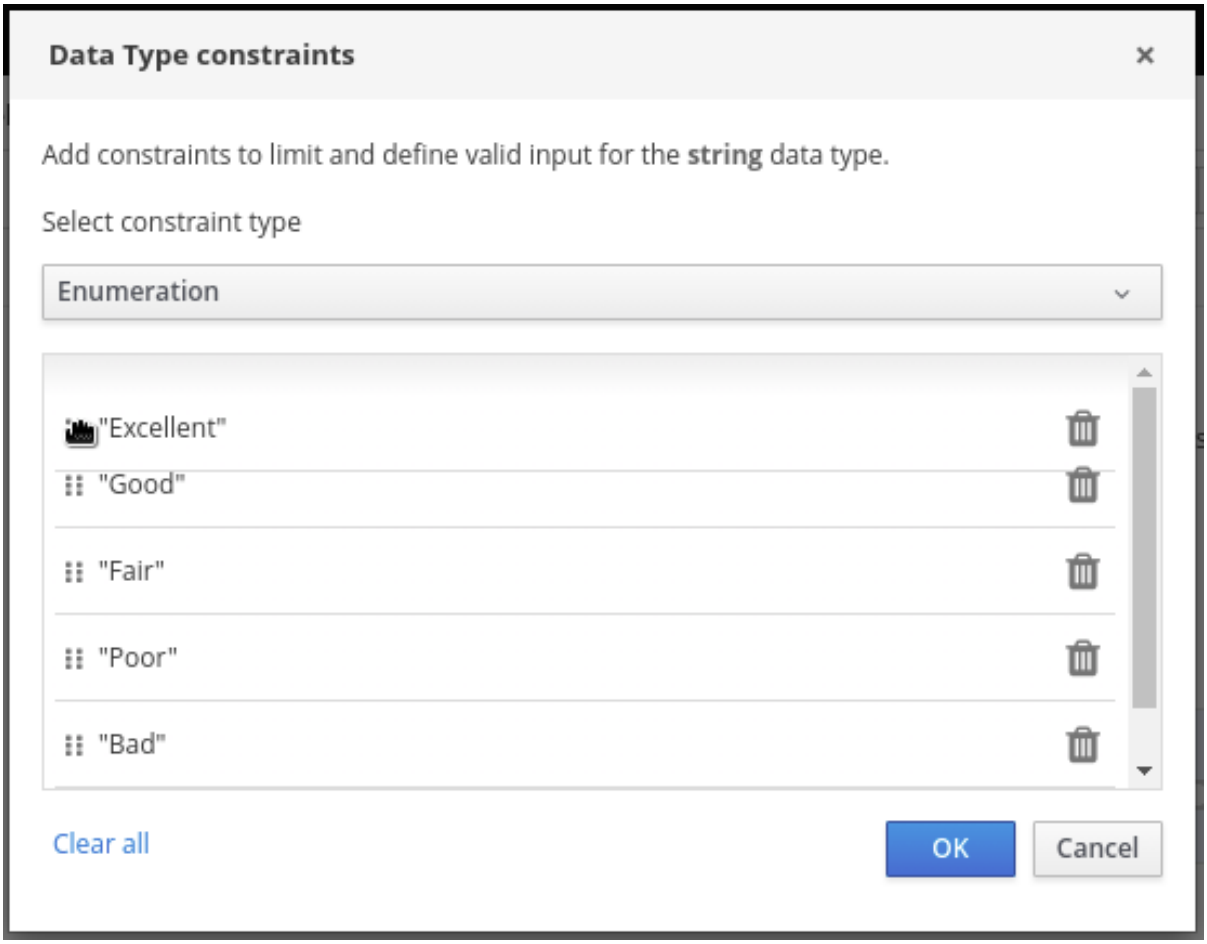
   - **"Excellent"**
   - **"Good"**
   - **"Fair"**
   - **"Poor"**
   - **"Bad"**

図6.28 新しいデータ型への制約の追加



データ型の制約の順序を変更するには、必要に応じて、制約の行の左端をクリックしてその行をドラッグします。

図6.29 制約をドラッグして制約順序を変更



指定のデータ型の制約タイプと構文要件に関する情報は、Decision Model and Notation specification を参照してください。

7. **OK** をクリックして制約を保存し、データ型の右側のチェックマークをクリックしてデータ型を保存します。

8. **Credit Score Rating** デシジョンテーブルに戻り、**Credit Score Rating** 列ヘッダーをクリックして、保存したデータ型をこの新規カスタムデータ型に設定し、指定した評価制約で、対象の列のルール値を定義します。

図6.30 クレジットスコア評価のデシジョンテーブル

« Back to Loan Pre-Qualification

Credit Score Rating *(Decision Table)*

U	Credit Score.FICO *(number)*	Credit Score Rating *(Credit_Score_Rating)*	Description
1	>= 750	"Excellent"	
2	[700..750)	"Good"	
3	[650..700)	"Fair"	
4	[600..650)	"Poor"	
5	< 600	"Bad"	

シナリオの DMN デシジョンモデルで、**Credit Score Rating** デシジョンが、以下の **Loan Prequalification** デシジョンに流れ、このデシジョンではカスタムのデータ型が必要です。

図6.31 ローン事前審査のデシジョンテーブル

Loan Pre-Qualification *(Decision Table)*

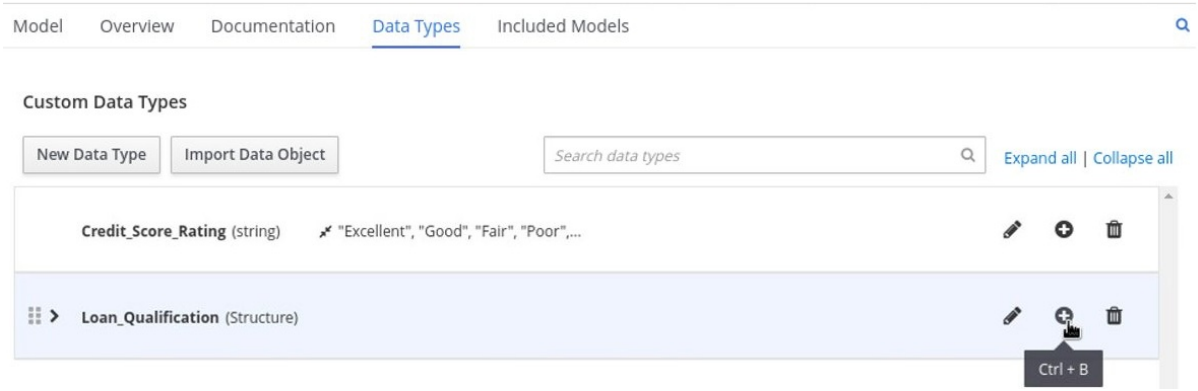
F	Credit Score Rating *(<Undefined>)*	Back End Ratio *(<Undefined>)*	Front End Ratio *(<Undefined>)*	Loan Pre-Qualification *(<Undefined>)*		Description
				Qualification *(string)*	Reason *(string)*	
1						

9. この例をそのまま使用し、**Data Types** ウィンドウに戻り、**New Data Type** をクリックして、**Loan_Qualification** データ型を制約なしの **Structure** として作成します。
新しい構造化データ型を保存すると、最初のサブフィールドが表示され、この親データ型で、ネスト化されたデータフィールドの定義を開始できます。デシジョンテーブル内にネスト化さ

れた列ヘッダー、コンテキストまたは関数式でネスト化されたパラメーターなど、ボックス式の構造化された親データ型と関連するこれらのサブフィールドを使用できます。

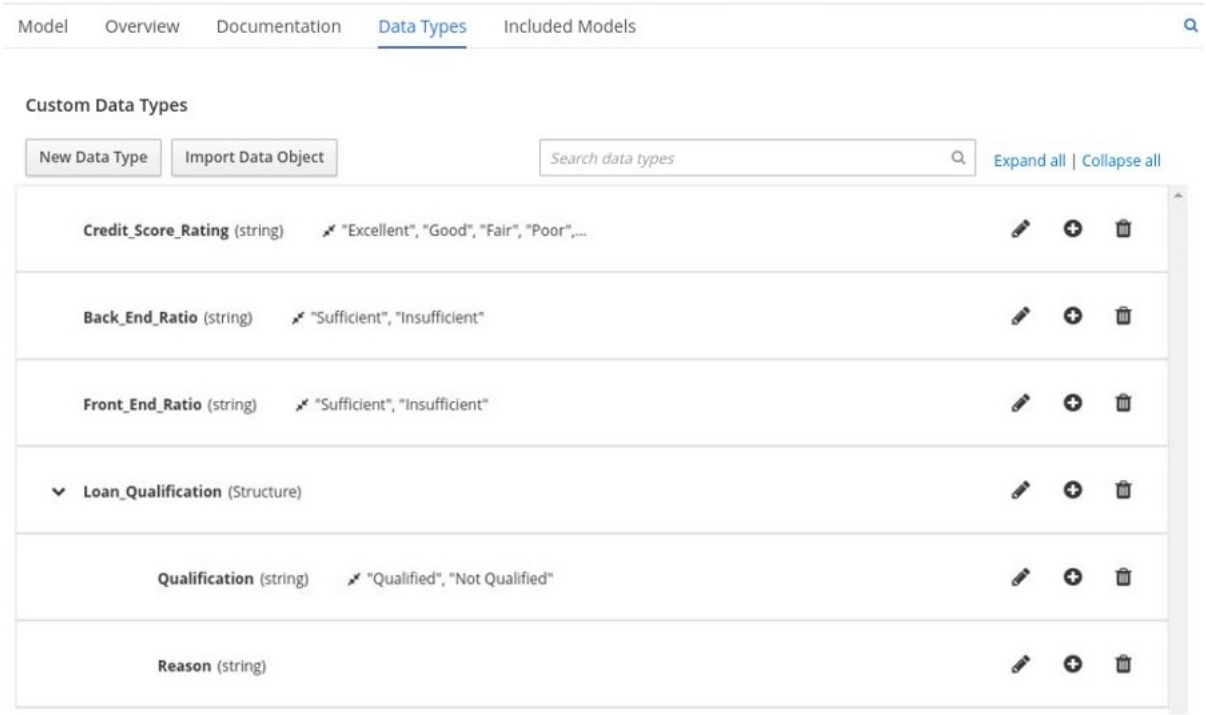
追加のサブフィールドについては **Loan_Qualification** のデータ型の横にある追加のアイコンをクリックします。

図6.32 新しい構造化データ型へのネスト化フィールドの追加



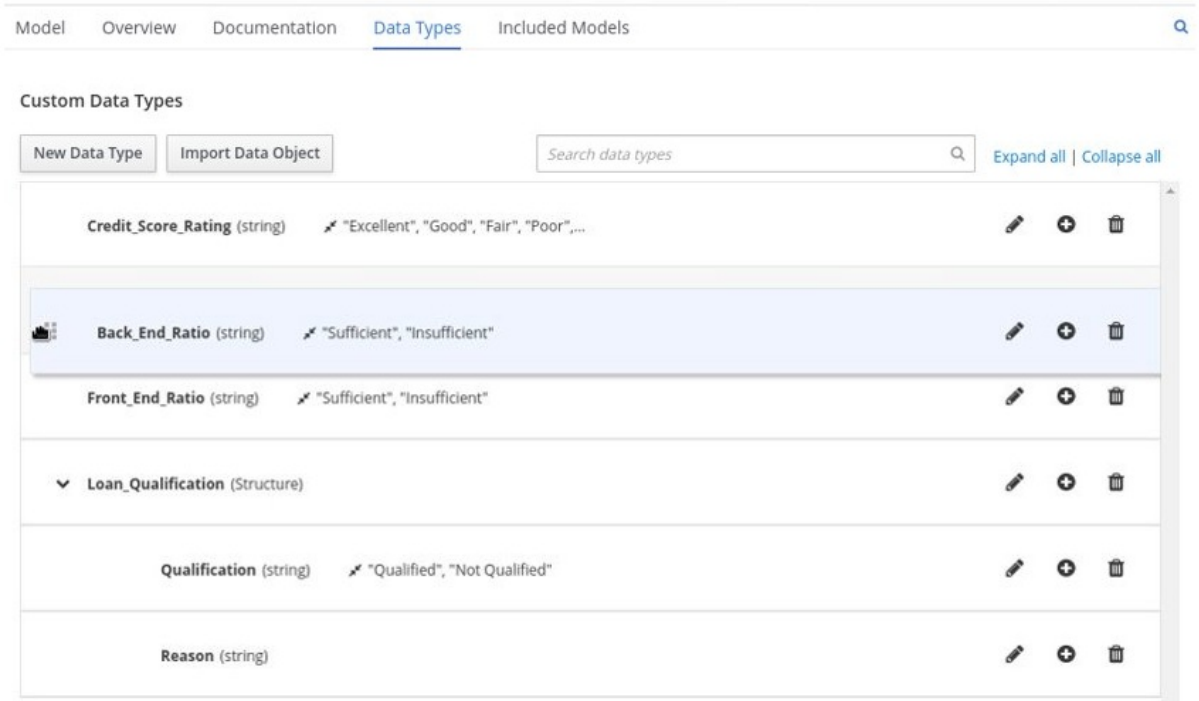
10. この例では、構造化された **Loan_Qualification** データ型に、**"Qualified"** と **"Not Qualified"** の列挙制約がある **Qualification** フィールドと、制約のない **Reason** フィールドを追加します。また、**"Sufficient"** と **"Insufficient"** の列挙制約がある、単純なデータ型 **Back_End_Ratio** と **Front_End_Ratio** を追加します。
作成する各データ型の右側にあるチェックマークをクリックして変更を保存します。

図6.33 ネスト化されたデータ型への制約の追加



データ型の順序やネスト化を変更するには、必要に応じて、データ型の行の左端をクリックしてその行をドラッグします。

図6.34 データ型をドラッグしてデータ型の順番やネスト化を変更



11. デシジョンテーブルに戻り、列ごとに、列ヘッダーセルをクリックし、対応する新規のカスタムデータ型に、このデータ型を設定して、(該当する場合には) 指定した制約で必要に応じて列のルール値を定義します。

図6.35 ローン事前審査のデシジョンテーブル

Loan Pre-Qualification *(Decision Table)*

F	Credit Score Rating *(Credit_Score_Rating)*	Back End Ratio *(Back_End_Ratio)*	Front End Ratio *(Front_End_Ratio)*	Loan Pre-Qualification *(Loan_Qualification)*		Description
				Qualification *(string)*	Reason *(string)*	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

デシジョンテーブル以外のボックス式タイプの場合は、ボックス式のテーブルを移動して、必要に応じてカスタムのデータ型を定義するのと同様に、これらのガイドラインに従うようにしてください。

たとえば、以下のボックス関数式はカスタムの **tCandidate** と **tProfile** の構造化データ型を使用して、データを関連付けてオンライン出会い系でのデート相手としての適合性を判断します。

図6.36 オンライン出会い系でデート相手としての適合性に使用するボックス関数式

Evaluate Match *(Function)*

F	Evaluate Match *(tCandidate)*		
	(Lonely Soul, Candidate)		
	1	Profile1 *(tProfile)*	Lonely Soul
	2	Profile2 *(tProfile)*	Candidate
	3	Is Match *(boolean)*	Is Soul a Match(Lonely Soul, Candidate) and Is Soul a Match(Candidate, Lonely Soul)
	4	Score *(number)*	Number of Matching Interests(Lonely Soul, Candidate) - absolute(Lonely Soul.Age - Candidate.Age)
		<result>	*Select expression*

図6.37 オンライン出会い系でデート相手としての適合性に使用するカスタムデータ型の定義

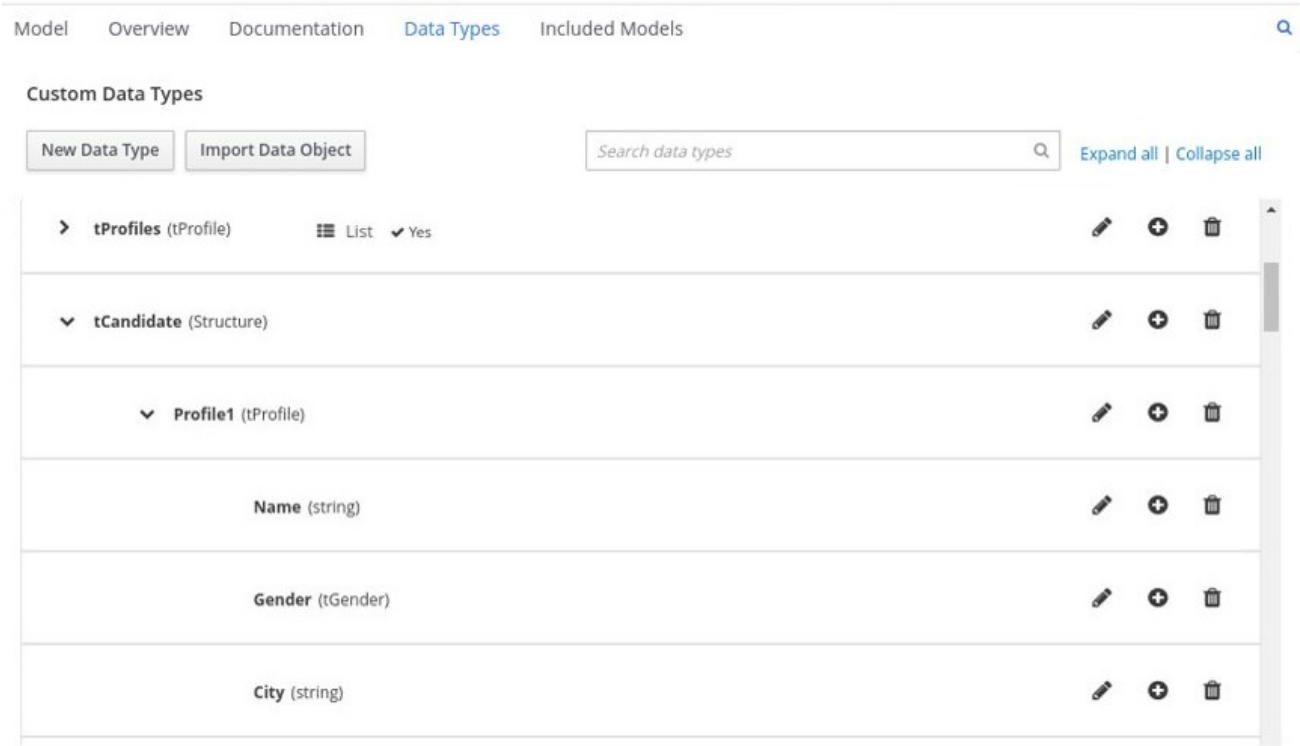
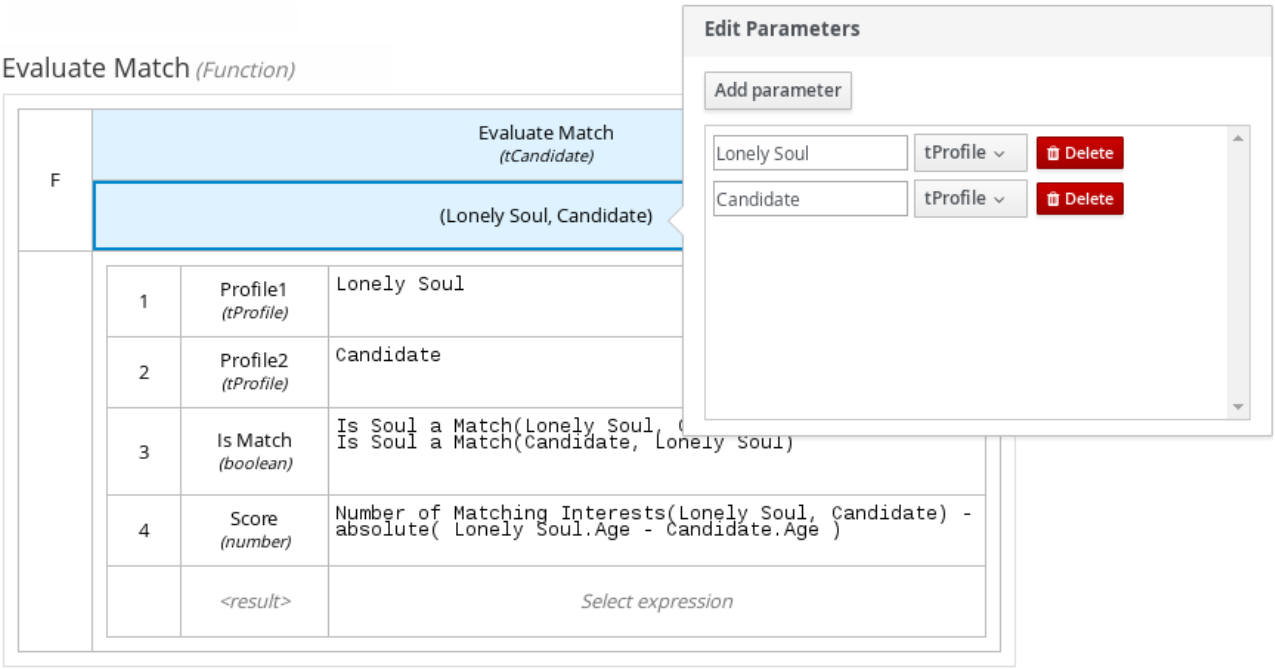


図6.38 オンライン出会い系でデート相手としての適合性に使用するカスタムデータ型を含むパラメーター定義



6.3. BUSINESS CENTRAL の DMN ファイルに含まれるモデル

Business Central の DMN デザイナーで **Included Models** タブを使用して、指定の DMN ファイルに、お使いのプロジェクトの他の DMN モデルや、Predictive Model Markup Language (PMML) モデルを使用できます。別の DMN ファイルに DMN モデルを追加すると、同じ意思決定要件ダイアグラム (DRD) にある両方のモデルのノードやロジックをすべて使用できます。DMN ファイルに PMML モデルを追加すると、その PMML モデルを DMN デシジョンノードまたはビジネスナレッジモデルノードのボックス関数式として呼び出すことができます。

Business Central では、他のプロジェクトの DMN または PMML モデルは追加できません。

6.3.1. Business Central の DMN ファイルへの他の DMN モデルの追加

Business Central では、指定の DMN ファイルに、プロジェクトの他の DMN モデルを追加できます。別の DMN ファイルに DMN モデルを追加すると、同じ意思決定要件ダイアグラム (DRD) にある両方のモデルのノードやロジックをすべて使用できますが、追加したモデルのノードは編集できません。追加したモデルのノードを編集するには、直接追加したモデルのソースファイルを更新する必要があります。追加した DMN モデルのソースファイルを更新した場合は、DMN モデルが追加されている DMN ファイルを表示して (または一旦閉じて、開き直して)、変更を確認してください。

Business Central では、他のプロジェクトの DMN モデルは追加できません。

前提条件

- Business Central で (**.dmn** ファイルとして) 同じプロジェクトに DMN モデルが作成されているか、インポートされている。

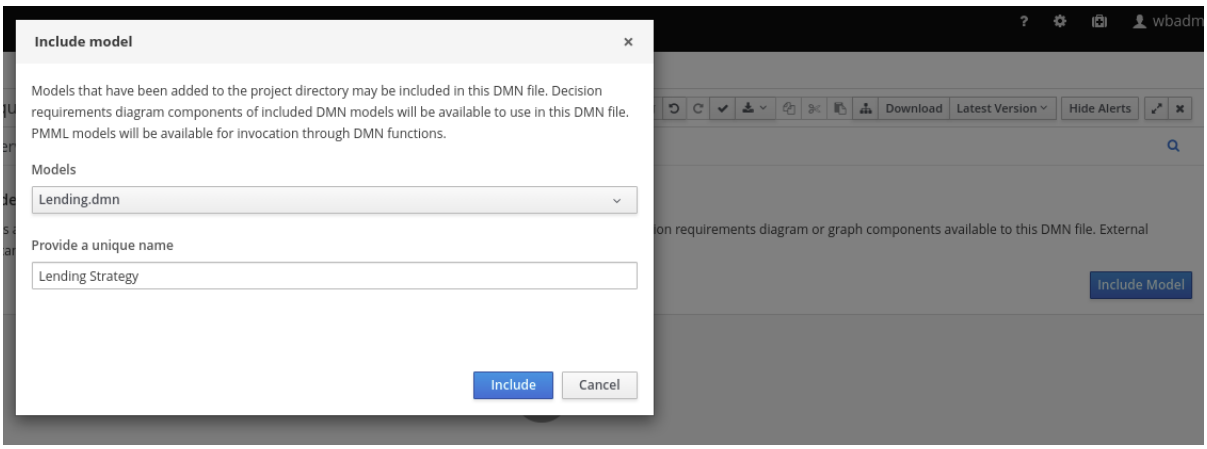
手順

1. Business Central で **Menu → Design → Projects** に移動して、プロジェクト名をクリックし、変更する DMN ファイルを選択します。

2. DMN デザイナーで、**Included Models** タブをクリックします。

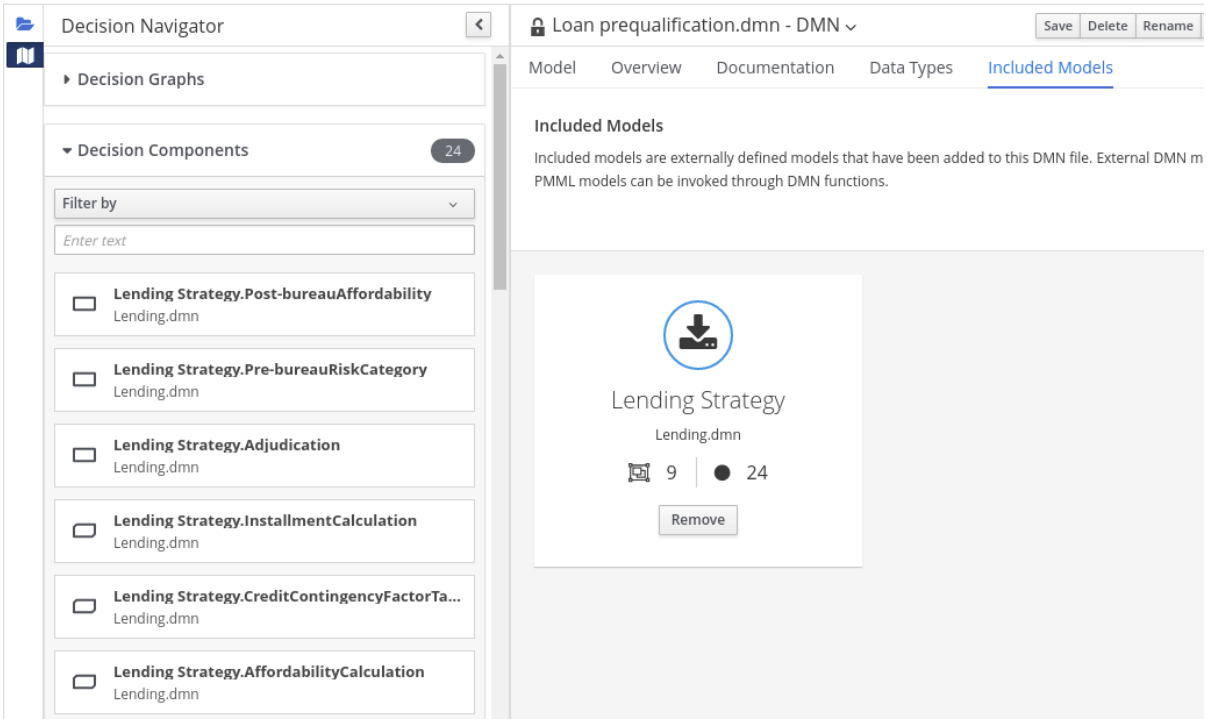
3. **Include Model** をクリックして、**Models** リストのプロジェクトから DMN モデルを選択し、追加するモデルの一意名を入力して、**Include** をクリックします。

図6.39 DMN モデルの追加



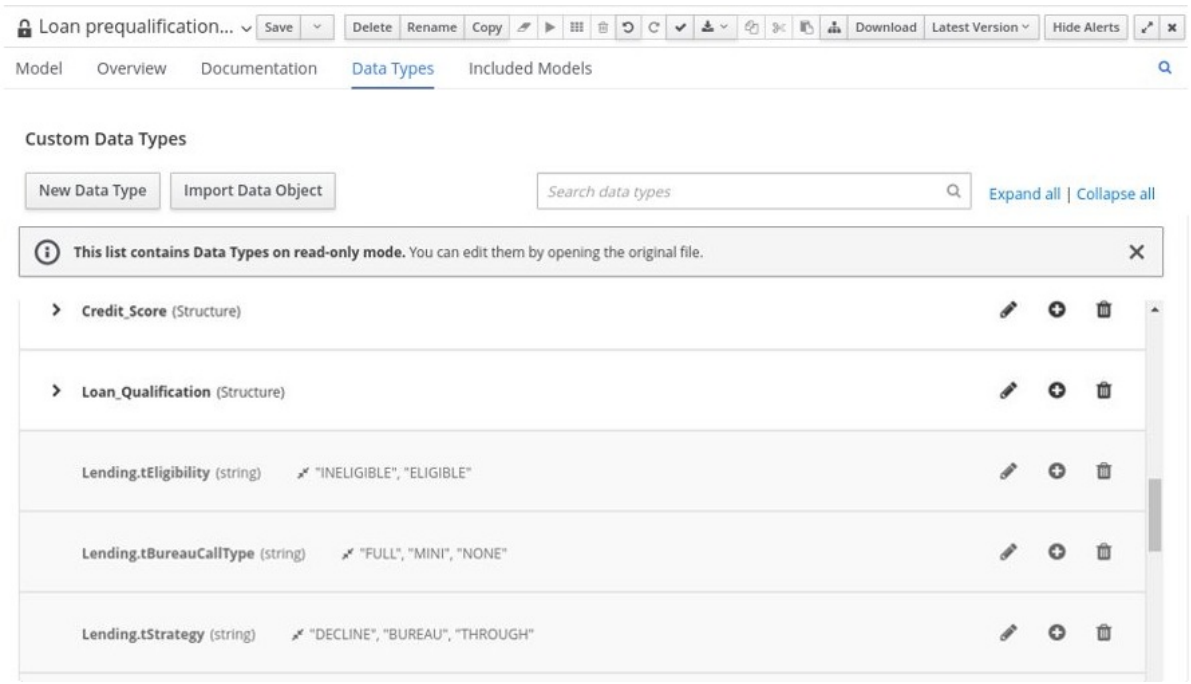
この DMN ファイルに、DMN ファイルが追加され、追加したモデルからの DRD ノードがすべて **Decision Navigator** ビューの **Decision Components** に表示されます。

図6.40 追加した DMN モデルのデシジョンコンポーネントを含む DMN ファイル



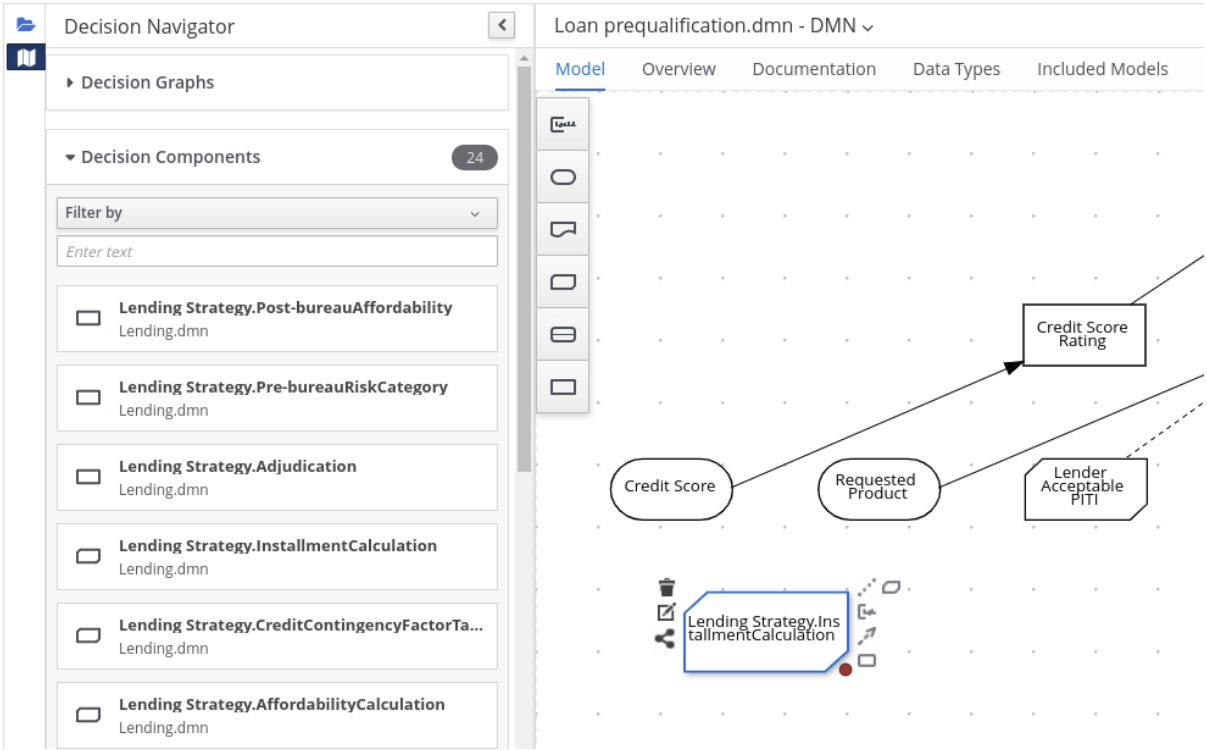
追加したモデルからの全データ型が DMN ファイルの **Data Types** タブにも、読み取り専用モードで表示されます。

図6.41 追加した DMN モデルのデータ型を含む DMN ファイル



4. DMN デザイナーの **Model** タブで、追加した DRD コンポーネントをクリックしてキャンバスにドラッグし、DRD での実装を開始します。

図6.42 追加した DMN モデルの DRD コンポーネントの追加



追加したモデルの DRD ノードまたはデータタイプを編集するには、追加したモデルのソースファイルを直接更新する必要があります。追加した DMN モデルのソースファイルを更新した場合は、DMN モデルが追加されている DMN ファイルを表示して (または一旦閉じて、開き直して)、変更を確認してください。

追加したモデル名を編集するか、追加したモデルを DMN ファイルから削除するには、DMN デザイナーの **Included Models** タブを使用します。

重要

追加したモデルを削除すると、DRD で現在使用している追加モデルのノードもすべて削除されます。

6.3.2. Business Central の DMN ファイルへの PMML モデルの追加

Business Central では、指定した DMN ファイルに、プロジェクトの Predictive Model Markup Language (PMML) を追加できます。DMN ファイルに PMML モデルを追加すると、その PMML モデルを DMN デシジョンノードまたはビジネスナレッジモデルノードのボックス関数式として呼び出すことができます。追加した PMML モデルのソースファイルを更新した場合は、DMN ファイルの PMML モデルを削除して追加し直し、ソースの変更を適用する必要があります。

Business Central では、他のプロジェクトからの PMML モデルを追加できません。

前提条件

- Business Central で (**.pmml** ファイルとして) 同じプロジェクトに PMML モデルがインポートされている。

手順

1. DMN プロジェクトで、プロジェクトの **pom.xml** ファイルに以下の依存関係を追加して、PMML 評価を有効にします。

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
  <scope>provided</scope>
</dependency>

<!-- Alternative dependencies for JPMML Evaluator, override `kie-pmml` dependency -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-jpmml</artifactId>
  <version>${rhpam.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jpmml</groupId>
  <artifactId>pmml-evaluator</artifactId>
  <version>1.5.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jpmml</groupId>
  <artifactId>pmml-evaluator-extension</artifactId>
  <version>1.5.1</version>
  <scope>provided</scope>
</dependency>
```

Business Central でプロジェクト **pom.xml** にアクセスするには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

Java Evaluator API for PMML (JPMML) と完全な PMML 仕様の実装を使用する場合は、DMN プロジェクトで代わりとなる JPMML の依存関係を使用してください。JPMML の依存関係と標準の **kie-pmml** の依存関係が両方ある場合は、**kie-pmml** の依存関係が無効になります。JPMML のライセンス条件に関する情報は、Openscoring.io を参照してください。

重要

従来の **kie-pmml** 依存関係は、Red Hat Process Automation Manager 7.10.0 で非推奨になり、将来の Red Hat Process Automation Manager リリースで **kie-pmml-trusty** 依存関係に置き換えられる予定です。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルの **dependencyManagement** のセクションに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. DMN プロジェクトに JPMML 依存関係を追加して JPMML Evaluator を使用する場合は、以下の JAR ファイルをダウンロードし、Red Hat Process Automation Manager ディストリビューションの **~/kie-server.war/WEB-INF/lib** と **~/business-central.war/WEB-INF/lib** のディレクトリーに追加します。

   - Red Hat カスタマーポータル から取得した **Red Hat Process Automation Manager 7.11.0 Maven Repository** ディストリビューションの JAR ファイル **kie-dmn-jpmml** (**rhpam-7.11.0-maven-repository/maven-repository/org/kie/kie-dmn-jpmml/7.52.0.Final-redhat-00007/kie-dmn-jpmml-7.52.0.Final-redhat-00007.jar**)
   - オンラインの Maven リポジトリーから取得した JPMML Evaluator 1.5.1 JAR ファイル
   - オンラインの Maven リポジトリーから取得した JPMML Evaluator Extensions 1.5.1 JAR ファイル

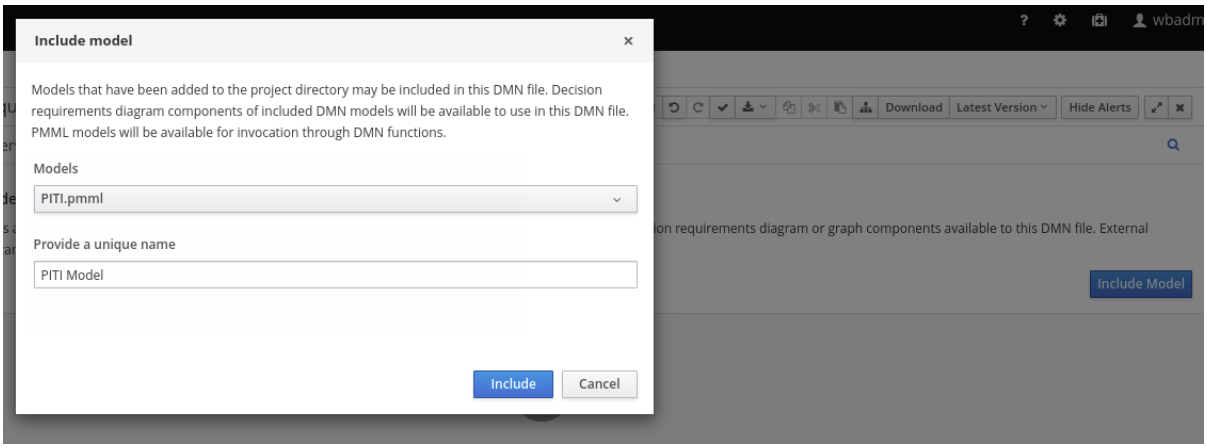
これらのアーティファクトは、KIE Server と Business Central で JPMML 評価を有効化するのに必要です。

重要

Red Hat は、Red Hat Process Automation Manager で PMML を実行するため、Java Evaluator API for PMML (JPMML) との統合をサポートしています。しかし、Red Hat は JPMML ライブラリーを直接サポートしません。Red Hat Process Automation Manager ディストリビューションに JPMML ライブラリーを含む場合は、JPMML の Openscoring.io ライセンス条件を確認してください。

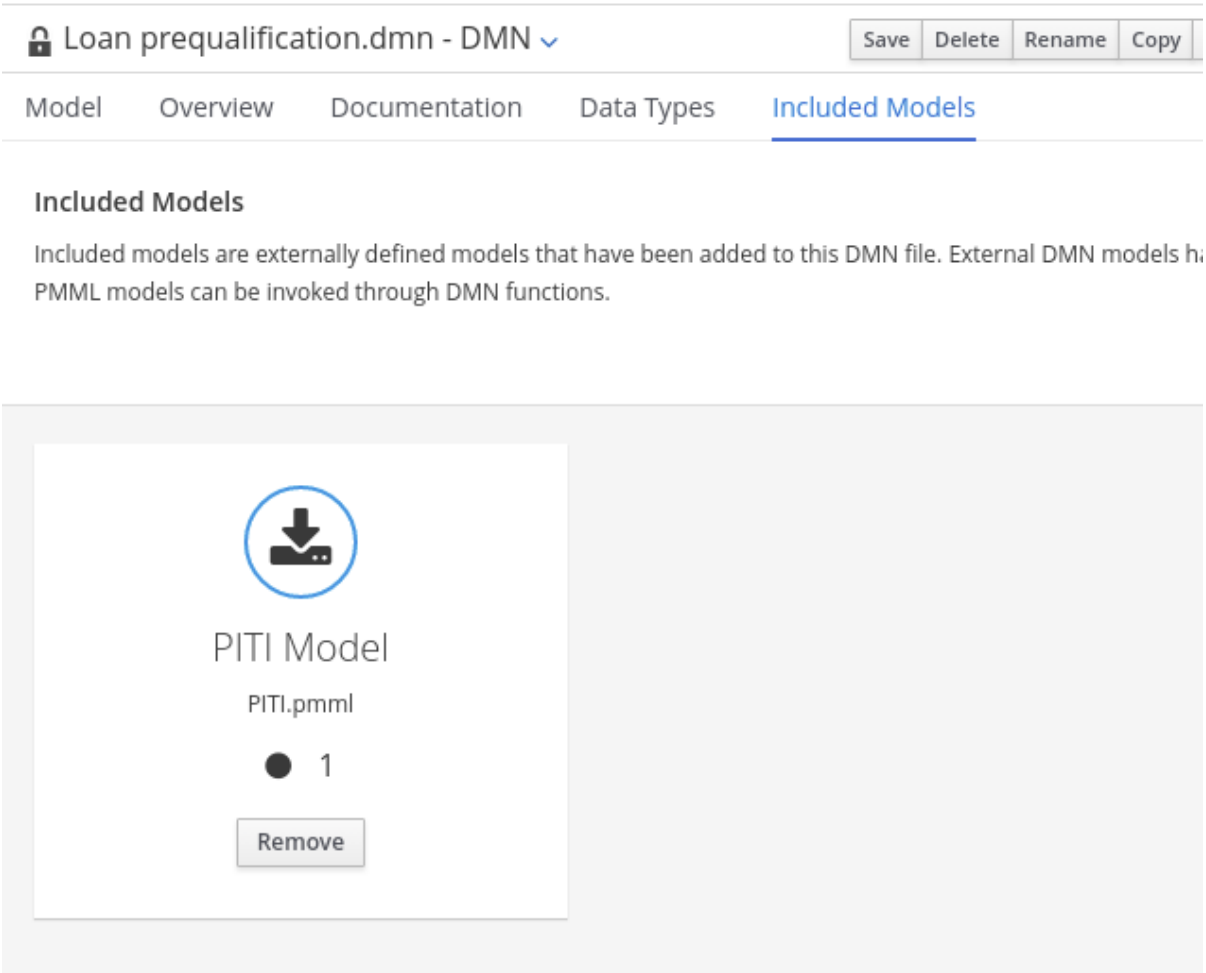
3. Business Central で **Menu → Design → Projects** に移動して、プロジェクト名をクリックし、変更する DMN ファイルを選択します。
4. DMN デザイナーで、**Included Models** タブをクリックします。
5. **Include Model** をクリックして、**Models** リストのプロジェクトから PMML モデルを選択し、追加するモデルの一意名を入力して、**Include** をクリックします。

図6.43 PMML モデルの追加



PMML モデルは、この DMN ファイルに追加されます。

図6.44 PMML モデルを含む DMN ファイル



6. DMN デザイナーの **Model** タブで、PMML モデルを呼び出すデシジョンノードまたはビジネスナレッジモデルノードを選択または作成して、**Edit** アイコンをクリックし、DMN ボックス式デザイナーを開きます。

図6.45 新規デシジョンノードのボックス式の表示

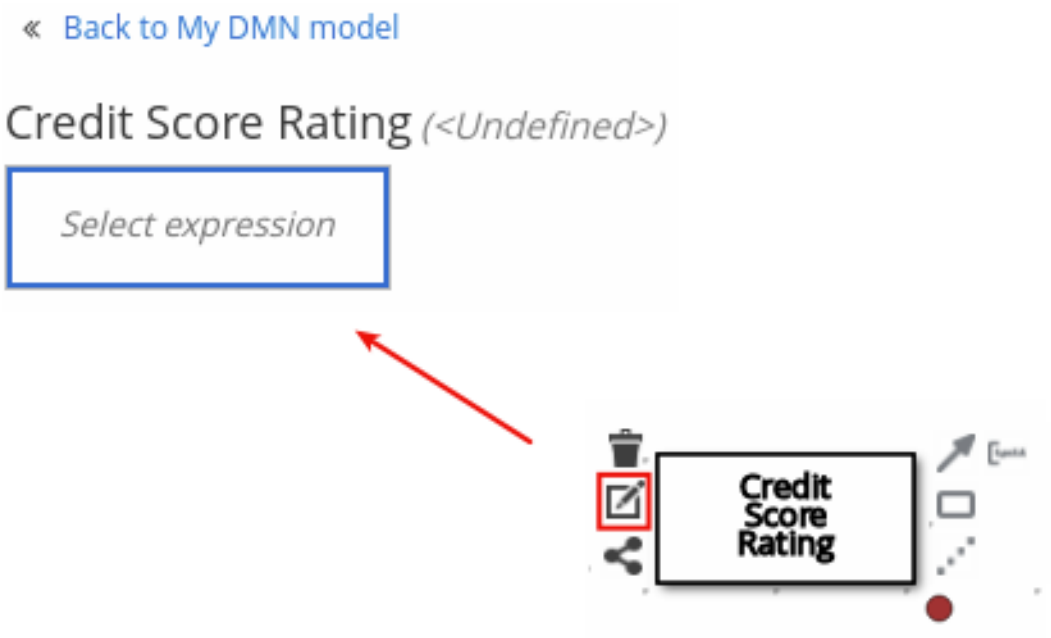
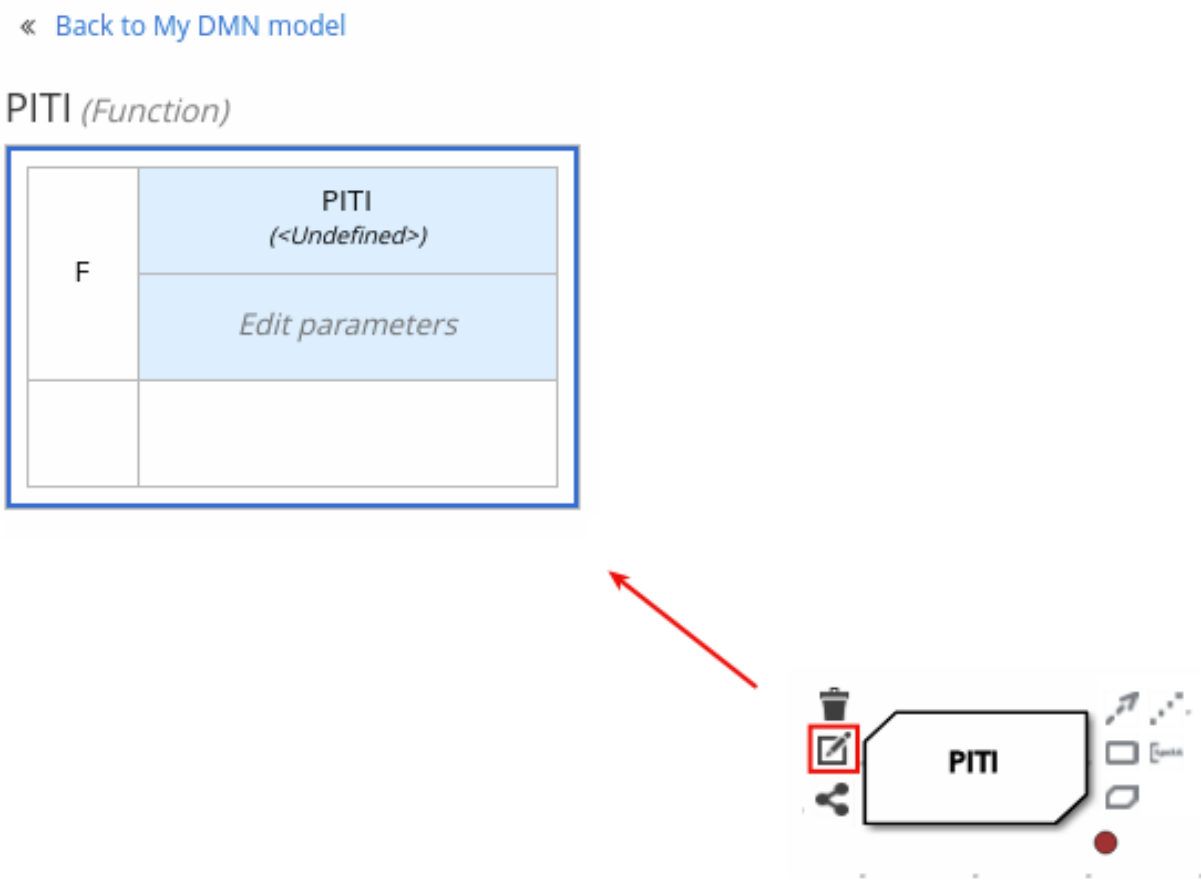


図6.46 新規ビジネスナレッジモデルのボックス式の表示



7. 式タイプを **Function** (ビジネスナレッジモデルノードのデフォルト) に設定し、左上の機能セルをクリックして **PMML** を選択します。

8. テーブルの **document** と **model** の行で、未定義のセルをダブルクリックして、PMML ドキュメント、およびそのドキュメント内の関連する PMML モデルを指定します。

図6.47 DMN ビジネスナレッジモデルへの PMML モデルの追加

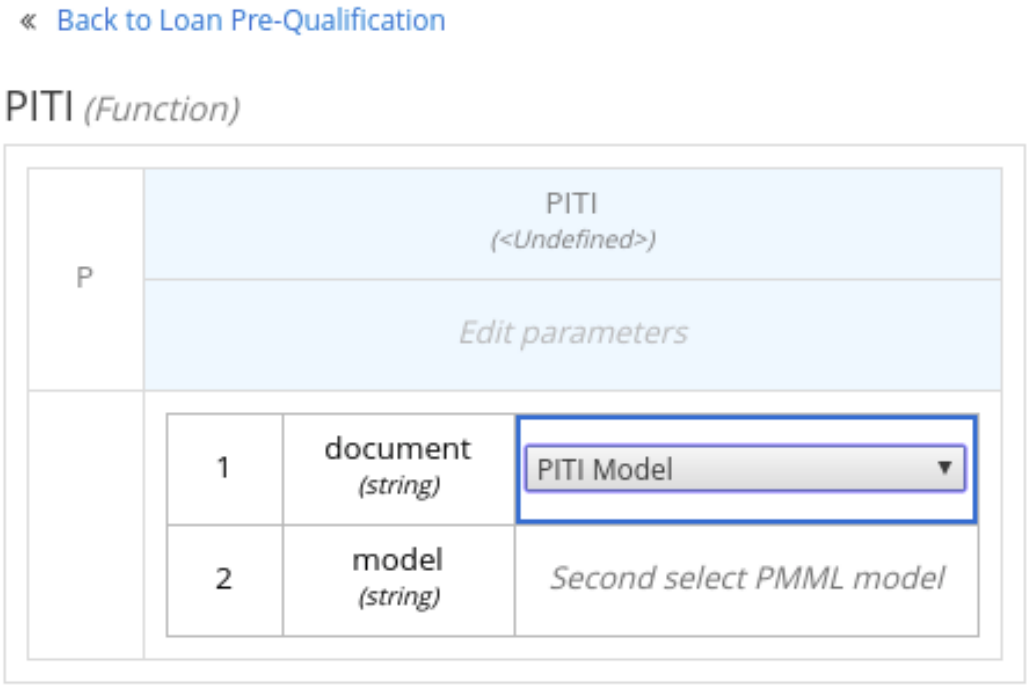
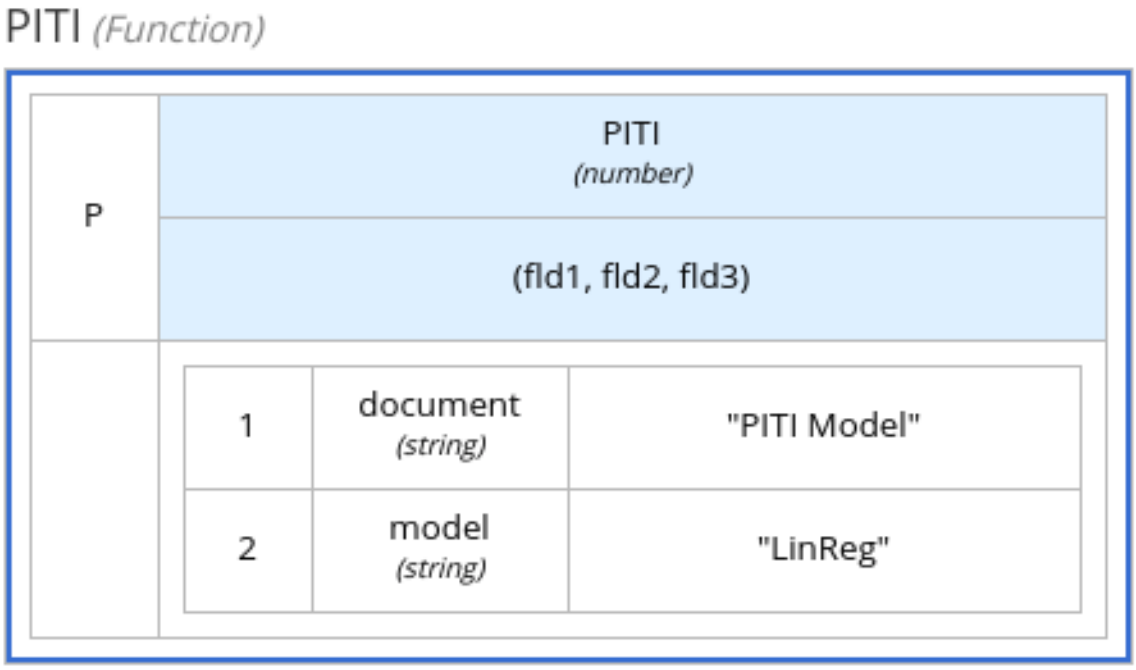


図6.48 DMN ビジネスナレッジモデルでの PMML 定義の例



追加した PMML モデルのソースファイルを更新した場合は、DMN ファイルの PMML モデルを削除して追加し直し、ソースの変更を適用する必要があります。

追加したモデル名を編集するか、追加したモデルを DMN ファイルから削除するには、DMN デザイナーの **Included Models** タブを使用します。

6.4. BUSINESS CENTRAL の複数の図を含む DMN モデルの作成

複雑な DMN モデルには、Business Central で DMN デザイナーを使用して、DMN デシジョンモデルに関する全体的な意識決定要件グラフ (DRG) の一部を表す複数の DMN の意思決定要件ダイアグラムを設計できます。単純な場合は、単一の DRD を使用して、デシジョンモデルに対して全体的な DRG をすべて表すことができますが、複雑な場合は、1 つの DRD のサイズが大きくなり、追跡が困難になる可能性があります。そのため、多くの意思決定要件で DMN の意思決定モデルをより適切に整理するために、このモデルを、DRG 全体の集中型 DRD を大きく設定する小規模のネストされた DRD に分割することができます。

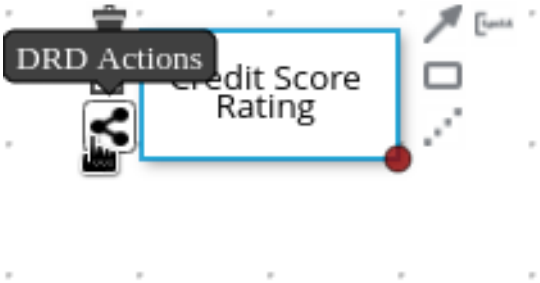
前提条件

- Business Central で DRD の設計方法を理解している。DRD の作成に関する詳細は、*6 章Business Central での DMN モデルの作成および編集* を参照してください。

手順

1. Business Central で DMN プロジェクトに移動し、プロジェクトに DMN ファイルを作成またはインポートします。
2. 新規またはインポートされた DMN ファイルを開いて DMN デザイナーで DRD を表示し、左側のツールバーの DMN ノードを使用して DRD の設計または変更を開始します。
3. 別のネストされた DRD で定義する DMN ノードについては、ノードを選択し、**DRD Actions** アイコンをクリックし、利用可能なオプションから選択します。

図6.49 DRD を細分化する DRD アクションアイコン



以下のオプションが利用できます。

- **Create**: このオプションを使用してネストされた DRD を作成します。この DRD は、選択したノードの DMN コンポーネントおよび図を個別に定義できます。
- **Add to**: ネストされた DRD がすでに作成されている場合は、このオプションを使用して選択したノードを既存の DRD に追加します。
- **Remove**: 選択したノードがネストされた DRD 内にある場合は、このオプションを使用して、ネストされた DRD からノードを削除します。

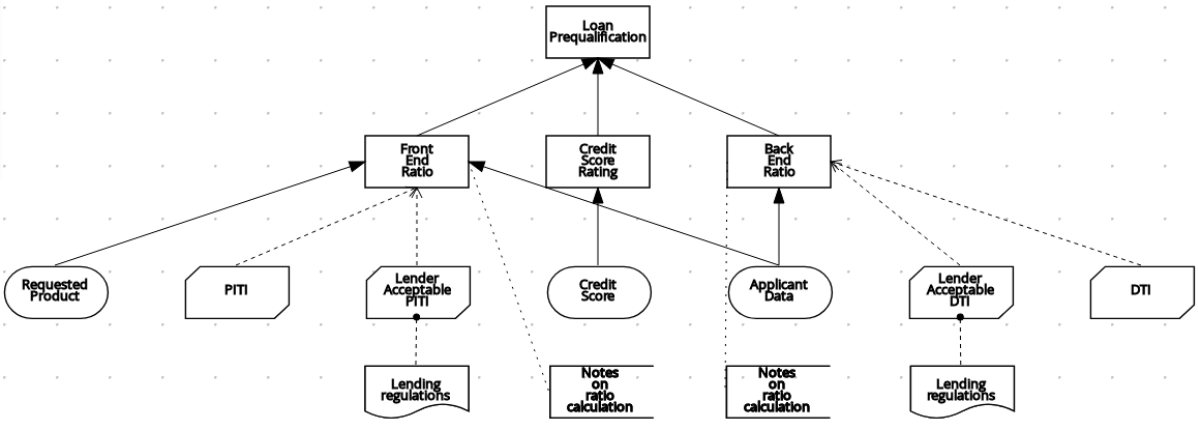
DMN のデシジョンモデル内にネストされた DRD を作成すると、新しい DRD は別の DRD キャンバスで開き、利用可能な DRD とコンポーネントが **Decision Navigator** の左側のメニューに表示されます。**Decision Navigator** メニューを使用して、ネストされた DRD を名前変更または削除できます。

図6.50 Decision Navigator メニューで新しいネストされた DRD の名前を変更

4. 新しいネストされた DRD の別のキャンバスで、DMN モデルのこの部分で必要なすべてのコンポーネントのフローと論理を設計します。

5. そのために、デシジョンモデルに他のネストされた DRD を追加および定義し、完了した DMN ファイルを保存します。
たとえば、ローン事前審査デシジョンモデルの次の DRD には、ネストされた DRD がないモデルのすべての DMN コンポーネントが含まれています。この例では、すべてのコンポーネントおよびロジックの単一の DRD に依存しているため、図は大きくて複雑になります。

図6.51 ローンの事前審査のための単一の DRD



または、この手順に従って、この例の DRD を複数のネストされた DRD に分割し、以下の例のように意思決定要件をより適切に整理できます。

図6.52 ローンの事前審査のための複数のネストされた DRD

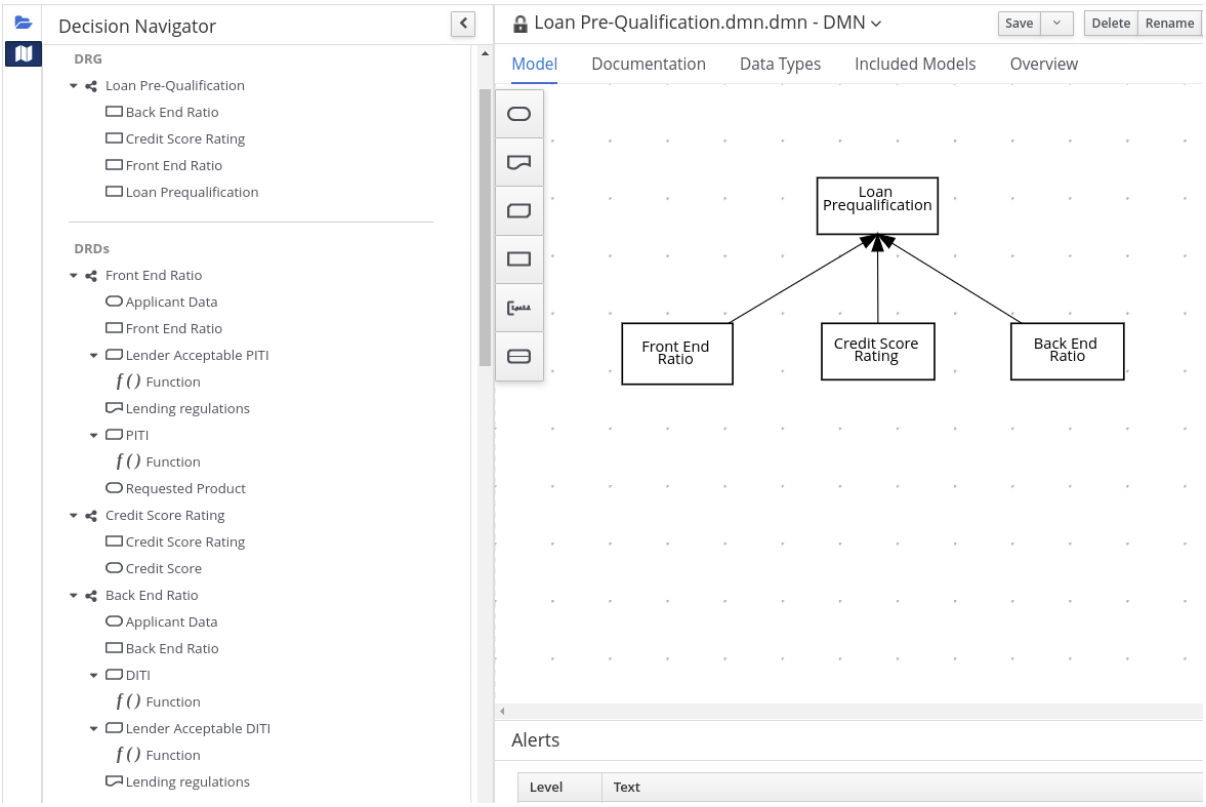


図6.53 フロントエンド比率 DRD の概要

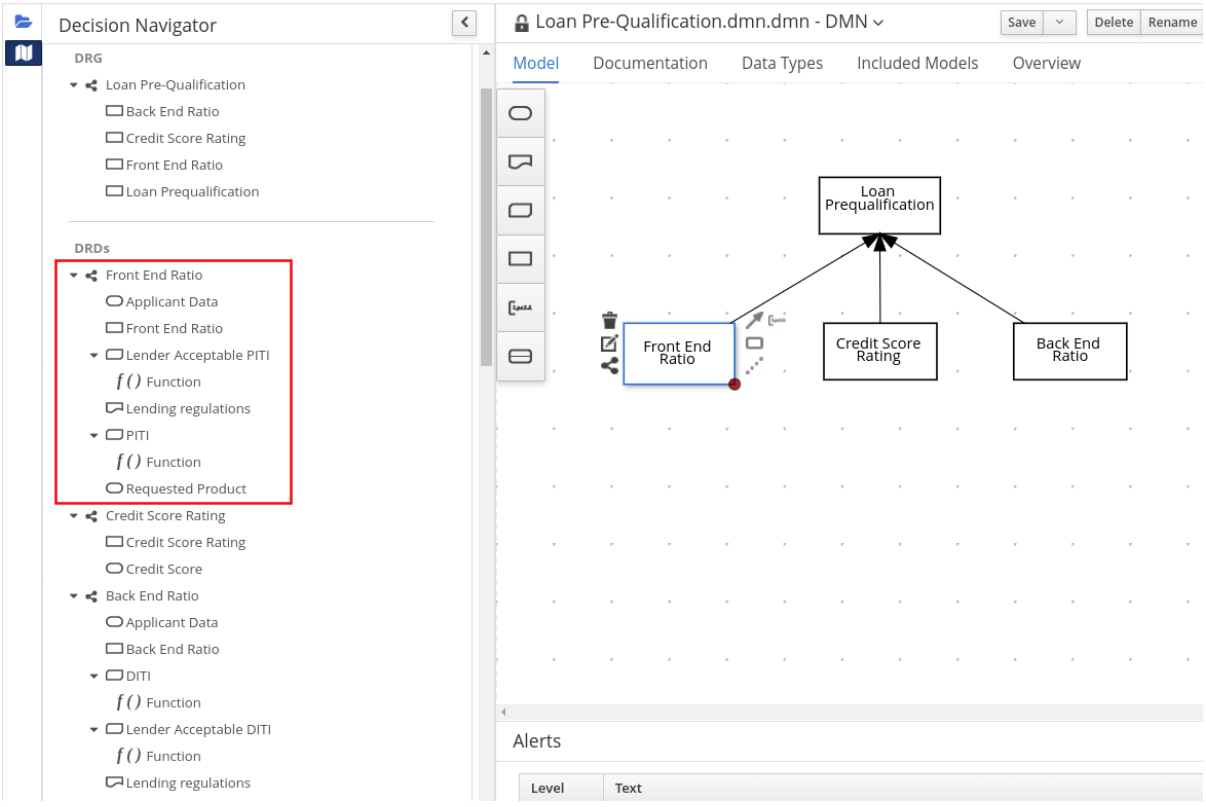


図6.54 フロントエンド比率の DRD

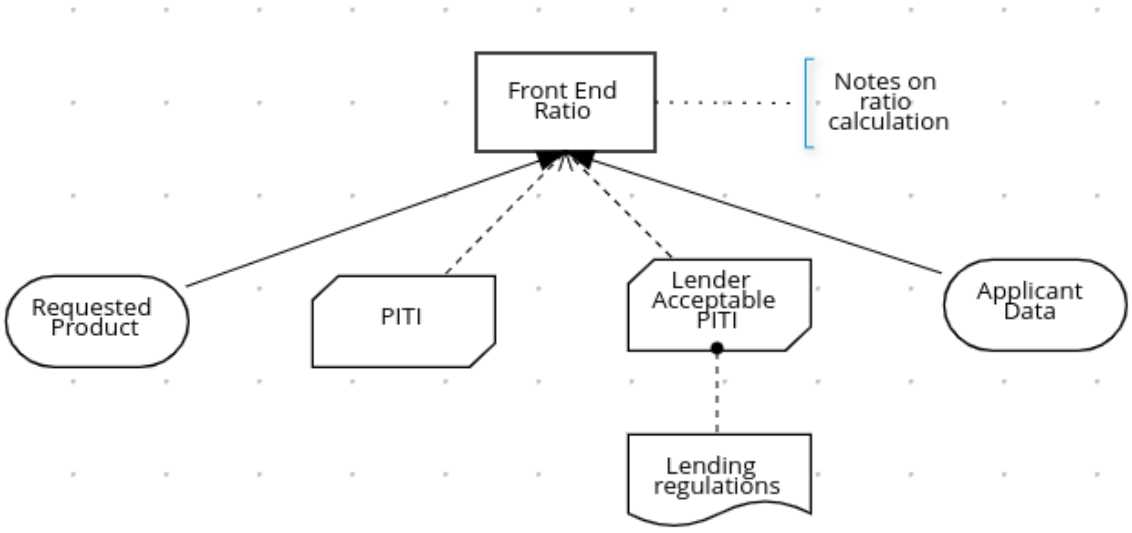


図6.55 クレジットカード評価 DRD の概要

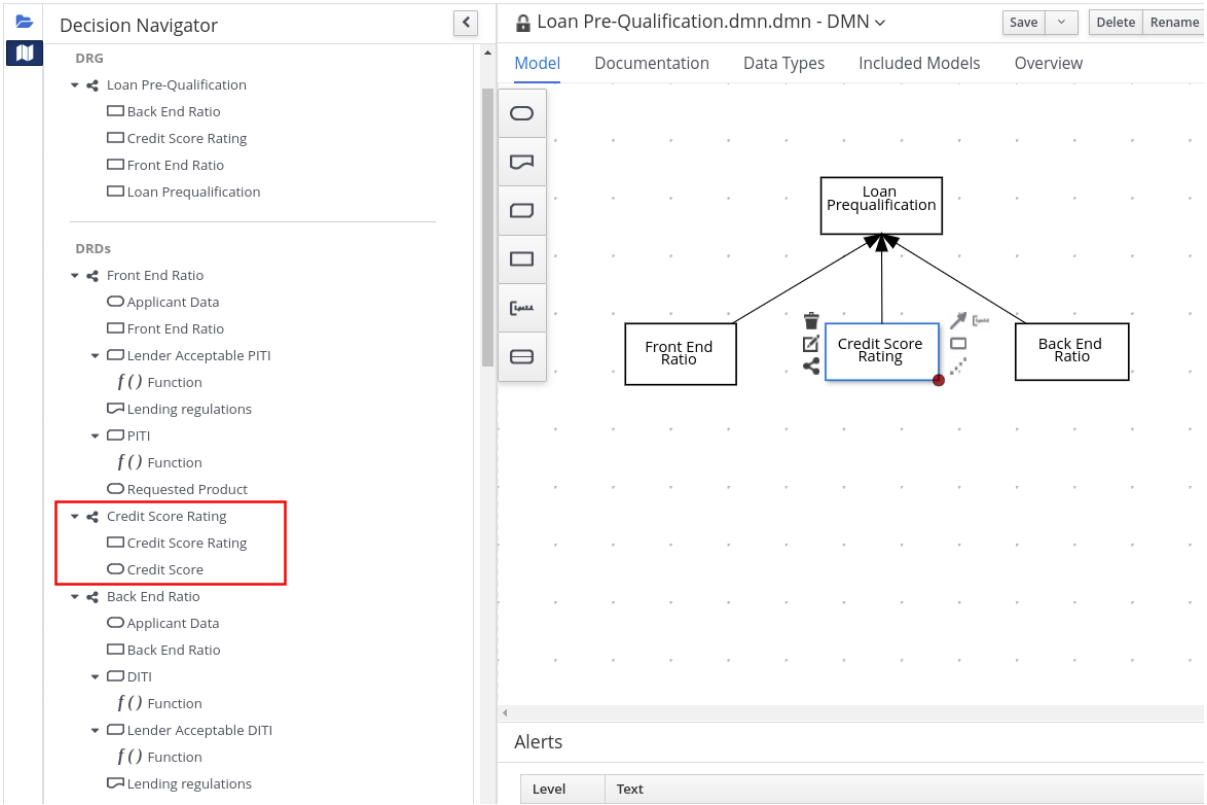


図6.56 クレジットカード評価の DRD

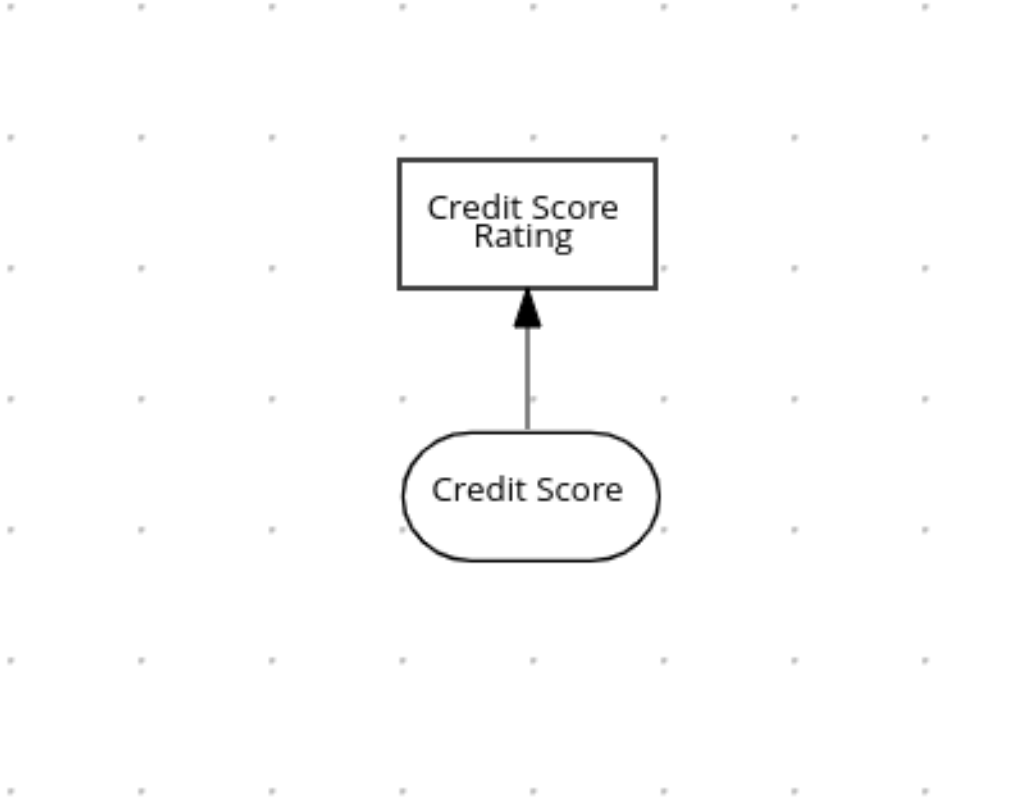


図6.57 バックエンド比率 DRD の概要

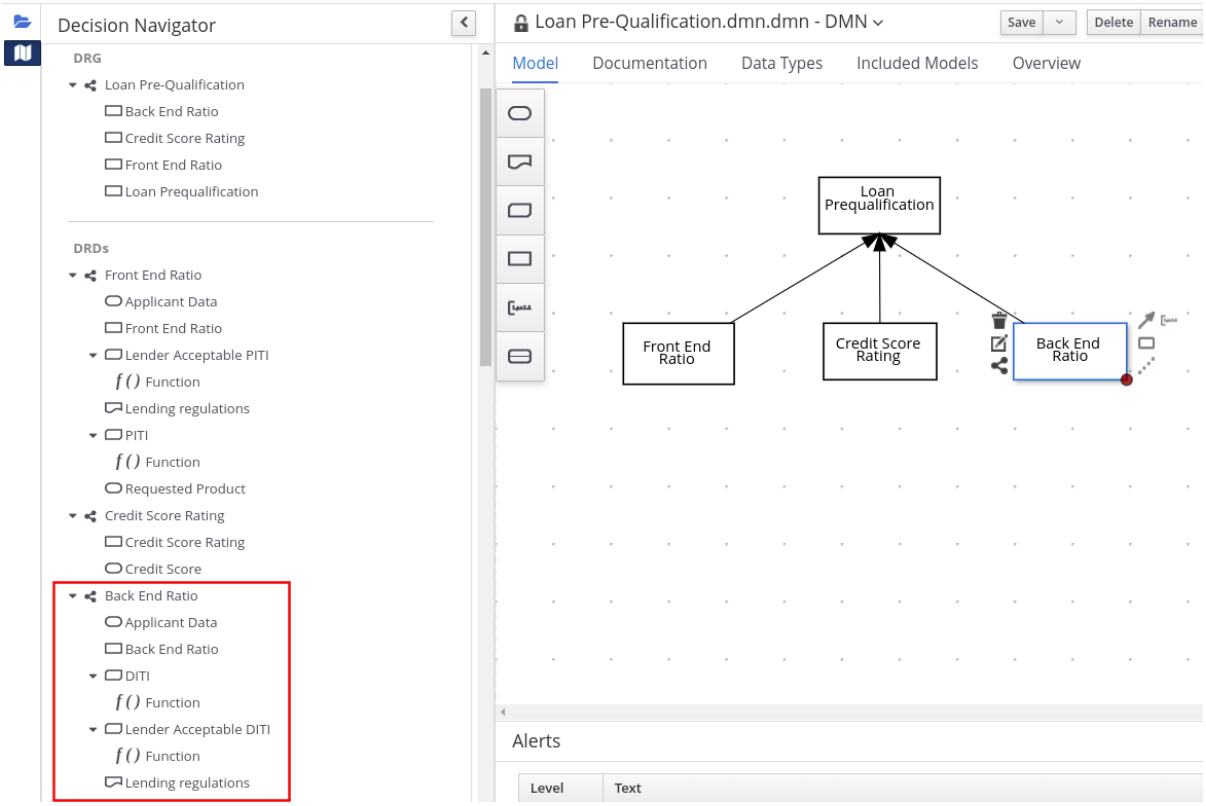
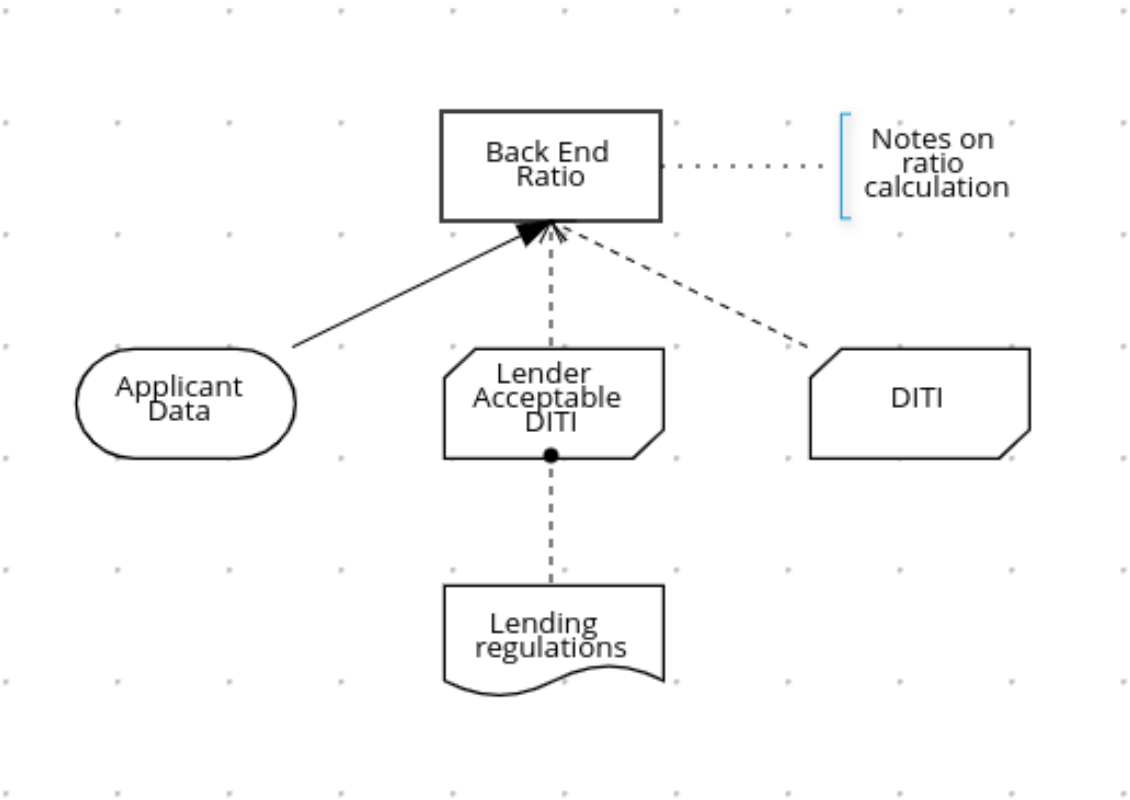


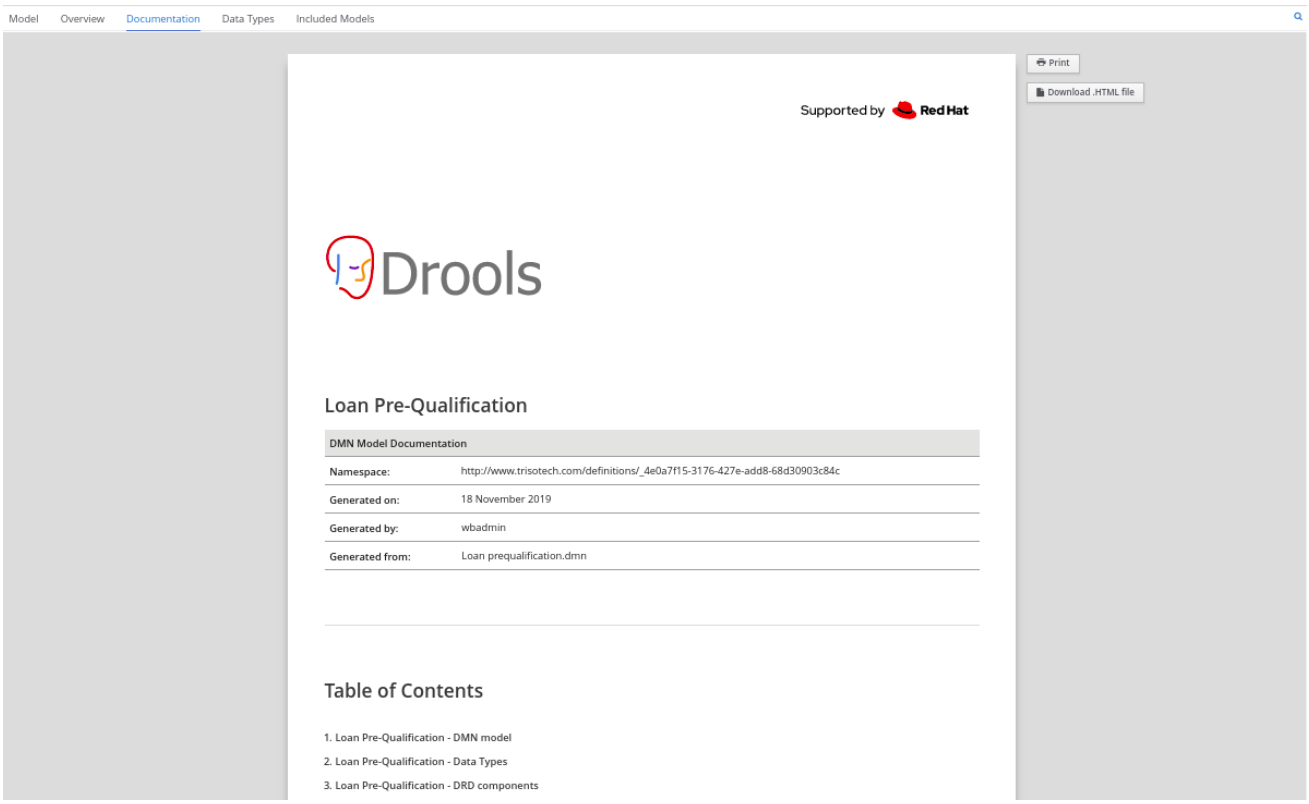
図6.58 バックエンド比率の DRD



6.5. BUSINESS CENTRAL の DMN モデルドキュメント

Business Central の DMN デザイナーで、**Documentation** タブを使用して、オフラインで使用できるように HTML ファイルとして出力またはダウンロード可能な、DMN モデルのレポートを生成します。DMN モデルのレポートには、DMN モデルの全意思決定要件ダイアグラム (DRD)、データタイプ、およびボックス式が含まれます。このレポートを使用して、DMN モデルの詳細を共有したり、社内のレポートワークフローの一部として共有できます。

図6.59 DMN モデルレポートの例



6.6. BUSINESS CENTRAL での DMN ナビゲーションとプロパティー

Business Central の DMN デザイナーには、以下の追加機能が含まれており、意思決定要件ダイアグラム (DRD) のコンポーネントやプロパティーの移動を容易にします。

DMN ファイルとダイアグラムのビュー

DMN デザイナーの左上隅で、**Project Explorer** ビューを選択して、全 DMN と他のファイルを移動するか、**Decision Navigator** ビューを選択して、選択した DRD のデシジョンコンポーネント、グラフ、ボックス式の間を移動します。

図6.60 Project Explorer のビュー

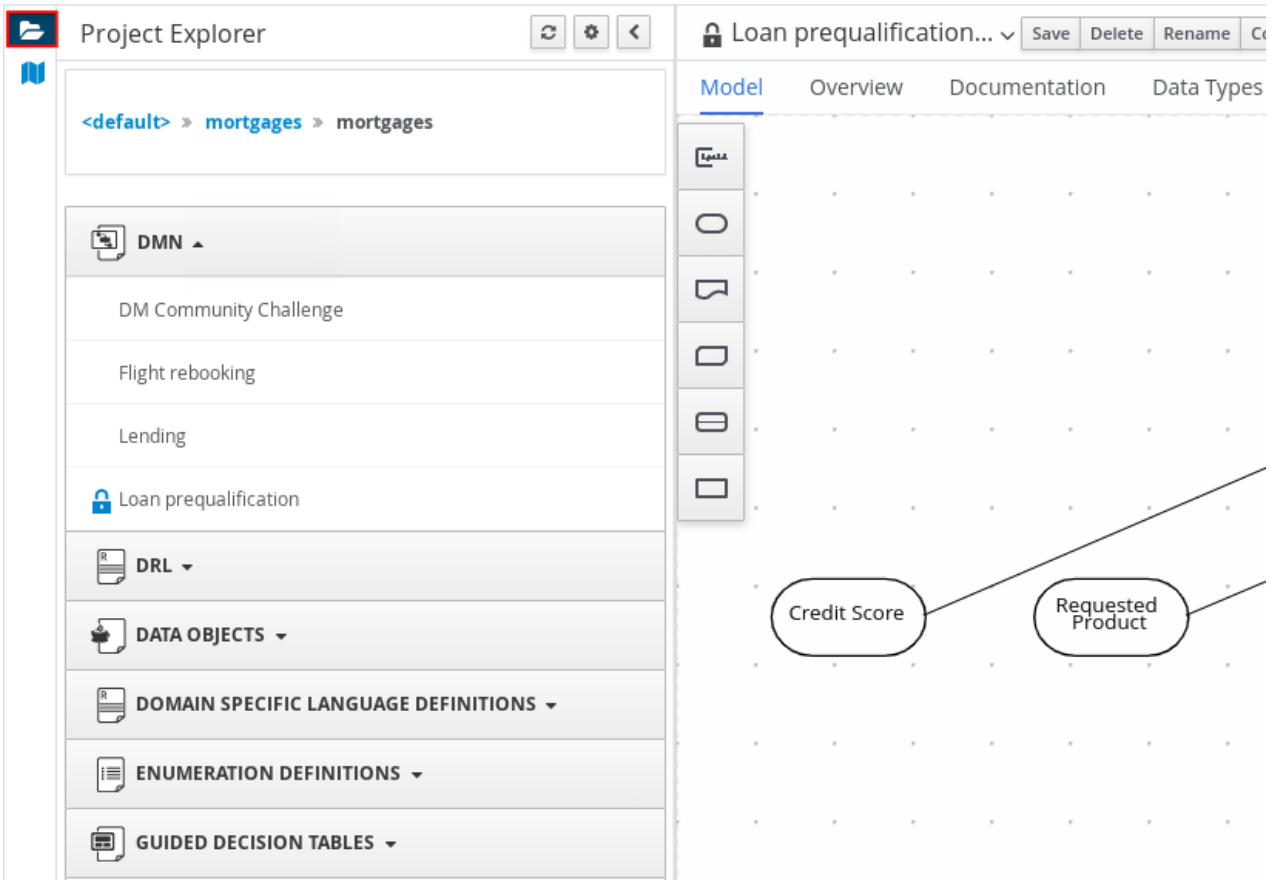
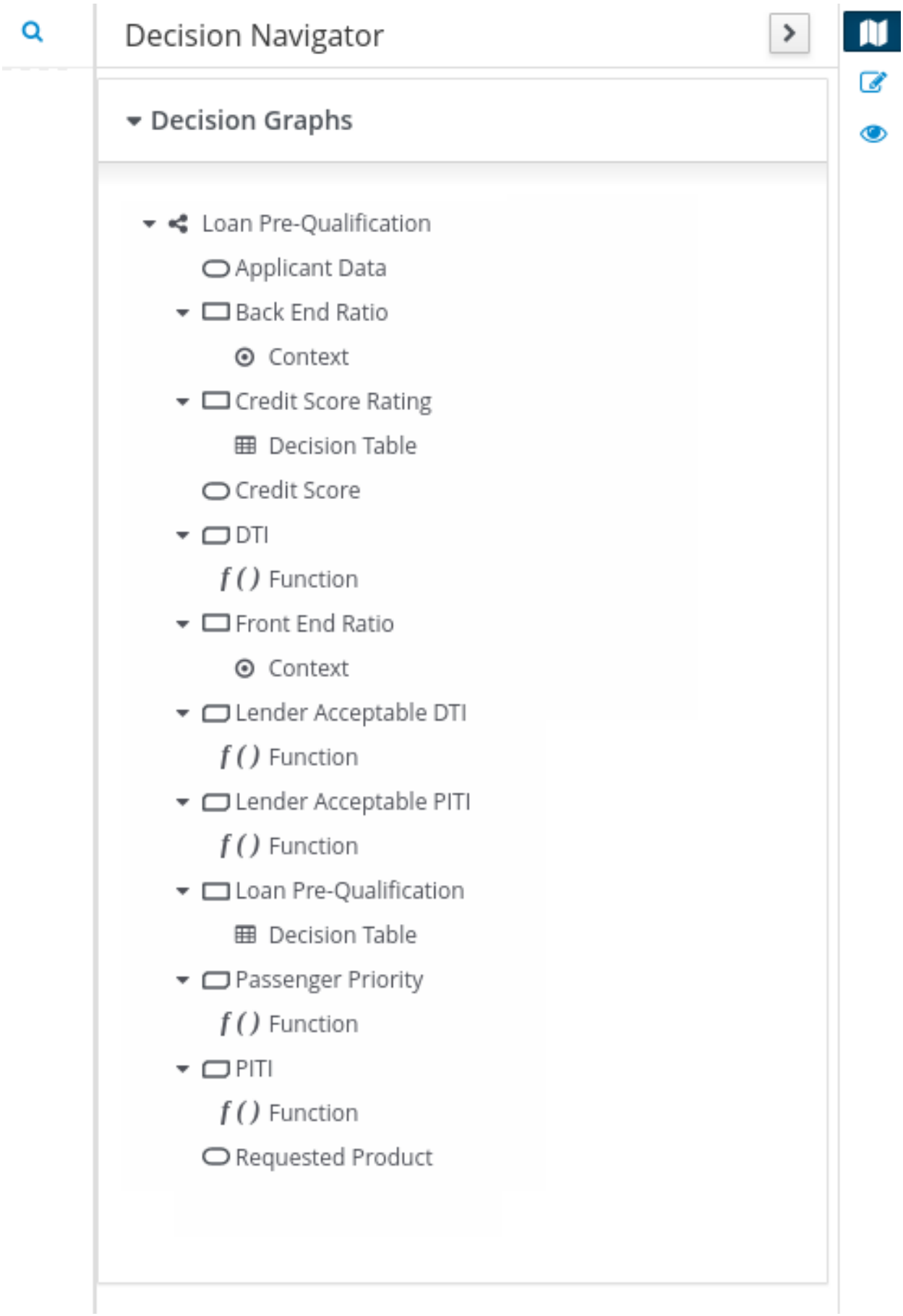
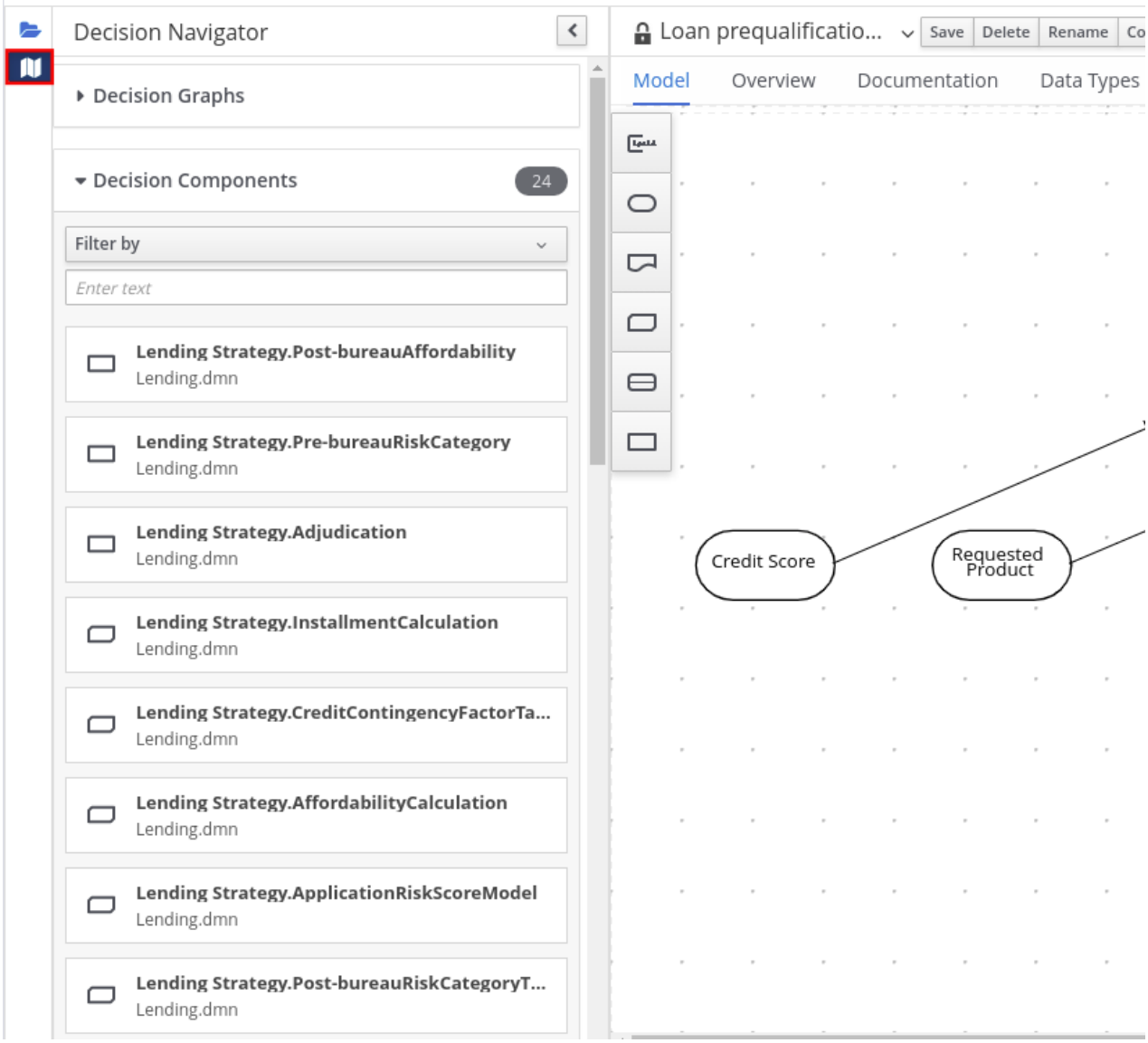
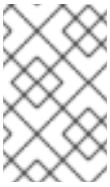


図6.61 Decision Navigator のビュー





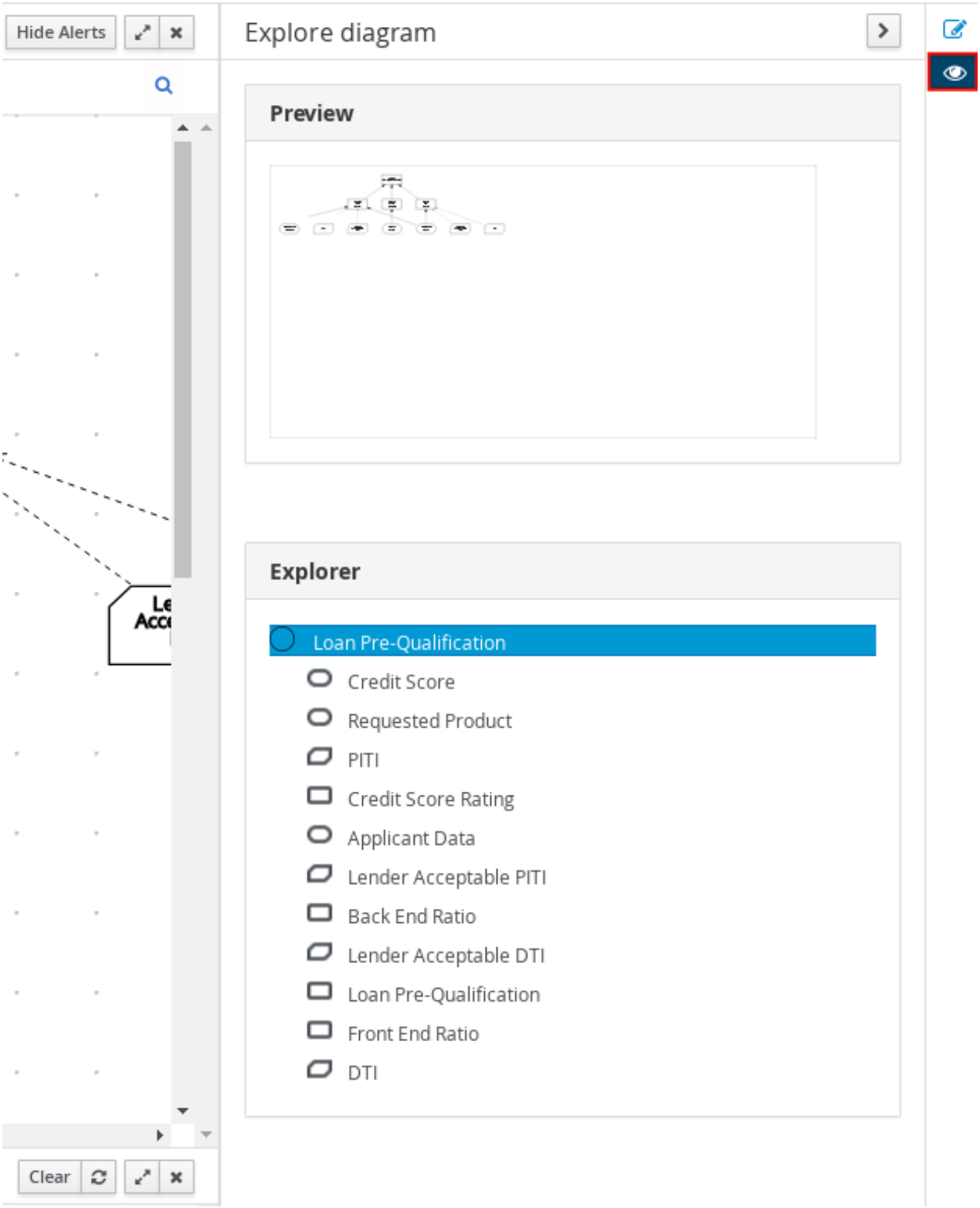


注記

DMN ファイルに含まれる DMN モデルからの DRD コンポーネント (**Included Models** タブ) は、DMN ファイルの **Decision Components** パネルにも表示されます。

DMN デザイナーの右上隅で、**Explore diagram** アイコンを選択して、選択した DRD で俯瞰プレビューを表示して、選択した DRD のノード間を移動します。

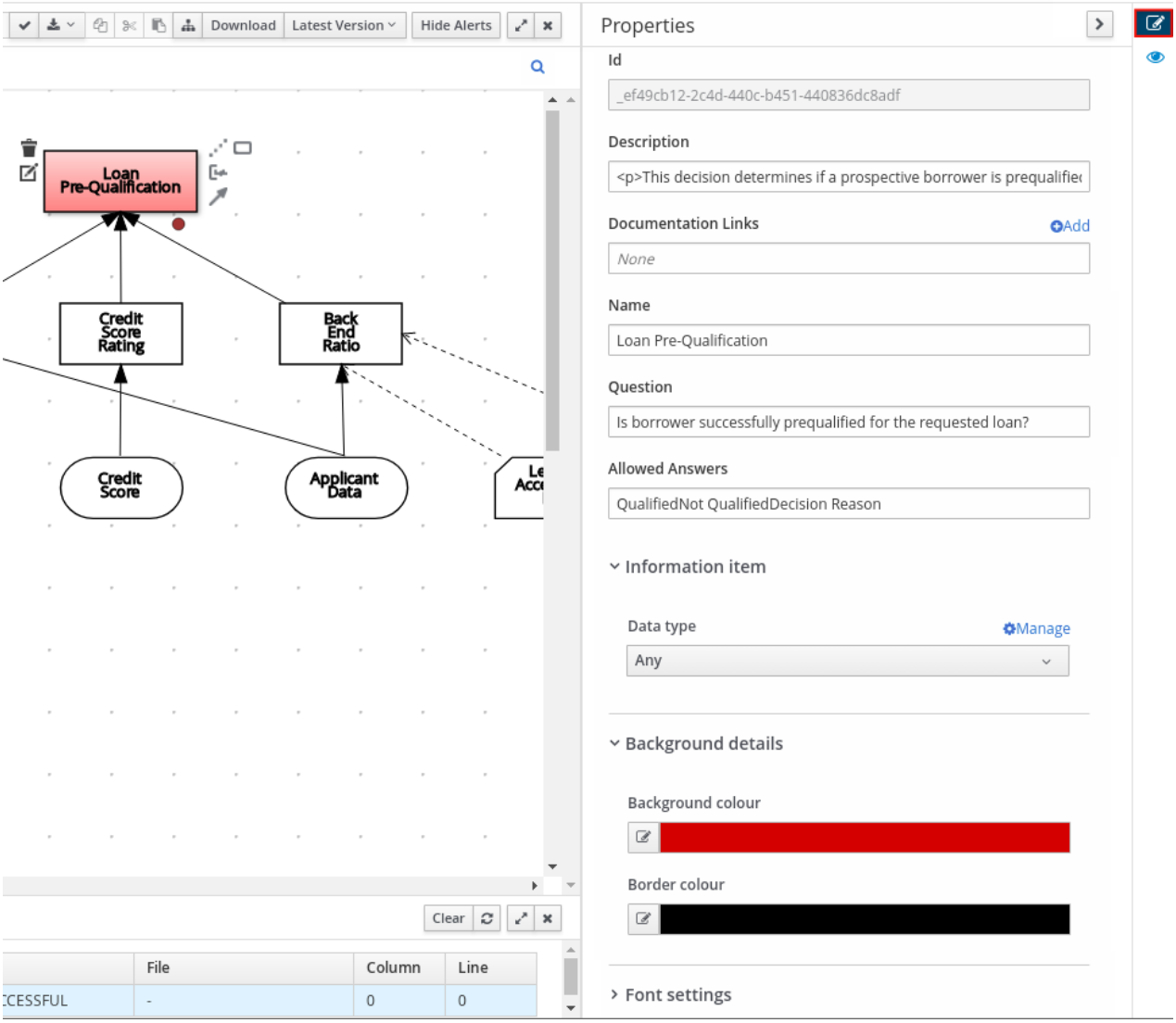
図6.62 Exploer Diagram ビュー



DRD プロパティーと設計

DMN デザイナーの右上隅で、**Properties** アイコンを選択して、情報、データ型、選択した DRD、DRD ノード、またはボックス式セルの外観を変更できます。

図6.63 DRD ノードのプロパティー

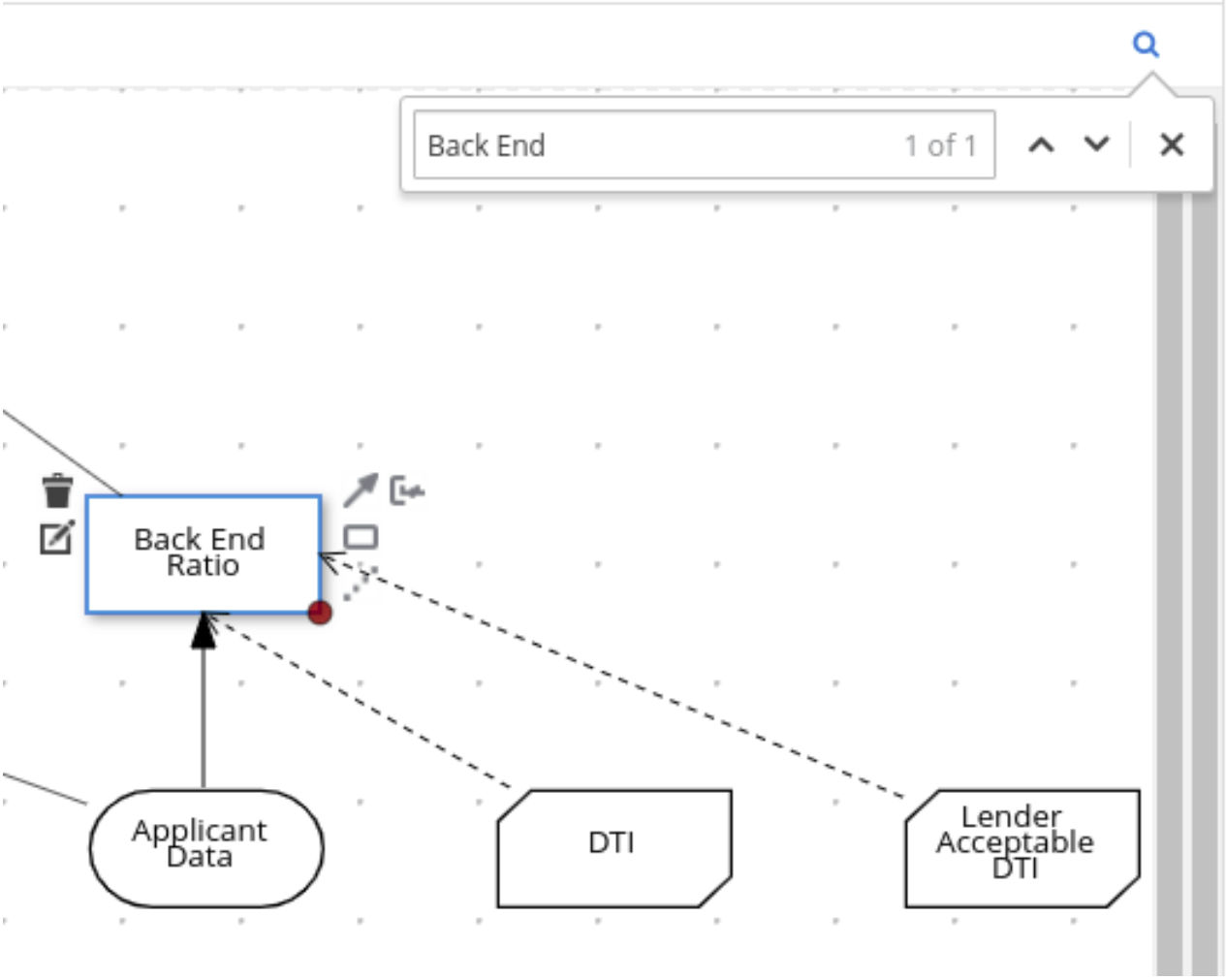


全 DRD のプロパティーを表示するには、特定のノードではなく、DRD キャンバスの背景をクリックします。

DRD 検索

DMN デザイナーの右上隅にある検索バーを使用して、DRD に表示されるテキストを検索します。検索機能は、特にノードが多数指定された複雑な DRD に有用です。

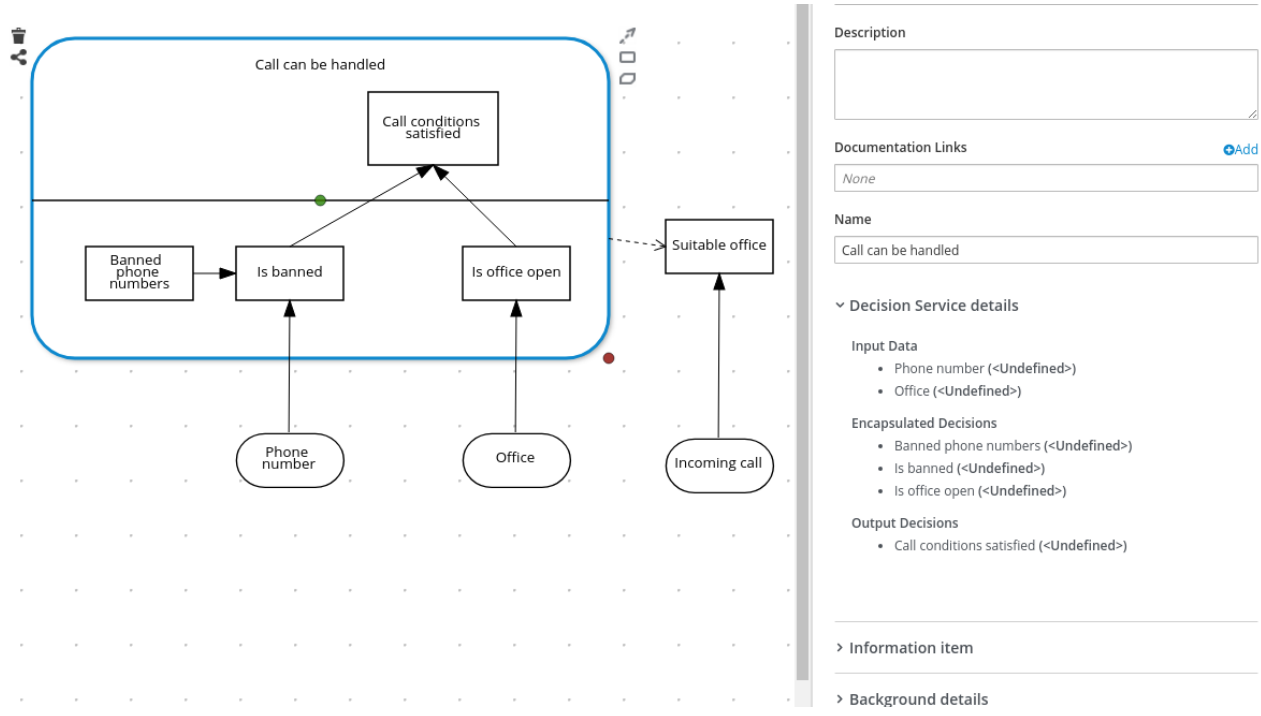
図6.64 DRD 検索



DMN デシジョンサービスの詳細

DMN デザイナーで意思決定サービスノードを選択して、**Properties** パネルの **Input Data**、**Encapsulated Decisions**、**Output Decisions** などの追加のプロパティーを表示します。

図6.65 デシジョンサービスの詳細



第7章 DMN モデルの実行

Business Central を使用して Red Hat Process Automation Manager のプロジェクトに DMN ファイルをインポートまたは作成するか、Business Central を使用しないプロジェクトのナレッジ JAR (KJAR) ファイルの一部として DMN ファイルをパッケージ化できます。Red Hat Process Automation Manager プロジェクトに DMN ファイルを実装した後に、KIE Server にこのファイルを含む KIE コンテナーをデプロイしてリモートアクセスするか、呼び出すアプリケーションの依存関係として KIE コンテナーを直接操作することで、DMN デシジョンサービスを実行できます。DMN ナレッジパッケージを作成しデプロイするその他のオプションも利用できますが、そのほとんどはナレッジアセットの全タイプ (DRL ファイルやプロセス定義など) と類似しています。

プロジェクトのパッケージングおよびデプロイメントの方法に外部 DMN アセットを含める方法は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

7.1. DMN コールの JAVA アプリケーションへの直接組み込み

KIE コンテナーは、呼び出しプログラムにナレッジアセットを直接組み込む場合や、KJAR 用 Maven 依存関係を使用して物理的にプルする場合は、ローカルとみなされます。コードのバージョンと、DML 定義のバージョンとの間に密接な関係がある場合は、通常、ナレッジアセットをプロジェクトに直接組み込みます。意思決定への変更は、アプリケーションを更新して再デプロイしないと有効になりません。このアプローチに対する利点は、適切なオペレーションがランタイムへの外部の依存関係に依存していないことですが、ロックされた環境の場合は制限になる可能性があります。

Maven の依存関係を使用すると、システムプロパティーを使用して、更新を定期的にスキャンして自動的に更新するなど、特定バージョンの意思決定が動的に変更するため、柔軟性が高まります。これにより、外部の依存関係がサービスのデプロイ時間に影響を及ぼしますが、意思決定はローカルで実行されるため、ランタイム時に利用可能な外部サービスに対する信頼が低くなります。

前提条件

- KJAR アーティファクトとして DMN プロジェクトをビルドして Maven リポジトリーにデプロイするか、プロジェクトクラスパスの一部として DMN アセットを追加している。理想的には、より効率的な実行ができるように、実行可能なモデルとして DMN プロジェクトをビルドしておいてください。

  ```
  mvn clean install -DgenerateDMNModel=yes
  ```

  プロジェクトのパッケージ化およびデプロイメント、および実行可能モデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

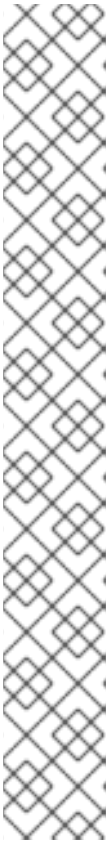
手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

```
<!-- Required for the DMN runtime API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

```
<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.52.0.Final-redhat-00007)。



注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. **namespace** モデルおよび **modelName** モデルを使用して、KIE コンテナーの **DMNRuntime** と、評価する DMN モデルへの参照を取得します。

```
DMNRuntime dmnRuntime =
KieRuntimeFactory.of(kieContainer.getKieBase()).get(DMNRuntime.class);

String namespace = "http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a";
```

```
String modelName = "dmn-movieticket-ageclassification";

DMNModel dmnModel = dmnRuntime.getModel(namespace, modelName);
```

4. 希望するモデルに対してデシジョンサービスを実行します。

```
DMNContext dmnContext = dmnRuntime.newContext(); (1)

for (Integer age : Arrays.asList(1,12,13,64,65,66)) {
    dmnContext.set("Age", age); (2)
    DMNResult dmnResult =
        dmnRuntime.evaluateAll(dmnModel, dmnContext); (3)

    for (DMNDecisionResult dr : dmnResult.getDecisionResults()) { (4)
        log.info("Age: " + age + ", " +
                "Decision: '" + dr.getDecisionName() + "', " +
                "Result: " + dr.getResult());
 }
}
```

(1) モデル評価に対する入力として使用する、新しい DMN コンテキストをインスタンス化します。この例では、Age Classification の意思決定を複数回ループさせています。

(2) 入力の DMN コンテキストに入力変数を割り当てます。

(3) DMN モデルに定義した DMN デシジョンをすべて評価します。

(4) 1 回の評価で結果が 1 つ以上になることがあり、ループを作成します。

この例では、以下の結果を出力します。

```
Age 1 Decision 'AgeClassification' : Child
Age 12 Decision 'AgeClassification' : Child
Age 13 Decision 'AgeClassification' : Adult
Age 64 Decision 'AgeClassification' : Adult
Age 65 Decision 'AgeClassification' : Senior
Age 66 Decision 'AgeClassification' : Senior
```

DMN モデルがより効率性の高い実行を行えるように、実行可能なモデルとしてこれまでにコンパイルされていない場合は、DMN モデルの実行時に、以下のプロパティーを有効化してください。

```
-Dorg.kie.dmn.compiler.execmodel=true
```

7.2. KIE SERVER JAVA クライアント API を使った DMN サービスの実行

KIE Server Java クライアント API は、KIE Server の REST または JMS インターフェイスを通してリモートの DMN サービスを呼び出す軽量なアプローチを提供します。これにより、KIE ベースと相互に作用するのに必要なランタイムの依存関係の数が減ります。これを有効にして適切なペースで個別に相互作用するようにし、意思決定の定義から呼び出しコードを切り離すと、柔軟性が上がります。

KIE Server Java クライアント API の詳細は、**KIE API を使用した Red Hat Process Automation Manager の操作** を参照してください。

前提条件

- KIE Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、Red Hat Process Automation Manager インストールの計画 を参照してください。
- KJAR アーティファクトとして DMN プロジェクトをビルドして、KIE Server にデプロイしておく。理想的には、より効率的な実行ができるように、実行可能なモデルとして DMN プロジェクトをビルドしておいてください。

```
mvn clean install -DgenerateDMNModel=yes
```

プロジェクトのパッケージ化およびデプロイメント、および実行可能モデルに関する詳細は、Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ を参照してください。

- KIE コンテナーの ID に DMN モデルを含んでいる。1 つ以上のモデルが存在する場合は、そのモデルの名前空間およびモデル名が必要です。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

```
<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.52.0.Final-redhat-00007)。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. 適切な接続情報で **KieServicesClient** をインスタンス化します。
以下に例を示します。

```
KieServicesConfiguration conf =
    KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD); (1)

conf.setMarshallingFormat(MarshallingFormat.JSON); (2)

KieServicesClient kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
```

(1) 接続情報:

- サンプル URL: **http://localhost:8080/kie-server/services/rest/server**
- この認証情報は、**kie-server** ロールを持つユーザーを参照します。

(2) マーシャリングの形式は、**org.kie.server.api.marshalling.MarshallingFormat** のインスタンスです。これは、メッセージが JSON であるか XML であるかを制御します。マーシャリング形式のオプションは、JSON、JAXB、 XSTREAM です。

3. KIE サーバーの Java クライアントインスタンスで **getServicesClient()** メソッドを呼び出すことで、関連する KIE Server に接続した KIE サーバーの Java クライアントから **DMNServicesClient** を取得します。

```
DMNServicesClient dmnClient =
kieServicesClient.getServicesClient(DMNServicesClient.class );
```

これで、**dmnClient** が、KIE Server でデシジョンサービスを実行できるようになりました。

4. 希望するモデルに対してデシジョンサービスを実行します。

以下に例を示します。

```
for (Integer age : Arrays.asList(1,12,13,64,65,66)) {
    DMNContext dmnContext = dmnClient.newContext(); (1)
    dmnContext.set("Age", age); (2)
    ServiceResponse<DMNResult> serverResp = (3)
        dmnClient.evaluateAll($kieContainerId,
                              $modelNamespace,
                              $modelName,
                              dmnContext);

    DMNResult dmnResult = serverResp.getResult(); (4)
    for (DMNDecisionResult dr : dmnResult.getDecisionResults()) {
        log.info("Age: " + age + ", " +
                 "Decision: '" + dr.getDecisionName() + "', " +
                 "Result: " + dr.getResult());
    }
}
```

(1) モデル評価に対する入力として使用する、新しい DMN コンテキストをインスタンス化します。この例では、Age Classification の意思決定を複数回ループさせています。

(2) 入力 DMN コンテキストに入力変数を割り当てます。

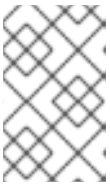
(3) DMN モデルに定義したすべての DMN の意思決定を評価します。

- **$kieContainerId** は、DMN モデルを含む KJAR がデプロイされているコンテナーの ID です。
- **$modelNamespace** は、モデルの名前空間です。
- **$modelName** は、モデルの名前です。

(4) DMN の結果オブジェクトは、サーバーの応答から利用できます。

この時点では、**dmnResult** には、評価した DMN モデルから得た意思決定の結果がすべて含まれます。

DMNServicesClient で利用可能なメソッドを使用して、モデルで特定の DMN 意思決定だけを実行することもできます。



注記

KIE コンテナーに DMN モデルが 1 つだけ含まれる場合は、KIE Server API によってデフォルトで選択されるため、**$modelNamespace** と **$modelName** を除外できます。

7.3. KIE SERVER REST API を使った DMN サービスの実行

KIE Server の REST エンドポイントで直接対話することで、呼び出しコードと、意思決定ロジックの定義の分離が最大になります。呼び出しコードに直接の依存関係がないため、**Node.js**、**.NET** など、完全に異なる開発プラットフォームに実装できます。このセクションの例では、Nix スタイルの curl コマンドを示しますが、REST クライアントに適用するための関連情報を提供します。

KIE Server の REST エンドポイントを使用する場合、標準の KIE Server マーシャリングアノテーションが付けられたドメインオブジェクト POJO Java クラスを定義することが推奨されます。たとえば、以下のコードは、適切にアノテーションが付けられたドメインオブジェクトの **Person** クラスを使用しています。

POJO Java クラスの例

```
@javax.xml.bind.annotation.XmlAccessorType(javax.xml.bind.annotation.XmlAccessType.FIELD)
public class Person implements java.io.Serializable {

    static final long serialVersionUID = 1L;

    private java.lang.String id;
    private java.lang.String name;

@javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter(org.kie.internal.jaxb.LocalDateXmlAdapter.c
lass)
    private java.time.LocalDate dojoining;

    public Person() {
    }

    public java.lang.String getId() {
        return this.id;
    }

    public void setId(java.lang.String id) {
        this.id = id;
    }

    public java.lang.String getName() {
        return this.name;
    }

    public void setName(java.lang.String name) {
        this.name = name;
    }

    public java.time.LocalDate getDojoining() {
        return this.dojoining;
    }

    public void setDojoining(java.time.LocalDate dojoining) {
        this.dojoining = dojoining;
    }

    public Person(java.lang.String id, java.lang.String name,
            java.time.LocalDate dojoining) {
        this.id = id;
        this.name = name;
        this.dojoining = dojoining;
    }

}
```

KIE Server REST API の詳細は、**KIE API を使用した Red Hat Process Automation Manager の操作**を参照してください。

前提条件

- KIE Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、**Red Hat Process Automation Manager インストールの計画** を参照してください。
- KJAR アーティファクトとして DMN プロジェクトをビルドして、KIE Server にデプロイしておく。理想的には、より効率的な実行ができるように、実行可能なモデルとして DMN プロジェクトをビルドしておいてください。

```
mvn clean install -DgenerateDMNModel=yes
```

  プロジェクトのパッケージ化およびデプロイメント、および実行可能モデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

- KIE コンテナーの ID に DMN モデルを含んでいる。1 つ以上のモデルが存在する場合は、そのモデルの名前空間およびモデル名が必要です。

手順

1. KIE Server REST API エンドポイントにアクセスするためのベース URL を決定します。これには、以下の値が必要です (例ではローカルデプロイメントのデフォルト値を使用しています)。
   - ホスト (**localhost**)
   - ポート (**8080**)
   - ルートコンテキスト (**kie-server**)
   - ベース REST パス (**services/rest/**)

   ローカルデプロイメントにおけるベース URL の例:

   http://localhost:8080/kie-server/services/rest/

2. ユーザー認証要件を決定します。
   ユーザーを KIE Server 設定に直接定義すると、ユーザー名およびパスワードを要求する HTTP Basic 認証が使用されます。要求を成功させるには、ユーザーに **kie-server** ルールが必要です。

   以下の例は、curl 要求に認証情報を追加する方法を示します。

```
curl -u username:password <request>
```

   Red Hat Single Sign-On を使用して KIE Server を設定している場合は、要求にベアラートークンが必要です。

```
curl -H "Authorization: bearer $TOKEN" <request>
```

3. 要求と応答の形式を指定します。REST API エンドポイントには JSON と XML の両方の書式が利用でき、要求ヘッダーを使用して設定されます。

JSON

```
curl -H "accept: application/json" -H "content-type: application/json"
```

XML

```
curl -H "accept: application/xml" -H "content-type: application/xml"
```

4. 必要に応じて、デプロイしたデシジョンモデルのリストに対するコンテナーのクエリーです。
[GET] server/containers/{containerId}/dmn

curl 要求例:

```
curl -u krisv:krisv -H "accept: application/xml" -X GET "http://localhost:8080/kie-server/services/rest/server/containers/MovieDMNContainer/dmn"
```

サンプルの XML 出力:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="OK models successfully retrieved from container 'MovieDMNContainer'">
   <dmn-model-info-list>
      <model>
         <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a</model-namespace>
         <model-name>dmn-movieticket-ageclassification</model-name>
         <model-id>_99</model-id>
         <decisions>
            <dmn-decision-info>
               <decision-id>_3</decision-id>
               <decision-name>AgeClassification</decision-name>
            </dmn-decision-info>
         </decisions>
      </model>
   </dmn-model-info-list>
</response>
```

サンプルの JSON 出力:

```
{
  "type" : "SUCCESS",
  "msg" : "OK models successfully retrieved from container 'MovieDMNContainer'",
  "result" : {
    "dmn-model-info-list" : {
      "models" : [ {
        "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a",
        "model-name" : "dmn-movieticket-ageclassification",
        "model-id" : "_99",
        "decisions" : [ {
          "decision-id" : "_3",
          "decision-name" : "AgeClassification"
        } ]
      } ]
```

```
    }
  }
}
```

5. モデルを実行します。
[POST] server/containers/{containerId}/dmn

curl 要求例:

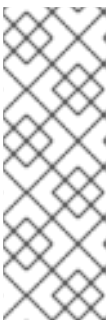
```
curl -u krisv:krisv -H "accept: application/json" -H "content-type: application/json" -X POST "http://localhost:8080/kie-server/services/rest/server/containers/MovieDMNContainer/dmn" -d "{ \"model-namespace\" : \"http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a\", \"model-name\" : \"dmn-movieticket-ageclassification\", \"decision-name\" : [ ], \"decision-id\" : [ ], \"dmn-context\" : {\"Age\" : 66}}"
```

JSON 要求例:

```
{
  "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a",
  "model-name" : "dmn-movieticket-ageclassification",
  "decision-name" : [ ],
  "decision-id" : [ ],
  "dmn-context" : {"Age" : 66}
}
```

XML 要求例 (JAXB 形式):

```
<?xml version="1.0" encoding="UTF-8"?>
<dmn-evaluation-context>
    <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a</model-namespace>
    <model-name>dmn-movieticket-ageclassification</model-name>
    <dmn-context xsi:type="jaxbListWrapper" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <type>MAP</type>
        <element xsi:type="jaxbStringObjectPair" key="Age">
            <value xsi:type="xs:int" xmlns:xs="http://www.w3.org/2001/XMLSchema">66</value>
        </element>
    </dmn-context>
</dmn-evaluation-context>
```



注記

要求には、その形式にかかわらず、以下の要素が必要です。

- モデルの名前空間
- モデル名
- 入力値を含むコンテキストオブジェクト

JSON 応答例:

```
{
```

```
  "type" : "SUCCESS",
  "msg" : "OK from container 'MovieDMNContainer'",
  "result" : {
    "dmn-evaluation-result" : {
      "messages" : [ ],
      "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a",
      "model-name" : "dmn-movieticket-ageclassification",
      "decision-name" : [ ],
      "dmn-context" : {
        "Age" : 66,
        "AgeClassification" : "Senior"
      },
      "decision-results" : {
        "_3" : {
          "messages" : [ ],
          "decision-id" : "_3",
          "decision-name" : "AgeClassification",
          "result" : "Senior",
          "status" : "SUCCEEDED"
        }
      }
    }
  }
}
```

XML (JAXB 形式) 応答例:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="OK from container 'MovieDMNContainer'">
    <dmn-evaluation-result>
        <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a</model-namespace>
        <model-name>dmn-movieticket-ageclassification</model-name>
        <dmn-context xsi:type="jaxbListWrapper"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <type>MAP</type>
            <element xsi:type="jaxbStringObjectPair" key="Age">
                <value xsi:type="xs:int"
xmlns:xs="http://www.w3.org/2001/XMLSchema">66</value>
            </element>
            <element xsi:type="jaxbStringObjectPair" key="AgeClassification">
                <value xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema">Senior</value>
            </element>
        </dmn-context>
        <messages/>
        <decisionResults>
            <entry>
                <key>_3</key>
                <value>
                    <decision-id>_3</decision-id>
                    <decision-name>AgeClassification</decision-name>
                    <result xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Senior</result>
```

```
                    <messages/>
                    <status>SUCCEEDED</status>
                </value>
            </entry>
        </decisionResults>
    </dmn-evaluation-result>
</response>
```

7.4. 特定の DMN モデルの REST エンドポイント

Red Hat Process Automation Manager には、Business Central ユーザーインターフェイスを使用せずに、特定の DMN モデルとの対話すに使用できるモデル固有の DMN KIE Server エンドポイントが含まれます。

Red Hat Process Automation Manager のコンテナーの DMN モデルごとに、KIE Server の以下の REST エンドポイントは、DMN モデルのコンテンツに基づいて自動的に生成されます。

- **POST /server/containers/{containerId}/dmn/models/{modelname}**: コンテナーで指定された DMN モデルを評価するための business-domain エンドポイント
- **POST /server/containers/{containerId}/dmn/models/{decisionServiceName}**: コンテナーで利用可能な特定の DMN モデルで指定のデシジョンサービスコンポーネントを評価するためのビジネスドメインエンドポイント
- **POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult**: 指定した DMN モデルの評価および business-domain コンテキスト、ヘルパーメッセージ、ヘルパーデシジョンポインターなど、**DMNResult** 応答を返すエンドポイント
- **POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult:** 特定の DMN モデルで指定のデシジョンサービスコンポーネントを評価して、ビジネスドメインのコンテキスト、ヘルパーメッセージ、およびヘルプデシジョンポインターなど、**DMNResult** 応答を返すエンドポイント
- **GET /server/containers/{containerId}/dmn/models/{modelname}**: デシジョンロジックなしの標準の DMN XML を返し、指定の DMN モデルの入力とデシジョンを含むエンドポイント
- **GET /server/containers/{containerId}/dmn/openapi.json(|.yaml)**: 指定のコンテナーで DMN モデルの Swagger または OAS を取得するエンドポイント

これらのエンドポイントを使用して、モデル内の DMN モデルまたは特定のデシジョンサービスと対話できます。これらの REST エンドポイントの business-domain と **dmnresult** を使用する場合は、以下の考慮事項を確認してください。

- **REST ビジネスドメインエンドポイント**: クライアントアプリケーションが正の評価結果だけを対象としていて、**Info** または **Warn** メッセージの解析を行わず、エラーに HTTP 5xx 応答だけが必要な場合にはこのエンドポイントタイプを使用します。また、このタイプのエンドポイントは、デシジョンサービスの結果のシングルトン強制が DMN のモデル動作に似ているので、単一ページアプリケーションのようなクライアントにも役立ちます。
- **REST dmnresult エンドポイント**: クライアントが **Info**、**Warn**、または **Error** メッセージの解析が必要な場合は、このエンドポイントタイプを使用します。

エンドポイントごとに、REST クライアントまたは curl ユーティリティーを使用して、以下のコンポーネントで要求を送信します。

- ベース URL: **http://HOST:PORT/kie-server/services/rest/**
- パスパラメーター:
  - **{containerId}**: **mykjar-project** などのコンテナーの文字列識別子
  - **{modelName}**: **Traffic Violation** などの DMN モデルの文字列識別子
  - **{decisionServiceName}**: **TrafficViolationDecisionService** などの DMN DRG のデシジョンサービスコンポーネントの文字列識別子
  - **dmnresult**: エンドポイントが、詳細にわたる **Info**、**Warn** および **Error** メッセージを含む、完全な **DMNResult** 応答を返すことができるようにする文字列識別子
- HTTP ヘッダー: **POST** 要求のみ:
  - **accept**: **application/json**
  - **content-type**: **application/json**
- HTTP メソッド: **GET** または **POST**

以下のエンドポイントの例は、**TrafficViolationDecisionService** デシジョンサービスコンポーネントが含まれる **Traffic Violation** DMN モデルを格納する **mykjar-project** コンテナーをもとにしています。

これらの全エンドポイントに対して、DMN 評価の **Error** メッセージが発生すると、**DMNResult** 応答が HTTP 5xx エラーと共に返されます。DMN の **Info** または **Warn** メッセージが表示されると、クライアント側のビジネスロジックに使用される **X-Kogito-decision-messages** 拡張 HTTP ヘッダーで、business- domain REST ボディーと共に関連する応答が返されます。クライアント側に詳細にわたるビジネスロジックの要件がある場合には、クライアントはエンドポイントの **dmnresult** バリアントを使用できます。

指定のコンテナー内の DMN モデルの Swagger または OAS 取得

GET /server/containers/{containerId}/dmn/openapi.json (|.yaml)

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/openapi.json (|.yaml)

デシジョンロジックのない DMN XML を返します。

GET /server/containers/{containerId}/dmn/models/{modelname}

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation

curl 要求例

```
curl -u wbadmin:wbadmin -X GET "http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic%20Violation" -H "accept: application/xml"
```

応答例 (XML)

```
<?xml version='1.0' encoding='UTF-8'?>
<dmn:definitions xmlns:dmn="http://www.omg.org/spec/DMN/20180521/MODEL/"
xmlns="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-
A2598F19ECFF" xmlns:di="http://www.omg.org/spec/DMN/20180521/DI/"
xmlns:kie="http://www.drools.org/kie/dmn/1.2"
xmlns:feel="http://www.omg.org/spec/DMN/20180521/FEEL/"
xmlns:dmndi="http://www.omg.org/spec/DMN/20180521/DMNDI/"
xmlns:dc="http://www.omg.org/spec/DMN/20180521/DC/" id="_1C792953-80DB-4B32-99EB-
25FBE32BAF9E" name="Traffic Violation"
expressionLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/"
typeLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/"
namespace="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-
A2598F19ECFF">
  <dmn:extensionElements/>
  <dmn:itemDefinition id="_63824D3F-9173-446D-A940-6A7F0FA056BB" name="tDriver"
isCollection="false">
    <dmn:itemComponent id="_9DAB5DAA-3B44-4F6D-87F2-95125FB2FEE4" name="Name"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_856BA8FA-EF7B-4DF9-A1EE-E28263CE9955" name="Age"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_FDC2CE03-D465-47C2-A311-98944E8CC23F" name="State"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_D6FD34C4-00DC-4C79-B1BF-BBCF6FC9B6D7" name="City"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_7110FE7E-1A38-4C39-B0EB-AEEF06BA37F4" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_40731093-0642-4588-9183-1660FC55053B" name="tViolation"
isCollection="false">
    <dmn:itemComponent id="_39E88D9F-AE53-47AD-B3DE-8AB38D4F50B3" name="Code"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_1648EA0A-2463-4B54-A12A-D743A3E3EE7B" name="Date"
isCollection="false">
      <dmn:typeRef>date</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_9F129EAA-4E71-4D99-B6D0-84EEC3AC43CC" name="Type"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
      <dmn:allowedValues kie:constraintType="enumeration" id="_626A8F9C-9DD1-44E0-9568-
0F6F8F8BA228">
        <dmn:text>"speed", "parking", "driving under the influence"</dmn:text>
      </dmn:allowedValues>
    </dmn:itemComponent>
    <dmn:itemComponent id="_DDD10D6E-BD38-4C79-9E2F-8155E3A4B438" name="Speed
```

```
Limit" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_229F80E4-2892-494C-B70D-683ABF2345F6" name="Actual
Speed" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_2D4F30EE-21A6-4A78-A524-A5C238D433AE" name="tFine"
isCollection="false">
    <dmn:itemComponent id="_B9F70BC7-1995-4F51-B949-1AB65538B405" name="Amount"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_F49085D6-8F08-4463-9A1A-EF6B57635DBD" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:inputData id="_1929CBD5-40E0-442D-B909-49CEDE0101DC" name="Violation">
    <dmn:variable id="_C16CF9B1-5FAB-48A0-95E0-5FCD661E0406" name="Violation"
typeRef="tViolation"/>
  </dmn:inputData>
  <dmn:decision id="_4055D956-1C47-479C-B3F4-BAEB61F1C929" name="Fine">
    <dmn:variable id="_8C1EAC83-F251-4D94-8A9E-B03ACF6849CD" name="Fine"
typeRef="tFine"/>
    <dmn:informationRequirement id="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
      <dmn:requiredInput href="#_1929CBD5-40E0-442D-B909-49CEDE0101DC"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmn:inputData id="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" name="Driver">
    <dmn:variable id="_A80F16DF-0DB4-43A2-B041-32900B1A3F3D" name="Driver"
typeRef="tDriver"/>
  </dmn:inputData>
  <dmn:decision id="_8A408366-D8E9-4626-ABF3-5F69AA01F880" name="Should the driver be
suspended?">
    <dmn:question>Should the driver be suspended due to points on his license?</dmn:question>
    <dmn:allowedAnswers>"Yes", "No"</dmn:allowedAnswers>
    <dmn:variable id="_40387B66-5D00-48C8-BB90-E83EE3332C72" name="Should the driver be
suspended?" typeRef="string"/>
    <dmn:informationRequirement id="_982211B1-5246-49CD-BE85-3211F71253CF">
      <dmn:requiredInput href="#_1F9350D7-146D-46F1-85D8-15B5B68AF22A"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement id="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
      <dmn:requiredDecision href="#_4055D956-1C47-479C-B3F4-BAEB61F1C929"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmndi:DMNDI>
    <dmndi:DMNDiagram>
      <di:extension/>
      <dmndi:DMNShape id="dmnshape-_1929CBD5-40E0-442D-B909-49CEDE0101DC"
dmnElementRef="_1929CBD5-40E0-442D-B909-49CEDE0101DC" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
```

```
        </dmndi:DMNStyle>
        <dc:Bounds x="708" y="350" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_4055D956-1C47-479C-B3F4-BAEB61F1C929"
dmnElementRef="_4055D956-1C47-479C-B3F4-BAEB61F1C929" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="709" y="210" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_1F9350D7-146D-46F1-85D8-15B5B68AF22A"
dmnElementRef="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="369" y="344" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_8A408366-D8E9-4626-ABF3-5F69AA01F880"
dmnElementRef="_8A408366-D8E9-4626-ABF3-5F69AA01F880" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="534" y="83" width="133" height="63"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNEdge id="dmnedge-_800A3BBB-90A3-4D9D-BA5E-A311DED0134F"
dmnElementRef="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
        <di:waypoint x="758" y="375"/>
        <di:waypoint x="759" y="235"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_982211B1-5246-49CD-BE85-3211F71253CF"
dmnElementRef="_982211B1-5246-49CD-BE85-3211F71253CF">
        <di:waypoint x="419" y="369"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_AEC4AA5F-50C3-4FED-A0C2-261F90290731"
dmnElementRef="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
        <di:waypoint x="759" y="235"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
    </dmndi:DMNDiagram>
  </dmndi:DMNDI>
```

指定されたコンテナーで指定の DMN モデルを評価します

POST /server/containers/{containerId}/dmn/models/{modelname}

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation

curl 要求例

```
curl -u wbadmin:wbadmin-X POST "http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation" -H  "accept: application/json" -H  "Content-Type: application/json" -d "{\"Driver\":{\"Points\":15},\"Violation\":{\"Date\":\"2021-04-08\",\"Type\":\"speed\",\"Actual Speed\":135,\"Speed Limit\":100}}"
```

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 15
  },
  "Violation": {
    "Date": "2021-04-08",
    "Type": "speed",
    "Actual Speed": 135,
    "Speed Limit": 100
  }
}
```

応答例 (JSON)

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 135,
    "Code": null,
    "Date": "2021-04-08"
  },
  "Driver": {
    "Points": 15,
    "State": null,
    "City": null,
    "Age": null,
    "Name": null
  },
  "Fine": {
    "Points": 7,
    "Amount": 1000
  },
  "Should the driver be suspended?": "Yes"
}
```

コンテナーで指定された DMN モデル内の指定のデシジョンサービスを評価します

POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}
このエンドポイントでは、要求ボディーにデシジョンサービスのすべての要件を含める必要があります。応答は、デシジョン値、元の入力値、およびシリアル化形式の他の parametric DRG コンポーネントすべてなど、デシジョンサービスの結果として返される DMN コンテキストです。たとえば、

ビジネスナレッジモデルは、署名の文字列のシリアル化形式で利用できます。

デシジョンサービスが単一の出力デシジョンで設定される場合には、応答はその特定のデシジョンから返される値になります。この動作は、モデル自体でデシジョンサービスを呼び出す時に、仕様機能の API レベルで同等の値を指定します。これにより、たとえば単一ページの Web アプリケーションから DMN デシジョンサービスと対話ができます。

図7.1 単一の出力デシジョンを含む TrafficViolationDecisionService デシジョンサービスの例

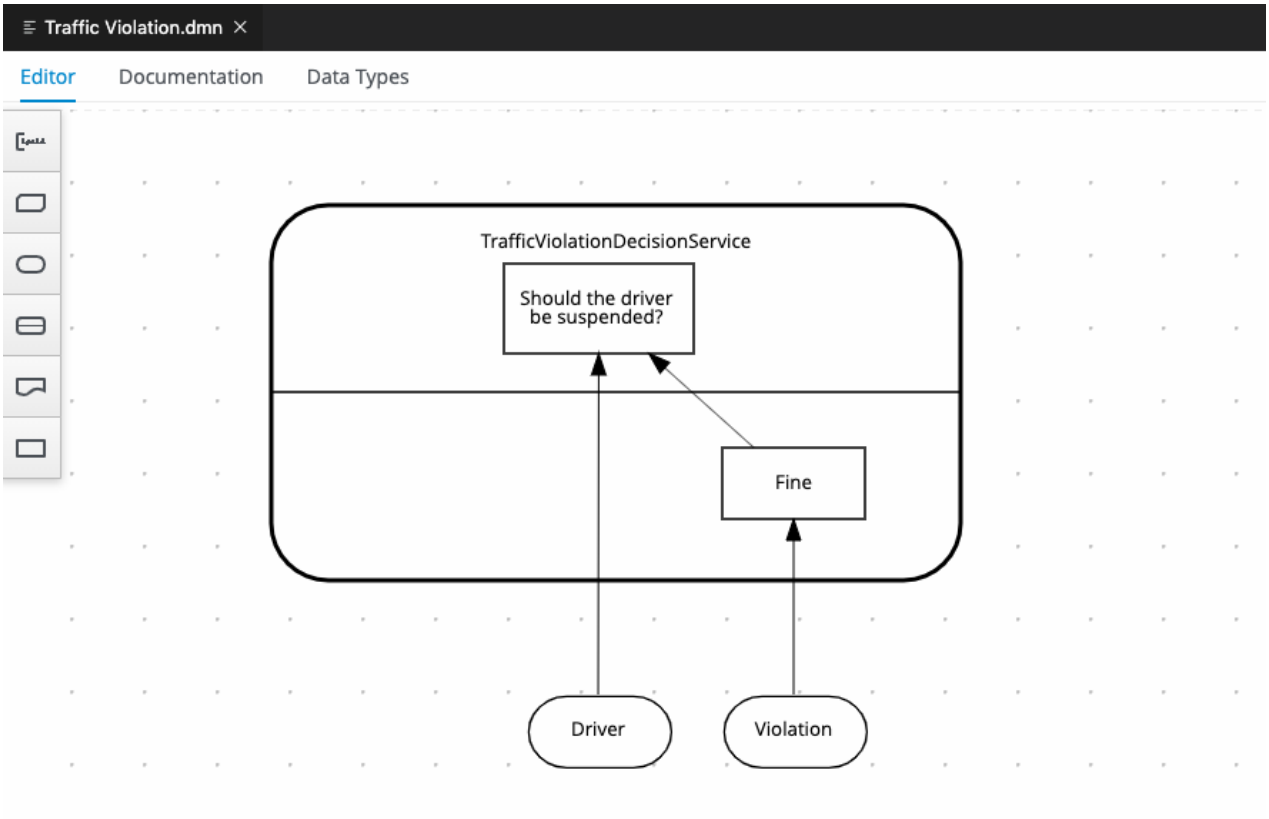
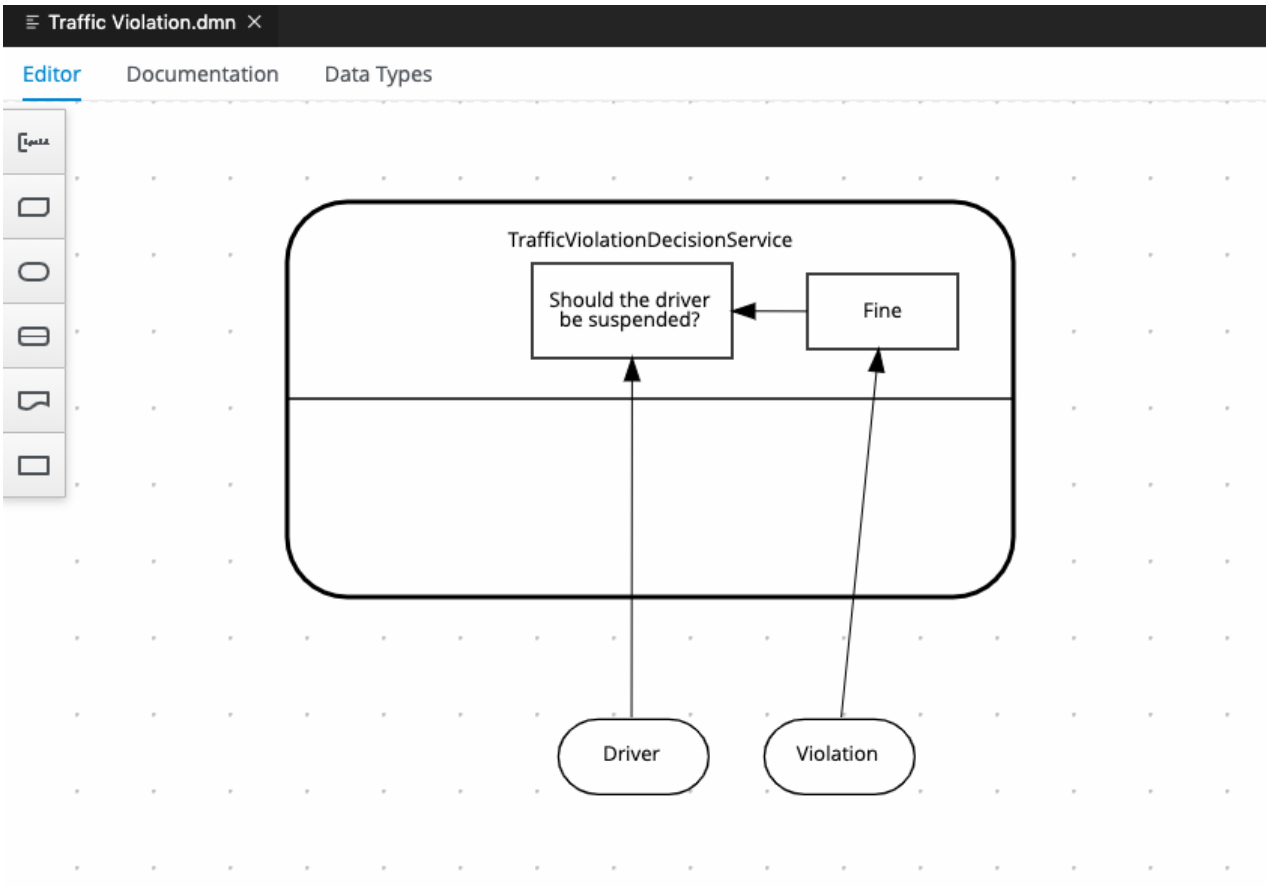


図7.2 複数の出力デシジョンを含むTrafficViolationDecisionService デシジョンサービスの例



REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

シングル出力デシジョンの応答例 (JSON)

```
"No"
```

–

複数出力デシジョンの応答例 (JSON)

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 120
  },
  "Driver": {
    "Points": 2
  },
  "Fine": {
    "Points": 3,
    "Amount": 500
  },
  "Should the driver be suspended?": "No"
}
```

指定したコンテナーで指定の DMN モデルを評価し、DMNResult 応答を返します。

POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/dmnresult

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

応答例 (JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
```

```
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Fine": {
      "Points": 3,
      "Amount": 500
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_4055D956-1C47-479C-B3F4-BAEB61F1C929",
      "decisionName": "Fine",
      "result": {
        "Points": 3,
        "Amount": 500
      },
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    },
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```


指定したコンテナーの DMN モデル内で指定のデシジョンサービスを評価し、DMNResult 応答を返します。

POST
/server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService/dmnresult

入力データでの POST 要求ボディーの例


```
{
```

```
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

応答例 (JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```

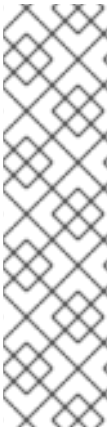
第8章 関連情報

- Decision Model and Notation specification
- DMN Technology Compatibility Kit
- **Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ**
- **KIE API を使った Red Hat Process Automation Manager の操作**

パート II. PMML モデルでのデシジョンサービスの作成

ビジネスルール開発者は、Predictive Model Markup Language (PMML) を使用して統計またはデータマイニングモデルを定義し、Red Hat Process Automation Manager のデシジョンサービスと統合できます。Red Hat Process Automation Manager には、回帰、スコアカード、ツリー、マイニングモデル向けの PMML 4.2.1 のコンシューマー適合サポートが含まれます。Red Hat Process Automation Manager には、PMML モデルエディターが同梱されていませんが、XML または PMML 固有のオーサリングツールを使用して、PMML モデルを作成してから Red Hat Process Automation Manager プロジェクトに統合できます。

PMML に関する詳細は、DMG の PMML 仕様 を参照してください。



注記

Decision Model and Notation (DMN) モデルを使用して独自のデシジョンサービスを設計し、DMN サービスの一部として PMML モデルを追加することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

第9章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表9.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第10章 PREDICTIVE MODEL MARKUP LANGUAGE (PMML)

Predictive Model Markup Language (PMML) は、統計およびデータマイニングモデルの定義用に、Data Miningu Group (DMG) が確立した XML ベースの標準です。PMML モデルは、ビジネスアナリストや開発者が一元的に PMML ベースのアセットやサービスを設計、分析、実装するために、PMML 準拠のプラットフォーム間や組織全体で共有することができます。

PMML の背景およびアプリケーションに関する詳細は、DMG の PMML 仕様 を参照してください。

10.1. PMML 適合レベル

PMML 仕様は、PMML モデルが確実に作成および統合されるように、ソフトウェア実装におけるプロデューサーおよびコンシューマーの適合レベルを定義します。各適合レベルの正式な定義については、DMG の PMML 適合 のページを参照してください。

以下の一覧では、PMML 適合レベルをまとめています。

プロデューサーの適合

ツールまたはアプリケーションは、少なくとも 1 種類のモデルに対して有効な PMML ドキュメントが生成されている場合に、プロデューサーの適合があるとされます。PMML のプロデューサー適合要件を満たすことで、モデル定義のドキュメントが構文的に正しく、このドキュメントで定義したモデルインスタンスが、モデル仕様に定義されている意味論の基準と整合性を保てるようにします。

コンシューマーの適合

ツールまたはアプリケーションは、少なくとも 1 種類のモデルに対して有効な PMML ドキュメントが生成されている場合に、コンシューマーの適合があるとされます。コンシューマー適合要件を満たすことで、プロデューサー適合にあわせて作成された PMML モデルが定義どおりに統合され、使用できるようにします。たとえば、アプリケーションが回帰モデルタイプに適合しているコンシューマーである場合は、有効な PMML ドキュメントで、別の適合プロデューサーにより作成されたこのタイプのモデルを定義していれば、アプリケーション内でどちらでも使用できます。

Red Hat Process Automation Manager には、以下の PMML 4.2.1 モデルタイプに対するコンシューマー適合サポートが含まれます。

- 回帰モデル
- スコアカードモデル
- ツリーモデル
- マイニングモデル (サブタイプとして **modelChain**、**selectAll**、および **selectFirst** が含まれる)

Red Hat Process Automation Manager でサポートされていないモデルを含め、すべての PMML モデルタイプの一覧は、DMG の PMML specification を参照してください。

第11章 PMML モデルの例

PMML は、XML スキーマ を定義しており、このスキーマを使用することで、PMML モデルが異なる PMML 準拠のプラットフォーム間で使用できるようになります。PMML 仕様では、プロデューサーとコンシューマー適合を満たしていることが前提で、複数のソフトウェアプラットフォームが同じファイルを使用してオーサリング、テスト、および実稼働環境での実行を可能にします。

以下は、PMM の回帰、スコアカード、ツリー、マイニングモデルの例です。これらの例では、Red Hat Process Automation Manager のデシジョンサービスと統合可能なモデルでサポートされているタイプを紹介します。

PMML の詳細例は、DMG の PMML Sample Files ページを参照してください。

PMML 回帰モデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="5">
    <DataField dataType="double" name="fld1" optype="continuous"/>
    <DataField dataType="double" name="fld2" optype="continuous"/>
    <DataField dataType="string" name="fld3" optype="categorical">
      <Value value="x"/>
      <Value value="y"/>
    </DataField>
    <DataField dataType="double" name="fld4" optype="continuous"/>
    <DataField dataType="double" name="fld5" optype="continuous"/>
  </DataDictionary>
  <RegressionModel algorithmName="linearRegression" functionName="regression"
modelName="LinReg" normalizationMethod="logit" targetFieldName="fld4">
    <MiningSchema>
      <MiningField name="fld1"/>
      <MiningField name="fld2"/>
      <MiningField name="fld3"/>
      <MiningField name="fld4" usageType="predicted"/>
      <MiningField name="fld5" usageType="target"/>
    </MiningSchema>
    <RegressionTable intercept="0.5">
      <NumericPredictor coefficient="5" exponent="2" name="fld1"/>
      <NumericPredictor coefficient="2" exponent="1" name="fld2"/>
      <CategoricalPredictor coefficient="-3" name="fld3" value="x"/>
      <CategoricalPredictor coefficient="3" name="fld3" value="y"/>
      <PredictorTerm coefficient="0.4">
        <FieldRef field="fld1"/>
        <FieldRef field="fld2"/>
      </PredictorTerm>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

PMML スコアカードモデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="4">
    <DataField name="param1" optype="continuous" dataType="double"/>
    <DataField name="param2" optype="continuous" dataType="double"/>
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="finalscore" optype="continuous" dataType="double" />
  </DataDictionary>
  <Scorecard modelName="ScorecardCompoundPredicate" useReasonCodes="true"
isScorable="true" functionName="regression"    baselineScore="15" initialScore="0.8"
reasonCodeAlgorithm="pointsAbove">
    <MiningSchema>
      <MiningField name="param1" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="param2" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="overallScore" usageType="target"/>
      <MiningField name="finalscore" usageType="predicted"/>
    </MiningSchema>
    <Characteristics>
      <Characteristic name="ch1" baselineScore="50" reasonCode="reasonCh1">
        <Attribute partialScore="20">
          <SimplePredicate field="param1" operator="lessThan" value="20"/>
        </Attribute>
        <Attribute partialScore="100">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="lessOrEqual" value="25"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="200">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="greaterThan" value="25"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
      <Characteristic name="ch2" reasonCode="reasonCh2">
        <Attribute partialScore="10">
          <CompoundPredicate booleanOperator="or">
            <SimplePredicate field="param2" operator="lessOrEqual" value="-5"/>
            <SimplePredicate field="param2" operator="greaterOrEqual" value="50"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="20">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param2" operator="greaterThan" value="-5"/>
            <SimplePredicate field="param2" operator="lessThan" value="50"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
    </Characteristics>
  </Scorecard>
</PMML>
```

PMML ツリーモデルの例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
 <Header copyright="JBOSS"/>
 <DataDictionary numberOfFields="5">
  <DataField dataType="double" name="fld1" optype="continuous"/>
  <DataField dataType="double" name="fld2" optype="continuous"/>
  <DataField dataType="string" name="fld3" optype="categorical">
   <Value value="true"/>
   <Value value="false"/>
  </DataField>
  <DataField dataType="string" name="fld4" optype="categorical">
   <Value value="optA"/>
   <Value value="optB"/>
   <Value value="optC"/>
  </DataField>
  <DataField dataType="string" name="fld5" optype="categorical">
   <Value value="tgtX"/>
   <Value value="tgtY"/>
   <Value value="tgtZ"/>
  </DataField>
 </DataDictionary>
 <TreeModel functionName="classification" modelName="TreeTest">
  <MiningSchema>
   <MiningField name="fld1"/>
   <MiningField name="fld2"/>
   <MiningField name="fld3"/>
   <MiningField name="fld4"/>
   <MiningField name="fld5" usageType="predicted"/>
  </MiningSchema>
  <Node score="tgtX">
   <True/>
   <Node score="tgtX">
    <SimplePredicate field="fld4" operator="equal" value="optA"/>
    <Node score="tgtX">
     <CompoundPredicate booleanOperator="surrogate">
      <SimplePredicate field="fld1" operator="lessThan" value="30.0"/>
      <SimplePredicate field="fld2" operator="greaterThan" value="20.0"/>
     </CompoundPredicate>
     <Node score="tgtX">
      <SimplePredicate field="fld2" operator="lessThan" value="40.0"/>
     </Node>
     <Node score="tgtZ">
      <SimplePredicate field="fld2" operator="greaterOrEqual" value="10.0"/>
     </Node>
    </Node>
    <Node score="tgtZ">
     <CompoundPredicate booleanOperator="or">
      <SimplePredicate field="fld1" operator="greaterOrEqual" value="60.0"/>
      <SimplePredicate field="fld1" operator="lessOrEqual" value="70.0"/>
     </CompoundPredicate>
     <Node score="tgtZ">
      <SimpleSetPredicate booleanOperator="isNotIn" field="fld4">
       <Array type="string">optA optB</Array>
      </SimpleSetPredicate>
     </Node>
```

```
      </Node>
    </Node>
    <Node score="tgtY">
      <CompoundPredicate booleanOperator="or">
        <SimplePredicate field="fld4" operator="equal" value="optA"/>
        <SimplePredicate field="fld4" operator="equal" value="optC"/>
      </CompoundPredicate>
      <Node score="tgtY">
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="fld1" operator="greaterThan" value="10.0"/>
          <SimplePredicate field="fld1" operator="lessThan" value="50.0"/>
          <SimplePredicate field="fld4" operator="equal" value="optA"/>
          <SimplePredicate field="fld2" operator="lessThan" value="100.0"/>
          <SimplePredicate field="fld3" operator="equal" value="false"/>
        </CompoundPredicate>
      </Node>
      <Node score="tgtZ">
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="fld4" operator="equal" value="optC"/>
          <SimplePredicate field="fld2" operator="lessThan" value="30.0"/>
        </CompoundPredicate>
      </Node>
    </Node>
  </Node>
 </TreeModel>
</PMML>
```

PMML マイニングモデルの例 (modelChain)

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-
1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
 <Header>
  <Application name="Drools-PMML" version="7.0.0-SNAPSHOT" />
 </Header>
 <DataDictionary numberOfFields="7">
  <DataField name="age" optype="continuous" dataType="double" />
  <DataField name="occupation" optype="categorical" dataType="string">
    <Value value="SKYDIVER" />
    <Value value="ASTRONAUT" />
    <Value value="PROGRAMMER" />
    <Value value="TEACHER" />
    <Value value="INSTRUCTOR" />
  </DataField>
  <DataField name="residenceState" optype="categorical" dataType="string">
    <Value value="AP" />
    <Value value="KN" />
    <Value value="TN" />
  </DataField>
  <DataField name="validLicense" optype="categorical" dataType="boolean" />
  <DataField name="overallScore" optype="continuous" dataType="double" />
  <DataField name="grade" optype="categorical" dataType="string">
    <Value value="A" />
    <Value value="B" />
    <Value value="C" />
    <Value value="D" />
```

```
      <Value value="F" />
    </DataField>
    <DataField name="qualificationLevel" optype="categorical" dataType="string">
      <Value value="Unqualified" />
      <Value value="Barely" />
      <Value value="Well" />
      <Value value="Over" />
    </DataField>
  </DataDictionary>
  <MiningModel modelName="SampleModelChainMine" functionName="classification">
    <MiningSchema>
      <MiningField name="age" />
      <MiningField name="occupation" />
      <MiningField name="residenceState" />
      <MiningField name="validLicense" />
      <MiningField name="overallScore" />
      <MiningField name="qualificationLevel" usageType="target"/>
    </MiningSchema>
    <Segmentation multipleModelMethod="modelChain">
      <Segment id="1">
        <True />
        <Scorecard modelName="Sample Score 1" useReasonCodes="true" isScorable="true"
functionName="regression"              baselineScore="0.0" initialScore="0.345">
          <MiningSchema>
            <MiningField name="age" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="occupation" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="residenceState" usageType="active" invalidValueTreatment="asMissing"
/>
            <MiningField name="validLicense" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="overallScore" usageType="predicted" />
          </MiningSchema>
          <Output>
            <OutputField name="calculatedScore" displayName="Final Score" dataType="double"
feature="predictedValue"                    targetField="overallScore" />
          </Output>
          <Characteristics>
            <Characteristic name="AgeScore" baselineScore="0.0" reasonCode="ABZ">
              <Extension name="cellRef" value="$B$8" />
              <Attribute partialScore="10.0">
                <Extension name="cellRef" value="$C$10" />
                <SimplePredicate field="age" operator="lessOrEqual" value="5" />
              </Attribute>
              <Attribute partialScore="30.0" reasonCode="CX1">
                <Extension name="cellRef" value="$C$11" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="5" />
                  <SimplePredicate field="age" operator="lessThan" value="12" />
                </CompoundPredicate>
              </Attribute>
              <Attribute partialScore="40.0" reasonCode="CX2">
                <Extension name="cellRef" value="$C$12" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="13" />
                  <SimplePredicate field="age" operator="lessThan" value="44" />
                </CompoundPredicate>
              </Attribute>
```

```
        <Attribute partialScore="25.0">
          <Extension name="cellRef" value="$C$13" />
          <SimplePredicate field="age" operator="greaterOrEqual" value="45" />
        </Attribute>
      </Characteristic>
      <Characteristic name="OccupationScore" baselineScore="0.0">
        <Extension name="cellRef" value="$B$16" />
        <Attribute partialScore="-10.0" reasonCode="CX2">
          <Extension name="description" value="skydiving is a risky occupation" />
          <Extension name="cellRef" value="$C$18" />
          <SimpleSetPredicate field="occupation" booleanOperator="isIn">
            <Array n="2" type="string">SKYDIVER ASTRONAUT</Array>
          </SimpleSetPredicate>
        </Attribute>
        <Attribute partialScore="10.0">
          <Extension name="cellRef" value="$C$19" />
          <SimpleSetPredicate field="occupation" booleanOperator="isIn">
            <Array n="2" type="string">TEACHER INSTRUCTOR</Array>
          </SimpleSetPredicate>
        </Attribute>
        <Attribute partialScore="5.0">
          <Extension name="cellRef" value="$C$20" />
          <SimplePredicate field="occupation" operator="equal" value="PROGRAMMER" />
        </Attribute>
      </Characteristic>
      <Characteristic name="ResidenceStateScore" baselineScore="0.0" reasonCode="RES">
        <Extension name="cellRef" value="$B$22" />
        <Attribute partialScore="-10.0">
          <Extension name="cellRef" value="$C$24" />
          <SimplePredicate field="residenceState" operator="equal" value="AP" />
        </Attribute>
        <Attribute partialScore="10.0">
          <Extension name="cellRef" value="$C$25" />
          <SimplePredicate field="residenceState" operator="equal" value="KN" />
        </Attribute>
        <Attribute partialScore="5.0">
          <Extension name="cellRef" value="$C$26" />
          <SimplePredicate field="residenceState" operator="equal" value="TN" />
        </Attribute>
      </Characteristic>
      <Characteristic name="ValidLicenseScore" baselineScore="0.0">
        <Extension name="cellRef" value="$B$28" />
        <Attribute partialScore="1.0" reasonCode="LX00">
          <Extension name="cellRef" value="$C$30" />
          <SimplePredicate field="validLicense" operator="equal" value="true" />
        </Attribute>
        <Attribute partialScore="-1.0" reasonCode="LX00">
          <Extension name="cellRef" value="$C$31" />
          <SimplePredicate field="validLicense" operator="equal" value="false" />
        </Attribute>
      </Characteristic>
    </Characteristics>
  </Scorecard>
</Segment>
<Segment id="2">
  <True />
```

```
      <TreeModel modelName="SampleTree" functionName="classification"
missingValueStrategy="lastPrediction" noTrueChildStrategy="returnLastPrediction">
        <MiningSchema>
          <MiningField name="age" usageType="active" />
          <MiningField name="validLicense" usageType="active" />
          <MiningField name="calculatedScore" usageType="active" />
          <MiningField name="qualificationLevel" usageType="predicted" />
        </MiningSchema>
        <Output>
          <OutputField name="qualification" displayName="Qualification Level" dataType="string"
feature="predictedValue"                    targetField="qualificationLevel" />
        </Output>
        <Node score="Well" id="1">
          <True/>
          <Node score="Barely" id="2">
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="age" operator="greaterOrEqual" value="16" />
              <SimplePredicate field="validLicense" operator="equal" value="true" />
            </CompoundPredicate>
            <Node score="Barely" id="3">
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="50.0" />
            </Node>
            <Node score="Well" id="4">
              <CompoundPredicate booleanOperator="and">
                <SimplePredicate field="calculatedScore" operator="greaterThan" value="50.0" />
                <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="60.0" />
              </CompoundPredicate>
            </Node>
            <Node score="Over" id="5">
              <SimplePredicate field="calculatedScore" operator="greaterThan" value="60.0" />
            </Node>
          </Node>
          <Node score="Unqualified" id="6">
            <CompoundPredicate booleanOperator="surrogate">
              <SimplePredicate field="age" operator="lessThan" value="16" />
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="40.0" />
              <True />
            </CompoundPredicate>
          </Node>
        </Node>
      </TreeModel>
    </Segment>
  </Segmentation>
 </MiningModel>
</PMML>
```

第12章 RED HAT PROCESS AUTOMATION MANAGER における PMML サポート

Red Hat Process Automation Manager には、以下の PMML モデルタイプに対するコンシューマー適合サポートが含まれます。

- 回帰モデル
- スコアカードモデル
- ツリーモデル
- マイニングモデル (サブタイプとして **modelChain**、**selectAll**、および **selectFirst** が含まれる)

Red Hat Process Automation Manager でサポートされていないモデルを含め、すべての PMML モデルタイプの一覧は、DMG の PMML specification を参照してください。

Red Hat Process Automation Manager は、PMML レガシーと PMML 信頼など、2 つの実装を提供します。

重要

PMML レガシーの実装は Red Hat Process Automation Manager 7.10.0 で非推奨となり、今後の Red Hat Process Automation Manager リリースで PMML 信頼実装に置き換えられます。

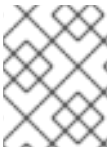
Red Hat Process Automation Manager には、PMML モデルエディターが同梱されていますが、XML または PMML 固有のオーサリングツールを使用して、Red Hat Process Automation Manager のデシジョンサービスで PMML モデルを作成して統合できます。Business Central (**Menu → Design → Projects → Import Asset**) でプロジェクトに PMML ファイルをインポートするか、Business Central なしにナレッジ JAR (KJAR) ファイルの一部として PMML ファイルをパッケージ化できます。

プロジェクトのパッケージ化およびデプロイメントの方法に、PMML ファイルなどのアセットを追加する方法の詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ**を参照してください

12.1. RED HAT PROCESS AUTOMATION MANAGER の PMML の信頼サポートおよび命名規則

Red Hat Process Automation Manager でプロジェクトに PMML ファイルを追加する場合は複数のアセットが生成されます。ツリーモデルおよびスコアカードモデルはルールに、リグレッションと最小モデルは Java クラスに変換されます。PMML モデルのタイプごとに、異なるアセットのセットが生成されますが、すべての PMML モデルタイプは少なくとも以下のアセットセットを生成します。

- PMML のファイル名から名前が派生する root パッケージ
- root パッケージでモデルをインスタンス化するために使用される Java ファクトリークラス
- モデル名から派生するモデル固有のサブパッケージ
- ルールモデルの場合にルールネットワークのインスタンス化に使用される **rule-mapper** クラス 2 つ
- マイニングモデルの場合に親モデルにネスト化された子モデルパッケージとクラス



注記

現時点では、PMML ファイルごとに 1 つのモデルのみが使用できます。また、拡張機能は一時的にサポートされていません。

以下は、生成された PMML パッケージ、クラス、ルールの命名規則です。

- root のパッケージ名は、小文字でスペースが含まれない、元の PMML ファイルの名前である (例: **sampleregression**)。
- 生成された **factory** Java クラスの名前は、 **fileName+"Factory"** の形式で、後ろに **Factory** を追加し、最初の文字が大文字になっている PMML ファイル名です (例: **SampleRegressionFactory**)。
- モデルのサブパッケージ名は小文字で、スペースのない元のモデルの名前です (例: **compoundnestedpredicatescorecard**)。
- 生成されたデータクラス名は、モデルタイプにより決まります。
  - ルールモデル: トップレベルの **PMMLRuleMappersImpl** は、サブパッケージでネストされた **PMMLRuleMapperImpl** クラスへの参照を含めて生成されます。
  - マイニングモデル:
    - 作成した **segmentation** サブパッケージの名前は、**modelName+"segmentation"** 形式で、元のモデルをスペースなし、小文字で表記し、後ろに **segmentation** が追加されています (例: **mixedminingsegmentation**)。
    - **segmentation** サブパッケージでは、ネスト化されたモデルへの参照が含まれる **segmentation** Java クラスが作成されます。作成された **segmentation** Java クラスの名前は、**modelName+Segmentation** の形式で、後ろに **Segmentation** が追加されたモデル名です (例: **MixedMiningSegmentation**)。
    - それぞれのセグメントに、特定のサブパッケージが作成されます。セグメント固有のサブパッケージの名前は、小文字の元のモデル名に **segment** 名と 0 で始まる進捗整数が追加されたものです。形式は、**modelName+segment+integer** のようになります。(例: **mixedminingsegment0**、**mixedminingsegment1**)

PMML 信頼実装における既知の制限

以下は、PMML 信頼には実装されない要素を記載しています。

- **Target** 要素は実装されていません
- **Extension** 要素が実装されていません
- 実装されない **MiningSchema** または **MiningField** 要素には以下が含まれます。
  - **importance**
  - **outliers**
  - **lowValue**
  - **highValue**
  - **invalidValueTreatment**

    - **invalidValueReplacement**

- 実装されない **OutputField** 要素には以下が含まれます。
    - 決定
    - 値
    - ルール機能
    - アルゴリズム
    - **isMultiValued**
    - **segmentId**
    - **isFinalResult**
- サポートされていない **TransformationDictionary** または **LocalTransformation** 式には以下が含まれます。
    - **NormContinuous**
    - **NormDiscrete**
    - **MapValues**
    - **TextIndex**
    - **Aggregate**
    - **Lag**
- **ModelStats**、**ModelExplanation** および **ModelExplanation** 要素は、リグレッション、ツリー、スコアカード、マイニングなど、どのモデルにも実装されていません。
- **verification** 要素はツリー、スコアカード、およびマイニングモデルには実装されていません。
- **VariableWeight** 要素はマイニングモデルに実装されていません
- 実装されないツリーモデル要素には、以下が含まれます。
    - **IsMissing** または **IsNotMissing**
    - **CompoundPredicate** の **Surrogate**
    - **missingValuePenalty**
    - **splitCharacteristic**
    - **isScorable**

12.2. RED HAT PROCESS AUTOMATION MANAGER の PMML のレガシーサポートおよび命名規則

Red Hat Process Automation Manager でプロジェクトに PMML ファイルを追加する場合は複数のアセットが生成されます。PMML モデルのタイプごとに、異なるアセットのセットが生成されますが、すべての PMML モデルタイプは少なくとも以下のアセットセットを生成します。

- PMML モデルに関連する全ルールを含む DRL ファイル
- 少なくとも以下の Java クラス 2 つが含まれている。
  - モデルタイプのデフォルトオブエジェクトタイプとして使用するデータクラス
  - データソースとルール実行の管理に使用する **RuleUnit** クラス

PMML ファイルに root モデルとして **MiningModel** が含まれている場合は、これらのファイルごとに複数のインスタンスが生成されます。

以下は、生成された PMML レガシーパッケージ、クラス、ルールの命名規則です。

- PMML モデルファイルにパッケージ名が指定されていない場合は、**"org.kie.pmml.pmml_4_2"+modelName** の形式で、生成されたルールのモデル名に、デフォルトのパッケージ名 **org.kie.pmml.pmml_4_2** がプリフィックスとして追加されます。
- 生成された **RuleUnit** Java クラスのパッケージ名は、生成されたルールのパッケージ名と同じです。
- 生成された **RuleUnit** Java クラスの名前は、 **RuleUnit** にモデル名を追加して **modelName+"RuleUnit"** の形式で指定します。
- PMML モデルにはそれぞれ、最低でも生成されたデータクラスが 1 つ含まれます。これらのクラスのパッケージ名は **org.kie.pmml.pmml_4_2.model** です。
- 生成されたデータクラス名は、モデルタイプにより決まり、プリフィックスとしてモデル名を指定します。
  - 回帰モデル: **modelName+"RegressionData"** という名前のデータクラス 1 つ
  - スコアカードモデル: **modelName+"ScoreCardData"** という名前のデータクラス 1 つ
  - ツリーモデル: **modelName+"TreeNode"** という名前の 1 つ目のデータクラスと、**modelName+"TreeToken"** という名前の 2 つ目のデータクラス 2 つ
  - マイニングモデル: **modelName+"MiningModelData"** という名前のデータクラス 1 つ

注記

マイニングモデルは、各セグメントに含まれるルールとクラスすべても生成します。

12.2.1. Red Hat Process Automation Manager における PMML 拡張

PMML 仕様は、PMML レガシーモデルのコンテンツを拡張する **Extension** 要素をサポートします。モデルの主要要素の最初と最後の子として、PMML モデル定義のほぼすべてのレベルで拡張を使用し、最大限に柔軟性をもたせることができます。PMML 拡張の詳細は DMG PMML の Extension Mechanism を参照してください。

Red Hat Process Automation Manager は、PMML 統合を最適化するために以下の追加の PMML 拡張をサポートします。

- **modelPackage**: 生成されたルールと Java クラスのパッケージ名を指定します。PMML モデルファイルの **Header** セクションにこの拡張を追加します。

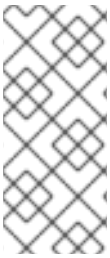
- **adapter**: ルールの入出力データを含めるのに使用するコンストラクトタイプ (**bean** または **trait**) を指定します。PMML モデルファイルの **MiningSchema** または **Output** セクション (または両方) にこの拡張を挿入します。

- **externalClass**: **adapter** 拡張と連携して使用し、**MiningField** または **OutputField** を定義します。この拡張には、**MiningField** または **OutputField** 要素名と一致する要素名のクラスが含まれます。

第13章 PMML モデルの実行

Business Central を使用して Red Hat Process Automation Manager に PMML ファイルをインポートする (**Menu → Design → Projects → Import Asset**) か、Business Central を使用しないでプロジェクトのナレッジ JAR (KJAR) ファイルの一部として DMN ファイルをパッケージ化できます。Red Hat Process Automation Manager プロジェクトに PMML ファイルに実装した後に、Java アプリケーションに直接 PMML 呼び出しを埋め込むか、設定済みの KIE Server に **ApplyPmmlModelCommand** コマンドを送信して、PMML ベースのデシジョンサービスを実行できます。

プロジェクトのパッケージ化およびデプロイメントの方法に PMML アセットを追加する方法の詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。



注記

Business Central の Decision Model and Notation (DMN) サービスの一部として、PMML モデルを追加することもできます。DMN ファイルに PMML モデルを追加すると、その PMML モデルを DMN デシジョンノードまたはビジネスナレッジモデルノードのボックス関数式として呼び出すことができます。DMN サービスへの PMML モデルの追加に関する詳細は、**DMN モデルを使用したデシジョンサービスの作成** を参照してください。

13.1. PMML 信頼呼び出しの JAVA アプリケーションへの直接組み込み

KIE コンテナーは、呼び出しプログラムにナレッジアセットを直接組み込む場合や、KJAR 用 Maven 依存関係を使用して物理的にプルする場合は、ローカルとみなされます。コードのバージョンと、PMML 定義のバージョンとの間に密接な関係がある場合は、ナレッジアセットをプロジェクトに直接組み込みます。意思決定への変更は、アプリケーションを更新して再デプロイしないと有効になりません。このアプローチに対する利点は、適切なオペレーションがランタイムへの外部の依存関係に依存していないことですが、ロックされた環境の場合は制限になる可能性があります。

前提条件

- 実行する PMML モデルを含む KJAR が作成されている。プロジェクトのパッケージ化に関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml-dependencies</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>
```

```
<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.52.0.Final-redhat-00007)。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. モデルの実行に使用する **PMMLRuntime** のインスタンスを作成します。

```
PMMLRuntime pmmlRuntime =
KieRuntimeFactory.of(kieContainer.getKieBase()).get(PMMLRuntime.class);
```

4. **PMMLRequestData** クラスのインスタンスを作成し、PMML モデルをデータセットに適用します。

```
PMMLRequestData pmmlRequestData = new PMMLRequestData({correlation_id},
{model_name});
pmmlRequestData.addRequestParam({parameter_name}, {parameter_value})
...
```

5. 入力データが含まれる **PMMLContext** クラスのインスタンスを作成します。

```
PMMLContext pmmlContext = new PMMLContextImpl(pmmlRequestData);
```

6. 作成した必須の PMML クラスインスタンスを使用して PMML の実行時に **PMML4Result** を取得します。

```
PMML4Result pmml4Result = pmmlRuntime.evaluate({model_name}, pmmlContext);
```

13.2. PMML レガシー呼び出しの JAVA アプリケーションへの直接組み込み

KIE コンテナーは、呼び出しプログラムにナレッジアセットを直接組み込む場合や、KJAR 用 Maven 依存関係を使用して物理的にプルする場合は、ローカルとみなされます。コードのバージョンと、PMML 定義のバージョンとの間に密接な関係がある場合は、ナレッジアセットをプロジェクトに直接組み込みます。意思決定への変更は、アプリケーションを更新して再デプロイしないと有効になりません。このアプローチに対する利点は、適切なオペレーションがランタイムへの外部の依存関係に依存していないことですが、ロックされた環境の場合は制限になる可能性があります。

Maven の依存関係を使用すると、システムプロパティーを使用して、更新を定期的にスキャンして自動的に更新するなど、特定バージョンの意思決定が動的に変更するため、柔軟性が高まります。これにより、外部の依存関係がサービスのデプロイ時間に影響を及ぼしますが、意思決定はローカルで実行されるため、ランタイム時に利用可能な外部サービスに対する信頼が低くなります。

前提条件

- 実行する PMML モデルを含む KJAR が作成されている。プロジェクトのパッケージ化に関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
```

```
</dependencies>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.52.0.Final-redhat-00007)。

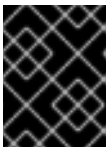
注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。



重要

レガシー実装を使用するには、**kie-pmml-implementation** システムプロパティーが **legacy** として設定されていることを確認します。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. **PMMLRequestData** クラスのインスタンスを作成し、PMML モデルをデータセットに適用します。

```
public class PMMLRequestData {
   private String correlationId; (1)
   private String modelName; (2)
   private String source; (3)
   private List<ParameterInfo<?>> requestParams; (4)
   ...
}
```

(1) 特定の要求または結果に関連のあるデータを特定

(2) 要求データに適用する必要のあるモデル名

(3) 要求を生成したセグメントを特定するために、内部で生成された **PMMLRequestData** オブジェクトが使用

(4) 入力データポイントを送信するデフォルトのメカニズム

4. **PMML4Result** クラスのインスタンスを作成します。このクラスでは、入力データに PMML ベースルールを適用した結果の出力情報を保持します。

```
public class PMML4Result {
   private String correlationId;
   private String segmentationId; (1)
   private String segmentId; (2)
   private int segmentIndex; (3)
   private String resultCode; (4)
   private Map<String, Object> resultVariables; (5)
   ...
}
```

(1) モデルタイプが **MiningModel** の場合に使用します。**segmentationId** は、複数のセグメントを区別化するために使用します。

(2) **segmentationId** と併用して、結果を生成したセグメントを特定します。

(3) セグメントの順番を維持するのに使用します。

(4) モデルが正常に適用されたかどうかを判断するのに使用します。**OK** は成功を示します。

(5) 結果の値として代入される変数とそれに関連する値の名前が含まれます。

通常の getter メソッドに加え、**PMML4Result** クラスも、結果となる変数の値を直接取得する以下の方法をサポートします。

```
public <T> Optional<T> getResultValue(String objName, String objField, Class<T> clazz,
Object...params)

public Object getResultValue(String objName, String objField, Object...params)
```

5. **ParameterInfo** クラスのインスタンスを作成します。このクラスは、**PMMLRequestData** クラスの一部として使用する、基本的なデータタイプオブジェクトのラッパーとして機能します。

```
public class ParameterInfo<T> { (1)
    private String correlationId;
    private String name; (2)
    private String capitalizedName;
    private Class<T> type; (3)
    private T value; (4)
    ...
}
```

(1) 多数の異なるタイプを処理するパラメーター化されたクラス

(2) モデルの入力として想定される変数の名前

(3) 変数の実際のタイプとなるクラス

(4) 変数の実際の値

6. 先ほど作成した対象の PMML クラスインスタンスをもとに PMML モデルを実行します。

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                                       modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                                  (InternalKnowledgeBase)kbase,
                                                                  possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}
```

```
protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

ルールは **RuleUnitExecutor** クラスにより実行されます。**RuleUnitExecutor** クラスは KIE セッションを作成し、必要な **DataSource** オブジェクトをこれらのセッションに追加してから、**run()** メソッドへのパラメーターとして渡される **RuleUnit** をもとに、ルールを実行します。**calculatePossiblePackageNames** と **getStartingRuleUnit** メソッドは、**run()** メソッドに渡される **RuleUnit** クラスの完全修飾名を決定します。

PMML モデル実行をスムーズに行うため、Red Hat Process Automation Manager でサポートされている **PMML4ExecutionHelper** クラスも使用できます。PMML ヘルパークラスに関する詳細は、「PMML 実行ヘルパークラス」 を参照してください。

13.2.1. PMML 実行ヘルパークラス

Red Hat Process Automation Manager には、PMML モデル実行に必要な **PMMLRequestData** クラスの作成や **RuleUnitExecutor** クラスを使用したルールの実行をサポートする **PMML4ExecutionHelper** クラスがあります。

以下では、比較の目的で **PMML4ExecutionHelper** クラスのある場合とない場合の PMML モデル実行の例を紹介しています。

PMML4ExecutionHelper の使用なしの PMML モデル実行例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                    modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                (InternalKnowledgeBase)kbase,
                                                possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}

protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
```

```
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

PMML4ExecutionHelper を使用した PMML モデル実行例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String modelPkgName,
                         String correlationId) {
  PMML4ExecutionHelper helper = PMML4ExecutionHelperFactory.getExecutionHelper(modelName,
kbase);
  helper.addPossiblePackageName(modelPkgName);

  PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
  variables.entrySet().forEach(entry -> {
    request.addRequestParam(entry.getKey(), entry.getValue);
  });

  PMML4Result resultHolder = helper.submitRequest(request);
  if ("OK".equals(resultHolder.getResultCode)) {
    // extract result variables here
  }
}
```

PMML4ExecutionHelper を使用する場合は、一般的な PMML モデル実行で行うように、考えられる名前または **RuleUnit** クラスを指定する必要はありません。

PMML4ExecutionHelper クラスを構築するには、**PMML4ExecutionHelperFactory** クラスを使用して、**PMML4ExecutionHelper** の取得方法を決定します。

以下は、**PMML4ExecutionHelper** を構築するのに利用可能な **PMML4ExecutionHelperFactory** クラスメソッドです。

KIE ベースにある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットがすでにコンパイルされており、既存の KIE ベースで使用されている場合には、これらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase,
boolean includeMiningDataSources)
```

プロジェクトクラスパスにある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットがプロジェクトクラスパスにある場合にこれらのメソッドを使用します。**classPath** の引数は、PMML ファイルのプロジェクトクラスパスの場所を指します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName,  String classPath,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName,String classPath,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

バイトアレイ形式の PMML アセット向けの PMML4ExecutionHelperFactory メソッド

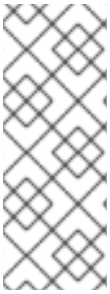
PMML アセットがバイトアレイ形式の場合にこれらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

Resource にある PMML アセット向けの PMML4ExecutionHelperFactory メソッド

PMML アセットが **org.kie.api.io.Resource** オブジェクトの形式の場合に、これらのメソッドを使用します。

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```



注記

クラスパス、バイトアレイ、リソース **PMML4ExecutionHelperFactory** メソッドは、生成されたルールおよび Java クラスの KIE コンテナーを作成します。このコンテナーは、**RuleUnitExecutor** が使用する KIE ベースのソースとして使用します。ただし、このコンテナーには永続性はありません。PMML アセットの **PMML4ExecutionHelperFactory** メソッドが KIE ベースにすでにあるには、この方法では KIE コンテナーは作成されません。

13.3. KIE SERVER を使用した PMML モデルの実行

ApplyPmmlModelCommand コマンドを設定済みの KIE Server に送信して KIE Server にデプロイした PMML モデルを実行できます。このコマンドを使用する場合は、**PMMLRequestData** オブジェクトが KIE Server に送信され、**PMML4Result** の結果オブジェクトを返信として受信します。設定済みの Java クラスからの KIE Server REST API または、REST クライアントから直接 KIE Server に PMML 要求を送信できます。

前提条件

- KIE Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、Red Hat Process Automation Manager インストールの計画 を参照してください。
- KIE コンテナーを、PMML モデルを含む KJAR の形式で KIE Server にデプロイしている。プロジェクトのパッケージ化に関する詳細は、Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ を参照してください。

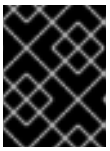
- PMML モデルを含む KIE コンテナーのコンテナー ID がある。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係を追加します。

レガシー実装の例

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```



重要

レガシー実装を使用するには、**kie-pmml-implementation** システムプロパティーが **legacy** として設定されていることを確認します。

信頼実装の例

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml-dependencies</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
```

```
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.52.0.Final-redhat-00007)。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. **classpath** または **ReleaseId** から KIE コンテナーを作成します。

```
KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

または、以下のオプションを使用します。

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. KIE Server への要求を送信して、応答を受信するクラスを作成します。

```
public class ApplyScorecardModel {
  private static final ReleaseId releaseId =
        new ReleaseId("org.acme","my-kjar","1.0.0");
  private static final String containerId = "SampleModelContainer";
  private static KieCommands commandFactory;
  private static ClassLoader kjarClassLoader; (1)
  private RuleServicesClient serviceClient; (2)

  // Attributes specific to your class instance
  private String rankedFirstCode;
  private Double score;

  // Initialization of non-final static attributes
  static {
    commandFactory = KieServices.Factory.get().getCommands();

    // Specifications for kjarClassLoader, if used
    KieMavenRepository kmp = KieMavenRepository.getMavenRepository();
    File artifactFile = kmp.resolveArtifact(releaseId).getFile();
    if (artifactFile != null) {
      URL urls[] = new URL[1];
      try {
        urls[0] = artifactFile.toURI().toURL();
        classLoader = new KieURLClassLoader(urls,PMML4Result.class.getClassLoader());
      } catch (MalformedURLException e) {
        logger.error("Error getting classLoader for "+containerId);
        logger.error(e.getMessage());
      }
    } else {
      logger.warn("Did not find the artifact file for "+releaseId.toString());
    }
  }

  public ApplyScorecardModel(KieServicesConfiguration kieConfig) {
    KieServicesClient clientFactory = KieServicesFactory.newKieServicesClient(kieConfig);
    serviceClient = clientFactory.getServicesClient(RuleServicesClient.class);
  }
  ...
  // Getters and setters
  ...

  // Method for executing the PMML model on KIE Server
  public void applyModel(String occupation, int age) {
    PMMLRequestData input = new PMMLRequestData("1234","SampleModelName"); (3)
    input.addRequestParam(new ParameterInfo("1234","occupation",String.class,occupation));
    input.addRequestParam(new ParameterInfo("1234","age",Integer.class,age));

    CommandFactoryServiceImpl cf = (CommandFactoryServiceImpl)commandFactory;
    ApplyPmmlModelCommand command = (ApplyPmmlModelCommand)
```

```
cf.newApplyPmmlModel(request); 4

    ServiceResponse<ExecutionResults> results =
        ruleClient.executeCommandsWithResults(CONTAINER_ID, command); 5

    if (results != null) { 6
      PMML4Result resultHolder = (PMML4Result)results.getResult().getValue("results");
      if (resultHolder != null && "OK".equals(resultHolder.getResultCode())) {
        this.score = resultHolder.getResultValue("ScoreCard","score",Double.class).get();
        Map<String,Object> rankingMap =
            (Map<String,Object>)resultHolder.getResultValue("ScoreCard","ranking");
        if (rankingMap != null && !rankingMap.isEmpty()) {
          this.rankedFirstCode = rankingMap.keySet().iterator().next();
        }
      }
    }
  }
}
```

1. クライアントプロジェクトの依存関係に KJAR を追加しなかった場合は、クラ出力ダーを定義します。
2. KIE Server REST API のアクセス認証情報など、設定設定で定義したサービスクライアントを特定します。
3. **PMMLRequestData** オブジェクトを初期化します。
4. **ApplyPmmlModelCommand** のインスタンスを作成します。
5. サービスクライアントを使用してコマンドを送信します。
6. 実行済みの PMML モデルの結果を取得します。

4. クラスインスタンスを実行して、PMML 呼び出し要求を KIE Server に送信します。または JMS および REST インターフェイスを使用して、**ApplyPmmlModelCommand** コマンドを KIE Server に送信できます。REST 要求については、**ApplyPmmlModelCommand** コマンドを、JSON、JAXB、または XStream 要求形式で、**http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}** への **POST** 要求として使用できます。

POST エンドポイントの例

```
http://localhost:8080/kie-
server/services/rest/server/containers/instances/SampleModelContainer
```

JSON リクエストボディの例

```
{
  "commands": [ {
    "apply-pmml-model-command": {
      "outIdentifier": null,
      "packageName": null,
      "hasMining": false,
      "requestData": {
```

```
        "correlationId": "123",
        "modelName": "SimpleScorecard",
        "source": null,
        "requestParams": [
          {
            "correlationId": "123",
            "name": "param1",
            "type": "java.lang.Double",
            "value": "10.0"
          },
          {
            "correlationId": "123",
            "name": "param2",
            "type": "java.lang.Double",
            "value": "15.0"
          }
        ]
      }
    }
  ]
}
```

エンドポイントよびボディーを含む curl 要求の例

```
curl -X POST "http://localhost:8080/kie-
server/services/rest/server/containers/instances/SampleModelContainer" -H "accept:
application/json" -H "content-type: application/json" -d "{ \"commands\": [ { \"apply-pmml-
model-command\": { \"outIdentifier\": null, \"packageName\": null, \"hasMining\": false,
\"requestData\": { \"correlationId\": \"123\", \"modelName\": \"SimpleScorecard\", \"source\":
null, \"requestParams\": [ { \"correlationId\": \"123\", \"name\": \"param1\", \"type\":
\"java.lang.Double\", \"value\": \"10.0\" }, { \"correlationId\": \"123\", \"name\": \"param2\",
\"type\": \"java.lang.Double\", \"value\": \"15.0\" } ] } } } ]}"
```

JSON の応答例

```
{
  "results" : [ {
    "value" : {"org.kie.api.pmml.DoubleFieldOutput":{
  "value" : 40.8,
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "name" : "OverallScore",
  "displayValue" : "OverallScore",
  "weight" : 1.0
}},
    "key" : "OverallScore"
  }, {
    "value" : {"org.kie.api.pmml.PMML4Result":{
  "resultVariables" : {
    "OverallScore" : {
      "value" : 40.8,
      "correlationId" : "123",
      "segmentationId" : null,
```

```
      "segmentId" : null,
      "name" : "OverallScore",
      "displayValue" : "OverallScore",
      "weight" : 1.0
    },
    "ScoreCard" : {
      "modelName" : "SimpleScorecard",
      "score" : 40.8,
      "holder" : {
        "modelName" : "SimpleScorecard",
        "correlationId" : "123",
        "voverallScore" : null,
        "moverallScore" : true,
        "vparam1" : 10.0,
        "mparam1" : false,
        "vparam2" : 15.0,
        "mparam2" : false
      },
      "enableRC" : true,
      "pointsBelow" : true,
      "ranking" : {
        "reasonCh1" : 5.0,
        "reasonCh2" : -6.0
      }
    }
  },
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "segmentIndex" : 0,
  "resultCode" : "OK",
  "resultObjectName" : null
}},
    "key" : "results"
  } ],
  "facts" : [ ]
}
```

第14章 関連情報

- PMML 仕様
- **Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ**
- **KIE API を使った Red Hat Process Automation Manager の操作**

パート III. DRL ルールを使用したデシジョンサービスの作成

ビジネスルール開発者は、Business Central で DRL (Drools Rule Language) デザイナーを使用してビジネスルールを定義できます。DRL ルールは、Business Central におけるその他のルールアセットとは異なり、ガイド付きまたは表形式ではなく、フリーフォームの **.drl** テキストファイルで直接定義できます。このような DRL ファイルは、プロジェクトのデシジョンサービスの中心となります。

注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

前提条件

- DRL ルールのチームおよびプロジェクトが Business Central に作成されている。各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は **デシジョンサービスのスタートガイド** を参照してください。

第15章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表15.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第16章 DRL (DROOLS RULE LANGUAGE) ルール

DRL (Drools Rule Language) ルールは、**.drl** テキストファイルに直接定義するビジネスルールです。このような DRL ファイルは、Business Central の他のすべてのルールアセットが最終的にレンダリングされるソースとなります。Business Central インターフェイスで DRL ファイルを作成して管理するか、Red Hat CodeReady Studio や別の統合開発環境 (IDE) を使用して Maven または Java プロジェクトの一部として外部で作成することができます。DRL ファイルには、最低でもルールの条件 (**when**) およびアクション (**then**) を定義するルールを 1 つ以上追加できます。Business Central の DRL デザイナーでは、Java、DRL、および XML の構文が強調表示されます。

DRL ファイルは、以下のコンポーネントで設定されます。

DRL ファイル内のコンポーネント

```
package

import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end

rule "rule2 name"

...
```

以下の DRL ルールの例では、ローン申し込みのデシジョンサービスで年齢制限を指定します。

申込者の年齢制限に関するルールの例

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

DRL ファイルには、ルール、クエリー、関数が 1 つまたは複数含まれており、このファイルで、ルールやクエリーで割り当て、使用するインポート、グローバル、属性などのリソース宣言を定義できます。

DRL パッケージは、DRL ファイルの一番上に表示され、ルールは通常最後に表示されます。他の DRL コンポーネントはどのような順番でも構いません。

ルールごとに、ルールパッケージ内で一意の名前を指定する必要があります。パッケージ内の DRL ファイルで、同じルール名を複数回使用すると、ルールのコンパイルに失敗します。特にルール名にスペースを使用する場合など、ルール名には必ず二重引用符 (**rule "rule name"**) を使用して、コンパイルエラーが発生しないようにしてください。

DRL ルールに関連するデータオブジェクトはすべて、DRL ファイルと同じ Business Central プロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。その他のパッケージの既存アセットは、DRL ルールを使用してインポートできます。

16.1. DRL のパッケージ

パッケージは、データオブジェクト、DRL ファイル、デシジョンテーブル、他のアセットタイプなど、Red Hat Process Automation Manager に含まれる関連アセットをまとめたディレクトリーです。また、パッケージは、各ルールグループに固有の namespace としても機能します。1 つのルールベースには、複数のパッケージを含めることができます。通常、パッケージだけで自己完結できるように、パッケージのすべてのルールはパッケージ宣言と同じファイルに保存します。ただし、対象のルールで使用するのに、他のパッケージからオブジェクトをインポートできます。

以下は、ローン申請デシジョンサービスの DRL ファイルのパッケージ名と名前空間の例です。

DRL ファイルのパッケージ定義例

```
package org.mortgages;
```

16.2. DRL のインポートステートメント

Java のインポートステートメントと同様に、DRL ファイルのインポートで、ルールで使用するオブジェクトの完全修飾パスとタイプ名を識別します。**packageName.objectName** の形式でパッケージとデータオブジェクトを指定し、複数のインポートを指定する場合には別の行に指定します。デシジョンエンジンは自動的に、DRL パッケージと同じ名前の Java パッケージおよび、**java.lang** パッケージからクラスを自動的にインポートします。

以下の例は、住宅ローン申請デシジョンサービスに含まれるローン申請オブジェクトのインポートステートメントです。

DRL ファイルのインポートステートメント例

```
import org.mortgages.LoanApplication;
```

16.3. DRL の機能

DRL ファイルの関数は、Java クラスではなく、ルールのソースファイルにセマンティックコードを追加します。関数は、特に、ルールのアクション (**then**) 部分が繰り返し使用され、パラメーターだけがルールごとに異なる場合に便利です。DRL ファイルのルールで、関数を宣言したり、ヘルパークラスから静的メソッドを関数としてインポートしたりすることで、ルールのアクション (**then**) 部分で、名前を指定して関数を使用できます。

以下は、DRL ファイルで宣言またはインポートされる関数の例です。

ルールを含む関数宣言の例 (オプション 1)

-

```
function String hello(String applicantName) {
    return "Hello " + applicantName + "!";
}

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

ルールを含む関数インポートの例 (オプション 2)

```
import function my.package.applicant.hello;

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

16.4. DRL のクエリー

DRL ファイルのクエリーは、デシジョンエンジンのワーキングメモリーで DRL ファイル内のルールに関連するファクトを検索します。DRL ファイルにクエリー定義を追加してから、アプリケーションコードで一致する結果を取得します。クエリーは、定義した条件セットを検索するため、**when** または **then** を指定する必要はありません。クエリー名は KIE ベースでグローバルとなるため、プロジェクトにあるその他のすべてのルールクエリーと重複しないようにする必要があります。クエリーの結果を返すには、**ksession.getQueryResults("name")** を使用して **QueryResults** 定義を設定します (**"name"** はクエリー名)。これにより、クエリーの結果が返り、クエリーに一致したオブジェクトを取得できるようになります。DRL ファイルのルールに、クエリーとクエリー結果パラメーターを定義します。

以下は、ローン申請デシジョンサービスの未成年の申請者に関する DRL ファイルのクエリー定義と、付属のアプリケーションコードの例です。

DRL ファイルにおけるクエリー定義の例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

クエリー結果を取得するためのアプリケーションコードの例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age  of 21" );
```

標準的な **for** ループを使用して、返される **QueryResults** を反復処理できます。各要素は **QueryResultsRow** で、これを使用してタプルの各列にアクセスできます。

クエリー結果を取得および反復するアプリケーションのコード例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
```

```
System.out.println( "we have " + results.size() + " people under the age of 21" );

System.out.println( "These people are under the age of 21:" );

for ( QueryResultsRow row : results ) {
    Person person = ( Person ) row.get( "person" );
    System.out.println( person.getName() + "\n" );
}
```

16.5. DRL でのタイプ宣言とメタデータ

DRL ファイルの宣言は、DRL ファイルのルールで使用するファクトタイプまたはメタデータを新たに定義します。

- **新規ファクトタイプ:** Red Hat Process Automation Manager に含まれる **java.lang** パッケージのデフォルトのファクトタイプは **Object** ですが、必要に応じて DRL ファイルで他のタイプを宣言できます。DRL ファイルにファクトタイプを宣言すると、Java などの低級言語でモデルを作成せず、デシジョンエンジンに直接新しいファクトモデルを定義するようになります。また、ドメインモデルがすでにビルドされていて、推論 (reasoning) のプロセスで主に使用する追加のエンティティーでこのモデルを補完する場合に、新規タイプを宣言することもできます。

- **ファクトタイプのメタデータ: @key(value)** 形式のメタデータを新規または既存のファクトを関連付けることができます。メタデータには、ファクト属性で表現されていない、該当のファクトタイプのすべてのインスタンス間で一貫性のあるすべての種類のデータを使用できます。メタデータはランタイム時に、デシジョンエンジンでクエリーでき、推論 (reasoning) のプロセスで使用できます。

16.5.1. DRL のメタデータを使用しないタイプ宣言

新規ファクトの宣言にメタデータは必要ありませんが、属性またはフィールドの一覧を含める必要があります。タイプ宣言に指定属性が含まれていない場合は、デシジョンエンジンがクラスパス内で既存のファクトクラスを検索し、クラスが見つからない場合はエラーを出します。

以下の例は、DRL ファイルにメタデータがない新規ファクトタイプ **Person** の宣言です。

ルールが 1 つ含まれる新規ファクトタイプの宣言の例

```
declare Person
  name : String
  dateOfBirth : java.util.Date
  address : Address
end

rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
    mark.setName( "Mark" );
    insert( mark );
end
```

この例では、新規ファクトタイプ **Person** には **name**、**dateOfBirth**、および **address** の 3 つの属性が含まれます。属性ごとにタイプがあり、このタイプには作成する別のクラスや以前に宣言したファクトタイプなどの、有効な Java タイプを指定できます。**dateOfBirth** 属性には、Java API からの

java.util.Date タイプがあり、**address** 属性には、以前に定義したファクトタイプ **Address** が含まれます。

宣言するたびにクラスの完全修飾名が記述されないように、完全なクラス名を **import** 句の一部として定義できます。

インポートの完全修飾クラス名でのタイプ宣言例

```
import java.util.Date

declare Person
    name : String
    dateOfBirth : Date
    address : Address
end
```

新規のファクトタイプを宣言すると、デシジョンエンジンはコンパイル時にファクトタイプを表す Java クラスを生成します。生成される Java クラスはタイプ定義の一対一の JavaBeans マッピングとなります。

たとえば、以下の Java クラスは **Person** タイプ宣言例から生成されます。

Person ファクトタイプの宣言用に生成された Java クラス

```
public class Person implements Serializable {
    private String name;
    private java.util.Date dateOfBirth;
    private Address address;

    // Empty constructor
    public Person() {...}

    // Constructor with all fields
    public Person( String name, Date dateOfBirth, Address address ) {...}

    // If keys are defined, constructor with keys
    public Person( ...keys... ) {...}

    // Getters and setters
    // `equals` and `hashCode`
    // `toString`
}
```

Person タイプ宣言が含まれる以前のルールの例が示すように、他のファクトと同様にルールで生成したクラスを使用できます。

宣言された Person ファクトタイプを使用するルールの例

```
rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
```

```
    mark.setName( "Mark" );
    insert( mark );
end
```

16.5.2. DRL の列挙タイプの宣言

DRL は、**declare enum <factType>** 形式の列挙タイプの宣言をサポートします。列挙タイプの後にはコンマ区切りの値の一覧が続き、最後はセミコロンで終了します。これにより、DRL ファイルのルールで列挙リストを使用できます。

たとえば、以下の列挙タイプの宣言では、従業員スケジュールルールの曜日を定義します。

スケジュールルールを使用した列挙タイプ宣言の例

```
declare enum DaysOfWeek

SUN("Sunday"),MON("Monday"),TUE("Tuesday"),WED("Wednesday"),THU("Thursday"),FRI("Friday"
),SAT("Saturday");

   fullName : String
end

rule "Using a declared Enum"
when
   $emp : Employee( dayOff == DaysOfWeek.MONDAY )
then
   ...
end
```

16.5.3. DRL の拡張タイプ宣言

DRL は、**declare <factType1> extends <factType2>** 形式のタイプ宣言の継承をサポートします。DRL で宣言したサブタイプで Java で宣言したタイプを拡張するには、フィールドなしの宣言ステートメントで親タイプを繰り返し使用します。

たとえば、以下のタイプ宣言はトップレベルの **Person** タイプから **Student** タイプを拡張し、さらに **Student** サブタイプから **LongTermStudent** タイプを拡張します。

拡張タイプ宣言の例

```
import org.people.Person

declare Person end

declare Student extends Person
    school : String
end

declare LongTermStudent extends Student
    years : int
    course : String
end
```

16.5.4. DRL のメタデータが含まれるタイプ宣言

@key(value) (値は任意) 形式のメタデータをファクトタイプまたはファクト属性に関連付けることができます。メタデータには、ファクト属性で表現されていない、該当のファクトタイプのすべてのインスタンス間で一貫性のあるすべての種類のデータを使用できます。メタデータはランタイム時に、デシジョンエンジンでクエリーでき、推論 (reasoning) のプロセスで使用できます。ファクトタイプの属性の前に宣言するメタデータはファクトタイプに割り当てられ、属性の後に宣言するメタデータはこの特定の属性に割り当てられます。

以下の例では、**@author** と **@dateOfCreation** のメタデータ属性 2 つが **Person** ファクトタイプに、**@key** と **@maxLength** のメタデータアイテム 2 つが **name** 属性に割り当てられています。**@key** メタデータ属性には必須の値がないので、括弧と値は省略されます。

ファクトタイプおよび属性のメタデータ宣言例

```
import java.util.Date

declare Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )

    name : String @key @maxLength( 30 )
    dateOfBirth : Date
    address : Address
end
```

既存タイプのメタデータ属性を宣言する場合には、すべての宣言の **import** 句の一部として、または個別の **declare** 句の一部として完全修飾クラス名を特定できます。

インポートタイプのメタデータ宣言の例

```
import org.drools.examples.Person

declare Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )
end
```

宣言タイプのメタデータ宣言例

```
declare org.drools.examples.Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )
end
```

16.5.5. DRL でのファクトタイプと属性宣言のメタデータタグ

DRL 宣言でカスタムメタデータ属性を定義することはできますが、デシジョンエンジンはファクトタイプまたはファクトタイプ属性の宣言についての、以下の事前定義メタデータタグもサポートします。

注記

VoiceCall クラスを参照する本セクションの例では、サンプルアプリケーションドメインモデルに以下のクラスの詳細が含まれていることを前提としています。

Telecom ドメインモデルの例における VoiceCall ファクトクラス

```
public class VoiceCall {
  private String  originNumber;
  private String  destinationNumber;
  private Date    callDateTime;
  private long    callDuration;  // in milliseconds

  // Constructors, getters, and setters
}
```

@role

このタグは、指定のファクトタイプが複雑なイベントの処理時にデシジョンエンジンにて通常のファクトまたはイベントとして処理されるかどうかを決定します。
デフォルトパラメーター: **fact**

サポート対象のパラメーター: **fact**、**event**

```
@role( fact | event )
```

例: イベントタイプとして VoiceCall の宣言

```
declare VoiceCall
  @role( event )
end
```

@timestamp

このタグは、デシジョンエンジンのすべてのイベントに自動的に割り当てられます。デフォルトでは、この時間はセッションクロックにより提供され、デシジョンエンジンのワーキングメモリーへの挿入時にイベントに割り当てられます。セッションクロックが追加するデフォルトのタイムスタンプの代わりに、カスタムのタイムスタンプ属性を指定できます。
デフォルトパラメーター: デシジョンエンジンのセッションクロックが追加する時間

サポート対象のパラメーター: セッションクロックタイムまたはカスタムのタイムスタンプ属性

```
@timestamp( <attributeName> )
```

例: VoiceCall のタイムスタンプ属性の宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
end
```

@duration

このタグは、デシジョンエンジンのイベントの持続期間を決定します。イベントは、interval-based

イベントまたは point-in-time イベントのいずれかになります。interval-based のイベントには持続期間があり、その持続期間が経過するまでデシジョンエンジンのワーキングメモリーで持続します。point-in-time イベントに持続期間はなく、基本的には期間がゼロの interval-based イベントになります。デフォルトでは、デシジョンエンジンのすべてのイベントの持続期間は 0 です。デフォルトの代わりに、カスタムの持続期間属性を指定できます。
デフォルトパラメーター: null (ゼロ)

サポート対象のパラメーター: カスタムの持続期間属性

```
@duration( <attributeName> )
```

例: VoiceCall の持続期間属性の宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
end
```

@expires

このタグは、デシジョンエンジンのワーキングメモリーでイベントの有効期限が切れるまでの時間を決定します。デフォルトでは、イベントは現在のルールのいずれにも一致せず、それらのいずれもアクティベートできなくなった時点で失効します。イベント失効までの期間を定義できます。また、このタグの定義は、KIE ベースの一時的な制約やスライディングウィンドウから算出した暗黙的な有効期限のオフセットもオーバーライドします。デシジョンエンジンがストリームモードで実行中の場合にのみ、このタグを使用できます。
デフォルトパラメーター: Null (イベントがルールに一致せず、ルールをアクティブにできなくなるとイベントの有効期限が切れる)

サポート対象のパラメーター: **[#d][#h][#m][#s][[ms]]** 形式のカスタムの **timeOffset** 属性

```
@expires( <timeOffset> )
```

例: VoiceCall イベントに対する有効期限のオフセットの宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
  @expires( 1h35m )
end
```

@typesafe

このタグは、型安全性を有効して/有効にせずに指定のファクトタイプをコンパイルするかどうかを決定します。デフォルトでは、すべてのタイプ宣言は型安全性が有効な状態でコンパイルされます。この動作を type-unsafe の評価にオーバーライドすることもできます。type-unsafe の評価の場合、すべての制約は MVEL 制約として生成され、動的に実行されます。これは、一般的なコレクションではない場合や、タイプが混同されているコレクションを処理する場合に便利です。
デフォルトパラメーター: **true**

サポート対象のパラメーター: **true**、**false**

```
@typesafe( <boolean> )
```

例: type-unsafe 評価の VoiceCall の宣言

```
declare VoiceCall
  @role( fact )
  @typesafe( false )
end
```

@serialVersionUID

このタグは、ファクト宣言でシリアル化可能なクラスの **serialVersionUID** の指定値を定義します。シリアル化可能なクラスで明示的に **serialVersionUID** が宣言されていない場合は、Java Object Serialization Specification に記載されているように、シリアル化のランタイムが、そのクラスのさまざまな側面に基づいてそのクラスのデフォルトの **serialVersionUID** 値を計算します。ただし、デシリアル化の結果を最適化し、シリアル化した KIE セッションの互換性を向上するには、関連のクラスや DRL 宣言の要件に応じて **serialVersionUID** を設定します。
デフォルトパラメーター: Null

サポート対象のパラメーター: カスタムの **serialVersionUID** 整数

```
@serialVersionUID( <integer> )
```

例: VoiceCall クラスの serialVersionUID の宣言

```
declare VoiceCall
  @serialVersionUID( 42 )
end
```

@key

このタグを指定すると、ファクトタイプ属性をファクトタイプのキー識別子として使用できるようになります。生成されたクラスは **equals()** メソッドと **hashCode()** メソッドを実装できるようになり、該当タイプの 2 つのインスタンスが同等であるかを判別できるようになります。デシジョンエンジンは、すべてのキー属性をパラメーターとして使用してコンストラクターを生成することもできます。
デフォルトパラメーター: なし

サポート対象のパラメーター: なし

```
<attributeDefinition> @key
```

例: Person タイプ属性をキーとして宣言する

```
declare Person
    firstName : String @key
    lastName : String @key
    age : int
end
```

この例では、デシジョンエンジンは **firstName** 属性と **lastName** 属性をチェックして、**Person** の 2 つのインスタンスが同等であるかどうかを判断しますが、**age** 属性はチェックしません。また、このデシジョンエンジンは暗黙的に 3 つのコンストラクターを生成します。1 つはパラメーターなし、

もう 1 つ目は **@key** フィールドのあるコンストラクター、3 つ目はすべてのフィールドのあるコンストラクターです。

キー宣言に基づくコンストラクターの例

```
Person() // Empty constructor

Person( String firstName, String lastName )

Person( String firstName, String lastName, int age )
```

以下の例のように、キーコンストラクターに基づいてタイプのインスタンスを作成できます。

キーコンストラクターを使用したインスタンスの例

```
Person person = new Person( "John", "Doe" );
```

@position

このタグは、位置引数で宣言されたファクトタイプ属性またはフィールドの位置を決定し、デフォルトで宣言した属性の順番をオーバーライドします。このタグを使用して、タイプ宣言または位置引数で制約の形式を維持しつつ、パターンの位置制約を変更できます。このタグは、クラスパスのクラスにあるフィールドに対してのみ使用できます。1 つのクラスのフィールドでこのタグを使用する場合も、使用しない場合もありますが、このタグのない属性は、宣言の順番では最後になります。クラスの継承はサポートされますが、メソッドのインターフェイスはサポートされません。
デフォルトパラメーター: なし

サポート対象のパラメーター: 整数

```
<attributeDefinition> @position ( <integer> )
```

例: ファクトタイプを宣言し、宣言した順番をオーバライドする

```
declare Person
    firstName : String @position( 1 )
    lastName : String @position( 0 )
    age : int @position( 2 )
    occupation: String
end
```

この例では、位置引数に含まれる属性の優先順位は以下のとおりです。

1. **lastName**
2. **firstName**
3. **age**
4. **occupation**

位置引数では、位置が既知の名前付きフィールドにマッピングされるため、フィールド名を指定する必要はありません。たとえば、**Person(lastName == "Doe")** の引数は **Person("Doe";)** と同じです。ここでは、**lastName** フィールドには、DRL 宣言の最上位の位置アノテーションが含まれます。セミコロン **;** は、その前にあるものはすべて位置引数であることを示します。セミコロンを使

用して引数を区切ることで、パターンで位置引数と名前付き引数を混ぜて使用できます。バインドされていない位置引数の変数は、その位置にマッピングされているフィールドにバインドされます。

以下のパターン例では、位置引数と名前付き引数をさまざまな方法で構築する方法を示します。このパターンには、制約が 2 つ、バインディングが 1 つ含まれており、セミコロンで位置引数のセクションと名前付き引数のセクションを分けています。位置引数では、変数とリテラル、およびリテラルのみを使用する式がサポートされていますが、変数だけの使用はサポートされていません。

位置引数および名前付き引数を含むパターン例

```
Person( "Doe", "John", $a; )

Person( "Doe", "John"; $a : age )

Person( "Doe"; firstName == "John", $a : age )

Person( lastName == "Doe"; firstName == "John", $a : age )
```

位置引数は、**入力引数** または **出力引数** とに分類できます。入力引数は、以前に宣言したバインディングと、ユニフィケーション (unification) を使用したそのバインディングに対する制約が含まれます。出力引数は、この宣言を生成して、バインディングがまだ存在しない場合は、これを位置引数で表現するフィールドにバインディングします。

拡張タイプの宣言で **@position** アノテーションを定義する場合は、属性の位置がサブタイプに継承されるため注意が必要です。このように継承されることで、属性の順番が混同し、混乱を生じさせる可能性があります。2 つのフィールドに同じ **@position** の値を指定でき、連続する値は宣言する必要がありません。位置が繰り返し使用される場合は、継承を使用することで競合が発生しないように解決されます。この場合は、親タイプの位置の値の優先順位が高く、その後は最初から最後の順番で宣言を使用します。

たとえば、以下の拡張タイプ宣言では、位置の優先順位が混合します。

位置アノテーションが混合した拡張ファクトタイプの例

```
declare Person
    firstName : String @position( 1 )
    lastName : String @position( 0 )
    age : int @position( 2 )
    occupation: String
end

declare Student extends Person
    degree : String @position( 1 )
    school : String @position( 0 )
    graduationDate : Date
end
```

この例では、位置引数に含まれる属性の優先順位は以下のとおりです。

1. **lastName** (親タイプでの位置 0)
2. **school** (サブタイプでの位置 0)
3. **firstName**(親タイプでの位置 1)

4. **degree** (サブタイプでの位置 1)
5. **age**(親タイプでの位置 2)
6. **occupation** (位置アノテーションがない最初のフィールド)
7. **graduationDate** (位置アノテーションがない 2 番目のフィールド)

16.5.6. ファクトタイプに対するプロパティー変更の設定およびリスナー

デフォルトでは、デシジョンエンジンは、ルールがトリガーされるたびに、ファクトタイプに対するすべてのファクトパターンを再評価しません。代わりに、指定のパターン内に制約またはバインドされている変更されたプロパティーのみに対応します。たとえば、ルールが、ルールアクションの一環として **modify()** を呼び出すものの、アクションが KIE ベースで新しいデータを生成しない場合は、データが変更されないため、デシジョンエンジンはすべてのファクトパターンを自動的に再評価しません。このプロパティーのリアクティビティー動作は、KIE ベースでの不要な再帰を阻止し、より効率的なルール評価をもたらします。また、この動作は無限再帰を回避するために **no-loop** ルール属性を必ずしも使用する必要がないことを意味します。

以下の **KnowledgeBuilderConfiguration** オプションを使用して、このプロパティーリアクティビティー動作を変更または無効にできます。次に、Java クラスまたは DRL ファイルでプロパティー変更設定を使用し、必要に応じてプロパティーリアクティビティーを調整します。

- **ALWAYS**: (デフォルト) すべてのタイプはプロパティーリアクティブです。ただし、**@classReactive** プロパティー変更設定を使用して、特定タイプのプロパティーリアクティビティーを無効にできます。
- **ALLOWED**: すべてのタイプはプロパティーリアクティブではありません。ただし、**@propertyReactive** プロパティー変更設定を使用して、特定タイプのプロパティーリアクティビティーを有効にできます。
- **DISABLED**: すべてのタイプはプロパティーリアクティブではありません。すべてのプロパティー変更リスナーは無視されます。

KnowledgeBuilderConfiguration におけるプロパティーリアクティビティー設定の例

```
KnowledgeBuilderConfiguration config =
KnowledgeBuilderFactory.newKnowledgeBuilderConfiguration();
config.setOption(PropertySpecificOption.ALLOWED);
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder(config);
```

または、Red Hat Process Automation Manager ディストリビューションにおける **standalone.xml** ファイルの **drools.propertySpecific** システムプロパティーを更新できます。

システムプロパティーにおけるプロパティーリアクティビティー設定の例

```
<system-properties>
  ...
  <property name="drools.propertySpecific" value="ALLOWED"/>
  ...
</system-properties>
```

デシジョンエンジンは、ファクトクラスまたは宣言された DRL ファクトタイプに対して、以下のプロパティー変更の設定およびリスナーをサポートします。

@classReactive

デシジョンエンジンでプロパティーリアクティビティーが **ALWAYS** に設定されている場合 (すべてのタイプはプロパティーリアクティブ)、このタグは特定の Java クラスまたは宣言された DRL ファクトタイプに対してデフォルトのプロパティーリアクティビティー動作を無効にします。このタグは、特定パターン内に制約またはバインドされる変更されたプロパティーのみに対応するのではなく、ルールがトリガーされるたびに指定されたファクトタイプのすべてのファクトパターンをデシジョンエンジンが再評価する必要がある場合に使用できます。

例: DRL タイプの宣言におけるデフォルトのプロパティーリアクティビティーの無効化

```
declare Person
  @classReactive
    firstName : String
    lastName : String
end
```

例: Java クラスにおけるデフォルトのプロパティーリアクティビティーの無効化

```
@classReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@propertyReactive

プロパティーリアクティビティーがデシジョンエンジンで **ALLOWED** に設定されている場合 (指定されていない場合、すべてのタイプはプロパティーリアクティブではない)、このタグは特定の Java クラスまたは宣言された DRL ファクトタイプに対してプロパティーリアクティビティーを有効にします。ルールがトリガーされるたびにファクトのすべてのファクトパターンを再評価するのではなく、指定されたファクトタイプの特定のパターン内で制約またはバインドされた変更済みプロパティーにのみデシジョンエンジンを反応させる場合は、このタグを使用できます。

例: DRL タイプの宣言におけるプロパティーリアクティビティーの有効化 (リアクティビティーがグローバルに無効にされる場合)

```
declare Person
  @propertyReactive
    firstName : String
    lastName : String
end
```

例: Java クラスでのプロパティーのリアクティビティーの有効化 (リアクティビティーがグローバルに無効にされる場合)

```
@propertyReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@watch

このタグは、DRL ルールのファクトパターンで、インラインで指定する追加のプロパティーに対す

るプロパティーリアクティビティーを有効にします。このタグがサポートされるのは、デシジョンエンジンでプロパティーリアクティビティーが **ALWAYS** に設定されている場合か、プロパティーリアクティビティーが **ALLOWED** に設定され、関連するファクトタイプが **@propertyReactive** タグを使用する場合に限られます。DRL ルールでこのタグを使用して、ファクトプロパティーリアクティビティー論理の指定されたプロパティーを追加または除外できます。
デフォルトパラメーター: なし

サポートされているパラメーター: プロパティー名、***** (all)、**!** (not)、**!*** (no properties)

```
<factPattern> @watch ( <property> )
```

例: ファクトパターンにおけるプロパティーリアクティビティーの有効化または無効化

```
// Listens for changes in both `firstName` (inferred) and `lastName`:
Person(firstName == $expectedFirstName) @watch( lastName )

// Listens for changes in all properties of the `Person` fact:
Person(firstName == $expectedFirstName) @watch( * )

// Listens for changes in `lastName` and explicitly excludes changes in `firstName`:
Person(firstName == $expectedFirstName) @watch( lastName, !firstName )

// Listens for changes in all properties of the `Person` fact except `age`:
Person(firstName == $expectedFirstName) @watch( *, !age )

// Excludes changes in all properties of the `Person` fact (equivalent to using `@classReactivity`
tag):
Person(firstName == $expectedFirstName) @watch( !* )
```

デシジョンエンジンは、**@classReactive** タグ (プロパティーリアクティビティーを無効にする) を使用するファクトタイプのプロパティーに対して **@watch** タグを使用する場合、またはデシジョンエンジンでプロパティーリアクティビティーが **ALLOWED** に設定され、関連するファクトタイプが **@propertyReactive** タグを使用しない場合は、コンパイルエラーを生成します。また、**@watch(firstName, ! firstName)** などのリスナーアノテーションでプロパティーを複製する場合でも、コンパイルエラーが生じます。

@propertyChangeSupport

JavaBeans Specification で定義されたプロパティー変更のサポートを実装するファクトの場合は、このタグによりデシジョンエンジンがファクトプロパティーの変更を監視できるようになります。

例: JavaBeans オブジェクトでのプロパティー変更のサポートの宣言

```
declare Person
    @propertyChangeSupport
end
```

16.5.7. アプリケーションコード内の DRL で宣言されたタイプへのアクセス

DRL の宣言タイプは通常 DRL ファイル内で使用され、 Java モデルは通常モデルをルールとアプリケーション間で共有する場合に使用されます。宣言タイプは KIE ベースのコンパイル時に生成されるため、アプリケーションはアプリケーションのランタイムまでこの宣言タイプにアクセスできません。状

況によっては、アプリケーションが宣言タイプからファクトに直接アクセスし、処理する必要があります (とくにアプリケーションがデシジョンエンジンをラップして、ルール管理用によりレベルの高い、ドメイン固有のユーザーインターフェイスを提供する場合)。

アプリケーションコードから宣言タイプを直接処理するには、Red Hat Process Automation Manager で **org.drools.definition.type.FactType** API を使用します。この API を使用して、宣言ファクトタイプでフィールドのインスタンス化、読み取り、書き込みを行います。

以下のコード例では、アプリケーションから **Person** ファクトタイプを直接変更します。

FactType API を使用した宣言されたファクトタイプを処理するアプリケーションコード例

```
import java.util.Date;

import org.kie.api.definition.type.FactType;
import org.kie.api.KieBase;
import org.kie.api.runtime.KieSession;

...

// Get a reference to a KIE base with the declared type:
KieBase kbase = ...

// Get the declared fact type:
FactType personType = kbase.getFactType("org.drools.examples", "Person");

// Create instances:
Object bob = personType.newInstance();

// Set attribute values:
personType.set(bob, "name", "Bob" );
personType.set(bob, "dateOfBirth", new Date());
personType.set(bob, "address", new Address("King's Road","London","404"));

// Insert the fact into a KIE session:
KieSession ksession = ...
ksession.insert(bob);
ksession.fireAllRules();

// Read attributes:
String name = (String) personType.get(bob, "name");
Date date = (Date) personType.get(bob, "dateOfBirth");
```

API には、一度にすべての属性を設定したり、**Map** コレクションから値を読み取ったり、すべての属性を一度に **Map** コレクションに読み込んだりするなど、他の便利なメソッドも含まれます。

API の動作は Java リフレクションと似ていますが、API はリフレクションを使用せず、生成されたバイトコードで実装され性能が良いアクセサーに依存します。

16.6. DRL のグローバル変数

DRL ファイルのグローバル変数は通常、ルールの結果で使用するアプリケーションサービスなど、ルールのデータやサービスを提供し、ルールの結果で追加されるログや値など、ルールからのデータを返します。KIE セッション設定や REST 操作を使用したデシジョンエンジンのワーキングメモリーにグロー

バル値を設定し、DRL ファイルのルールの上にグローバル変数を宣言してから、ルールのアクション部分 (**then**) でこれを使用します。グローバル変数が複数ある場合は、DRL ファイルで行を分けて使用してください。

以下は、DRL ファイルにおけるデシジョンエンジンのグローバル変数一覧の設定と、対応するグローバル変数定義の例です。

デシジョンエンジンに対するグローバルリストの設定例

```
List<String> list = new ArrayList<>();
KieSession kieSession = kiebase.newKieSession();
kieSession.setGlobal( "myGlobalList", list );
```

ルールを使用したグローバル変数定義の例

```
global java.util.List myGlobalList;

rule "Using a global"
  when
    // Empty
  then
    myGlobalList.add( "My global list" );
end
```

警告

グローバル変数に定数イミュータブル値がない場合は、ルールの条件設定にグローバル変数を使用しないでください。グローバル変数はデシジョンエンジンのワーキングメモリーに挿入されないため、デシジョンエンジンでは変数の値の変更を追跡できません。

グローバル変数を使用してルール間でデータを共有しないでください。ルールは常に、ワーキングメモリーの状態に関して推論し、これに対応するため、ルールからルールにデータを渡す必要がある場合は、データをファクトとしてデシジョンエンジンのワーキングメモリーにアサートしてください。

グローバル変数のユースケースとして、メールサービスの例があります。デシジョンエンジンを呼び出す統合コードで、**emailService** オブジェクトを取得してから、デシジョンエンジンのワーキングメモリーでそのオブジェクトを設定します。DRL ファイルで、**emailService** のグローバルタイプがあり、名前が **"email"** であることを宣言すると、ルールの結果で **email.sendSMS(number, message)** などのアクションを使用できるようになります。

複数のパッケージで同じ ID のグローバル変数を宣言した場合には、すべてのパッケージを同じタイプで設定し、すべてが同じグローバル値を参照するようにする必要があります。

16.7. DRL でのルール属性

ルール属性は、ルールの動作を修正するビジネスルールを指定する追加設定です。DRL ファイルでは、ルール属性は、以下の形式で、通常ルールの条件とアクションの上に定義します。複数の属性がある場合には、別々の行に指定します。

```
rule "rule_name"
    // Attribute
    // Attribute
    when
        // Conditions
    then
        // Actions
end
```

次の表では、ルールに割り当て可能な属性の名前と、対応する値を紹介します。

表16.1 ルールの属性

属性	値
salience	ルールの優先順位を定義する整数。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。 例: **salience 10**
enabled	ブール値。このオプションを選択すると、ルールが有効になります。このオプションを選択しないと、ルールは無効になります。 例: **enabled true**
date-effective	日付定義および時間定義を含む文字列。現在の日時が **date-effective** 属性よりも後の場合は、このルールがアクティブになります。 例: **date-effective "4-Sep-2018"**
date-expires	日付定義および時間定義を含む文字列。現在日時が **date-expires** 属性よりも後になると、このルールをアクティブにすることはできません。 例: **date-expires "4-Oct-2018"**
no-loop	ブール値。このオプションが選択される場合は、ルールの結果が以前一致した条件を再度トリガーすると、ルールは再アクティブ化 (ループ) されません。条件を選択しないと、この状況でルールがループされます。 例: **no-loop true**
agenda-group	ルールを割り当てるアジェンダグループを指定する文字列。アジェンダグループを使用すると、アジェンダをパーティションで区切り、ルールのグループに対する実行をさらに制御できます。フォーカスを取得したアジェンダグループのルールだけがアクティブになります。 例: **agenda-group "GroupName"**

属性	値
activation-group	ルールを割り当てるアクティベーション (または XOR) グループを指定する文字列。アクティベーショングループでは、1 つのルールのみをアクティブ化できます。最初のルールが実行すると、アクティベーショングループの中で、アクティベーションが保留中のルールはすべてキャンセルされます。 例: **activation-group "GroupName"**
duration	ルールの条件が一致している場合に、ルールがアクティブになってからの時間をミリ秒で定義する長整数値。 例: **duration 10000**
timer	ルールのスケジュールに対する **int** (間隔) または **cron** タイマー定義を指定する文字列。 例: **timer (cron:* 0/15 * * * ?)** (15 分ごと)
calendar	ルールのスケジュールを指定する Quartz カレンダーの定義。 例: **calendars "* * 0-7,18-23 ?* *"** (営業時間外を除く)
auto-focus	アジェンダグループ内のルールにのみ適用可能なブール値。このオプションが選択されている場合は、次にルールがアクティブになった場合に、そのルールが割り当てられたアジェンダグループにフォーカスが自動的に指定されます。 例: **auto-focus true**
lock-on-active	ルールフローグループまたはアジェンダグループ内のルールにのみ適用可能なブール値。このオプションを選択すると、次回、ルールのルールフローグループがアクティブになるか、ルールのアジェンダグループがフォーカスを受け取ると、(ルールフローグループがアクティブでなくなるか、アジェンダグループがフォーカスを失うまで) ルールをアクティブにすることができません。これは、**no-loop** 属性を強力にしたものです。なぜなら、一致するルールのアクティベーションが、(ルールそのものによるものだけでなく) 更新元にかかわらず破棄されるためです。この属性は、ファクトを修正するルールが多数あり、ルールの再一致と再発行を希望しない計算ルールに適しています。 例: **lock-on-active true**
ruleflow-group	ルールフローグループを指定する文字列。ルールフローグループで、関連するルールフローによってそのグループがアクティブになった場合に限りルールを発行できます。 例: **ruleflow-group "GroupName"**

属性	値
dialect	ルールのコード表記に使用される言語を指定する文字列 (**JAVA** または **MVEL**)。デフォルトでは、ルールは、パッケージレベルに指定されている方言を使用します。ここで指定した方言は、ルールのパッケージ方言設定を上書きします。 例: **dialect "JAVA"**  **注記** 実行可能モデルなしで Red Hat Process Automation Manager を使用する場合には、**dialect "JAVA"** ルールの結果は、Java 5 構文のみをサポートします。実行可能モデルに関する詳細は、Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ を参照してください。

16.7.1. DRL でのタイマーおよびカレンダールール属性

タイマーおよびカレンダーは、DRL ルール属性であり、これを使用してスケジュールと時間の制約を DRL ルールに適用できます。これらの属性を使用するには、ユースケースによって追加の設定が必要になります。

DRL ルールの **timer** 属性は、ルールのスケジュール設定を行うための **int** (間隔) または **cron** タイマー定義を指定する文字列で、以下の形式をサポートします。

タイマー属性の形式

```
timer ( int: <initial delay> <repeat interval> )

timer ( cron: <cron expression> )
```

間隔タイマー属性の例

```
// Run after a 30-second delay
timer ( int: 30s )

// Run every 5 minutes after a 30-second delay each time
timer ( int: 30s 5m )
```

cron タイマー属性の例

```
// Run every 15 minutes
timer ( cron:* 0/15 * * * ? )
```

間隔タイマーは、最初に遅延があり、オプションで間隔が繰り返されるという **java.util.Timer** オブジェクトのセマンティクスに従います。Cron タイマーは標準の Unix cron 式に準拠します。

以下の DRL ルール例では、cron タイマーを使用して 15 分ごとに SMS テキストメッセージを送信します。

cron タイマーを使用した DRL ルールの例

```
rule "Send SMS message every 15 minutes"
  timer ( cron:* 0/15 * * * ? )
  when
    $a : Alarm( on == true )
  then
    channels[ "sms" ].insert( new Sms( $a.mobileNumber, "The alarm is still on." );
end
```

一般的に、タイマーが制御するルールは、ルールがトリガーされた時点でアクティブになり、タイマーの設定に合わせてルールの結果が繰り返し実行されます。実行は、ルールの条件が受信ファクトに一致しなくなる時点で停止します。ただし、デシジョンエンジンがタイマー付きのルールを処理する方法は、デシジョンエンジンが **アクティブモード** か、**パッシブモード** かによって異なります。

デフォルトでは、ユーザーまたはアプリケーションが明示的に **fireAllRules()** を呼び出した場合に、デシジョンエンジンは **パッシブモード** で実行され、定義されたタイマー設定によりルールが評価されます。一方、ユーザーまたはアプリケーションが **fireUntilHalt()** を呼び出す場合、デシジョンエンジンは **アクティブモード** でで実行され、ユーザーまたはアプリケーションが **halt()** を明示的に呼び出すまでルールを継続的に評価します。

デシジョンエンジンがアクティブモードの場合は、**fireUntilHalt()** への呼び出しからコントロールが戻った後でもルールの結果が実行され、デシジョンエンジンは、ワーキングメモリーに加えられた変更に対して **リアクティブ** のままになります。たとえば、タイマールールの実行のトリガーに関連したファクトを削除すると、反復実行が停止になり、ファクトが挿入されるので、ルールが一致するとそのルールが実行します。ただし、デシジョンエンジンは継続して **アクティブ** な訳ではなく、ルールの実行後にだけアクティブになります。そのため、タイマーで制御されるルールが次回に実行されるまで、デシジョンエンジンは非同期のファクト挿入には反応しません。KIE セッションを破棄するとタイマーアクティビティーはすべて中断されます。

デシジョンエンジンがパッシブモードの場合は、タイマー付きのルールの結果は、**fireAllRules()** が再度呼び出される場合にのみ評価されます。ただし、以下の例にあるように、パッシブモードでは **TimedRuleExecutionOption** オプションで KIE セッションを設定することで、デフォルトのタイマー実行動作を変更できます。

パッシブモードでタイマー付きルールを自動的に実行するための KIE セッションの設定

```
KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();
ksconf.setOption( TimedRuleExecutionOption.YES );
KSession ksession = kbase.newKieSession(ksconf, null);
```

以下の例のように、**TimedRuleExecutionOption** オプションに追加で **FILTERED** 仕様を設定することで、対象のルールをフィルターするコールバックを定義できるようになります。

どのタイマー付きのルールを自動的に実行するかをフィルターするための KIE セッション

```
KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();
conf.setOption( new TimedRuleExecutionOption.FILTERED(new TimedRuleExecutionFilter() {
    public boolean accept(Rule[] rules) {
```

```
        return rules[0].getName().equals("MyRule");
    }
}) );
```

間隔タイマーの場合は、**int** ではなく、**expr** を指定した式タイマーを使用して、固定値の代わりに、式として遅延と間隔の両方を定義できます。

以下の DRL ファイルの例では、遅延と期間でファクトタイプを宣言し、後続のルールの式タイマーでこの遅延と期間を使用します。

式タイマーが含まれるルールの例

```
declare Bean
  delay   : String = "30s"
  period  : long = 60000
end

rule "Expression timer"
  timer ( expr: $d, $p )
  when
    Bean( $d : delay, $p : period )
  then
    // Actions
end
```

この例の **$d** と **$p** などの式は、ルールのパターンが一致する部分に定義した変数を使用できます。この変数には **String** の値を使用でき、これは期間や、期間の **long** 値 (ミリ秒単位) に内部で変換される数値に解析できます。

間隔と式のタイマーはいずれも、以下のオプションのパラメーターを使用できます。

- **start** および **end**: **Date**、または **Date** または **long** 値を表す **String**。この値には、**Number** も指定でき、数字は **new Date(((Number) n).longValue())** 形式の Java の **Date** に変換されます。
- **repeat-limit**: タイマーが許可する最大反復回数を定義する整数。**end** および **repeat-limit** の両パラメーターが設定されている場合は、2 つのどちらかに先に到達した時点でタイマーが停止します。

任意の start、end、および repeat-limit パラメーターが使用されるタイマー属性の例

```
timer (int: 30s 1h; start=3-JAN-2020, end=4-JAN-2020, repeat-limit=50)
```

この例では、ルールは 1 時間ごとに 30 秒の遅延の後に実行されるようにスケジュールされ、2020 年 1 月 3 日に開始し、2020 年 1 月 4 日またはサイクルが 50 回繰り返された後に終了するようにスケジュールされています。

システムが一時停止すると (セッションがシリアライズされた後にデシリアライズされる場合など)、ルールは、一時停止中に実施されなかったアクティベーションの数にかかわらず、実施されなかったアクティベーションからの回復を 1 回のみ実施するようにスケジュールされており、ルールはその後にタイマー設定と同期して再度スケジュールされます。

DRL ルールの **calendar** 属性は、ルールのスケジュールのための Quartz カレンダー定義であり、以下の形式をサポートします。

カレンダー属性の形式

```
calendars "<definition or registered name>"
```

カレンダー属性の例

```
// Exclude non-business hours
calendars "* * 0-7,18-23 ? * *"

// Weekdays only, as registered in the KIE session
calendars "weekday"
```

以下の例ように、Quartz カレンダー API に基づいて Quartz カレンダーを適用し、そのカレンダーを KIE セッションに登録することができます。

Quartz カレンダーの適用

```
Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)
```

KIE セッションでのカレンダーの登録

```
ksession.getCalendars().set( "weekday", weekDayCal );
```

カレンダーは、標準のルールや、タイマーを使用するルールと共に使用できます。カレンダー属性には、**String** リテラルとして記述された 1 つ以上のコンマ区切りのカレンダー名を含めることができます。

以下のルールの例では、カレンダーとタイマーの両方を使用してルールをスケジュールします。

カレンダーとタイマーを使用したルールの例

```
rule "Weekdays are high priority"
  calendars "weekday"
  timer ( int:0 1h )
  when
    Alarm()
  then
    send( "priority high - we have an alarm" );
end

rule "Weekends are low priority"
  calendars "weekend"
  timer ( int:0 4h )
  when
    Alarm()
  then
    send( "priority low - we have an alarm" );
end
```

16.8. DRL のルール条件 (WHEN)

DRL ルールの **when** 部分 (ルールの**左辺 (LHS)**とも言う) には、アクションを実行するのに満たされな

ければならない条件が含まれます。条件は、パッケージ内で使用可能なデータオブジェクトに基づいて、指定された一連の **パターン** および **制約**、オプションの **バインディング** およびサポートされるルール条件要素 (キーワード) で設定されます。たとえば、銀行のローンの申し込みに年齢制限 (21 歳以上) が必要な場合、**"Underage"** ルールの **when** 条件は **Applicant(age < 21)** となります。

注記

DRL は **if** ではなく **when** を使用します。これは、**if** が一般に手続き型の実行フローの一部であり、条件が特定の時点でチェックされるためです。一方、**when** は、条件の評価が特定の評価シーケンスや時点に限定されず、いつでも継続的に行われることを示しています。条件が満たされるたびに、これらのアクションが実施されます。

when セクションを空にすると、条件は true であると見なされ、デシジョンエンジンで **fireAllRules()** 呼び出しが最初に実施された場合に、**then** セクションのアクションが実行されます。これは、デシジョンエンジンのステートを設定するルールを使用する場合に便利です。

以下のルール例では、空の条件を使用して、ルールが実行するたびにファクトを挿入します。

条件のないルール例

```
rule "Always insert applicant"
  when
    // Empty
  then   // Actions to be executed once
    insert( new Applicant() );
end

// The rule is internally rewritten in the following way:

rule "Always insert applicant"
  when
    eval( true )
  then
    insert( new Applicant() );
end
```

ルールの条件が、定義されたキーワード接続詞 (**and**、**or**、または **not** など) なしで複数のパターンを使用する場合、デフォルトの接続詞は **and** になります。

キーワード接続詞なしのルールの例

```
rule "Underage"
  when
    application : LoanApplication()
    Applicant( age < 21 )
  then
    // Actions
end

// The rule is internally rewritten in the following way:

rule "Underage"
  when
    application : LoanApplication()
    and Applicant( age < 21 )
```

```
  then
    // Actions
end
```

16.8.1. パターンと制約

DRL ルール条件の **パターン** は、デシジョンエンジンによって照合されるセグメントです。パターンは、デシジョンエンジンのワーキングメモリーに挿入される各ファクトと一致する可能性があります。パターンには **制約** を含めることもでき、これにより一致するファクトをより詳細に定義できます。

制約のない最も単純なフォームでは、パターンは指定されたタイプのファクトに一致します。以下の例ではタイプが **Person** であるため、このパターンはデシジョンエンジンのワーキングメモリーのすべての **Person** オブジェクトに一致します。

ファクトタイプが 1 つの場合のパターン例

```
Person()
```

このタイプは、ファクトオブジェクトの実際のクラスである必要はありません。パターンは、複数の異なるクラスのファクトと一致する可能性のあるスーパーユーザーやインターフェイスも参照できます。たとえば、以下のパターンは、デシジョンエンジンのワーキングメモリーにあるすべてのオブジェクトと一致します。

すべてのオブジェクトの場合のパターン例

```
Object() // Matches all objects in the working memory
```

パターンの括弧は制約を囲みます (以下のユーザーの年齢に関する制約など)。

制約のあるパターンの例

```
Person( age == 50 )
```

制約は、**true** または **false** を返す式です。DRL 内のパターンの制約は、基本的にはプロパティーのアクセスなどの拡張が設定された Java の式ですが、**==** および **!=** に対する **equals()** および **!equals()** セマンティクスなど、若干の違いがあります (通常の **same** および **not same** セマンティクスではありません)。

JavaBean プロパティーはパターンの制約から直接アクセスできます。Bean プロパティーは、引数を使用せずに何かを返す標準の JavaBeans の getter を使用して内部的に公開されます。たとえば、**age** プロパティーは、DRL で getter の **getAge()** ではなく、**age** として記述されます。

JavaBeans プロパティーを使用した DRL 制約構文

```
Person( age == 50 )

// This is the same as the following getter format:

Person( getAge() == 50 )
```

Red Hat Process Automation Manager は標準の JDK **leavingspector** クラスを使用してこのマッピングを行うため、標準の JavaBeans 仕様に従います。デシジョンエンジンのパフォーマンスの最適化には、**getAge()** のように getter を明示的に使用するのではなく、 **age** のようなプロパティーアクセスの

形式を使用します。

警告

デシジョンエンジンは効率化のために呼び出し間で一致した結果をキャッシュするため、プロパティーアクセサーを使用してルールに影響を与える可能性がある方法でオブジェクトの状態を変更しないでください。

たとえば、プロパティーアクセサーを以下のように使用しないでください。

```
public int getAge() {
    age++; // Do not do this.
    return age;
}
```

```
public int getAge() {
    Date now = DateUtil.now(); // Do not do this.
    return DateUtil.differenceInYears(now, birthday);
}
```

2 番目の例に従う代わりに、ワーキングメモリーに現在の日付をラップするファクトを挿入し、必要に応じてそのファクトを **fireAllRules()** の間で更新します。

ただし、プロパティーの getter が見つからなかった場合、コンパイラーは、以下のようにこのプロパティー名をフォールバックメソッド名として引数なしで使用します。

オブジェクトが見つからない場合のフォールバックメソッド

```
Person( age == 50 )

// If `Person.getAge()` does not exist, the compiler uses the following syntax:

Person( age() == 50 )
```

以下の例のように、パターンでアクセスプロパティーをネストすることもできます。ネストされたプロパティーにはデシジョンエンジンでインデックス化されます。

ネストされたプロパティーアクセスを使用するパターンの例

```
Person( address.houseNumber == 50 )

// This is the same as the following format:

Person( getAddress().getHouseNumber() == 50 )
```

警告

ステートフルな KIE セッションでは、ネストされたアクセサーの使用に注意が必要です。デシジョンエンジンのワーキングメモリーではネストされた値は認識されず、これらの値の変更は検出されません。ネストされた値の親参照がワーキングメモリーに挿入されている場合はこれらの値を不変と見なすか、ネストされた値を変更する必要がある場合は、すべての外部ファクトを更新済みとしてマークします。前の例では、**houseNumber** プロパティーが変更された場合は、この **Address** が指定された **Person** は更新済みとしてマークされる必要があります。

パターンの括弧内では **boolean** 値を制約として返す任意の Java 式を使用できます。Java 式は、プロパティーアクセスなどの他の式の拡張機能と組み合わせることができます。

プロパティーアクセスと Java 式を使用する制約が設定されたパターンの例

```
Person( age == 50 )
```

評価の優先度は、論理式や数式のように括弧を使用して変更できます。

制約の評価順序の例

```
Person( age > 100 && ( age % 10 == 0 ) )
```

以下の例のように、制約で Java メソッドを再利用することもできます。

再利用される Java メソッドによる制約の例

```
Person( Math.round( weight / ( height * height ) ) < 25.0 )
```

警告

デシジョンエンジンは効率化を図るために呼び出し間で一致の結果をキャッシュするため、ルールに影響を与える可能性のある方法でオブジェクトの状態を変更するために制約を使用しないでください。ルール条件のファクトで実行されるメソッドは、読み取り専用のメソッドである必要があります。また、ファクトの状態は、ファクトがワーキングメモリーで更新済みとしてマークされているのでない限り、毎回変更されるたびにルールの呼び出し間で変更されません。

たとえば、以下のような方法でパターンの制約を使用しないでください。

```
Person( incrementAndGetAge() == 10 ) // Do not do this.
```

```
Person( System.currentTimeMillis() % 1000 == 0 ) // Do not do this.
```

DRL 内の制約演算子には、標準の Java 演算子の優先順位が適用されます。DRL 演算子は、**==** および **!=** 演算子を除き、標準の Java セマンティクスに従います。

== 演算子は、通常の **same** セマンティクスではなく、null 安全な **equals()** セマンティクスを使用します。たとえば、**Person(firstName == "John")** というパターンは **java.util.Objects.equals(person.getFirstName(), "John")** と同様であり、**"John"** は null でないため、このパターンは **"John".equals(person.getFirstName())** にも似ています。

!= 演算子は、通常の **not same** セマンティクスではなく null 安全な **!equals()** セマンティクスを使用します。たとえば、**Person(firstName != "John")** というパターンは、**!java.util.Objects.equals(person.getFirstName(), "John")** に似ています。

フィールドと制約の値が異なるタイプの場合、デシジョンエンジンは型強制 (type coercion) を使用して競合を解決し、コンパイルエラーを減らします。たとえば、**"ten"** が数値エバリュエーターで文字列として指定される場合は、コンパイルエラーが発生しますが、**"10"** は数値 10 に型強制されます。型強制では、フィールドのタイプは常に値のタイプより優先されます。

型強制された値を使用する制約の例

```
Person( age == "10" ) // "10" is coerced to 10
```

制約のグループの場合は、コンマ区切り (**,**) を使って、暗黙的な **and** の接続的なセマンティクスを使用することができます。

複数の制約があるパターンの例

```
// Person is at least 50 years old and weighs at least 80 kilograms:
Person( age > 50, weight > 80 )

// Person is at least 50 years old, weighs at least 80 kilograms, and is taller than 2 meters:
Person( age > 50, weight > 80, height > 2 )
```

注記

&& 演算子および **,** 演算子のセマンティクスは同じですが、これらは異なる優先順位で解決されます。**&&** 演算子は || 演算子より優先され、**&&** 演算子および || 演算子はどちらも **,** 演算子より優先されます。デシジョンエンジンのパフォーマンスと人による可読性を最適化するために、コンマ演算子は最上位レベルの制約で使用してください。

括弧内など、複合制約式にコンマ演算子を埋め込むことはできません。

複合制約式での不適切なコンマの例

```
// Do not use the following format:
Person( ( age > 50, weight > 80 ) || height > 2 )

// Use the following format instead:
Person( ( age > 50 && weight > 80 ) || height > 2 )
```

16.8.2. パターンと制約でバインドされた変数

パターンおよび制約に変数をバインドして、ルールの他の部分で一致するオブジェクトを参照することができます。バインドされた変数は、ルールをより効率的に、かつデータモデルでのファクトへのアノ

テーションの付け方と一貫した方法で定義するのに役立ちます。(とくに複雑なルールの場合に) ルール内で変数とフィールドを簡単に区別するには、変数に対して標準の形式である **$variable** を使用します。この規則は便利ですが、DRL で必須ではありません。

たとえば、以下の DRL ルールでは、**Person** ファクトが指定されたパターンに対して変数 **$p** が使用されています。

バインドされた変数が使用されているパターン

```
rule "simple rule"
  when
    $p : Person()
  then
    System.out.println( "Person " + $p );
end
```

同様に、以下の例のように、パターンの制約で変数をプロパティーにバインドすることもできます。

```
// Two persons of the same age:
Person( $firstAge : age ) // Binding
Person( age == $firstAge ) // Constraint expression
```

注記

より明確で効率的なルールを定義するには、制約のバインディングと制約式を必ず分離します。バインディングと式の組み合わせはサポートされますが、パターンが複雑になり、評価の効率に影響が及ぶ可能性があります。

```
// Do not use the following format:
Person( $age : age * 2 < 100 )

// Use the following format instead:
Person( age * 2 < 100, $age : age )
```

デシジョンエンジンは同じ宣言に対するバインディングをサポートしませんが、複数のプロパティー間での引数の **ユニフィケーション** をサポートします。位置引数は、常にユニフィケーションで常に処理され、名前付き引数の場合はユニフィケーション記号 **:=** が使用されます。

以下のパターンの例では、2 つの **Person** ファクト間で **age** プロパティーを統合します。

ユニフィケーションが使用されるパターンの例

```
Person( $age := age )
Person( $age := age )
```

ユニフィケーションは、シーケンスオカレンスのバインドされたフィールドの同じ値に対して、最初のオカレンスと制約のバインディングを宣言します。

16.8.3. ネストされた制約とインラインキャスト

以下の例のように、ネストされたオブジェクトの複数のプロパティーにアクセスしなければならない場合があります。

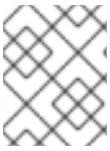
複数のプロパティーにアクセスするパターンの例

```
Person( name == "mark", address.city == "london", address.country == "uk" )
```

以下の例のように、これらのプロパティーのアクセサーを、**.(<constraints>)** という構文を使用してネストされたオブジェクトに対してグループ化することで、ルールを読みやすくすることができます。

グループ化された制約を使用したパターンの例

```
Person( name == "mark", address.( city == "london", country == "uk") )
```



注記

ピリオドのプリフィックス **.** は、メソッド呼び出しとネストされたオブジェクト制約を区別します。

パターンでネストされたオブジェクトを使用する場合は、構文 **<type>#<subtype>** を使用してサブタイプにキャストし、親タイプの getter をサブタイプに対して利用可能にします。以下の例のように、オブジェクト名または完全修飾クラス名のいずれかを使用して、1 つまたは複数のサブタイプにキャストできます。

サブタイプへのインラインキャストを使用したパターンの例

```
// Inline casting with subtype name:
Person( name == "mark", address#LongAddress.country == "uk" )

// Inline casting with fully qualified class name:
Person( name == "mark", address#org.domain.LongAddress.country == "uk" )

// Multiple inline casts:
Person( name == "mark", address#LongAddress.country#DetailedCountry.population > 10000000 )
```

これらのパターン例では、**Address** を **LongAddress** に、さらに最後の例にある **DetailedCountry** にキャストし、各ケースのサブタイプで親の getter を利用可能にします。

以下の例のように、**instanceof** 演算子を使用して、パターンを使用した後続のフィールドで指定されたタイプの結果を推測できます。

```
Person( name == "mark", address instanceof LongAddress, address.country == "uk" )
```

インラインキャストが使用できない場合 (たとえば **instanceof** が **false** を返す場合)、評価は **false** と見なされます。

16.8.4. 制約内の日付リテラル

デフォルトで、デシジョンエンジンは **dd-mmm-yyyy** という日付形式をサポートします。この日付形式 (必要に応じて時間形式マスクを含む) は、システムプロパティー **drools.dateformat="dd-mmm-yyyy hh:mm"** を使用して、別の形式マスクを指定することによってカスタマイズすることができます。日付形式は、**drools.defaultlanguage** および **drools.defaultcountry** システムプロパティーを使用し、言語ロケールを変更することによってカスタマイズすることもできます (たとえば、タイのロケールは **drools.defaultlanguage=th** および **drools.defaultcountry=TH** と設定します)。

日付のリテラル制限を使用したパターンの例

```
Person( bornBefore < "27-Oct-2009" )
```

16.8.5. DRL のパターン制約でサポートされている演算子

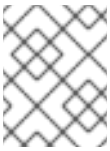
DRL では、パターン制約の演算子で標準の Java セマンティクスがサポートされていますが、いくつかの例外があり、追加となる DRL 固有の演算子もいくつかあります。以下の一覧は、標準の Java セマンティクスとは異なる方法で処理される DRL の制約の演算子や DRL の制約に固有の演算子をまとめています。

.(), #

.() 演算子を使用すると、プロパティーのアクセサーをネストされたオブジェクトにグループ化でき、**#** 演算子を使用すると、ネストされたオブジェクトのサブタイプにキャストできます。サブタイプにキャストすることで、親タイプの getter をサブタイプに対して使用できるようになります。オブジェクト名または完全修飾クラス名のいずれかを使用でき、1 つまたは複数のサブタイプにキャストできます。

ネストされたオブジェクトが使用されるパターンの例

```
// Ungrouped property accessors:
Person( name == "mark", address.city == "london", address.country == "uk" )

// Grouped property accessors:
Person( name == "mark", address.( city == "london", country == "uk") )
```



注記

ピリオドのプリフィックス **.** は、メソッド呼び出しとネストされたオブジェクト制約を区別します。

サブタイプへのインラインキャストを使用したパターンの例

```
// Inline casting with subtype name:
Person( name == "mark", address#LongAddress.country == "uk" )

// Inline casting with fully qualified class name:
Person( name == "mark", address#org.domain.LongAddress.country == "uk" )

// Multiple inline casts:
Person( name == "mark", address#LongAddress.country#DetailedCountry.population > 10000000
)
```

!.

この演算子を使用すると、null 安全な方法でプロパティーを逆参照します。パターンマッチングの適切な結果を得るには、**!.** 演算子の左側の値を null にすることはできません (**!= null** と解釈される)。

null 安全な逆参照を使用した制約の例

```
Person( $streetName : address!.street )

// This is internally rewritten in the following way:

Person( address != null, $streetName : address.street )
```

_

[]

この演算子を使用して、インデックスで **List** 値にアクセスするか、またはキーで **Map** 値にアクセスします。

List および Map アクセスを使用する制約の例

```
// The following format is the same as `childList(0).getAge() == 18`:
Person(childList[0].age == 18)

// The following format is the same as `credentialMap.get("jdoe").isValid()`:
Person(credentialMap["jdoe"].valid)
```

<、<=、>、>=

これらの演算子は、自然順序付けのあるプロパティーに使用されます。たとえば、**<** 演算子は、**Date** フィールドでは **前** を意味し、**String** フィールドでは **アルファベット順で前** であることを意味します。これらのプロパティーは、比較可能なプロパティーにのみ適用されます。

before 演算子を使用した制約の例

```
Person( birthDate < $otherBirthDate )

Person( firstName < $otherFirstName )
```

==, !=

制約では、これらの演算子を、通常の **same** および **not same** セマンティクスではなく、**equals()** および **!equals()** メソッドとして使用します。

null 安全な等価性を使用する制約の例

```
Person( firstName == "John" )

// This is similar to the following formats:

java.util.Objects.equals(person.getFirstName(), "John")
"John".equals(person.getFirstName())
```

null 安全な非等価性を使用する制約の例

```
Person( firstName != "John" )

// This is similar to the following format:

!java.util.Objects.equals(person.getFirstName(), "John")
```

&&, ||

これらの演算子を使用して、フィールドに複数の制約を追加する略記組合せ比較条件を作成します。再帰的な構文パターンを作成するには、括弧 **()** を使用して制約をグループ化します。

略記組合せ比較を使用する制約の例

```
// Simple abbreviated combined relation condition using a single `&&`:
Person(age > 30 && < 40)

// Complex abbreviated combined relation using groupings:
Person(age ((> 30 && < 40) || (> 20 && < 25)))

// Mixing abbreviated combined relation with constraint connectives:
Person(age > 30 && < 40 || location == "london")
```

matches, not matches

これらの演算子を使用して、指定された Java 正規表現にフィールドが一致するか、または一致しないかを示します。一般に、正規表現は **String** リテラルですが、有効な正規表現に解決される変数もサポートされます。これらの演算子は **String** プロパティーのみに適用されます。**null** 値に対して **matches** を使用する場合は、結果の評価が常に **false** になります。**null** 値に対して **not matches** を使用する場合は、結果の評価が常に **true** になります。Java の場合のように、**String** リテラルとして記述された正規表現は二重のバックスラッシュ \\ を使用してエスケープする必要があります。

正規表現と一致する制約または一致しない制約の例

```
Person( country matches "(USA)?\\S*UK" )

Person( country not matches "(USA)?\\S*UK" )
```

contains, not contains

これらの演算子を使用して、フィールドの **Array** または **Collection** が指定された値を含むか、または含まないかを検証します。これらの演算子は **Array** プロパティーまたは **Collection** プロパティーに適用されますが、これらの演算子を **String.contains()** および **!String.contains()** の制約チェックの代わりとして使用することもできます。

コレクションに対して contains および not contains が使用された制約の例

```
// Collection with a specified field:
FamilyTree( countries contains "UK" )

FamilyTree( countries not contains "UK" )


// Collection with a variable:
FamilyTree( countries contains $var )

FamilyTree( countries not contains $var )
```

String リテラルに対して contains および not contains が使用された制約の例

```
// Sting literal with a specified field:
Person( fullName contains "Jr" )

Person( fullName not contains "Jr" )


// String literal with a variable:
```

```
Person( fullName contains $var )

Person( fullName not contains $var )
```

注記

下位互換性を確保するため、**excludes** 演算子は **not contains** の同義語としてサポートされます。

memberOf, not memberOf

これらの演算子を使用して、フィールドが変数として定義されている **Array** または **Collection** のメンバーであるかどうかを検証します。**Array** または **Collection** は変数でなければなりません。

コレクションと memberOf および not memberOf を使用する制約の例

```
FamilyTree( person memberOf $europeanDescendants )

FamilyTree( person not memberOf $europeanDescendants )
```

soundslike

この演算子を使用して、ある単語を英語で発音した場合に、指定された値と発音がほぼ同じであるかどうかを検証します (**matches** 演算子に類似)。この演算子は Soundex アルゴリズムを使用します。

soundslike を使用した制約の例

```
// Match firstName "Jon" or "John":
Person( firstName soundslike "John" )
```

str

この演算子を使用して、**String** であるフィールドが指定された値で開始しているか、または終了しているかを検証します。この演算子を使用して、**String** の長さを検証することもできます。

str を使用する制約の例

```
// Verify what the String starts with:
Message( routingValue str[startsWith] "R1" )

// Verify what the String ends with:
Message( routingValue str[endsWith] "R2" )

// Verify the length of the String:
Message( routingValue str[length] 17 )
```

in、notin

これらの演算子を使用して、制約の中で一致する可能性がある複数の値を指定します (複合値の制約)。複合値の制約についての機能をサポートするのは **in** 演算子および **not in** 演算子のみです。これらの演算子の 2 番目のオペランドは、括弧で囲み、コンマ区切り値の一覧で指定する必要があります。値は変数、リテラル、戻り値、または修飾された識別子として指定できます。これらの演算子は、**==** または **!=** 演算子を使用し、複数の制約のリストとして内部に再書き込みされます。

in および notin を使用した制約の例

```
Person( $color : favoriteColor )
Color( type in ( "red", "blue", $color ) )

Person( $color : favoriteColor )
Color( type notin ( "red", "blue", $color ) )
```

16.8.6. DRL のパターン制約における演算子の優先順位

DRL では、適用可能な制約演算子の場合は標準的な Java 演算子の優先順位をサポートしていますが、一部の例外と、DRL に固有の追加の演算子がいくつかあります。以下の表には、適用可能な DRL 演算子を優先順位の高いものから低いものの順で記載しています。

表16.2 DRL のパターン制約における演算子の優先順位

演算子のタイプ	演算子	注記
ネストされているか、null 安全なプロパティーアクセス	**.**、**.()**、**!.**	標準の Java セマンティクスではない
List または **Map** アクセス	**[]**	標準の Java セマンティクスではない
制約のバインディング	**:**	標準の Java セマンティクスではない
乗法	***、/、%**	
加法	**+**、**-**	
シフト	**>>**、**>>>**、**<<**	
リレーショナル	**<**、**<=**、**>**、**>=**、**instanceof**	
等価性	**== !=**	標準の Java の **same** および **not same** セマンティクスではなく、**equals()** および **!equals()** を使用
非短絡 (Non-short-circuiting) **AND**	**&**	
非短絡の排他的 **OR**	**^**	
非短絡の包含的 **OR**	**|**	
論理 **AND**	**&&**	
論理 **OR**	**||**	

演算子のタイプ	演算子	注記
三項	**? :**	
コンマ区切り **AND**	**,**	標準の Java セマンティクスではない

16.8.7. DRL でサポートされるルール条件要素 (キーワード)

DRL では、DRL のルール条件で定義するパターンで使用できる以下のルール条件要素 (キーワード) がサポートされます。

and

条件コンポーネントを論理積に分類します。インフィックスおよびプリフィックスの **and** がサポートされます。括弧 **()** を使用することにより、パターンを明示的にグループ化できます。デフォルトでは、結合演算子を指定しないと、リストされているパターンがすべて **and** で結合されます。

and を使用したパターンの例

```
//Infix `and`:
Color( colorType : type ) and Person( favoriteColor == colorType )

//Infix `and` with grouping:
(Color( colorType : type ) and (Person( favoriteColor == colorType ) or Person( favoriteColor ==
colorType ))

// Prefix `and`:
(and Color( colorType : type ) Person( favoriteColor == colorType ))

// Default implicit `and`:
Color( colorType : type )
Person( favoriteColor == colorType )
```

注記

先頭の宣言のバインディングには **and** キーワードを使用しないでください (**or** などは使用できます)。宣言が参照できるのは一度に 1 つのファクトのみであり、**and** と宣言のバインディングを使用すると、**and** が満たされた場合にこの要素が両方のファクトと一致してしまうため、エラーが発生します。

and の誤った使用例

```
// Causes compile error:
$person : (Person( name == "Romeo" ) and Person( name == "Juliet"))
```

または

条件コンポーネントを論理和にグループ化します。インフィックスおよびプリフィックスの **or** がサポートされます。括弧 **()** を使用することにより、パターンを明示的にグループ化できます。**or** と共にパターンバインディングを使用することもできますが、各パターンは個別にバインディングする必要があります。

or を使用したパターンの例

```
//Infix `or`:
Color( colorType : type ) or Person( favoriteColor == colorType )

//Infix `or` with grouping:
(Color( colorType : type ) or (Person( favoriteColor == colorType ) and Person( favoriteColor ==
colorType ))

// Prefix `or`:
(or Color( colorType : type ) Person( favoriteColor == colorType ))
```

or とパターンのバインディングを使用したパターンの例

```
pensioner : (Person( sex == "f", age > 60 ) or Person( sex == "m", age > 65 ))

(or pensioner : Person( sex == "f", age > 60 )
    pensioner : Person( sex == "m", age > 65 ))
```

or 条件要素の動作は、制約や、フィールド制約の制限を対象とした接続演算子 (||) とは異なります。デシジョンエンジンは **or** 要素を直接解釈しませんが、論理変換を使用して、**or** が使用されているルールを複数のサブルールに書き換えます。このプロセスにより、最終的には、ルートノードおよび各条件要素のサブルールとして、1 つの **or** を使用するルールが生成されます。サブルール間での操作や、特別な動作なしに、各サブルールは通常のルールと同様に有効にされ、実行されます。

したがって、**or** 条件要素については複数の類似したルールを生成するための近道であり、複数の論理和が true である場合は、複数のアクティベーションが作成される可能性があることに留意してください。

exists

存在している必要のあるファクトおよび制約を指定します。このオプションは、最初に一致したものだけが適用され、その後一致するものは無視されます。この要素を複数のパターンで使用する場合は、これらのパターンを括弧 **()** で囲みます。

exists を使用したパターンの例

```
exists Person( firstName == "John")

exists (Person( firstName == "John", age == 42 ))

exists (Person( firstName == "John" ) and
        Person( lastName == "Doe" ))
```

not

存在していてはならないファクトと制約を指定します。この要素を複数のパターンで使用する場合は、これらのパターンを括弧 **()** で囲みます。

not を使用したパターンの例

```
not Person( firstName == "John")

not (Person( firstName == "John", age == 42 ))
```

```
not (Person( firstName == "John" ) and
    Person( lastName == "Doe" ))
```

forall

最初のパターンと一致するすべてのファクトが残りのパターンのすべてと一致するかどうかを検証します。**forall** 設定が満たされると、このルールが **true** と評価されます。この要素は範囲の区切りであるため、以前にバインドされたすべての変数を使用できますが、内部でバインドされた変数を外部で使用することはできません。

forall を使用したルールの例

```
rule "All full-time employees have red ID badges"
  when
    forall( $emp : Employee( type == "fulltime" )
                Employee( this == $emp, badgeColor = "red" ) )
  then
    // True, all full-time employees have red ID badges.
end
```

この例では、ルールによってタイプが **"fulltime"** であるすべての **Employee** オブジェクトが選択されます。このパターンに一致するそれぞれのファクトに対して、ルールは、従うパターン (バッジの色) を評価し、一致すると、ルールは **true** と評価されます。

デシジョンエンジンのワーキングメモリー内の特定のタイプのすべてのファクトが一連の制約と一致する必要があることを示すために、単純化するために単一のパターンで **forall** を使用できます。

forall と 1 つのパターンを使用するルールの例

```
rule "All full-time employees have red ID badges"
  when
    forall( Employee( badgeColor = "red" ) )
  then
    // True, all full-time employees have red ID badges.
end
```

複数のパターンと **forall** 設定を使用するか、**not** 要素設定内など他の条件要素でネスト化することができます。

forall と複数のパターンを使用するルールの例

```
rule "All employees have health and dental care programs"
  when
    forall( $emp : Employee()
            HealthCare( employee == $emp )
            DentalCare( employee == $emp )
          )
  then
    // True, all employees have health and dental care.
end
```

forall と not を使用するルールの例

```
rule "Not all employees have health and dental care"
  when
    not ( forall( $emp : Employee()
                  HealthCare( employee == $emp )
                  DentalCare( employee == $emp ) )
        )
  then
    // True, not all employees have health and dental care.
end
```

注記

forall(p1 p2 p3 ...) の形式は **not(p1 and not(and p2 p3 ...))** と等価です。

from

これを使用してパターンのデータソースを指定します。これにより、デシジョンエンジンがワーキングメモリーにないデータに対して推論できるようになります。データソースには、バインドされた変数のサブフィールド、またはメソッド呼び出しの結果を指定できます。オブジェクトソースの定義に使用される式として、通常の MVEL 構文に準拠する任意の式を使用できます。このため、**from** 要素により、オブジェクトプロパティーのナビゲーションを使用して、メソッド呼び出しを実行し、マップとコレクション要素にアクセスすることが簡単にできます。

from およびパターンのバインディングを使用するルールの例

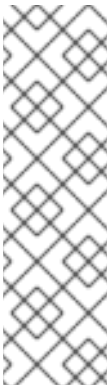
```
rule "Validate zipcode"
  when
    Person( $personAddress : address )
    Address( zipcode == "23920W" ) from $personAddress
  then
    // Zip code is okay.
end
```

from とグラフ表記を使用するルールの例

```
rule "Validate zipcode"
  when
    $p : Person()
    $a : Address( zipcode == "23920W" ) from $p.address
  then
    // Zip code is okay.
end
```

すべてのオブジェクトに対して反復処理される from のルールの例

```
rule "Apply 10% discount to all items over US$ 100 in an order"
  when
    $order : Order()
    $item  : OrderItem( value > 100 ) from $order.items
  then
    // Apply discount to `$item`.
end
```



注記

オブジェクトの大規模なコレクションの場合は、デシジョンエンジンが頻繁に繰り返す必要がある大きなグラフを持つオブジェクトを追加するのではなく、次の例に示すように、コレクションを直接 KIE セッションに追加して、コレクションを条件に結合します。

```
when
  $order : Order()
  OrderItem( value > 100, order == $order )
```

from および lock-on-active ルール属性を使用するルールの例

```
rule "Assign people in North Carolina (NC) to sales region 1"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person()
    $a : Address( state == "NC" ) from $p.address
  then
    modify ($p) {} // Assign the person to sales region 1.
end

rule "Apply a discount to people in the city of Raleigh"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person()
    $a : Address( city == "Raleigh" ) from $p.address
  then
    modify ($p) {} // Apply discount to the person.
end
```

重要

from と **lock-on-active** のルール属性を同時に使用すると、ルールが実行しなくなります。この問題に対しては、以下のいずれかの方法で対処できます。

- すべてのファクトをデシジョンエンジンのワーキングメモリーに挿入したり、制約式でネストされたオブジェクト参照を使用したりする場合は、**from** 要素は使用しないでください。
- ルール条件の最後の文として、**modify()** ブロックで使用される変数を配置します。
- 同じルールフローグループ内のルールがアクティベーションを相互に組み込む方法を明示的に管理できる場合は、**lock-on-active** ルール属性を使用しないでください。

from 句を含むパターンの後に、括弧から始まる別のパターンを使用することはできません。この制限がある理由は、DRL パーサーが **from** 式を **"from $l (String() or Number())"** として読み取り、この式を関数呼び出しと区別できないためです。この最も単純な回避策は、以下の例に示すように、**from** 句を括弧でラップする方法です。

from が適切に使用されていないルールと適切に使用されているルールの例

```
// Do not use `from` in this way:
rule R
  when
    $l : List()
    String() from $l
    (String() or Number())
  then
    // Actions
end

// Use `from` in this way instead:
rule R
  when
    $l : List()
    (String() from $l)
    (String() or Number())
  then
    // Actions
end
```

entry-point

エントリーポイントまたはパターンのデータソースに対応した **イベントストリーム** を定義します。この要素は通常、**from** 条件要素と共に使用します。イベントのエントリーポイントを宣言し、デシジョンエンジンがそのエントリーポイントからのデータのみを使用してルールを評価することが可能です。エントリーポイントは、DRL ルールで参照することで暗黙的に宣言することも、Java アプリケーションで明示的に宣言することもできます。

from entry-point を使用したルールの例

```
rule "Authorize withdrawal"
  when
    WithdrawRequest( $ai : accountId, $am : amount ) from entry-point "ATM Stream"
    CheckingAccount( accountId == $ai, balance > $am )
  then
    // Authorize withdrawal.
end
```

EntryPoint オブジェクトが使用され、ファクトが挿入された Java アプリケーションコードの例

```
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.EntryPoint;

// Create your KIE base and KIE session as usual:
KieSession session = ...

// Create a reference to the entry point:
EntryPoint atmStream = session.getEntryPoint("ATM Stream");

// Start inserting your facts into the entry point:
atmStream.insert(aWithdrawRequest);
```

collect

ルールで条件の一部として使用できるオブジェクトのコレクションを定義します。このルールは、指定されたソースまたはデシジョンエンジンのワーキングメモリーのいずれかからコレクションを取得します。**collect** 要素の結果パターンには、**java.util.Collection** インターフェイスを実装し、デフォルトの引数を持たないパブリックコンストラクターを指定する任意の具象クラスを使用できます。**List**、**LinkedList**、および **HashSet** のような Java コレクションを使用することも、独自のクラスを使用することもできます。条件内で **collect** 要素の前に変数がバインドされている場合は、その変数を使用してソースおよび結果パターンの両方を制限することができます。ただし、**collect** 要素内で作成されるバインディングをその外部で使用することはできません。

collect を使用するルールの例

```
import java.util.List

rule "Raise priority when system has more than three pending alarms"
  when
    $system : System()
    $alarms : List( size >= 3 )
              from collect( Alarm( system == $system, status == 'pending' ) )
  then
    // Raise priority because `$system` has three or more `$alarms` pending.
end
```

この例では、ルールは指定された各システムのデシジョンエンジンのワーキングメモリーの保留中のすべてのアラームを評価し、それらを **List** にグループ化します。指定されたシステムについての 3 つ以上のアラームが見つかると、ルールが実行します。

以下の例のように、ネストされた **from** 要素と共に **collect** 要素を使用することもできます。

collect とネストされた from を使用するルールの例

```
import java.util.LinkedList;

rule "Send a message to all parents"
  when
    $town : Town( name == 'Paris' )
    $mothers : LinkedList()
               from collect( Person( children > 0 )
                             from $town.getPeople()
                           )
  then
    // Send a message to all parents.
end
```

accumulate

オブジェクトのコレクションを反復処理し、各要素に対してカスタムアクションを実行し、結果オブジェクトを返します (制約が **true** に評価される場合)。この要素は、**collect** 条件要素のより柔軟性が高い、強化された形式です。**accumulate** 条件で事前に定義した関数を使用するか、必要に応じてカスタム関数を実装できます。また、ルール条件で **accumulate** の短縮形である **acc** を使用することもできます。
以下の形式を使用して、ルールに **accumulate** 条件を定義します。

accumulate の推奨形式

-

```
accumulate( <source pattern>; <functions> [;<constraints>] )
```

注記

デシジョンエンジンは下位互換性を確保するために **accumulate** 要素の代替形式をサポートしますが、この形式は、ルールとアプリケーションの最適なパフォーマンスという点ではより適しています。

デシジョンエンジンは、以下の事前に定義された **accumulate** 関数をサポートします。これらの関数は、任意の式を入力として受け入れます。

- **average**
- **min**
- **max**
- **count**
- **sum**
- **collectList**
- **collectSet**

以下のルールの例では、**min**、**max**、および **average** は **accumulate** 関数であり、各センサーの読み取り値での最低、最高、および平均の温度値を算出します。

温度値を算出する accumulate を使用したルールの例

```
rule "Raise alarm"
  when
    $s : Sensor()
    accumulate( Reading( sensor == $s, $temp : temperature );
                $min : min( $temp ),
                $max : max( $temp ),
                $avg : average( $temp );
                $min < 20, $avg > 70 )
  then
    // Raise the alarm.
end
```

以下のルールの例では、**accumulate** を指定した **average** 関数を使用して、ある注文のすべてのアイテムの平均収益を計算します。

平均収益を計算する accumulate を使用したルールの例

```
rule "Average profit"
  when
    $order : Order()
    accumulate( OrderItem( order == $order, $cost : cost, $price : price );
                $avgProfit : average( 1 - $cost / $price ) )
  then
    // Average profit for `$order` is `$avgProfit`.
end
```

accumulate の条件でカスタムかつドメイン固有の関数を使用するには、**org.kie.api.runtime.rule.AccumulateFunction** インターフェイスを実装する Java クラスを作成します。たとえば、以下の Java クラスは **AverageData** 関数のカスタム実装を定義します。

average 関数がカスタムで実装された Java クラスの例

```
// An implementation of an accumulator capable of calculating average values

public class AverageAccumulateFunction implements
org.kie.api.runtime.rule.AccumulateFunction<AverageAccumulateFunction.AverageData> {

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {

    }

    public void writeExternal(ObjectOutput out) throws IOException {

    }

    public static class AverageData implements Externalizable {
        public int    count = 0;
        public double total = 0;

        public AverageData() {}

        public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
            count   = in.readInt();
            total   = in.readDouble();
        }

        public void writeExternal(ObjectOutput out) throws IOException {
            out.writeInt(count);
            out.writeDouble(total);
        }

    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#createContext()
     */
    public AverageData createContext() {
        return new AverageData();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#init(java.io.Serializable)
     */
    public void init(AverageData context) {
        context.count = 0;
        context.total = 0;
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#accumulate(java.io.Serializable,
java.lang.Object)
```

```
     */
    public void accumulate(AverageData context,
                           Object value) {
        context.count++;
        context.total += ((Number) value).doubleValue();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#reverse(java.io.Serializable,
java.lang.Object)
     */
    public void reverse(AverageData context, Object value) {
        context.count--;
        context.total -= ((Number) value).doubleValue();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#getResult(java.io.Serializable)
     */
    public Object getResult(AverageData context) {
        return new Double( context.count == 0 ? 0 : context.total / context.count );
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#supportsReverse()
     */
    public boolean supportsReverse() {
        return true;
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#getResultType()
     */
    public Class< ? > getResultType() {
        return Number.class;
    }

}
```

DRL ルールでカスタム関数を使用するため、**import accumulate** ステートメントを使用してその関数をインポートします。

カスタム関数をインポートするための形式

```
import accumulate <class_name> <function_name>
```

インポートされた average 関数を使用するルールの例

```
import accumulate AverageAccumulateFunction.AverageData average

rule "Average profit"
  when
    $order : Order()
    accumulate( OrderItem( order == $order, $cost : cost, $price : price );
                $avgProfit : average( 1 - $cost / $price ) )
```

```
  then
    // Average profit for `$order` is `$avgProfit`.
end
```

16.8.8. DRL ルール条件内でオブジェクトのグラフが使用される OOPath 構文

OOPath は、DRL ルールの条件の制約でオブジェクトのグラフを参照するために設計された XPath のオブジェクト指向構文の拡張です。OOPath は、コレクションおよびフィルター制約を処理する間に XPath からのコンパクト表記を使用して関連要素を移動します。また、OOPath は特にオブジェクトグラフの場合に役に立ちます。

ファクトのフィールドがコレクションである場合は、**from** 条件要素 (キーワード) を使用してバインドし、そのコレクションのすべての項目を 1 つずつ判断することができます。ルール条件制約でオブジェクトのグラフを参照する必要がある場合は、**from** 条件要素を過度に使用すると、以下の例のように冗長かつ繰り返しの多い構文になります。

from を使用してオブジェクトのグラフを参照するルールの例

```
rule "Find all grades for Big Data exam"
  when
    $student: Student( $plan: plan )
    $exam: Exam( course == "Big Data" ) from $plan.exams
    $grade: Grade() from $exam.grades
  then
    // Actions
end
```

この例では、ドメインモデルには **Student** オブジェクトと学習の **Plan** が含まれています。**Plan** には、ゼロ以上の **Exam** インスタンスを指定でき、**Exam** にはゼロ以上の **Grade** インスタンスを指定できます。このルール設定が機能するためにデシジョンエンジンのワーキングメモリーに存在する必要があるのは、グラフのルートオブジェクト (この例では **Student**) のみです。

from ステートメントを使用するより効率的な別の方法として、以下の例のように短縮された OOPath 構文を使用できます。

OOPath 構文でオブジェクトのグラフを参照するルールの例

```
rule "Find all grades for Big Data exam"
  when
    Student( $grade: /plan/exams[course == "Big Data"]/grades )
  then
    // Actions
end
```

通常、OOPath 式の中核となる文法は、以下のように拡張された Backus-Naur form (EBNF) 表記で定義されます。

OOPath 式の EBNF 表記

```
OOPExpr = [ID ( ":" | ":=" )] ( "/" | "?/" ) OOPSegment { ( "/" | "?/" | "." ) OOPSegment } ;
OOPSegment = ID ["#" ID] ["[" ( Number | Constraints ) "]"]
```

OOPath 式の実際の特性および機能は以下のとおりです。

- 非リアクティブな OOPath 式の場合は、スラッシュ / または疑問符とスラッシュ **?/** で始まります (このセクションで後述します)。
- オブジェクトの単一プロパティーを、ピリオド **.** 演算子を使用して逆参照できます。
- オブジェクトの複数のプロパティーをスラッシュ / 演算子を使用して逆参照できます。コレクションが返される場合、この式はコレクションの値を反復処理します。
- 1 つまたは複数の制約を満たさない、走査されたオブジェクトをフィルターで除外できます。この制約は、以下の例のように、角括弧内の述語式として記述されます。

  述語式としての制約

  ```
  Student( $grade: /plan/exams[ course == "Big Data" ]/grades )
  ```

- 汎用コレクションで宣言されたクラスのサブクラスに走査されたオブジェクトをダウンキャストできます。以下の例に示すように、後続の制約も、このサブクラスで宣言されたプロパティーにのみ安全にアクセスすることができます。このインラインキャストで指定されたクラスのインスタンスではないオブジェクトは、自動的にフィルターで除外されます。

  ダウンキャストオブジェクトが使用される制約

  ```
  Student( $grade: /plan/exams#AdvancedExam[ course == "Big Data", level > 3 ]/grades )
  ```

- 現在の反復処理されたグラフの前に走査されたグラフのオブジェクトを前方参照できます。たとえば、以下の OOPath 式は、合格した試験の平均を上回るグレードにのみ一致します。

  前方参照オブジェクトを使用する制約

  ```
  Student( $grade: /plan/exams/grades[ result > ../averageResult ] )
  ```

- 以下の例のように、別の OOPath 式を再帰的に使用できます。

  再帰的な制約式

  ```
  Student( $exam: /plan/exams[ /grades[ result > 20 ] ] )
  ```

- 以下の例のように、角括弧 **[]** 内のオブジェクトのインデックスを使用することにより、そのオブジェクトにアクセスすることができます。Java の規則に従うために、OOPath のインデックスは 0 をベースとし、XPath のインデックスは 1 をベースとします。

  インデックスによるオブジェクトへのアクセスが設定された制約

  ```
  Student( $grade: /plan/exams[0]/grades )
  ```

OOPath 式は、リアクティブまたは非リアクティブにできます。デシジョンエンジンは、深くネスト化されており、OOPath 式の評価中に走査されたオブジェクトを含む更新には反応しません。

これらのオブジェクトが変更に反応するようにするには、**org.drools.core.phreak.ReactiveObject** クラスを拡張するようにオブジェクトを変更します。オブジェクトを変更して **ReactiveObject** クラスを拡張すると、ドメインオブジェクトはいずれかのフィールドが更新されている場合に、以下のように継承されたメソッド **notifyModification** を呼び出して、デシジョンエンジンに通知します。

試験が別のコースに移動したことをデシジョンエンジンに通知するオブジェクトメソッドの例

```
public void setCourse(String course) {
        this.course = course;
        notifyModification(this);
        }
```

次の対応する OOPath 式を使すると、試験が別のコースに移動された場合にルールが再実行され、そのルールに一致するグレードのリストが再計算されます。

Big Data ルールの OOPath 式の例

```
Student( $grade: /plan/exams[ course == "Big Data" ]/grades )
```

以下の例に示すように、/ セパレーターの代わりに **?**/ セパレーターを使用して、OOPath 式の一部のみについてリアクティビティーを無効にすることもできます。

部分的に非リアクティブである OOPath 式の例

```
Student( $grade: /plan/exams[ course == "Big Data" ]?/grades )
```

この例では、デシジョンエンジンは試験に対して実施された変更や、計画に試験が追加された場合に反応しますが、既存の試験に新しいグレードが追加された場合には反応しません。

OOPath の一部が非リアクティブである場合は、OOPath 式の残りの部分も非リアクティブになります。たとえば、以下の OOPath 式は完全に非リアクティブです。

完全に非リアクティブである OOPath 式の例

```
Student( $grade: ?/plan/exams[ course == "Big Data" ]/grades )
```

こうした理由から、同じ OOPath 式内で **?**/ セパレーターを複数回使用することはできません。たとえば、以下の式はコンパイルエラーの原因となります。

重複した非リアクティビティーマーカーを使用する OOPath 式の例

```
Student( $grade: /plan?/exams[ course == "Big Data" ]?/grades )
```

OOPath 式のリアクティビティーを有効にするもう 1 つの方法は、Red Hat Process Automation Manager で **List** インターフェイスおよび **Set** インターフェイスの専用の実装を使用することです。これらの実装は、**ReactiveList** クラスおよび **ReactiveSet** クラスです。また、**ReactiveCollection** クラスも使用できます。これらの実装により、**Iterator** クラスおよび **ListIterator** クラスを使用した可変操作の実行もリアクティブにサポートされます。

以下のクラスの例では、これらのクラスを使用して OOPath 式のリアクティビティーを設定します。

OOPath 式のリアクティビティーを設定する Java クラスの例

```
public class School extends AbstractReactiveObject {
    private String name;
    private final List<Child> children = new ReactiveList<Child>(); (1)

    public void setName(String name) {
        this.name = name;
        notifyModification(); (2)
```

```
    }

    public void addChild(Child child) {
        children.add(child); 3
        // No need to call `notifyModification()` here
    }
}
```

1 標準の Java **List** インスタンスに対するリアクティブサポートのために、**ReactiveList** インスタンスを使用します。

2 フィールドがリアクティブなサポートで変更された場合は、必須の **notifyModification()** メソッドを使用します。

3 **children** フィールドは **ReactiveList** のインスタンスであるため、**notifyModification()** メソッド呼び出しは必要ありません。通知は、**children** フィールドで実行される他のすべての変更操作と同様に、自動的に処理されます。

16.9. DRL におけるルールアクション (THEN)

ルールの **then** 部分 (または、ルールの **右辺 (RHS)**) には、ルールの条件部分が満たされる場合に実行されるアクションが含まれます。アクションは、1 つ以上の **メソッド** で設定されます。これらのメソッドは、ルール条件とパッケージ内で使用できる利用可能なデータオブジェクトに応じて結果を実行します。たとえば、銀行がローン申請者が 21 歳を超えていることが要件となっているにもかかわらず (ルール条件が **Applicant(age < 21)**)、ローン申請者が 21 歳未満の場合は、**"Underage"** ルールの **then** アクションが **setApproved(false)** となり、年齢が基準に達していないためローンの申し込みは承認されません。

ルールアクションの主な目的は、デシジョンエンジンのワーキングメモリーでデータの挿入、削除、または変更を行うことです。有効なルールアクションは小規模かつ宣言的で、可読性があるものです。ルールアクションで必須または条件付きコードを使用する必要がある場合は、ルールを小規模かつより宣言的な複数のルールに分割します。

申込者の年齢制限に関するルールの例

```
rule "Underage"
  when
    application : LoanApplication()
    Applicant( age < 21 )
  then
    application.setApproved( false );
    application.setExplanation( "Underage" );
end
```

16.9.1. DRL でサポートされるルールアクションメソッド

DRL では DRL ルールアクションで使用できる以下のルールアクションメソッドがサポートされます。これらのメソッドを使用することで、最初にワーキングメモリーインスタンスを参照せずに、デシジョンエンジンのワーキングメモリーを変更できます。これらのメソッドは Red Hat Process Automation Manager ディストリビューションの **KnowledgeHelper** クラスで提供されるメソッドへのショートカットとして機能します。

すべてのルールアクションメソッドの方法は、Red Hat カスタマーポータル から ZIP ファイル **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**~/rhpam-7.11.0-**

sources/src/drools-$VERSION/drools-core/src/main/java/org/drools/core/spi/KnowledgeHelper.java に移動してください。

set

これを使用してフィールドの値を設定します。

```
set<field> ( <value> )
```

ローン申し込みの承認の値を設定するルールアクションの例

```
$application.setApproved ( false );
$application.setExplanation( "has been bankrupt" );
```

modify

ファクトの変更するフィールドを指定し、デシジョンエンジンに変更を通知します。このメソッドは、ファクトの更新に対する構造化されたアプローチを提供します。このメソッドは、**update** 操作とオブジェクトフィールドを変更する setter 呼び出しを組み合わせたものです。

```
modify ( <fact-expression> ) {
    <expression>,
    <expression>,
    ...
}
```

ローン申し込み件数および承認を変更するルールアクションの例

```
modify( LoanApplication ) {
        setAmount( 100 ),
        setApproved ( true )
}
```

update

フィールドと更新される関連ファクト全体を指定して、その変更をデシジョンエンジンに通知します。ファクトが変更したら、更新した値の影響を受ける可能性がある別のファクトを変更する前に、**update** を呼び出す必要があります。この追加設定を回避するには、**modify** メソッドを代わりに使用します。

```
update ( <object, <handle> )  // Informs the decision engine that an object has changed

update ( <object> )  // Causes `KieSession` to search for a fact handle of the object
```

ローンの申し込み件数および承認を更新するルールアクションの例

```
LoanApplication.setAmount( 100 );
update( LoanApplication );
```

注記

property-change リスナーを指定する場合は、オブジェクトの変更時にこのメソッドを呼び出す必要はありません。property-change リスナーの詳細は、Red Hat Process Automation Manager のデシジョンエンジン を参照してください。

insert

new ファクトをデシジョンエンジンのワーキングメモリーに挿入し、ファクトに必要な結果として生成されるフィールドと値を定義します。

```
insert( new <object> );
```

新しいローン申請者オブジェクトを挿入するルールアクションの例

```
insert( new Applicant() );
```

insertLogical

デシジョンエンジンに **new** ファクトを論理挿入する場合に使用します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。通常の挿入または記述による挿入の後には、ファクトは明示的に取り消される必要があります。論理挿入の後、挿入されたファクトは、ファクトを挿入したルールの条件が true でなくなると自動的に取り消されます。

```
insertLogical( new <object> );
```

新しいローン申請者オブジェクトを論理的に挿入するルールアクションの例

```
insertLogical( new Applicant() );
```

delete

デシジョンエンジンからオブジェクトを削除します。キーワード **retract** も DRL でサポートされており、同じアクションを実行しますが、DRL のコードでは、キーワード **insert** との整合性を考慮して **delete** が通常推奨されます。

```
delete( <object> );
```

ローン申請者オブジェクトを削除するルールアクションの例

```
delete( Applicant );
```

16.9.2. drools および kcontext 変数のその他のルールアクションメソッド

標準のルールアクションメソッドに加えて、デシジョンエンジンでは、ルールアクションで使用できる事前定義された **drools** 変数および **kcontext** 変数と組み合わせて使用できるメソッドもサポートしています。

drools 変数を使用して、Red Hat Process Automation Manager ディストリビューションの **KnowledgeHelper** クラスからメソッドを呼び出すことができます。これは、標準のアクションメソッドのベースとなるクラスでもあります。すべての **drools** ルールアクションのオプションは、Red Hat Customer Portal から ZIP ファイル **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**~/rhpam-7.11.0-sources/src/drools-$VERSION/drools-core/src/main/java/org/drools/core/spi/KnowledgeHelper.java** に移動してください。

以下の例は、**drools** 変数と共に使用できる一般的なメソッドです。

- **drools.halt()**: ユーザーまたはアプリケーションで以前に **fireUntilHalt()** が呼び出されている場合は、ルールの実行を終了します。ユーザーまたはアプリケーションが **fireUntilHalt()** を呼び

出す場合、デシジョンエンジンは **アクティブモード** で開始し、ユーザーまたはアプリケーションが **halt()** を明示的に呼び出すまでルールの評価を継続します。それ以外の場合、デシジョンエンジンはデフォルトで **パッシブモード** で実行し、ユーザーまたはアプリケーションが明示的に **fireAllRules()** を呼び出す場合にのみルールを評価します。

- **drools.getWorkingMemory()**: **WorkingMemory** オブジェクトを返します。
- **drools.setFocus("<agenda_group>")**: ルールが属する指定されたアジェンダグループにフォーカスを設定します。
- **drools.getRule().getName()**: ルールの名前を返します。
- **drools.getTuple()**、**drools.getActivation()**: 現在実行されているルールに一致する **Tuple** を返し、対応する **Activation** を提供します。これらの呼び出しは、ロギングやデバッグを行う場合に役立ちます。

kcontext 変数を **getKieRuntime()** メソッドと共に使用して、**KieContext** クラスや、拡張によりご使用の Red Hat Process Automation Manager ディストリビューションの **RuleContext** クラスから別のメソッドを呼び出すことができます。完全な Knowledge Runtime API は **kcontext** 変数を通じて公開され、詳細なルールアクションメソッドを提供します。**kcontext** ルールアクションのすべてのオプションは、Red Hat カスタマーポータル から ZIP ファイル **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**~/rhpam-7.11.0-sources/src/kie-api-parent-$VERSION/kie-api/src/main/java/org/kie/api/runtime/rule/RuleContext.java** に移動してください。

以下の例は **kcontext.getKieRuntime()** の変数とメソッドの組み合わせで使用できる一般的なメソッドです。

- **kcontext.getKieRuntime().halt()**: ユーザーまたはアプリケーションで以前に **fireUntilHalt()** が呼び出されている場合は、ルールの実行を終了します。このメソッドは、**drools.halt()** メソッドと同等です。ユーザーまたはアプリケーションが **fireUntilHalt()** を呼び出す場合、デシジョンエンジンは **アクティブモード** で開始し、ユーザーまたはアプリケーションが **halt()** を明示的に呼び出すまでルールの評価を継続します。それ以外の場合、デシジョンエンジンはデフォルトで **パッシブモード** で実行し、ユーザーまたはアプリケーションが明示的に **fireAllRules()** を呼び出す場合にのみルールを評価します。
- **kcontext.getKieRuntime().getAgenda()**: KIE セッション **Agenda** への参照を返し、次にルールアクティベーショングループ、ルールアジェンダグループ、およびルールフローグループにアクセスできるようにします。

  アジェンダグループ CleanUp にアクセスしてフォーカスを設定する呼び出しの例

  ```
  kcontext.getKieRuntime().getAgenda().getAgendaGroup( "CleanUp" ).setFocus();
  ```

  この例は、**drools.setFocus("CleanUp")** と同等です。
- **kcontext.getKieRuntime().getQueryResults(<string> query)**: クエリーを実行し、結果を返します。このメソッドは **drools.getKieRuntime().getQueryResults()** と同等です。
- **kcontext.getKieRuntime().getKieBase()**: **KieBase** オブジェクトを返します。KIE ベースはルールシステムのすべてのナレッジのソースであり、現在の KIE セッションの開始元です。
- **kcontext.getKieRuntime().setGlobal()**、**~.getGlobal()**、**~.getGlobals()**: グローバル変数を設定するか、または取得します。
- **kcontext.getKieRuntime().getEnvironment()**: 使用するオペレーティングシステム環境と類似したランタイムの **Environment** を返します。

16.9.3. 条件付きおよび名前付きの結果を伴う高度なルールアクション

一般的に、有効なルールアクションは小規模かつ宣言的であり、可読性があります。ただし、場合によっては、ルールごとに結果を 1 つに制限することが困難であり、以下のルールの例のように、ルールの構文が冗長になったり、繰り返しが多くなる可能性があります。

冗長で繰り返しの構文の多いルールの例

```
rule "Give 10% discount to customers older than 60"
  when
    $customer : Customer( age > 60 )
  then
    modify($customer) { setDiscount( 0.1 ) };
end

rule "Give free parking to customers older than 60"
  when
    $customer : Customer( age > 60 )
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
end
```

繰り返しを部分的に解決する手段として、以下の変更例にあるように、2 番目のルールで最初のルールを拡張します。

拡張された条件を使用して部分的に強化されたルールの例

```
rule "Give 10% discount to customers older than 60"
  when
    $customer : Customer( age > 60 )
  then
    modify($customer) { setDiscount( 0.1 ) };
end

rule "Give free parking to customers older than 60"
    extends "Give 10% discount to customers older than 60"
  when
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
end
```

より効率的な代替方法として、以下の例に示すように、変更した条件およびラベルが付与された対応するルールアクションを使用して、2 つのルールを 1 つのルールに統合することができます。

条件および名前付きの結果を使用した統合されたルールの例

```
rule "Give 10% discount and free parking to customers older than 60"
  when
    $customer : Customer( age > 60 )
    do[giveDiscount]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
```

```
  then[giveDiscount]
    modify($customer) { setDiscount( 0.1 ) };
end
```

このルールの例では、通常のデフォルトアクションと、**giveDiscount** という名前のもう 1 つの別のアクションが使用されています。**giveDiscount** アクションは、KIE ベースで年齢が 60 歳を超えた顧客が見つかると、その顧客が車を所有しているかどうかにかかわらず、キーワード **do** でアクティブにされます。

名前付きの結果のアクティベーションは、次の例の **if** ステートメントのように、追加の条件を使用して設定できます。**if** ステートメント内の条件は、その直前にあるパターンで常に評価されます。

追加の条件が指定される統合されたルールの例

```
rule "Give free parking to customers older than 60 and 10% discount to golden ones among them"
  when
    $customer : Customer( age > 60 )
    if ( type == "Golden" ) do[giveDiscount]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
  then[giveDiscount]
    modify($customer) { setDiscount( 0.1 ) };
end
```

以下のより複雑な例のように、ネストされた **if** および **else if** 設定を使用して、さまざまなルール条件を評価することもできます。

より複雑な条件を使用して統合されたルールの例

```
rule "Give free parking and 10% discount to over 60 Golden customer and 5% to Silver ones"
  when
    $customer : Customer( age > 60 )
    if ( type == "Golden" ) do[giveDiscount10]
    else if ( type == "Silver" ) break[giveDiscount5]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
  then[giveDiscount10]
    modify($customer) { setDiscount( 0.1 ) };
  then[giveDiscount5]
    modify($customer) { setDiscount( 0.05 ) };
end
```

このルールの例では、60 歳を超えた Golden 顧客には 10% の割引きと無料駐車サービスが提供されますが、Silver 顧客に提供されるのは 5% の割引きのみで、無料駐車サービスは提供されません。このルールでは、**do** ではなく **break** というキーワードによって、**giveDiscount5** という名前の結果がアクティブにされます。キーワード **do** は、デシジョンエンジンのアジェンダで結果をスケジュール設定し、ルール条件の残りの部分が引き続き評価されるようにします。一方、**break** は追加の条件の評価を行いません。名前付きの結果が **do** のいずれの条件にも一致せず、**break** でアクティブにされる場合は、ルールの条件部分に到達することはないため、ルールはコンパイルされません。

16.10. DRL ファイルのコメント

DRL では、二重のスラッシュ (//) が先頭に付けられた単一行コメントと、スラッシュおよびアスタリスク (/* ... */) で囲まれた複数行のコメントがサポートされます。DRL コメントを使用して、DRL ファイルのルールまたは関連コンポーネントにアノテーションを付けることができます。DRL ファイルの処理時に、デシジョンエンジンでは DRL コメントは無視されます。

コメントが使用されるルールの例

```
rule "Underage"
  // This is a single-line comment.
  when
    $application : LoanApplication()  // This is an in-line comment.
    Applicant( age < 21 )
  then
    /* This is a multi-line comment
    in the rule actions. */
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

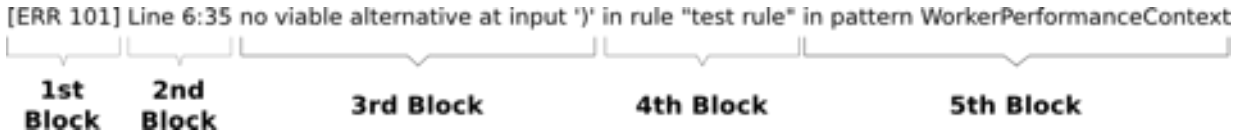
重要

DRL コメントでは、ハッシュ記号 # はサポートされていません。

16.11. DRL トラブルシューティングのエラーメッセージ

Red Hat Process Automation Manager は、DRL ファイルの問題のトラブルシューティングと解決に役立つ、DRL エラーの標準化されたメッセージを提供します。エラーメッセージでは、以下の形式が使用されています。

図16.1 DRL ファイルの問題に関するエラーメッセージの形式



- **最初のブロック**: エラーコード
- **2 番目のブロック**: DRL ソースのエラーが発生している行および列
- **3 番目のブロック**: 問題の説明
- **4 番目のブロック**: DRL ソース内のエラーが発生しているコンポーネント (ルール、関数、クエリー)
- **5 番目のブロック**: DRL ソース内のエラーが発生しているパターン (該当する場合)

Red Hat Process Automation Manager では、以下の標準化されたエラーメッセージがサポートされます。

101: no viable alternative

パーサーが決定ポイントに到達したにもかかわらず、選択肢を特定できなかったことを示します。

誤ったスペリングを含むルールの例

```
1: rule "simple rule"
2:   when
3:     exists Person()
4:     exits Student()  // Must be `exists`
5:   then
6: end
```

エラーメッセージ

```
[ERR 101] Line 4:4 no viable alternative at input 'exits' in rule "simple rule"
```

ルール名がないルールの例

```
1: package org.drools.examples;
2: rule    // Must be `rule "rule name"` (or `rule rule_name` if no spacing)
3:   when
4:     Object()
5:   then
6:     System.out.println("A RHS");
7: end
```

エラーメッセージ

```
[ERR 101] Line 3:2 no viable alternative at input 'when'
```

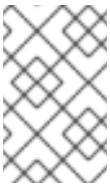
この例では、パーサーがキーワード **when** を検出しましたが、予想していたのはルール名であったため、パーサーは **when** に予想外の不適切なトークンというフラグを付けます。

誤った構文が使用されるルールの例

```
1: rule "simple rule"
2:   when
3:     Student( name == "Andy )  // Must be `"Andy"`
4:   then
5: end
```

エラーメッセージ

```
[ERR 101] Line 0:-1 no viable alternative at input '<eof>' in rule "simple rule" in pattern Student
```



注記

列と行の値が **0:-1** の場合は、パーサーがソースファイルの終わり (**<eof>**) に到達しましたが、不完全な設定を検出したことを示します。よくある原因として、引用符 **"..."**、アポストロフィー **'...'**、または括弧 **(...)** が欠落している場合があります。

102: mismatched input

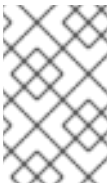
パーサーが特定の記号を予想していましたが、これが現在の入力位置で欠落していることを示します。

不完全なルールステートメントが使用されるルールの例

```
1: rule simple_rule
2:   when
3:     $p : Person(
       // Must be a complete rule statement
```

エラーメッセージ

```
[ERR 102] Line 0:-1 mismatched input '<eof>' expecting ')' in rule "simple rule" in pattern Person
```



注記

列と行の値が **0:-1** の場合は、パーサーがソースファイルの終わり (**<eof>**) に到達しましたが、不完全な設定を検出したことを示します。よくある原因として、引用符 **"..."**、アポストロフィー **'...'**、または括弧 **(...)** が欠落している場合があります。

誤った構文が使用されるルールの例

```
1: package org.drools.examples;
2:
3: rule "Wrong syntax"
4:   when
5:     not( Car( ( type == "tesla", price == 10000 ) || ( type == "kia", price == 1000 ) ) from $carList )
       // Must use `&&` operators instead of commas `,`
6:   then
7:     System.out.println("OK");
8: end
```

エラーメッセージ

```
[ERR 102] Line 5:36 mismatched input ',' expecting ')' in rule "Wrong syntax" in pattern Car
[ERR 101] Line 5:57 no viable alternative at input 'type' in rule "Wrong syntax"
[ERR 102] Line 5:106 mismatched input ')' expecting 'then' in rule "Wrong syntax"
```

この例では、構文に関する問題が原因で相互に関連する複数のエラーメッセージが発生しています。**&&** 演算子で **,** を置き換えるという 1 つの解決策により、すべてのエラーが解消されます。複数のエラーが発生した場合に、エラーが前のエラーの結果である場合があるため、一度に 1 つずつ解決します。

103: failed predicate

セマンティクスの述語の検証が **false** と評価されたことを示します。これらのセマンティクスの述語は一般に **declare**、**rule**、**exists**、**not** など、DRL ファイルのコンポーネントのキーワードを特定するために使用されます。

無効なキーワードが使用されているルールの例

```
1: package nesting;
2:
3: import org.drools.compiler.Person
4: import org.drools.compiler.Address
5:
6: Some text  // Must be a valid DRL keyword
7:
```

```
 8: rule "test something"
 9:   when
10:     $p: Person( name=="Michael" )
11:   then
12:     $p.name = "other";
13:     System.out.println(p.name);
14: end
```

エラーメッセージ

```
[ERR 103] Line 6:0 rule 'rule_key' failed predicate:
{(validateIdentifierKey(DroolsSoftKeywords.RULE))}? in rule
```

Some text の行は、DRL キーワード設定で始まっていないか、または DRL キーワード設定の一部ではないため、パーサーが DRL ファイルの残りの部分の評価に失敗します。

注記

このエラーは **102: mismatched input** に似ていますが、通常は DRL キーワードが関係しています。

104: trailing semi-colon not allowed

ルール条件の **eval()** 句でセミコロン **;** が使用されていますが、セミコロンは使用できません。

eval() と末尾のセミコロンが使用されているルールの例

```
1: rule "simple rule"
2:   when
3:     eval( abc(); )  // Must not use semicolon `;`
4:   then
5: end
```

エラーメッセージ

```
[ERR 104] Line 3:4 trailing semi-colon not allowed in rule "simple rule"
```

105: did not match anything

パーサーが文法内で、少なくとも 1 回は代替の選択肢に一致する必要があるサブルールに到達しましたが、サブルールがいすれにも一致しなかったことを示します。パーサーは出口のないブランチに入ります。

空の条件内に無効なテキストがあるルールの例

```
1: rule "empty condition"
2:   when
3:     None  // Must remove `None` if condition is empty
4:   then
5:     insert( new Person() );
6: end
```

エラーメッセージ

> [ERR 105] Line 2:2 required (...)+ loop did not match anything at input 'WHEN' in rule "empty condition"

この例では、条件は空であることが意図されていますが、**None** という単語が使用されています。このエラーは、DRL の有効でないキーワード、データタイプ、またはパターン設定である **None** を削除することによって解決できます。

注記

解決できないその他の DRL エラーメッセージが発生した場合は、Red Hat のテクニカルアカウントマネージャーにお問い合わせください。

16.12. DRL ルールセットのルールユニット

ルールユニットは、データソース、グローバル変数、および DRL ルールのグループで、特定の目的に向けて互いに機能し合います。ルールユニットを使用して、ルールセットを小さなユニットに分割し、それらのユニットにさまざまなデータソースをバインドしてから、個別のユニットを実行します。ルールユニットは、実行制御用のルールアジェンダグループまたはアクティブ化グループなどの、ルールをグループ化する DRL 属性に代わるものとして、強化されています。

ルールユニットは、ルールの実行を調整することで、あるルールユニットが完全に実行されると別のルールユニットの開始をトリガーする場合などに便利です。たとえば、データ強化用の一連のルール、そのデータを処理する別の一連のルール、および処理されたデータを抽出して出力する別の一連のルールがあるとします。これらのルールセットを 3 つの異なるルールユニットに追加する場合は、これらのルールユニットを調整することで、1 つ目のユニットが完全に実行すると 2 つ目のユニットの開始をトリガーし、2 つ目のユニットが完全に実行すると 3 つ目のユニットの開始をトリガーすることができます。

ルールユニットを定義するには、以下の例に示すように **RuleUnit** インターフェイスを実装します。

ルールユニットクラスの例

```
package org.mypackage.myunit;

public static class AdultUnit implements RuleUnit {
    private int adultAge;
    private DataSource<Person> persons;

    public AdultUnit( ) { }

    public AdultUnit( DataSource<Person> persons, int age ) {
        this.persons = persons;
        this.age = age;
    }

    // A data source of `Persons` in this rule unit:
    public DataSource<Person> getPersons() {
        return persons;
    }

    // A global variable in this rule unit:
    public int getAdultAge() {
        return adultAge;
    }
```

```
    // Life-cycle methods:
    @Override
    public void onStart() {
        System.out.println("AdultUnit started.");
    }

    @Override
    public void onEnd() {
        System.out.println("AdultUnit ended.");
    }
}
```

この例では、**persons** はタイプ **Person** のファクトのソースです。ルールユニットのデータソースは、指定のルールユニットで処理されるデータのソースで、デシジョンエンジンがルールユニットの評価に使用するエントリーポイントを表します。**adultAge** グローバル変数は、このルールユニットに属するすべてのルールからアクセスできます。最後の 2 つのメソッドは、ルールユニットのライフサイクルの一部で、デシジョンエンジンによって呼び出されます。

デシジョンエンジンは、以下のようなルールユニットの任意のライフサイクルメソッドをサポートします。

表16.3 ルールユニットのライフサイクルメソッド

メソッド	呼び出されるタイミング
onStart()	ルールユニット実行開始時
onEnd()	ルールユニット実行終了時
onSuspend()	ルールユニット実行の一時停止時 (**runUntilHalt()** でのみを使用される)
onResume()	ルールユニット実行の再開時 (**runUntilHalt()** でのみ使用される)
onYield(RuleUnit other)	ルールユニットにおけるルールの結果が異なるルールユニットの実行をトリガー

ルールユニットに、ルールを 1 つ以上追加することができます。デフォルトでは、DRL ファイルのすべてのルールは、DRL ファイル名の命名規則に従うルールユニットに自動的に関連付けられます。DRL ファイルが同じパッケージにあり、**RuleUnit** インターフェイスを実装するクラスと同じ名前を持つ場合、その DRL ファイルのすべてのルールは、そのルールユニットに暗黙的に属します。たとえば、**org.mypackage.myunit** パッケージの **AdultUnit.drl** ファイルにあるすべてのルールは、自動的にルールユニット **org.mypackage.myunit.AdultUnit** の一部となります。

この命名規則をオーバーライドし、DRL ファイル内のルールが属するルールユニットを明示的に宣言するには、DRL ファイル内でキーワード **unit** を使用します。**unit** 宣言は、すぐに package 宣言に従い、DRL ファイルのルールが一部となっているパッケージ内のクラス名を含む必要があります。

DRL ファイルのルールユニット宣言の例

```
package org.mypackage.myunit
```

```
unit AdultUnit

rule Adult
  when
    $p : Person(age >= adultAge) from persons
  then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
```

警告

同じ KIE ベースで、ルールユニットありのルールとルールユニットなしのルールを混在させないでください。KIE ベースで 2 つのルールのパラダイムを混在させると、コンパイルエラーが発生します。

以下の例のように OOPath 表記を使用して、より便利な方法で同じパターンを書き換えることもできます。

OOPath 表記を使用する DRL ファイルのルールユニット宣言の例

```
package org.mypackage.myunit
unit AdultUnit

rule Adult
  when
    $p : /persons[age >= adultAge]
  then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
```

注記

OOPath は、DRL ルールの条件の制約でオブジェクトのグラフを参照するために設計された XPath のオブジェクト指向構文の拡張です。OOPath は、コレクションおよびフィルター制約を処理する間に XPath からのコンパクト表記を使用して関連要素を移動します。また、OOPath は特にオブジェクトグラフの場合に役に立ちます。

この例では、ルール条件で一致するファクトはすべて、ルールユニットクラスの **DataSource** 定義で定義される **persons** のデータソースから取得されます。ルール条件およびアクションは、グローバル変数が DRL ファイルレベルで定義されるのと同じ方法で **adultAge** 変数を使用します。

KIE ベースに定義されたルールユニットを 1 つ以上実行するには、KIE ベースにバインドされている新規の **RuleUnitExecutor** クラスを作成し、関連するデータソースからルールユニットを作成して、ルールユニットエグゼキューターを実行します。

ルールユニット実行の例

```
// Create a `RuleUnitExecutor` class and bind it to the KIE base:
KieBase kbase = kieContainer.getKieBase();
```

```
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

// Create the `AdultUnit` rule unit using the `persons` data source and run the executor:
RuleUnit adultUnit = new AdultUnit(persons, 18);
executor.run( adultUnit );
```

ルールは **RuleUnitExecutor** クラスにより実行されます。**RuleUnitExecutor** クラスは KIE セッションを作成し、必要な **DataSource** オブジェクトをこれらのセッションに追加してから、**run()** メソッドへのパラメーターとして渡される **RuleUnit** をもとに、ルールを実行します。

例の実行コードは、関連する **Person** ファクトが **persons** データソースに挿入されると、以下の出力を生成します。

ルールユニット実行出力の例

```
org.mypackage.myunit.AdultUnit started.
Jane is adult and greater than 18
John is adult and greater than 18
org.mypackage.myunit.AdultUnit ended.
```

ルールユニットインスタンスを明示的に作成するのではなく、エグゼキューターにルールユニット変数を登録し、実行するルールユニットクラスをエグゼキューターに渡すと、エグゼキューターがルールユニットのインスタンスを作成します。続いて、ルールユニットを実行する前に **DataSource** 定義および他の変数を設定できます。

登録変数を含む別のルールユニット実行オプション

```
executor.bindVariable( "persons", persons );
        .bindVariable( "adultAge", 18 );
executor.run( AdultUnit.class );
```

RuleUnitExecutor.bindVariable() メソッドに渡す名前は、実行時に、同じ名前のルールユニットクラスのフィールドに変数をバインドするために使用されます。前述の例では、**RuleUnitExecutor** は、新しいルールユニットに **"persons"** の名前にバインドされているデータソースを挿入します。また、**AdultUnit** クラス内の対応する名前のフィールドに、文字列 **"adultAge"** にバインドされている値 **18** を挿入します。

このデフォルトの変数バインディング動作をオーバーライドするには、**@UnitVar** アノテーションを使用してルールユニットクラスの各フィールドに対して論理バインディング名を明示的に定義します。たとえば、以下のクラスのフィールドバインディングは、代替名で再度定義されます。

@UnitVar を使用した変数バインディング名を変更するコード例

```
package org.mypackage.myunit;

public static class AdultUnit implements RuleUnit {
    @UnitVar("minAge")
    private int adultAge = 18;

    @UnitVar("data")
    private DataSource<Person> persons;
}
```

次に、これらの代替名を使用して、変数をエグゼキューターにバインドし、ルールユニットを実行できます。

変更した変数名を使用したルールユニット実行の例

```
executor.bindVariable( "data", persons );
        .bindVariable( "minAge", 18 );
executor.run( AdultUnit.class );
```

ルールユニットは、**run()** メソッド (KIE セッションで **fireAllRules()** を呼び出す場合と同じ) を使用して **パッシブモード** で、または **runUntilHalt()** メソッド (KIE セッションで **fireUntilHalt()** を呼び出す場合と同じ) を使用して **アクティブモード** で実行できます。デフォルトでは、デシジョンエンジンは **パッシブモード** で実行し、ユーザーまたはアプリケーションが明示的に **run()** (標準ルールでは **fireAllRules()**) を呼び出す場合にのみルールユニットを評価します。ユーザーまたはアプリケーションがルールユニットに **runUntilHalt()** (標準ルールでは **fireAllRules()**) を呼び出す場合、デシジョンエンジンは **アクティブモード** で開始し、ユーザーまたはアプリケーションが明示的に **halt()** を呼び出すまで、継続的にルールユニットを評価します。

runUntilHalt() メソッドを使用する場合は、メインスレッドをブロックしないように、別の実行スレッド上でメソッドを呼び出します。

別のスレッド上の runUntilHalt() を使用したルールユニットの実行例

```
new Thread( () -> executor.runUntilHalt( adultUnit ) ).start();
```

16.12.1. ルールユニットのデータソース

ルールユニットのデータソースは、指定のルールユニットで処理されるデータのソースで、デシジョンエンジンがルールユニットの評価に使用するエントリーポイントを表します。ルールユニットは、ゼロまたは複数のデータソースを持つことができ、ルールユニット内で宣言された各 **DataSource** の定義は、ルールユニットエグゼキューターへの異なるエントリーポイントに対応することができます。複数のルールユニットは、単一データソースを共有できます。ただし、各ルールユニットは、同じオブジェクトが挿入される異なるエントリーポイントを使用する必要があります。

以下の例で示すように、ルールユニットクラスの固定されたデータセットを使用して **DataSource** 定義を作成できます。

データソース定義の例

```
DataSource<Person> persons = DataSource.create( new Person( "John", 42 ),
                                                new Person( "Jane", 44 ),
                                                new Person( "Sally", 4 ) );
```

データソースはルールユニットのエントリーポイントを表すため、ルールユニットでファクトを挿入、更新、または削除できます。

ルールユニットでファクトを挿入、更新、削除するコード例

```
// Insert a fact:
Person john = new Person( "John", 42 );
FactHandle johnFh = persons.insert( john );

// Modify the fact and optionally specify modified properties (for property reactivity):
john.setAge( 43 );
```

```
persons.update( johnFh, john, "age" );

// Delete the fact:
persons.delete( johnFh );
```

16.12.2. ルールユニットの実行制御

一方のルールユニットの実行により、もう一方のルールユニットの開始がトリガーされるようにルールの実行を調整する必要がある場合に、ルールユニットは役に立ちます。

ルールユニットの実行制御を容易にするために、デシジョンエンジンは以下のルールユニットメソッドをサポートします。このメソッドは、DRL ルールアクションで使用して、ルールユニットの実行を調整することができます。

- **drools.run()**: 指定されたルールユニットクラスの実行をトリガーします。このメソッドでは、ルールユニットの実行を命令的に中断し、他の指定されたルールユニットを有効化します。
- **drools.guard()**: 関連付けられたルール条件が満たされるまで、指定されたルールユニットクラスが実行されないようにします (保護します)。このメソッドは、他の指定されたルールユニットの実行を宣言的にスケジュールします。デシジョンエンジンが、保護ルールの条件に対して少なくとも 1 つの一致をもたらす場合は、保護されたルールユニットが有効とみなされます。ルールユニットには、複数の保護ルールを含めることができます。

drools.run() メソッドの例として、それぞれが指定されたルールユニットに属す以下の DRL ルールを検討してください。**NotAdult** ルールは **drools.run(AdultUnit.class)** メソッドを使用して **AdultUnit** ルールユニットの実行をトリガーします。

drools.run() を使用した制御された実行を含む DRL ルールの例

```
package org.mypackage.myunit
unit AdultUnit

rule Adult
  when
    Person(age >= 18, $name : name) from persons
  then
    System.out.println($name + " is adult");
end
```

```
package org.mypackage.myunit
unit NotAdultUnit

rule NotAdult
  when
    $p : Person(age < 18, $name : name) from persons
  then
    System.out.println($name + " is NOT adult");
    modify($p) { setAge(18); }
    drools.run( AdultUnit.class );
end
```

この例では、これらのルールからビルドされた KIE ベースから作成された **RuleUnitExecutor** クラスと、これにバインドされている **persons** の **DataSource** 定義も使用します。

ルールエグゼキューターとデータソース定義の例

```
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );
DataSource<Person> persons = executor.newDataSource( "persons",
                                        new Person( "John", 42 ),
                                        new Person( "Jane", 44 ),
                                        new Person( "Sally", 4 ) );
```

この例では、**RuleUnitExecutor** クラスから **DataSource** 定義を直接作成し、これを単一ステートメントで **"persons"** 変数にバインドします。

例の実行コードは、関連する **Person** ファクトが **persons** データソースに挿入されると、以下の出力を生成します。

ルールユニット実行出力の例

```
Sally is NOT adult
John is adult
Jane is adult
Sally is adult
```

NotAdult ルールは、**"Sally"** という人物の評価時に一致を検出します。この人物は 18 歳未満です。続いてこのルールは、この人物の年齢を **18** に変更し、**drools.run(AdultUnit.class)** メソッドを使用して **AdultUnit** ルールユニットの実行をトリガーします。**AdultUnit** ルールユニットには、**DataSource** 定義の 3 人の **persons** 全員に対して実行可能となったルールが含まれています。

drools.guard() メソッドの例として、以下の **BoxOffice** クラスと **BoxOfficeUnit** ルールユニットクラスを検討してください。

BoxOffice クラスの例

```
public class BoxOffice {
    private boolean open;

    public BoxOffice( boolean open ) {
        this.open = open;
    }

    public boolean isOpen() {
        return open;
    }

    public void setOpen( boolean open ) {
        this.open = open;
    }
}
```

BoxOfficeUnit ルールユニットクラスの例

```
public class BoxOfficeUnit implements RuleUnit {
    private DataSource<BoxOffice> boxOffices;

    public DataSource<BoxOffice> getBoxOffices() {
        return boxOffices;
    }
}
```

また、この例では、以下の **TicketIssuerUnit** ルールユニットクラスを使用して、少なくとも 1 つのボックスオフィス (チケット売り場) が営業中である限り、ボックスオフィスでのイベントチケットの販売を続行します。このルールユニットは **persons** および **tickets** の **DataSource** 定義を使用します。

TicketIssuerUnit ルールユニットクラスの例

```
public class TicketIssuerUnit implements RuleUnit {
    private DataSource<Person> persons;
    private DataSource<AdultTicket> tickets;

    private List<String> results;

    public TicketIssuerUnit() { }

    public TicketIssuerUnit( DataSource<Person> persons, DataSource<AdultTicket> tickets ) {
        this.persons = persons;
        this.tickets = tickets;
    }

    public DataSource<Person> getPersons() {
        return persons;
    }

    public DataSource<AdultTicket> getTickets() {
        return tickets;
    }

    public List<String> getResults() {
        return results;
    }
}
```

BoxOfficeUnit ルールユニットには、DRL ルール **BoxOfficeIsOpen** が含まれます。これは、**drools.guard(TicketIssuerUnit.class)** メソッドを使用して、イベントチケットを配布する **TicketIssuerUnit** ルールユニットの実行を保護します。以下に DRL ルールの例を示します。

drools.guard() を使用した制御された実行を含む DRL ルールの例

```
package org.mypackage.myunit;
unit TicketIssuerUnit;

rule IssueAdultTicket when
    $p: /persons[ age >= 18 ]
then
    tickets.insert(new AdultTicket($p));
end
rule RegisterAdultTicket when
    $t: /tickets
then
    results.add( $t.getPerson().getName() );
end
```

```
package org.mypackage.myunit;
unit BoxOfficeUnit;
```

```
rule BoxOfficeIsOpen
  when
    $box: /boxOffices[ open ]
  then
    drools.guard( TicketIssuerUnit.class );
end
```

この例では、少なくとも 1 つのボックスオフィスが **open** である限り、保護された **TicketIssuerUnit** ルールユニットが有効なため、イベントチケットは配布されます。**open** 状態のボックスオフィスがなくなると、保護された **TicketIssuerUnit** ルールユニットは実行されなくなります。

以下のクラスの例は、より完全なボックスオフィスのシナリオを説明します。

ボックスオフィスシナリオのクラスの例

```
DataSource<Person> persons = executor.newDataSource( "persons" );
DataSource<BoxOffice> boxOffices = executor.newDataSource( "boxOffices" );
DataSource<AdultTicket> tickets = executor.newDataSource( "tickets" );

List<String> list = new ArrayList<>();
executor.bindVariable( "results", list );

// Two box offices are open:
BoxOffice office1 = new BoxOffice(true);
FactHandle officeFH1 = boxOffices.insert( office1 );
BoxOffice office2 = new BoxOffice(true);
FactHandle officeFH2 = boxOffices.insert( office2 );

persons.insert(new Person("John", 40));

// Execute `BoxOfficeIsOpen` rule, run `TicketIssuerUnit` rule unit, and execute `RegisterAdultTicket`
rule:
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "John", list.get(0) );
list.clear();

persons.insert(new Person("Matteo", 30));

// Execute `RegisterAdultTicket` rule:
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "Matteo", list.get(0) );
list.clear();

// One box office is closed, the other is open:
office1.setOpen(false);
boxOffices.update(officeFH1, office1);
persons.insert(new Person("Mark", 35));
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "Mark", list.get(0) );
```

```
list.clear();

// All box offices are closed:
office2.setOpen(false);
boxOffices.update(officeFH2, office2); // Guarding rule is no longer true.
persons.insert(new Person("Edson", 35));
executor.run(BoxOfficeUnit.class); // No execution

assertEquals( 0, list.size() );
```

16.12.3. ルールユニットのアイデンティティーの競合

保護されたルールユニットを使用したルール実行のシナリオでは、1 つのルールが複数のルールユニットを保護することができます。同時に、複数のルールが 1 つのルールユニットを保護してから有効にすることもできます。このような 2 通りの保護シナリオでは、ルールユニットには、アイデンティティーの競合を避けるための明確に定義されたアイデンティティーが必要です。

デフォルトでは、ルールユニットのアイデンティティーはルールユニットクラス名で、**RuleUnitExecutor** によりシングルトンクラスとして処理されます。この識別動作は、**RuleUnit** インターフェイスの **getUnitIdentity()** のデフォルトメソッドにエンコードされています。

RuleUnit インターフェイスのデフォルトのアイデンティティーメソッド

```
default Identity getUnitIdentity() {
    return new Identity( getClass() );
}
```

場合によっては、ルールユニット間のアイデンティティーの競合を避けるために、このデフォルトの識別動作をオーバーライドする必要があります。

たとえば、以下の **RuleUnit** クラスには、あらゆる種類のオブジェクトを許可する **DataSource** 定義が含まれています。

Unit0 ルールユニットクラスの例

```
public class Unit0 implements RuleUnit {
    private DataSource<Object> input;

    public DataSource<Object> getInput() {
        return input;
    }
}
```

このルールユニットには、2 つの条件 (OOPath 表記) に基づいて別のルールユニットを保護する、以下の DRL ルールが含まれています。

ルールユニットの DRL ルール GuardAgeCheck の例

```
package org.mypackage.myunit
unit Unit0

rule GuardAgeCheck
  when
    $i: /input#Integer
```

```
    $s: /input#String
  then
    drools.guard( new AgeCheckUnit($i) );
    drools.guard( new AgeCheckUnit($s.length()) );
end
```

保護された **AgeCheckUnit** ルールユニットは、一連の **persons** の年齢を検証します。**AgeCheckUnit** には、確認用の **persons** の **DataSource** の定義、検証用の **minAge** 変数、および結果を集計する **List** が含まれます。

AgeCheckUnit ルールユニットの例

```
public class AgeCheckUnit implements RuleUnit {
    private final int minAge;
    private DataSource<Person> persons;
    private List<String> results;

    public AgeCheckUnit( int minAge ) {
        this.minAge = minAge;
    }

    public DataSource<Person> getPersons() {
        return persons;
    }

    public int getMinAge() {
        return minAge;
    }

    public List<String> getResults() {
        return results;
    }
}
```

AgeCheckUnit ルールユニットには、データソースの **persons** の検証を実行する以下の DRL ルールが含まれます。

ルールユニットの DRL ルール CheckAge の例

```
package org.mypackage.myunit
unit AgeCheckUnit

rule CheckAge
  when
    $p : /persons{ age > minAge }
  then
    results.add($p.getName() + ">" + minAge);
end
```

この例では、**RuleUnitExecutor** クラスを作成し、これらの 2 つのルールユニットが含まれる KIE ベースにクラスをバインドして、同じルールユニットの **DataSource** 定義を 2 つ作成します。

executor 定義とデータソース定義の例

```
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

DataSource<Object> input = executor.newDataSource( "input" );
DataSource<Person> persons = executor.newDataSource( "persons",
                                        new Person( "John", 42 ),
                                        new Person( "Sally", 4 ) );

List<String> results = new ArrayList<>();
executor.bindVariable( "results", results );
```

一部のオブジェクトを入力データソースに挿入し、**Unit0** ルールユニットを実行できるようになりました。

挿入されたオブジェクトを使用したルールユニット実行の例

```
ds.insert("test");
ds.insert(3);
ds.insert(4);
executor.run(Unit0.class);
```

実行結果一覧の例

```
[Sally>3, John>3]
```

この例では、**AgeCheckUnit** という名前のルールユニットはシングルトンクラスと見なされ、1回のみ実行されます。この時、**minAge** 変数は **3** に設定されます。入力データソースに挿入された文字列 **"test"** および整数 **4** の両方は、**minAge** 変数が **4** に設定された2回目の実行をトリガーする可能性もあります。しかし、同じアイデンティティーを持つ別のルールユニットがすでに評価されているため、2回目の実行はありません。

このルールユニットのアイデンティティーの競合を解決するには、**AgeCheckUnit** クラスの **getUnitIdentity()** メソッドをオーバーライドして、ルールユニットアイデンティティーに **minAge** 変数も含めます。

getUnitIdentity() メソッドをオーバーライドする変更された AgeCheckUnit ルールユニット

```
public class AgeCheckUnit implements RuleUnit {

    ...

    @Override
    public Identity getUnitIdentity() {
        return new Identity(getClass(), minAge);
    }
}
```

このオーバーライドにより、以前のルールユニットの実行例は、以下の出力を生成します。

変更したルールユニットの実行結果一覧の例

```
[John>4, Sally>3, John>3]
```

minAge が **3** と **4** に設定されたルールユニットは、2 つの異なるルールユニットとみなされるようになり、両方とも実行されます。

第17章 データオブジェクト

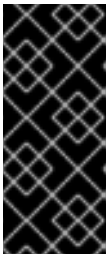
データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

17.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

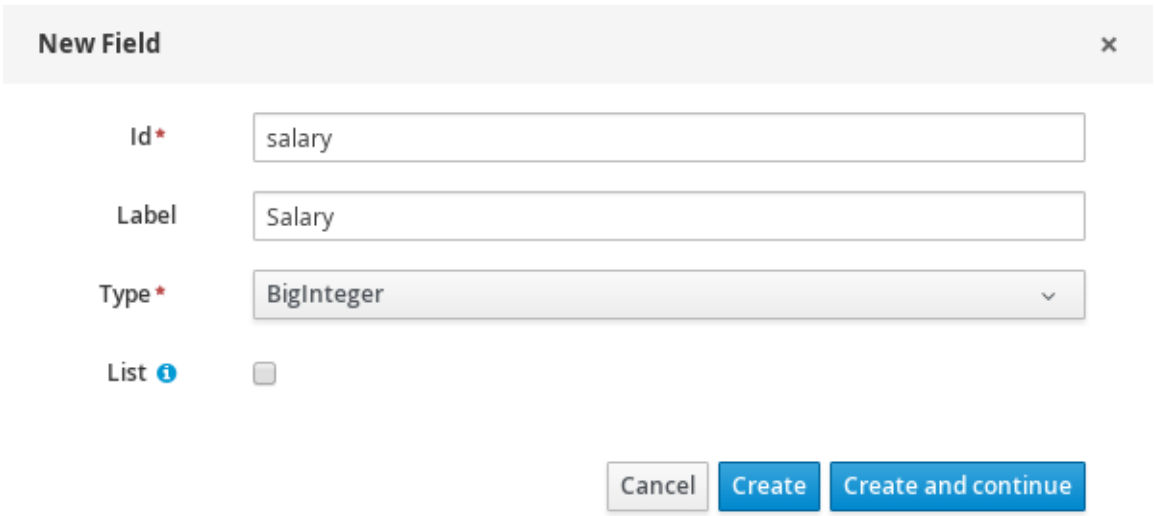


別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図17.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

第18章 BUSINESS CENTRAL における DRL ルールの作成

Business Central で、プロジェクトに対して DRL ルールを作成して管理できます。パッケージに作成またはインポートするデータオブジェクトに基づいて、各 DRL ファイルで、ルールの条件、アクション、そしてルールに関連するその他のコンポーネントを定義します。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → DRL file** をクリックします。
3. 参考となる **DRL ファイル** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
   ドメイン固有言語 (DSL) アセットがプロジェクトに定義されている場合は、**Show declared DSL sentences** を選択することもできます。この DSL アセットは、DRL デザイナーで定義する条件およびアクションに使用できるオブジェクトです。
4. **OK** をクリックして、ルールアセットを作成します。
   新しい DRL ファイルが、**Project Explorer** の **DRL** パネルに追加されます。**Show declared DSL sentences** オプションを選択した場合は、**DSLR** パネルに追加されます。この DRL ファイルを割り当てたパッケージは、ファイルの上位にリストされます。
5. DRL デザイナーの左パネルの **Fact types** リストで、ルールに必要なすべてのデータオブジェクトとデータオブジェクトフィールドがリストされていることを確認します (それぞれを展開します)。リストされていない場合は、DRL ファイルの **import** 命令文を使用して、その他のパッケージから関連するデータオブジェクトをインポートするか、パッケージに データオブジェクトを作成 します。
6. データオブジェクトをすべて配置したら、DRL デザイナーの **Model** タブに戻り、以下のいずれかのコンポーネントで DRL ファイルを定義します。

   DRL ファイル内のコンポーネント

```
package

import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end
```

```
rule "rule2 name"

...
```

- **package**: (自動) これは、DRL ファイルを作成し、パッケージを選択すると定義されます。
- **import**: このパッケージ、または DRL ファイルで使用するその他のパッケージのデータオブジェクトを指定します。パッケージとデータオブジェクトは **packageName.objectName** の形式で指定し、複数のインポートは別々の行に指定します。

  データオブジェクトのインポート

  ```
  import org.mortgages.LoanApplication;
  ```

- **function**: (任意) DRL ファイルのルールが使用する関数を指定します。DRL ファイルの関数は、Java クラスではなく、ルールのソースファイルにセマンティックコードを追加します。関数は、特に、ルールのアクション (**then**) 部分が繰り返し使用され、パラメーターだけがルールごとに異なる場合に便利です。DRL ファイルのルールで、関数を宣言したり、ヘルパークラスから静的メソッドを関数としてインポートしたりすることで、ルールのアクション (**then**) 部分で、名前を指定して関数を使用できます。

  ルールに関数を宣言して使用 (オプション 1)

  ```
  function String hello(String applicantName) {
      return "Hello " + applicantName + "!";
  }

  rule "Using a function"
    when
      // Empty
    then
      System.out.println( hello( "James" ) );
  end
  ```

  ルールに関数をインポートして使用 (オプション 2)

  ```
  import function my.package.applicant.hello;

  rule "Using a function"
    when
      // Empty
    then
      System.out.println( hello( "James" ) );
  end
  ```

- **query**: (任意) DRL ファイルのルールに関連するファクトに対してデシジョンエンジンを検索するのに使用します。DRL ファイルにクエリー定義を追加してから、アプリケーションコードで一致する結果を取得します。クエリーは、定義した条件セットを検索するため、**when** または **then** を指定する必要はありません。クエリー名は KIE ベースでグローバルとなるため、プロジェクトにあるその他のすべてのルールクエリーと重複しないようにする必要があります。クエリーの結果に戻るには、**ksession.getQueryResults("name")**

を使用して、従来の **QueryResults** 定義を設定します (**"name"** はクエリー名)。これにより、クエリーの結果が返り、クエリーに一致したオブジェクトを取得できるようになります。DRL ファイルのルールに、クエリーと、クエリー結果パラメーターを定義します。

DRL ファイルにおけるクエリー定義の例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

クエリー結果を取得するためのアプリケーションコードの例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age  of 21" );
```

- **declare**: (任意) DRL ファイルのルールが使用する新しいファクトタイプを宣言します。Red Hat Process Automation Manager の **java.lang** パッケージのデフォルトのファクトタイプは **Object** ですが、必要に応じて DRL ファイルに別のタイプを宣言することもできます。DRL ファイルにファクトタイプを宣言すると、Java などの低級言語でモデルを作成せず、デシジョンエンジンに直接新しいファクトモデルを定義するようになります。

新しいファクトタイプの宣言および使用

```
declare Person
  name : String
  dateOfBirth : java.util.Date
  address : Address
end

rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
    mark.setName( "Mark" );
    insert( mark );
end
```

- **global**: (任意) DRL ファイルのルールで使用するグローバル変数を組み込みます。グローバル変数は通常、ルールの結果で使用するアプリケーションサービスなど、ルールのデータやサービスを提供し、ルールの結果で追加されるログや値など、ルールからのデータを返します。KIE セッション設定や REST 操作を使用してデシジョンエンジンのワーキングメモリーにグローバル値を設定し、DRL ファイルのルールの上にグローバル変数を宣言してから、これをルールのアクション部分 (**then**) で使用します。グローバル変数が複数ある場合は、DRL ファイルで行を分けて使用してください。

デシジョンエンジンに対するグローバルリストの設定

```
List<String> list = new ArrayList<>();
KieSession kieSession = kiebase.newKieSession();
kieSession.setGlobal( "myGlobalList", list );
```

ルールでのグローバルリストの定義

```
global java.util.List myGlobalList;

rule "Using a global"
  when
    // Empty
  then
    myGlobalList.add( "My global list" );
end
```

警告

グローバル変数に定数イミュータブル値がない場合は、ルールの条件設定にグローバル変数を使用しないでください。グローバル変数はデシジョンエンジンのワーキングメモリーに挿入されないため、デシジョンエンジンでは変数の値の変更を追跡できません。

グローバル変数を使用してルール間でデータを共有しないでください。ルールは常に、ワーキングメモリーの状態に関して推論し、これに対応するため、ルールからルールにデータを渡す必要がある場合は、データをファクトとしてデシジョンエンジンのワーキングメモリーにアサートしてください。

- **rule**: DRL ファイルで各ルールを定義します。ルールは、**rule "name"** 形式のルール名、ルールの動作 (**salience**、**no-loop** など) を定義する任意の属性、**when** および **then** 定義が続きます。ルールごとに、ルールパッケージ内で一意の名前を指定する必要があります。ルールの **when** 部分には、アクションを実行するのに必要な条件が含まれます。たとえば、銀行が、ローンの申し込みを 21 歳以上に限定した場合、**"Underage"** ルールの **when** 条件は **Applicant(age < 21)** になります。ルールの **then** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、ローンの申込者が 21 歳に満たない場合は、**then** アクションが **setApproved(false)** になり、申込者の年齢条件を満たしていないためにローンの申し込みは承認されません。

申込者の年齢制限に関するルール

```
rule "Underage"
  salience 15
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

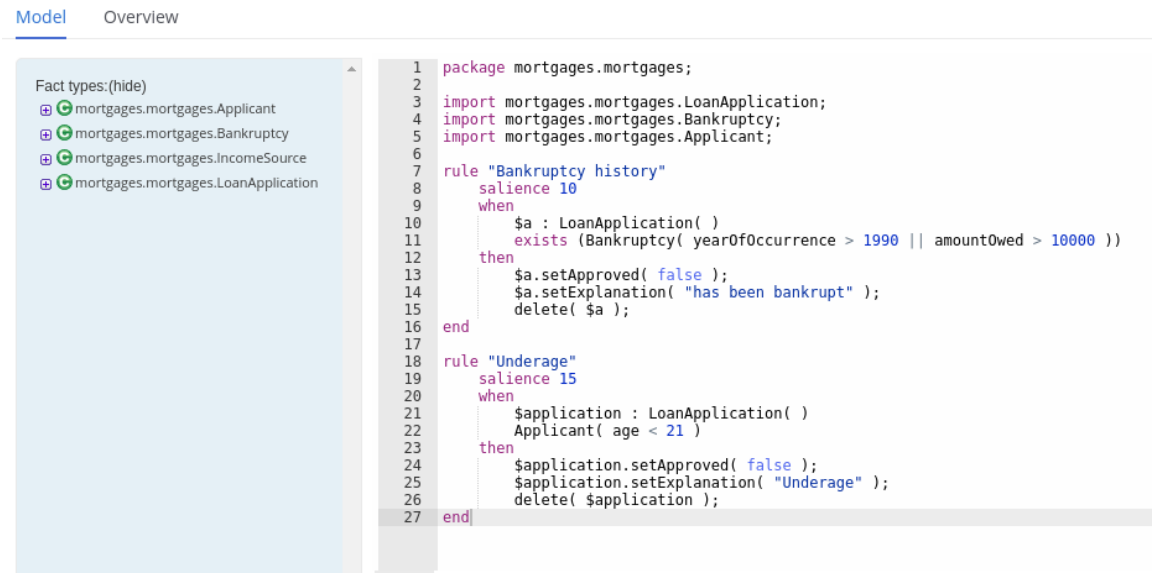
少なくても、各 DRL ファイルは **package** コンポーネント、**import** コンポーネント、**rule** コンポーネントを指定する必要があります。他のすべてのコンポーネントは任意です。

以下は、ローン申し込みのデシジョンサービスの DRL ファイルの例です。

ローン申し込みの DRL ファイルの例

```
package org.mortgages;

import org.mortgages.LoanApplication;
import org.mortgages.Bankruptcy;
import org.mortgages.Applicant;

rule "Bankruptcy history"
 salience 10
 when
  $a : LoanApplication()
  exists (Bankruptcy( yearOfOccurrence > 1990 || amountOwed > 10000 ))
 then
  $a.setApproved( false );
  $a.setExplanation( "has been bankrupt" );
  delete( $a );
end

rule "Underage"
 salience 15
 when
  $application : LoanApplication()
  Applicant( age < 21 )
 then
  $application.setApproved( false );
  $application.setExplanation( "Underage" );
  delete( $application );
end
```

図18.1 Business Central のロール申し込み用の DRL ファイルの例



7. ルールのコンポーネントをすべて定義したら、DRL デザイナーの右上ツールバーで **Validate** をクリックし、DRL ファイルの妥当性を確認します。ファイルの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、DRL ファイルの構文およびコンポーネントをすべて見直し、エラーが表示されなくなるまで再度、ファイルを検証します。

8. DRL デザイナーで **Save** をクリックして、設定した内容を保存します。

18.1. DRL ルールへの WHEN 条件の追加

ルールの **when** 部分には、アクションを実行するのに必要な条件が含まれます。たとえば、銀行のローンの申し込みに年齢制限 (21 歳以上) が必要な場合、**"Underage"** ルールの **when** 条件は **Applicant(age < 21)** となります。パッケージで利用可能なデータオブジェクトに基づいて、指定した一連のパターンおよび制約と、任意のバインディング、その他のサポートされる DRL 要素で設定されます。

前提条件

- **package** は DRL ファイルに定義されます。これは、ファイルの作成時に行われます。
- ルールで使用したデータオブジェクトの **import** リストが、DRL ファイルの **package** 行の下に定義されます。データオブジェクトは、このパッケージ、または別の Business Central パッケージから使用できます。
- **rule** 名は、**package**、**import**、または DRL ファイル全体に適用されるその他の行の下に、**rule "name"** という形式で定義されます。同じパッケージでルール名を重複させることはできません。ルールの動作 (**salience**、**no-loop** など) を定義する任意のルール属性は、ルール名の下、**when** セクションの前に定義します。

手順

1. DRL デザイナーで、ルールに **when** を入力して、条件命令文を追加します。**when** セクションは、ルールの条件を定義するファクトパターンで設定されますが、ファクトパターンが 1 つも追加されない場合もあります。
   when セクションを空にすると、条件は true であると見なされ、デシジョンエンジンで **fireAllRules()** 呼び出しが最初に実施された場合に、**then** セクションのアクションが実行されます。これは、デシジョンエンジンのステートを設定するルールを使用する場合に便利です。

   条件のないルール例

   ```
   rule "Always insert applicant"
     when
       // Empty
     then   // Actions to be executed once
       insert( new Applicant() );
   end

   // The rule is internally rewritten in the following way:

   rule "Always insert applicant"
     when
       eval( true )
     then
       insert( new Applicant() );
   end
   ```

2. 一致させる最初の条件のパターンを入力し、任意で制約、バインディング、およびサポートされる DRL 要素を入力します。基本的なパターンフォーマットは **<patternBinding> : <patternType> (<constraints>)** です。パターンは、パッケージで利用可能なデータオブジェクトに基づいており、**then** セクションのアクションを発生させるのに必要な条件を定義します。

   - **単純なパターン:** 制約のない単純なパターンは、指定したタイプのファクトに一致します。たとえば、次は、申込者が存在することだけが条件になります。

```
when
  Applicant()
```

- **制約のあるパターン:** 制約を持つパターンは、指定したタイプのファクトと、追加制限を括弧で指定したパターン (true または false) に一致します。たとえば、次は、申込者が 21 歳に満たないことを条件としています。

```
when
  Applicant( age < 21 )
```

- **バインディングのあるパターン:** パターンのバインディングは簡単な参照となり、ルールのその他のコンポーネントが、定義したバターンに戻って参照します。たとえば、次の例では、**LoanApplication** のバインディング **a** が、underage の申込者に関連するアクションとして使用されます。

```
when
  $a : LoanApplication()
  Applicant( age < 21 )
then
  $a.setApproved( false );
  $a.setExplanation( "Underage" )
```

3. 引き続き、このルールに適用する条件パターンをすべて定義します。以下は、DRL 条件を定義するいくつかのキーワードオプションです。

- **and**: 条件コンポーネントを論理積に分類します。インフィックスおよびプリフィックスの **and** がサポートされます。デフォルトでは、結合演算子を指定しないと、リストされているパターンがすべて **and** で結合されます。

```
// All of the following examples are interpreted the same way:
$a : LoanApplication() and Applicant( age < 21 )

$a : LoanApplication()
and Applicant( age < 21 )

$a : LoanApplication()
Applicant( age < 21 )

(and $a : LoanApplication() Applicant( age < 21 ))
```

- **or**: 条件コンポーネントを論理和に分類します。インフィックスおよびプリフィックスの **or** がサポートされます。

```
// All of the following examples are interpreted the same way:
Bankruptcy( amountOwed == 100000 ) or IncomeSource( amount == 20000 )

Bankruptcy( amountOwed == 100000 )
or IncomeSource( amount == 20000 )

(or Bankruptcy( amountOwed == 100000 ) IncomeSource( amount == 20000 ))
```

- **exists**: 存在すべきファクトおよび制約を指定します。このオプションは、最初に一致したものだけが適用され、その後一致するものは無視されます。この要素を複数のパターンで使用する場合は、これらのパターンを括弧 **()** で囲みます。

```
exists ( Bankruptcy( yearOfOccurrence > 1990 || amountOwed > 10000 ) )
```

- **not**: 存在するべきでないファクトおよび制約を指定します。

```
not ( Applicant( age < 21 ) )
```

- **forall**: 最初のパターンに一致するすべてのファクトが残りのすべてのパターンに一致するかどうかを検証します。**forall** 設定が満たされると、このルールが **true** と評価されます。

```
forall( $app : Applicant( age < 21 )
        Applicant( this == $app, status = 'underage' ) )
```

- **from**: パターンのデータソースを指定します。

```
Applicant( ApplicantAddress : address )
Address( zipcode == "23920W" ) from ApplicantAddress
```

- **entry-point**: パターンのデータソースに対応する **エントリーポイント** を定義します。通常は **from** とともに使用します。

```
Applicant() from entry-point "LoanApplication"
```

- **collect**: ルールを条件の一部として使用できる、オブジェクトのコレクションを定義します。この例では、指定したそれぞれのローンについてデシジョンエンジンで保留されているすべての申し込みが **List** に分類されます。申し込みが 3 つ以上ある場合は、このルールが実行します。

```
$m : Mortgage()
$a : List( size >= 3 )
   from collect( LoanApplication( Mortgage == $m, status == 'pending' ) )
```

- **accumulate**: オブジェクトのコレクションを処理し、各要素のカスタムアクションを実行し、(制約が **true** と評価されると) 結果オブジェクトを 1 つ以上返します。このオプションは、**collect** よりも強力で、柔軟性が高いオプションです。**accumulate(<source pattern>; <functions> [;<constraints>])** 形式を使用します。この例では、**min**、**max**、および **average** は累積関数で、各センサーのすべての測定値から、最低気温、最高気温、および平均気温の値を計算します。その他のサポートされる関数には、**count**、**sum**、**variance**、**standardDeviation**、**collectList**、および **collectSet** があります。

```
$s : Sensor()
accumulate( Reading( sensor == $s, $temp : temperature );
        $min : min( $temp ),
        $max : max( $temp ),
        $avg : average( $temp );
        $min < 20, $avg > 70 )
```

注記

DRL ルール条件の詳細については、「DRL のルール条件 (WHEN)」 を参照してください。

4. ルールの条件コンポーネントをすべて定義したら、DRL デザイナーの右上のツールバーの **Validate** をクリックして、DRL ファイルの妥当性を確認します。ファイルの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、DRL ファイルの構文およびコンポーネントをすべて見直し、エラーが表示されなくなるまで再度、ファイルを検証します。

5. DRL デザイナーで **Save** をクリックして、設定した内容を保存します。

18.2. DRL ルールへの THEN アクションの追加

ルールの **then** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、ローンの申請者が 21 歳に満たない場合は、**"Underage"** ルールの **then** アクションが **setApproved(false)** となり、年齢が基準に達していないためローンの申し込みは承認されません。アクションは、ルールの条件とパッケージで利用可能オブジェクトに基づいて結果を実行する 1 つ以上のメソッドで設定されます。ルールアクションの主な目的は、デシジョンエンジンのワーキングメモリーでデータの挿入、削除、または変更を行うことです。

前提条件

- **package** は DRL ファイルに定義されます。これは、ファイルの作成時に行われます。
- ルールで使用したデータオブジェクトの **import** リストが、DRL ファイルの **package** 行の下に定義されます。データオブジェクトは、このパッケージ、または別の Business Central パッケージから使用できます。
- **rule** 名は、**package**、**import**、または DRL ファイル全体に適用されるその他の行の下に、**rule "name"** という形式で定義されます。同じパッケージでルール名を重複させることはできません。ルールの動作 (**salience**、**no-loop** など) を定義する任意のルール属性は、ルール名の下、**when** セクションの前に定義します。

手順

1. DRL デザイナーで、ルールの **when** セクションの後に **then** を入力して、アクション命令文を追加します。
2. ルールの条件に基づいて、ファクトパターンに対して実行するアクションを 1 つ以上入力します。
   次は、DRL アクションを定義するキーワードオプションの例です。
   - **set**: フィールドの値を設定します。

     ```
     $application.setApproved ( false );
     $application.setExplanation( "has been bankrupt" );
     ```

   - **modify**: ファクトに対して修正するフィールドを指定し、変更をデシジョンエンジンに通知します。このメソッドは、ファクトの更新に対する構造化されたアプローチを提供します。このメソッドは、**update** 操作とオブジェクトフィールドを変更する setter 呼び出しを組み合わせたものです。

     ```
     modify( LoanApplication ) {
           setAmount( 100 ),
           setApproved ( true )
     }
     ```

   - **update**: フィールドと、更新される関連ファクト全体を指定して、その変更をデシジョンエンジンに通知します。ファクトが変更したら、更新した値の影響を受ける可能性がある別

のファクトを変更する前に、**update** を呼び出す必要があります。この追加設定を回避するには、**modify** メソッドを代わりに使用します。

```
LoanApplication.setAmount( 100 );
update( LoanApplication );
```

- **insert**: **new** ファクトをデシジョンエンジンに挿入します。

```
insert( new Applicant() );
```

- **insertLogical**: デシジョンエンジンに **new** ファクトを論理挿入する場合に使用します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。通常の挿入または記述による挿入の後には、ファクトは明示的に取り消される必要があります。論理挿入の後、挿入されたファクトは、ファクトを挿入したルールの条件が true でなくなると自動的に取り消されます。

```
insertLogical( new Applicant() );
```

- **delete**: デシジョンエンジンからオブジェクトを削除します。キーワード **retract** も DRL でサポートされており、同じアクションを実行しますが、DRL のコードでは、キーワード **insert** との整合性を考慮して **delete** が通常推奨されます。

```
delete( Applicant );
```

注記

DRL ルールアクションの詳細については、「DRL におけるルールアクション (THEN)」 を参照してください。

3. ルールのアクションコンポーネントをすべて定義したら、DRL デザイナーの右上のツールバーの **Validate** をクリックして、DRL ファイルの妥当性を確認します。ファイルの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、DRL ファイルの構文およびコンポーネントをすべて見直し、エラーが表示されなくなるまで再度、ファイルを検証します。
4. DRL デザイナーで **Save** をクリックして、設定した内容を保存します。

第19章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントのオプションに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合は、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれていることを確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

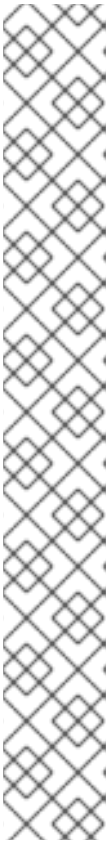
3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、Business Central 外に Maven または Java プロジェクトが作成されていない場合は作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例は、その他の DRL ルールの作成および実行方法 を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。

- **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルに読み込みます。
- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (任意) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける Red Hat Process Automation Manager 7.11 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。



注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

5. モデルクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイルの両方に表示されます。

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースを読み込む **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
```

```
    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

KIE Server でのルールの実行

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);
```

```
        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert Person into the session:
        KieCommands kieCommands = KieServices.Factory.get().getCommands();
        List<Command> commandList = new ArrayList<Command>();
        commandList.add(kieCommands.newInsert(p, "personReturnId"));

        // Fire all rules:
        commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
        BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

        // Use rule services client to send request:
        RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
        ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
        System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
}
```

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリーにおける) Java の実行例

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

第20章 その他の DRL ルールの作成および実行方法

Business Central インターフェイスで DRL ルールを作成し、管理する代わりに、Red Hat CodeReady Studio やその他の統合開発環境 (IDE) を使用して、Maven または Java プロジェクトの一部として DRL ルールファイルを作成できます。このスタンドアロンプロジェクトは、ナレッジ JAR (kJAR) 依存関係として、Business Central の既存の Red Hat Process Automation Manager プロジェクトに統合できます。スタンドアロンプロジェクトの DRL ファイルには、少なくても、必要な **package** 仕様、**import** リスト、および **rule** 定義が含まれる必要があります。グローバル変数や関数など、その他の DRL コンポーネントは任意です。DRL ルールに関連するすべてのデータオブジェクトは、スタンドアロンの DRL プロジェクトまたはデプロイメントに含まれる必要があります。

Maven または Java プロジェクトで実行可能なルールモデルを使用して、ビルド時に実行するルールセットの Java ベース表記を提供します。実行可能モデルは Red Hat Process Automation Manager の標準アセットパッケージングの代わりとなるもので、より効率的です。KIE コンテナーと KIE ベースの作成がより迅速にでき、DRL (Drools Rule Language) ファイルリストや他の Red Hat Process Automation Manager アセットが多い場合は、特に有効です。

20.1. RED HAT CODEREADY STUDIO での DRL ルールの作成および実行

Red Hat JBoss CodeReady Studio を使用して、ルールが含まれる DRL ファイルを作成し、Red Hat Process Automation Manager デシジョンサービスにファイルを統合します。DRL ルールを作成する方法は、デシジョンサービスに Red Hat CodeReady Studio を使用する場合や、同じワークフローを継続する場合に便利です。この方法を使用していない場合は、Red Hat Process Automation Manager の代わりに Business Central インターフェイスを使用して、DRL ファイルや、その他のルールアセットを作成することが推奨されます。

前提条件

- Red Hat カスタマーポータル から Red Hat CodeReady Studio をインストールしている。

手順

1. Red Hat CodeReady Studio で、**File → New → Project** をクリックします。
2. 開いた **New Project** ウィンドウで、**Drools → Drools Project** を選択し、**Next** をクリックします。
3. **Create a project and populate it with some example files to help you get started quickly** の 2 番目のアイコンをクリックします。**Next** をクリックします。
4. **Project name** を入力し、プロジェクトのビルドオプションで **Maven** ラジオボタンを選択します。GAV 値が自動的に生成されます。必要に応じて、プロジェクトに対してこの値を更新できます。
   - Group ID: **com.sample**
   - Artifact ID: **my-project**
   - Version: **1.0.0-SNAPSHOT**
5. **Finish** をクリックしてプロジェクトを作成します。
   これは、基本的なプロジェクト構造、クラスパス、サンプルルールを設定します。以下は、プロジェクト構造の概要です。

```
my-project
```

```
`-- src/main/java
  | `-- com.sample
  |    `-- DecisionTableTest.java
  |    `-- DroolsTest.java
  |    `-- ProcessTest.java
  |
`-- src/main/resources
  | `-- dtables
  |    `-- Sample.xls
  | `-- process
  |    `-- sample.bpmn
  | `-- rules
  |    `-- Sample.drl
  | `-- META-INF
  |
`-- JRE System Library
  |
`-- Maven Dependencies
  |
`-- Drools Library
  |
`-- src
  |
`-- target
  |
`-- pom.xml
```

以下の要素に注目してください。

- **src/main/resources** ディレクトリーの **Sample.drl** ルールファイル。これには、サンプルの **Hello World** ルールおよび **GoodBye** ルールが含まれます。
- **com.sample** パッケージの **src/main/java** ディレクトリーにある **DroolsTest.java** ファイル。**Sample.drl** ルールの実行には、**DroolsTest** クラスを使用できます。
- 実行するのに必要な JAR ファイルを含むカスタムのクラスパスとなる **Drools Library** ディレクトリー。

既存の **Sample.drl** ファイルおよび **DroolsTest.java** ファイルを必要に応じて新しい設定に変更するか、ルールファイルおよびオブジェクトファイルを新たに作成します。この手順では、ルールと Java オブジェクトを新たに作成します。

6. ルールが有効な Java オブジェクトを作成します。
この例では、**my-project/src/main/java/com.sample** に **Person.java** ファイルが作成されます。**Person** クラスには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。

```
public class Person {
  private String firstName;
  private String lastName;
  private Integer hourlyRate;
  private Integer wage;

  public String getFirstName() {
    return firstName;
  }
```

```
    public void setFirstName(String firstName) {
      this.firstName = firstName;
    }

    public String getLastName() {
      return lastName;
    }

    public void setLastName(String lastName) {
      this.lastName = lastName;
    }

    public Integer getHourlyRate() {
      return hourlyRate;
    }

    public void setHourlyRate(Integer hourlyRate) {
      this.hourlyRate = hourlyRate;
    }

    public Integer getWage(){
      return wage;
    }

    public void setWage(Integer wage){
      this.wage = wage;
    }
  }
```

7. **File → Save** をクリックして、ファイルを保存します。

8. **my-project/src/main/resources/rules** に、**.drl** 形式のルールファイルを作成します。DRL ファイルには、少なくともパッケージの指定と、(1 つまたは複数の) ルールで使用されるデータオブジェクトのインポートリストと、**when** 条件および **then** アクションを持つ 1 つ以上のルールが含まれます。
   以下の **Wage.drl** ファイルには、**Person** クラスをインポートする **Wage** ルールが含まれ、賃金および時給の値を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

9. **File → Save** をクリックして、ファイルを保存します。

10. メインクラスを作成し、Java オブジェクトを作成したディレクトリーに保存します。メインクラスは KIE ベースを読み込み、ルールを実行します。

注記

また、**DroolsTest.java** サンプルファイルと同様に、**main()** メソッドと **Person** クラスを 1 つの Java オブジェクトファイルに追加できます。

11. メインクラスに、KIE サービス、KIE コンテナー、および KIE セッションをインポートするのに必要な **import** 命令文を追加します。つぎに、KIE ベースを読み込み、ファクトを挿入し、ファクトモデルをルールに渡す **main()** メソッドからルールを実行します。
この例では、必要なインポートと **main()** メソッドを使用して、**my-project/src/main/java/com.sample** に **RulesTest.java** ファイルを作成します。

```
package com.sample;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class RulesTest {
  public static final void main(String[] args) {
    try {
      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.getKieClasspathContainer();
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

12. **File → Save** をクリックして、ファイルを保存します。

13. プロジェクトで DRL アセットをすべて作成して保存したあと、プロジェクトフォルダーを右クリックして、**Run As → Java Application** を選択してプロジェクトをビルドします。プロジェクトのビルドに失敗したら、CodeReady Studio の下部ウィンドウの **Problems** タブに記載されている問題に対応し、プロジェクトがビルドされるまでプロジェクトの検証を行います。

RUN AS → JAVA APPLICATION オプションが利用できない場合

プロジェクトを右クリックして、**Run As** を選択した場合に **Java Application** が選択肢にない場合は、**Run As → Run Configurations** に移動して **Java Application** を右クリックし、**New** をクリックします。次に、**Main** タブで、**Project** と、関連する **Main class** を参照して選択します。**Apply** をクリックし、**Run** をクリックしてプロジェクトをテストします。再度プロジェクトフォルダーを右クリックすると、**Java Application** オプションが表示されます。

Red Hat Process Automation Manager で既存のプロジェクトと新しいルールアセットを統合するには、ナレッジ JAR (KJAR) として新規プロジェクトをコンパイルし、Business Central でプロジェクトの **pom.xml** ファイルに、依存関係としてこのプロジェクトを追加します。Business Central でプロジェクト **pom.xml** にアクセスするには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

20.2. JAVA を使用した DRL ルールの作成および実行

Java オブジェクトを使用して、ルールを持つ DRL ファイルを作成し、オブジェクトを Red Hat Process Automation Manager デシジョンサービスに統合します。DRL ルールを作成する方法は、デシジョンサービスに外部 Java オブジェクトを使用している場合や、同じワークフローを継続する場合に便利です。この方法を使用していない場合は、Red Hat Process Automation Manager の代わりに Business Central インターフェイスを使用して、DRL ファイルや、その他のルールアセットを作成することが推奨されます。

手順

1. ルールが有効な Java オブジェクトを作成します。
   この例では、**my-project** ディレクトリーに **Person.java** ファイルが作成されます。**Person** クラスには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。

```
public class Person {
  private String firstName;
  private String lastName;
  private Integer hourlyRate;
  private Integer wage;

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public Integer getHourlyRate() {
```

```
    return hourlyRate;
  }

  public void setHourlyRate(Integer hourlyRate) {
    this.hourlyRate = hourlyRate;
  }

  public Integer getWage(){
    return wage;
  }

  public void setWage(Integer wage){
    this.wage = wage;
  }
}
```

2. **my-project** ディレクトリーに、**.drl** 形式のルールファイルを作成します。DRL ファイルには、少なくともパッケージの指定 (該当する場合) と、(1 つまたは複数の) ルールで使用されるデータオブジェクトのインポートリストと、**when** 条件および **then** アクションを持つ 1 つ以上のルールが含まれます。
以下の **Wage.drl** ファイルには、賃金と時給を計算し、その結果に基づいてメッセージを表示する **Wage** ルールが含まれています。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

3. メインクラスを作成し、Java オブジェクトを作成したディレクトリーに保存します。メインクラスは KIE ベースを読み込み、ルールを実行します。

4. メインクラスに、KIE サービス、KIE コンテナー、および KIE セッションをインポートするのに必要な **import** 命令文を追加します。つぎに、KIE ベースを読み込み、ファクトを挿入し、ファクトモデルをルールに渡す **main()** メソッドからルールを実行します。
この例では、必要なインポートと **main()** メソッドを使用して、**my-project** に **RulesTest.java** ファイルを作成します。

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class RulesTest {
  public static final void main(String[] args) {
    try {
      // Load the KIE base:
```

```
        KieServices ks = KieServices.Factory.get();
        KieContainer kContainer = ks.getKieClasspathContainer();
        KieSession kSession = kContainer.newKieSession();

        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

5. Red Hat カスタマーポータル から ZIP ファイル **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**my-project/pam-engine-jars/** で展開します。

6. **my-project/META-INF** ディレクトリーに、以下の内容の **kmodule.xml** メタデータファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

この **kmodule.xml** ファイルは、KIE ベースへのリソースを選択し、セッションを設定する KIE モジュールの記述子です。このファイルを使用すると、KIE ベースを 1 つ以上定義して設定し、特定の KIE ベースの特定の **packages** から DRL ファイルを含めることができます。各 KIE ベースから KIE セッションを 1 つ以上作成することもできます。

次の例は、より高度な **kmodule.xml** ファイルを表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
      <fileLogger file="debugInfo" threaded="true" interval="10" />
```

```
    <workItemHandlers>
     <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
    </workItemHandlers>
    <listeners>
     <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
     <agendaEventListener type="org.domain.FirstAgendaListener" />
     <agendaEventListener type="org.domain.SecondAgendaListener" />
     <processEventListener type="org.domain.ProcessListener" />
    </listeners>
   </ksession>
  </kbase>
</kmodule>
```

この例は、KIE ベースを 2 つ定義します。KIE ベース **KBase1** から 2 つの KIE セッションをインスタンス化し、**KBase2** から KIE セッションを 1 つインスタンス化します。**KBase2** の KIE セッションは **ステートレス** な KIE セッションですが、これは 1 つ前の KIE セッションで呼び出されたデータ (1 つ前のセッションの状態) が、セッションの呼び出しと呼び出しの間で破棄されることを示しています。ルールアセットの特定の **パッケージ** は両方 KIE ベースに含まれます。この方法でパッケージを指定した場合は、指定したパッケージを反映するディレクトリー構造で DRL ファイルを整理する必要があります。

7. Java オブジェクトですべての DRL アセットを作成して保存したあと、コマンドラインで **my-project** ディレクトリーに移動し、以下のコマンドを実行して Java ファイルをビルドします。**RulesTest.java** を、Java のメインクラスの名前に置き換えます。

```
javac -classpath "./pam-engine-jars/*:." RulesTest.java
```

ビルドに失敗したら、コマンドラインのエラーメッセージに記載されている問題に対応し、エラーが表示されなくなるまで Java オブジェクトの妥当性確認を行います。

8. Java ファイルが問題なくビルトできたら、以下のコマンドを実行してローカルでルールを実行します。**RulesTest** を、Java のメインクラスのプリフィックスに置き換えます。

```
java -classpath "./pam-engine-jars/*:." RulesTest
```

9. ルールを見直して、適切に実行したことを確認し、Java ファイルで必要な変更に対応します。

Red Hat Process Automation Manager で既存のプロジェクトと新しいルールアセットを統合するには、ナレッジ JAR (KJAR) として新規 Java プロジェクトをコンパイルし、Business Central でプロジェクトの **pom.xml** ファイルに、依存関係としてこのプロジェクトを追加します。Business Central でプロジェクト **pom.xml** にアクセスするには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

20.3. MAVEN を使用した DRL ルールの作成および実行

Maven アーキタイプを使用して、ルールを持つ DRL ファイルを作成し、アーキタイプを Red Hat Process Automation Manager デシジョンサービスに統合します。DRL ルールを作成する方法は、デシジョンサービスに外部 Maven アーキタイプを使用している場合や、同じワークフローを継続する場合に便利です。この方法を使用していない場合は、Red Hat Process Automation Manager の代わりに Business Central インターフェイスを使用して、DRL ファイルや、その他のルールアセットを作成することが推奨されます。

手順

1. Maven アーキタイプを作成するディレクトリーに移動して、次のコマンドを実行します。

```
mvn archetype:generate -DgroupId=com.sample.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

これにより、**my-app** という名前のディレクトリーが、以下の構造で作成されます。

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- sample
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- sample
                    `-- app
                        `-- AppTest.java
```

my-app ディレクトリーには、以下の重要なコンポーネントが含まれます。

- アプリケーションソースを保存する **src/main** ディレクトリー
- テストソースを保存する **src/test** ディレクトリー
- プロジェクト設定ファイル **pom.xml**

2. Maven アーキタイプに、ルールが有効な Java オブジェクトを作成します。
この例では、**my-app/src/main/java/com/sample/app** ディレクトリーに **Person.java** ファイルが作成されます。**Person** クラスには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。

```
package com.sample.app;

 public class Person {

  private String firstName;
  private String lastName;
  private Integer hourlyRate;
  private Integer wage;

  public String getFirstName() {
   return firstName;
  }

  public void setFirstName(String firstName) {
   this.firstName = firstName;
  }

  public String getLastName() {
   return lastName;
```

```
    }

    public void setLastName(String lastName) {
      this.lastName = lastName;
    }

    public Integer getHourlyRate() {
      return hourlyRate;
    }

    public void setHourlyRate(Integer hourlyRate) {
      this.hourlyRate = hourlyRate;
    }

    public Integer getWage(){
      return wage;
    }

    public void setWage(Integer wage){
      this.wage = wage;
    }
  }
```

3. **my-app/src/main/resources/rules** に、**.drl** 形式のルールファイルを作成します。DRL ファイルには、少なくともパッケージの指定と、(1 つまたは複数の) ルールで使用されるデータオブジェクトのインポートリストと、**when** 条件および **then** アクションを持つ 1 つ以上のルールが含まれます。
以下の **Wage.drl** ファイルには、**Person** クラスをインポートする **Wage** ルールが含まれ、賃金および時給の値を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample.app;

import com.sample.app.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

4. **my-app/src/main/resources/META-INF** ディレクトリーに、以下の内容の **kmodule.xml** メタデータファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

この **kmodule.xml** ファイルは、KIE ベースへのリソースを選択し、セッションを設定する KIE モジュールの記述子です。このファイルを使用すると、KIE ベースを 1 つ以上定義して設定し、特定の KIE ベースの特定の **packages** から DRL ファイルを含めることができます。各 KIE ベー

スから KIE セッションを 1 つ以上作成することもできます。

次の例は、より高度な **kmodule.xml** ファイルを表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
      <fileLogger file="debugInfo" threaded="true" interval="10" />
      <workItemHandlers>
        <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
      </workItemHandlers>
      <listeners>
        <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
        <agendaEventListener type="org.domain.FirstAgendaListener" />
        <agendaEventListener type="org.domain.SecondAgendaListener" />
        <processEventListener type="org.domain.ProcessListener" />
      </listeners>
    </ksession>
  </kbase>
</kmodule>
```

この例は、KIE ベースを 2 つ定義します。KIE ベース **KBase1** から 2 つの KIE セッションをインスタンス化し、**KBase2** から KIE セッションを 1 つインスタンス化します。**KBase2** の KIE セッションは **ステートレス** な KIE セッションですが、これは 1 つ前の KIE セッションで呼び出されたデータ (1 つ前のセッションの状態) が、セッションの呼び出しと呼び出しの間で破棄されることを示しています。ルールアセットの特定の **パッケージ** は両方 KIE ベースに含まれます。この方法でパッケージを指定した場合は、指定したパッケージを反映するディレクトリー構造で DRL ファイルを整理する必要があります。

5. **my-app/pom.xml** 設定ファイルで、アプリケーションが要求するライブラリーを指定します。Red Hat Process Automation Manager の依存関係と、アプリケーションの **group ID**、**artifact ID**、および **version** (GAV) を提供します。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.sample.app</groupId>
<artifactId>my-app</artifactId>
<version>1.0.0</version>
<repositories>
  <repository>
    <id>jboss-ga-repository</id>
    <url>http://maven.repository.redhat.com/ga/</url>
  </repository>
```

```
</repositories>
<dependencies>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>VERSION</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>VERSION</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>
```

Red Hat Process Automation Manager における Maven 依存関係および BOM (Bill of Materials) の詳細は、What is the mapping between Red Hat Process Automation Manager and Maven library version? を参照してください。

6. **my-app/src/test/java/com/sample/app/AppTest.java** の **testApp** メソッドを使用してルールをテストします。Maven によって、**AppTest.java** ファイルがデフォルトで作成されます。

7. **AppTest.java** ファイルで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするのに必要な **import** 命令文を追加します。次に、KIE ベースを読み込み、ファクトを挿入し、ファクトモデルをルールに渡す **testApp()** メソッドからルールを実行します。

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public void testApp() {

  // Load the KIE base:
  KieServices ks = KieServices.Factory.get();
  KieContainer kContainer = ks.getKieClasspathContainer();
  KieSession kSession = kContainer.newKieSession();

  // Set up the fact model:
  Person p = new Person();
  p.setWage(12);
  p.setFirstName("Tom");
  p.setLastName("Summers");
  p.setHourlyRate(10);

  // Insert the person into the session:
  kSession.insert(p);

  // Fire all rules:
```

```
    kSession.fireAllRules();
    kSession.dispose();
  }
```

8. Maven アーキタイプにすべての DRL アセットを作成して保存したあと、コマンドラインで **my-app** ディレクトリーに移動し、以下のコマンドを実行してファイルを作成します。

```
mvn clean install
```

ビルドに失敗したら、コマンドラインのエラーメッセージに記載されている問題に対応し、ビルドに成功するまでファイルの妥当性確認を行います。

9. ファイルが問題なくビルドできたら、以下のコマンドを実行してローカルでルールを実行します。**com.sample.app** をパッケージ名に置き換えます。

```
mvn exec:java -Dexec.mainClass="com.sample.app"
```

10. ルールを見直して、適切に実行したことを確認し、ファイルで必要な変更に対応します。

Red Hat Process Automation Manager で既存のプロジェクトと新しいルールアセットを統合するには、ナレッジ JAR (KJAR) として新規 Maven プロジェクトをコンパイルし、Business Central でプロジェクトの **pom.xml** ファイルに、依存関係としてこのプロジェクトを追加します。Business Central でプロジェクト **pom.xml** にアクセスするには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

第21章 RED HAT PROCESS AUTOMATION MANAGER の IDE 向けのデシジョン例

Red Hat Process Automation Manager は、統合開発環境 (IDE: integrated development environment) にインポートできるように Java クラスとして配信されるデシジョン例を提供します。これらの例は、デシジョンエンジン機能をさらに理解するために使用する目的か、Red Hat Process Automation Manager プロジェクトに定義するデシジョンの参考として使用してください。

以下のデシジョンセットの例は、Red Hat Process Automation Manager で利用可能な例の一部です。

- **Hello World の例**: 基本的なルール実行や、デバッグ出力の使用方法を例示します。
- **状態の例**: ルールの顕著性やアジェンダグループを使用した前向き連鎖や競合解決を例示します。
- **フィボナッチの例**: ルールの顕著性を使用した再帰や競合解決を例示します。
- **銀行の例**: パターン一致、基本的なソート、計算を例示します。
- **ペットショップの例**: ルールアジェンダグループ、グローバル変数、コールバック、GUI 統合を例示します。
- **数独の例**: 複雑なパターン一致、問題解決、コールバック、GUI 統合を例示します。
- **House of Doom の例**: 後向き連鎖と再帰を例示します。

注記

Red Hat ビルドの OptaPlanner で提供される最適化の例は、**Red Hat ビルドの OptaPlanner のスタートガイド** を参照してください。

21.1. IDE での RED HAT PROCESS AUTOMATION MANAGER のデシジョン例のインポートおよび実行

Red Hat Process Automation Manager のデシジョン例を統合開発環境 (IDE) にインポートして実行し、ルールとコードがどのように機能するかをチェックできます。これらの例は、デシジョンエンジン機能をさらに理解するために使用する目的か、Red Hat Process Automation Manager プロジェクトに定義するデシジョンの参考として使用してください。

前提条件

- Java 8 以降がインストールされている。
- Maven 3.5.x 以降がインストールされている。
- Red Hat CodeReady Studio などの IDE がインストールされている。

手順

1. Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**/rhpam-7.11.0-sources** などの一時的なディレクトリーに展開します。

2. IDE を開き、**File → Import → Maven → Existing Maven Projects**を選択するか、同等のオプションを選択して、Maven プロジェクトをインポートします。

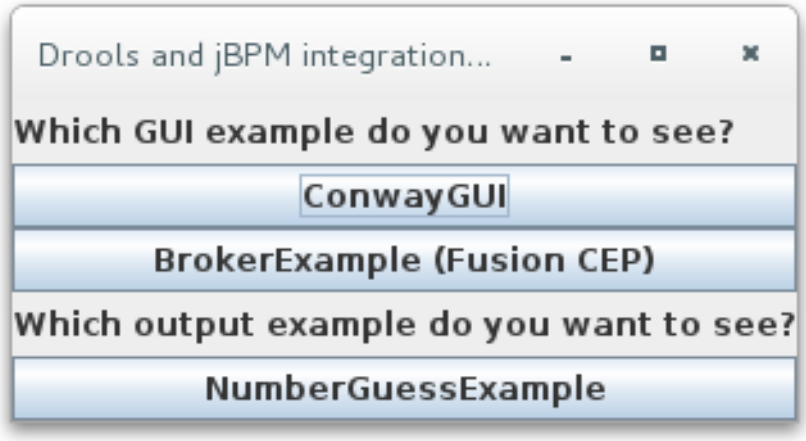
3. **Browse** をクリックして、**~/rhpam-7.11.0-sources/src/drools-$VERSION/drools-examples** (または、Conway の Game of Life の例の場合は、**~/rhpam-7.11.0-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples**) に移動して、プロジェクトをインポートします。

4. 実行するパッケージ例に移動して、**main** メソッドが含まれる Java クラスを検索します。

5. Java クラスを右クリックし、**Run As → Java Application** を選択して例を実行します。
基本的なユーザーインターフェイスですべての例を実行するには、Main クラス **org.drools.examples** の **DroolsExamplesApp.java** クラス (または Conway の Game of Life の場合は **DroolsJbpmIntegrationExamplesApp.java** クラス) を実行します。

図21.1 drools-examples (DroolsExamplesApp.java) 内のすべての例のインターフェイス

図21.2 droolsjbpm-integration-examples (DroolsJbpmIntegrationExamplesApp.java) のすべての例のインターフェイス



21.2. HELLO WORLD の例のデシジョン (基本ルールおよびデバッグ)

Hello World のデシジョンセットの例では、オブジェクトをデシジョンエンジンのワーキングメモリーに挿入する方法、ルールを使用してオブジェクトを一致させる方法、エンジンの内部アクティビティーを追跡するロギングの設定方法を説明します。

以下は、Hello World の例の概要です。

- **名前**: **helloworld**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.helloworld.HelloWorldExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.helloworld.HelloWorld.drl**
- **目的**: 基本的なルール実行とデバッグ出力の使用方法を例示します。

Hello World の例では、KIE セッションが生成されて、ルールの実行が可能になります。すべてのルールは、実行するのに KIE セッションが必要です。

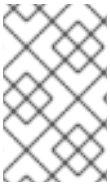
ルール実行の KIE セッション

```
KieServices ks = KieServices.Factory.get(); (1)
KieContainer kc = ks.getKieClasspathContainer(); (2)
KieSession ksession = kc.newKieSession("HelloWorldKS"); (3)
```

(1) **KieServices** ファクトリーを取得します。これは、アプリケーションがデシジョンエンジンとの対話に使用する主なインターフェイスです。

(2) プロジェクトクラスパスから **KieContainer** を作成します。これは、**KieModule** で **KieContainer** を設定してインスタンス化し、そこから **/META-INF/kmodule.xml** ファイルを検出します。

(3) **/META-INF/kmodule.xml** ファイルに定義された KIE セッション設定 **"HelloWorldKS"** をもとに **KieSession** を作成します。



注記

Red Hat Process Automation Manager プロジェクトのパッケージ化に関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

Red Hat Process Automation Manager には、内部エンジンアクティビティーを公開するイベントモデルがあります。デフォルトのデバッグリスナー **DebugAgendaEventListener** と **DebugRuleRuntimeEventListener** により、デバッグイベント情報が **System.err** の出力に表示されます。**KieRuntimeLogger** では、実行監査と、グラフィックビューワーで確認可能な結果が提供されます。

リスナーと監査ロガーのデバッグ

```
// Set up listeners.
ksession.addEventListener( new DebugAgendaEventListener() );
ksession.addEventListener( new DebugRuleRuntimeEventListener() );

// Set up a file-based audit logger.
KieRuntimeLogger logger = KieServices.get().getLoggers().newFileLogger( ksession,
"./target/helloworld" );

// Set up a ThreadedFileLogger so that the audit view reflects events while debugging.
KieRuntimeLogger logger = ks.getLoggers().newThreadedFileLogger( ksession, "./target/helloworld",
1000 );
```

ロガーは、**Agenda** と **RuleRuntime** リスナーにビルドされる特別な実装です。デシジョンエンジンが実行を終えると、**logger.close()** が呼び出されます。

この例では、**"Hello World"** というメッセージを含む **Message** オブジェクトを作成し、ステータス **HELLO** を **KieSession** に挿入して、**fireAllRules()** でルールを実行します。

データの挿入および実行

```
// Insert facts into the KIE session.
final Message message = new Message();
message.setMessage( "Hello World" );
message.setStatus( Message.HELLO );
ksession.insert( message );

// Fire the rules.
ksession.fireAllRules();
```

ルール実行は、データモデルを使用して、**KieSession** への出入力としてデータを渡します。この例のデータモデルには **message** (**String**) と **status** (**HELLO** または **GOODBYE**) の 2 つのフィールドが含まれます。

データモデルクラス

```
public static class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String         message;
```

```
    private int           status;
    ...
}
```

この 2 つのルールは、**src/main/resources/org/drools/examples/helloworld/HelloWorld.drl** ファイルに配置されます。

"Hello World" ルールの **when** 条件では、ステータスが **Message.HELLO** の KIE セッションに、**Message** オブジェクトが挿入されるたびに、このルールをアクティベートすると記述しています。さらに、変数のバインドが 2 つ作成されます (**message** 変数を **message** 属性に、**m** 変数を一致する **Message** オブジェクト自体にバインド)。

ルールの **then** アクションは、バインドされた変数 **message** のコンテンツを **System.out** に出力するよう指定し、続いて **m** にバインドされている **Message** オブジェクトの **message** と **status** 属性値を変更します。このルールは **modify** ステートメントを使用して、1 つのステートメントに割り当てブロックを適用し、ブロックの最後にデシジョンエンジンにこの変更について通知します。

"Hello World" のルール

```
rule "Hello World"
  when
    m : Message( status == Message.HELLO, message : message )
  then
    System.out.println( message );
    modify ( m ) { message = "Goodbye cruel world",
                   status = Message.GOODBYE };
end
```

"Good Bye" ルールは、ステータスが **Message.GOODBYE** の **Message** オブジェクトと一致する点を除き、**"Hello World"** ルールによく似ています。

"Good Bye" ルール

```
rule "Good Bye"
  when
    Message( status == Message.GOODBYE, message : message )
  then
    System.out.println( message );
end
```

この例を実行するには、**org.drools.examples.helloworld.HelloWorldExample** クラスを IDE で Java アプリケーションとして実行します。このルールは **System.out** に、デバッグリスナーは **System.err** に書き込み、監査ロガーは **target/helloworld.log** のログファイルを作成します。

IDE コンソールの System.out 出力

```
Hello World
Goodbye cruel world
```

IDE コンソールでの System.err の出力

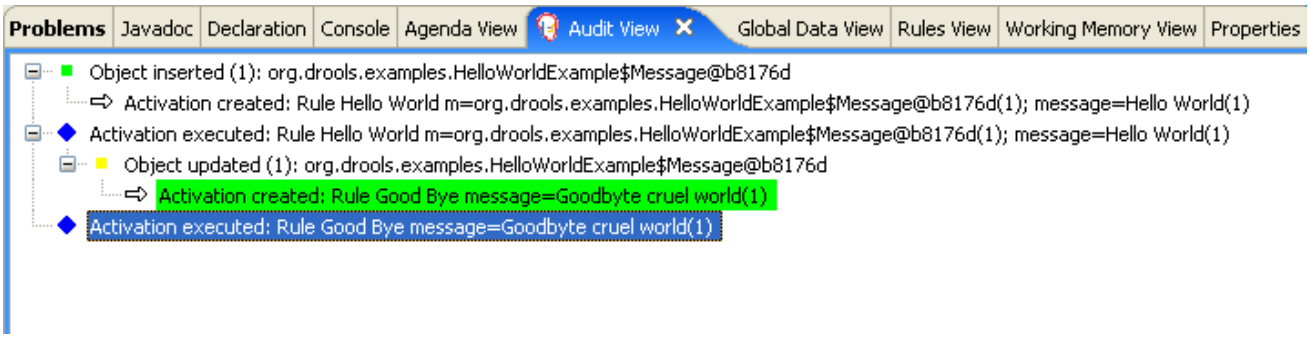
```
==>[ActivationCreated(0): rule=Hello World;
                  tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectInserted: handle=
```

```
[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
            object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[BeforeActivationFired: rule=Hello World;
             tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
==>[ActivationCreated(4): rule=Good Bye;
             tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectUpdated: handle=
[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
          old_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96;
          new_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[AfterActivationFired(0): rule=Hello World]
[BeforeActivationFired: rule=Good Bye;
             tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[AfterActivationFired(4): rule=Good Bye]
```

この例の実行フローをさらに理解するには、**target/helloworld.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**Audit view** で、オブジェクトが挿入され、**"Hello World"** ルールのアクティベーションが作成されます。次に、このアクティベーションが実行され、**Message** オブジェクトを更新して、**"Good Bye"** ルールのアクティベーションをトリガーします。最後に、**"Good Bye"** ルールが実行します。**Audit View** でイベントが選択されると、この例の **"Activation created"** イベントである元のイベントが緑色にハイライトされます。

図21.3 Hello World の例の監査ビュー



21.3. 状態の例のデシジョン (前向き連鎖および競合解決)

状態の例のディジョンセットでは、デシジョンエンジンが前向き連鎖と、ワーキングメモリー内のファクトへの変更をどのように使用してルールの実行競合を順番に解決していくのかを例示します。この例では、ルールで定義可能な顕著性の値またはアジェンダグループを使用して競合を解決することにフォーカスします。

以下は、状態の例の概要です。

- **名前**: **state**
- Main **クラス**: (**src/main/java** 内の) **org.drools.examples.state.StateExampleUsingSalience**、**org.drools.examples.state.StateExampleUsingAgendaGroup**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション

- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.state.*.drl**

- **目的**: ルールの顕著性やアジェンダグループを使用した前向き連鎖や競合解決を例示します。

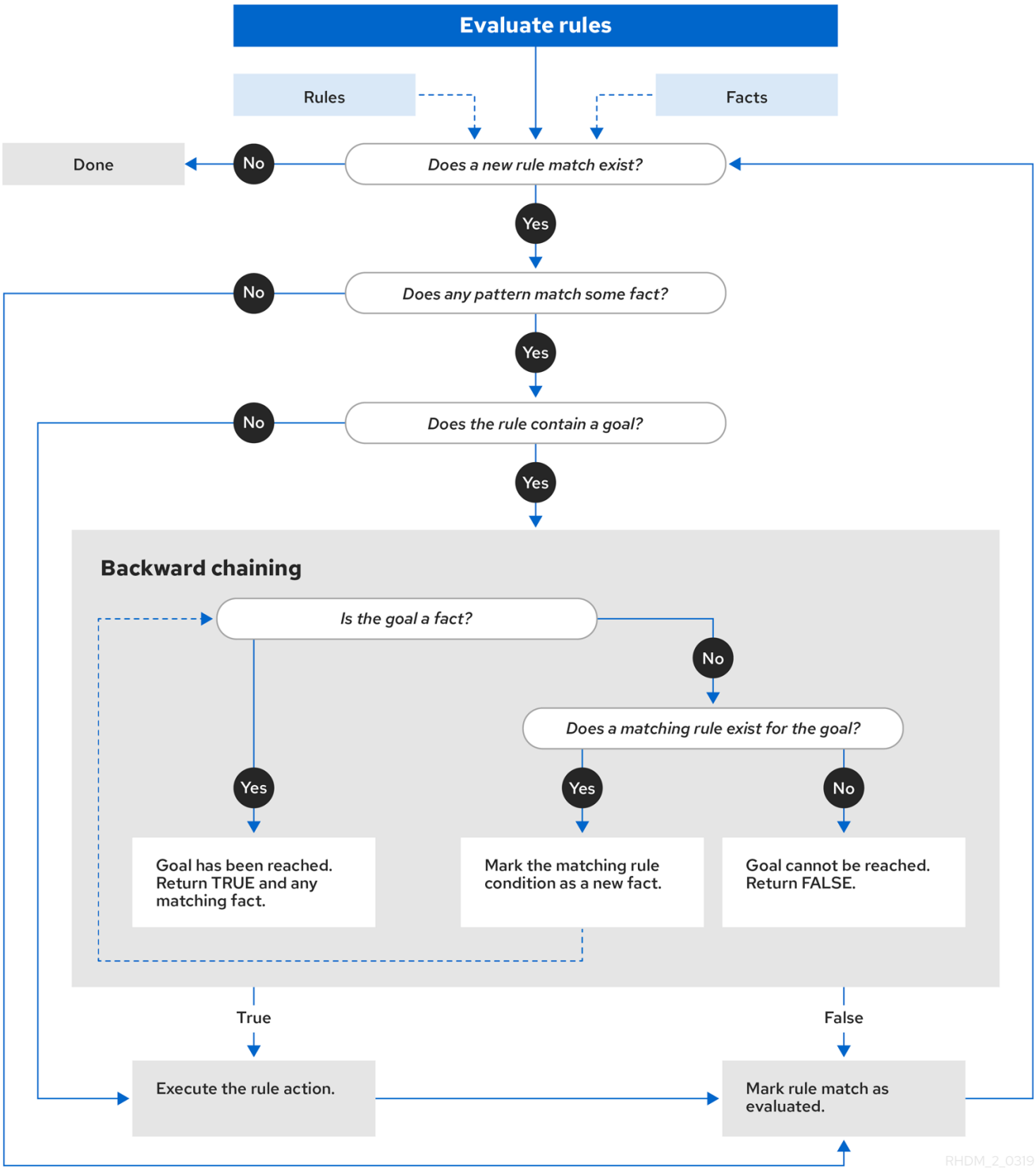
前向き連鎖のルールシステムは、ディジョンエンジンのワーキングメモリーにあるファクトで開始して、そのファクトへの変更に反応するデータ駆動型のシステムです。オブジェクトがワーキングメモリーに挿入されると、その変更の結果として True となるルールの条件はすべて、アジェンダによって実行されるようにスケジュールされます。

反対に、後向き連鎖のルールシステムは、しばしば再帰を使用して、デシジョンエンジンが満たそうとする結論から開始する目的駆動型のシステムです。システムが結論または目的に到達できない場合は、サブとなる目的、つまり、現在の目的の一部を完了する結論を検索します。システムは、最初の結論が満たされるか、すべてのサブとなる目的が満たされるまでこのプロセスを続行します。

Red Hat Process Automation Manager のデシジョンエンジンは、前向き連鎖と後向き連鎖の両方を使用してルールを評価します。

以下の図は、デシジョンエンジンが、ロジックフローで後向き連鎖のセグメントと、前向き連鎖全体を使用してルールを評価する方法を例示します。

図21.4 前向き連鎖と後向き連鎖を使用したルール評価のロジック



状態の例では、**State** クラスごとに、名前や現在の状態のフィールドが含まれます (**org.drools.examples.state.State** のクラス参照)。以下の状態は、各プロジェクトで考えられる 2 つの状態です。

- **NOTRUN**
- **FINISHED**

State クラス

```
public class State {
```

```
    public static final int NOTRUN   = 0;
    public static final int FINISHED = 1;

    private final PropertyChangeSupport changes =
        new PropertyChangeSupport( this );

    private String name;
    private int    state;

    ... setters and getters go here...
}
```

状態の例には、同じ例が 2 つのバージョンとして提供されており、それぞれルール実行の競合を解決します。

- ルールの顕著性を使用して競合を解決する **StateExampleUsingSalience** バージョン
- ルールアジェンダグループを使用して競合を解決する **StateExampleUsingAgendaGroups** バージョン

状態の例のバージョンはいずれも、**A**、**B**、**C**、および **D** の 4 つの **State** オブジェクトを使用します。最初に、それぞれの状態は、**NOTRUN** に設定されます。NOTRUN は、例が使用するコンストラクターのデフォルト値です。

顕著性を使用した状態の例

状態の例の **StateExampleUsingSalience** バージョンでは、ルールで顕著性の値を使用し、ルール実行の競合を解決します。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。

この例では、各 **State** インスタンスを KIE セッションに挿入して、**fireAllRules()** を呼び出します。

顕著性の状態例の実行

```
final State a = new State( "A" );
final State b = new State( "B" );
final State c = new State( "C" );
final State d = new State( "D" );

ksession.insert( a );
ksession.insert( b );
ksession.insert( c );
ksession.insert( d );

ksession.fireAllRules();

// Dispose KIE session if stateful (not required if stateless).
ksession.dispose();
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.state.StateExampleUsingSalience** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの顕著性の状態例の出力

```
A finished
B finished
C finished
D finished
```

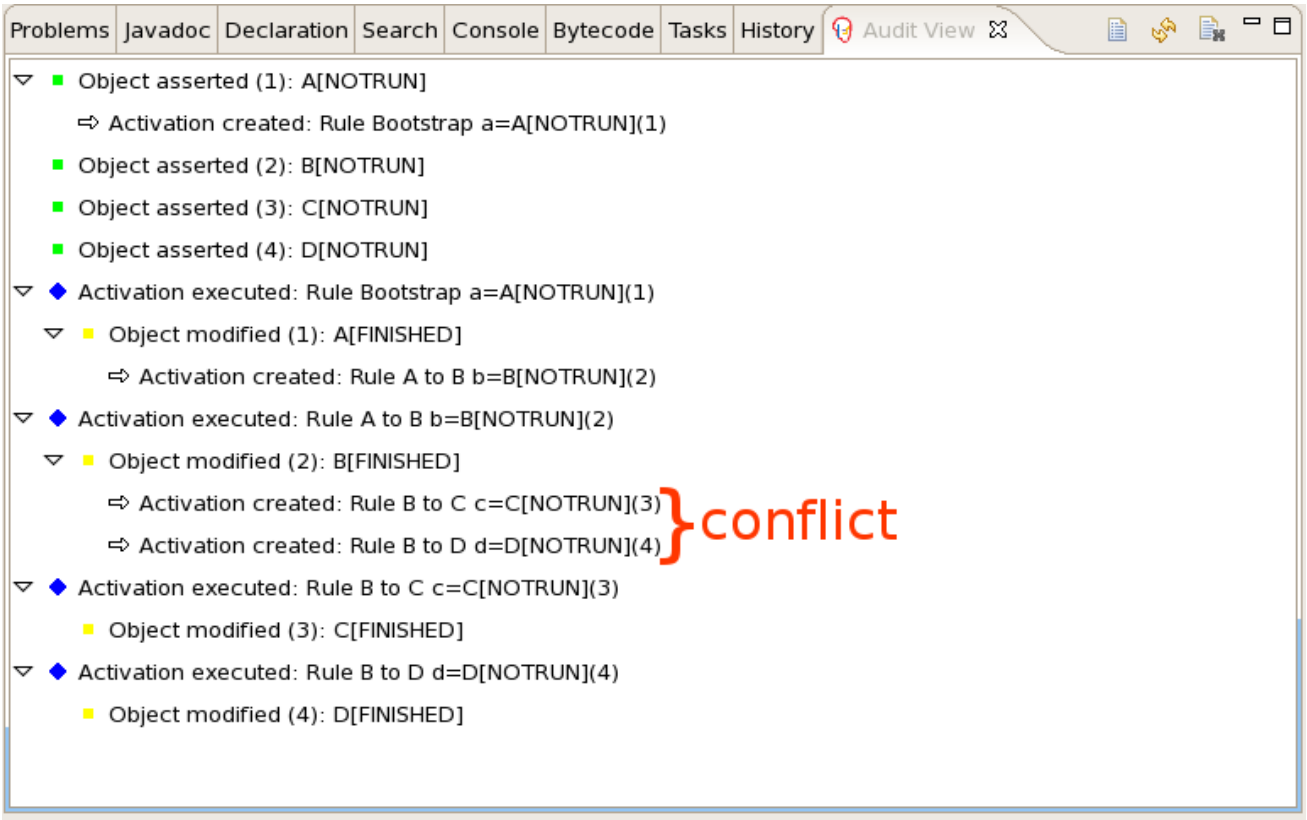
4 つのルールが存在します。

まず、**"Bootstrap"** ルールが実行され、**A** の状態が **FINISHED** に設定されます。次に、**B** の状態が **FINISHED** に変更されます。オブジェクト **C** と **D** はいずれも **B** に依存するため競合が発生しますが、顕著性の値で解決されます。

この例の実行フローをさらに理解するには、**target/state.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**Audit View** は、状態が **NOTRUN** のオブジェクト **A** のアサーションが **"Bootstrap"** ルールをアクティベートしますが、他のオブジェクトのアサーションはすぐに有効になりません。

図21.5 顕著性の状態例の監査ビュー



顕著性の状態例の "Bootstrap" ルール

```
rule "Bootstrap"
  when
    a : State(name == "A", state == State.NOTRUN )
  then
    System.out.println(a.getName() + " finished" );
    a.setState( State.FINISHED );
end
```

"Bootstrap" ルールを実行すると、**A** の状態が **FINISHED** に変わり、ルール **"A to B"** をアクティベートします。

顕著性の状態例の "A to B" ルール

```
rule "A to B"
  when
    State(name == "A", state == State.FINISHED )
    b : State(name == "B", state == State.NOTRUN )
  then
    System.out.println(b.getName() + " finished" );
    b.setState( State.FINISHED );
end
```

"A to B" ルールを実行すると、**B** の状態を **FINISHED** に変更し、**"B to C"** と **"B to D"** の両方のルールをアクティベートして、これらのアクティベーションをデシジョンエンジンアジェンダに配置します。

顕著性の状態例の "B to C" および "B to D" ルール

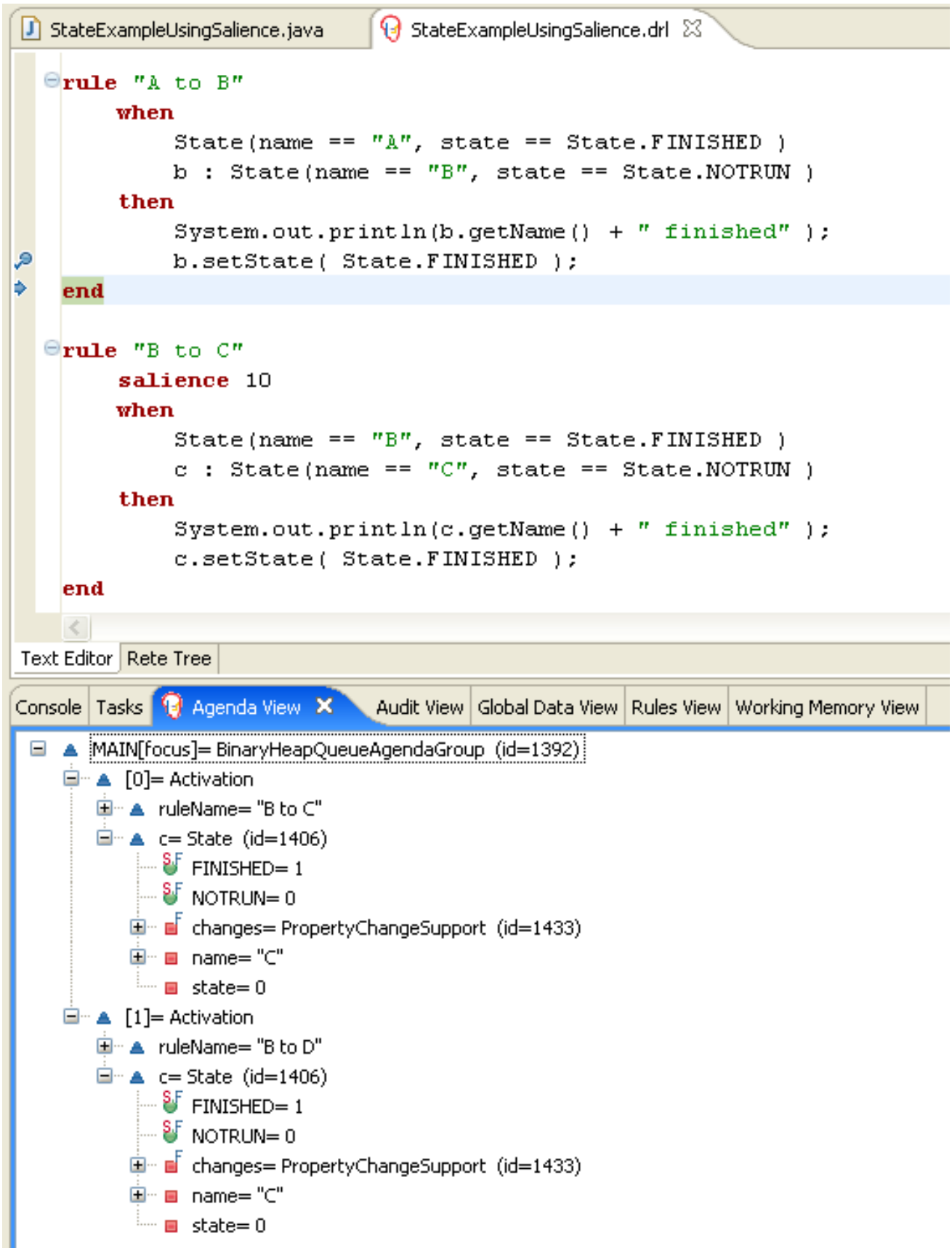
```
rule "B to C"
    salience 10
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
end

rule "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

この時点から、両方のルールが実行される可能性があるため、これらのルールは競合しています。競合解決ストラテジーを使用すると、デシジョンエンジンアジェンダがどのルールを実行するかを決定できます。**"B to C"** は、顕著性の値が高いため (デフォルトの顕著性の値 **0** に対して **10**) 先に実行し、オブジェクト **C** の状態が **FINISHED** に変更されます。

IDE の **Audit View** では、ルール **"A to B"** の **State** オブジェクトが変更され、2 つのアクティベーションが競合する結果になることが分かります。

IDE で **Agenda View** を使用して、デシジョンエンジンアジェンダの状態を調査できます。この例では **Agenda View** で、ルール **"A to B"** のブレークポイントと、2 つの競合するルールを持つアジェンダの状態が分かります。最後にルール **"B to D"** が実行され、オブジェクト **D** の状態が **FINISHED** に変更されます。

図21.6 顕著性の状態例のアジェンダビュー



アジェンダグループを使用した状態の例

状態の例の **StateExampleUsingAgendaGroups** バージョンでは、ルールでアジェンダグループを使用し、ルール実行における競合を解決します。アジェンダグループを使用すると、デシジョンエンジンアジェンダが分割され、ルールのグループの実行に対してこれまで以上に制御ができるようになります。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

最初は、ワーキングメモリーは、アジェンダグループ **MAIN** にフォーカスを当てます。アジェンダグループのルールは、グループがこのフォーカスを受けた場合のみ実行されます。**setFocus()** メソッドか、**auto-focus** ルール属性を使用してフォーカスを設定できます。**auto-focus** 属性を使用すると、ルールが一致してアクティベートされた場合のみ、ルールにアジェンダグループのフォーカスが自動的に当てられます。

この例では、**auto-focus** 属性を使用すると **"B to D"** の前に **"B to C"** ルールを実行できます。

アジェンダグループの状態例のルール "B to C"

```
rule "B to C"
    agenda-group "B to C"
    auto-focus true
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "B to D" ).setFocus();
end
```

ルール **"B to C"** は、アジェンダグループ **"B to D"** の **setFocus()** を呼び出し、アクティブなルールを実行できるようにします。その後にルール **"B to D"** が実行できるようになります。

アジェンダグループの状態例のルール "B to D"

```
rule "B to D"
    agenda-group "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.state.StateExampleUsingAgendaGroups** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます (状態の例の顕著性バージョンと同じ)。

IDE コンソールでのアジェンダグループの状態例の出力

```
A finished
B finished
C finished
D finished
```

状態の例の含まれる動的なファクト

状態の例に含まれる主なコンセプトとしては、他にも **PropertyChangeListener** オブジェクトを実装するオブジェクトに基づいて **動的ファクト** を使用するというものがあります。デシジョンエンジンがファクトプロパティーへの変更を確認し、対応するためには、アプリケーションがデシジョンエンジン

に対して、変更があったことを通知する必要があります。**modify** ステートメントを使用して、このコミュニケーションをルールで明示的に設定するか、JavaBeans 仕様で定義されているようにファクトが **PropertyChangeSupport** インターフェイスを実装するように指定することで暗黙的に設定できます。

この例は、ルールで **modify** ステートメントを明示的に指定しなくても良いように **PropertyChangeSupport** インターフェイスを使用する方法が示されています。このインターフェイスを使用するには、**org.drools.example.State** クラスと同じ方法で、ファクトに **PropertyChangeSupport** が実装されていることを確認し、DRL ルールファイルで以下のコードを使用して、これらのファクトでプロパティー変更がないかをリッスンするようにデシジョンエンジンを設定してください。

動的ファクトの宣言

```
declare type State
  @propertyChangeSupport
end
```

PropertyChangeListener オブジェクトを使用する場合に、各セッターは通知用に追加のコードを実装する必要があります。たとえば、**state** の以下のセッターは **org.drools.examples** のクラスに含まれます。

PropertyChangeSupport のセッター例

```
public void setState(final int newState) {
    int oldState = this.state;
    this.state = newState;
    this.changes.firePropertyChange( "state",
                                     oldState,
                                     newState );
}
```

21.4. フィボナッチの例のデシジョン (再帰および競合解決)

フィボナッチの例のデシジョンセットでは、デシジョンエンジンが再帰をどのように使用してルールの実行競合を順番に解決していくのかを例示します。この例では、ルールで定義可能な顕著性の値を使用して競合を解決することにフォーカスします。

以下は、フィボナッチの例の概要です。

- **名前**: **フィボナッチ**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.fibonacci.FibonacciExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.fibonacci.Fibonacci.drl**
- **目的**: ルールの顕著性を使用した再帰や競合解決を例示します。

フィボナッチ数は、0 または 1 で開始する数列です。0、1、1、2、3、5、8、13、21、34、55、89、144、233、377、610、987、1597、2584、4181、6765、10946 などのように、2 つの先行する数を足すことにより、次にくるフィボナッチ数が求められます。

フィボナッチの例では、**Fibonacci** のファクトクラスを 1 つ使用し、このクラスに以下の 2 つのフィールドが含まれています。

- **sequence**
- **value**

sequence フィールドは、フィボナッチ数列のオブジェクトの位置を示します。**value** フィールドは、その数列の位置のフィボナッチオブジェクトの値を示します。**-1** は、計算する必要がある値という意味です。

フィボナッチクラス

```
public static class Fibonacci {
    private int  sequence;
    private long value;

    public Fibonacci( final int sequence ) {
        this.sequence = sequence;
        this.value = -1;
    }

    ... setters and getters go here...
}
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.fibonacci.FibonacciExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでのフィボナッチの例の出力

```
recurse for 50
recurse for 49
recurse for 48
recurse for 47
...
recurse for 5
recurse for 4
recurse for 3
recurse for 2
1 == 1
2 == 1
3 == 2
4 == 3
5 == 5
6 == 8
...
47 == 2971215073
48 == 4807526976
49 == 7778742049
50 == 12586269025
```

Java でこの動作を実現するには、sequence フィールドに **50** を指定して、**Fibonacci** オブジェクトを挿入します。この例では、次に再帰ルールを使用して、他の 49 個の **Fibonacci** オブジェクトを挿入します。

PropertyChangeSupport インターフェイスを実装して動的ファクトを使用する代わりに、この例では MVEL 方言の **modify** キーワードを使用して、ブロックセッターアクションを有効にしてデシジョンエンジンに変更を通知しています。

フィボナッチの例の実行

```
ksession.insert( new Fibonacci( 50 ) );
ksession.fireAllRules();
```

この例では、以下の 3 つのルールを使用します。

- **"Recurse"**
- **"Bootstrap"**
- **"Calculate"**

"Recurse" ルールは、値が **-1** の、アサートされた各 **Fibonacci** オブジェクトを照合して、現在の値よりも数列が 1 つ小さい **Fibonacci** オブジェクトを新たに作成し、アサートします。数列フィールドが **1** に相当するオブジェクトが存在しない場合に、フィボナッチオブジェクトが追加されると毎回、このルールは再度照合され、実行されます。メモリーにフィボナッチオブジェクト 50 個がすべて存在する場合は、**not** 条件要素を使用して、ルールの合致を停止します。また、**"Bootstrap"** ルールを実行する前に **Fibonacci** オブジェクト 50 個をすべてアサートする必要があるため、このルールには **salience** の値も含まれます。

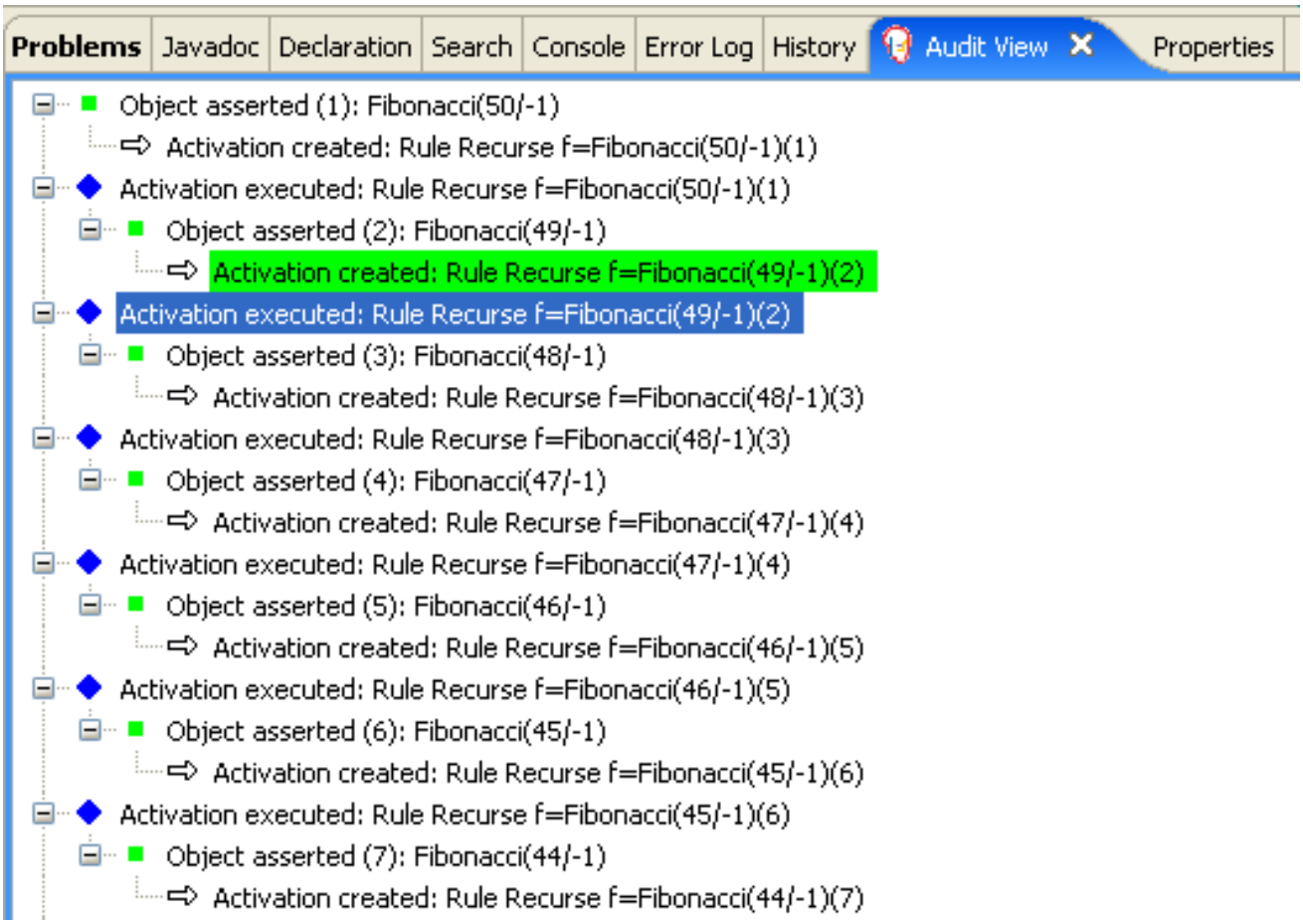
ルール "Recurse"

```
rule "Recurse"
    salience 10
  when
    f : Fibonacci ( value == -1 )
    not ( Fibonacci ( sequence == 1 ) )
  then
    insert( new Fibonacci( f.sequence - 1 ) );
    System.out.println( "recurse for " + f.sequence );
end
```

この例の実行フローをさらに理解するには、**target/fibonacci.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**監査ビュー** に、**sequence** フィールドが **50** に指定された、**Fibonacci** の元のアサーションが表示されます。これは Java コードで実行されています。これ以降、**監査ビュー** で、ルールの再帰が継続して行われ、アサートされた **Fibonacci** オブジェクトにより、**"Recurse"** ルールがアクティベートされて、再度実行されます。

図21.7 監査ビューでのルール "Recurse"



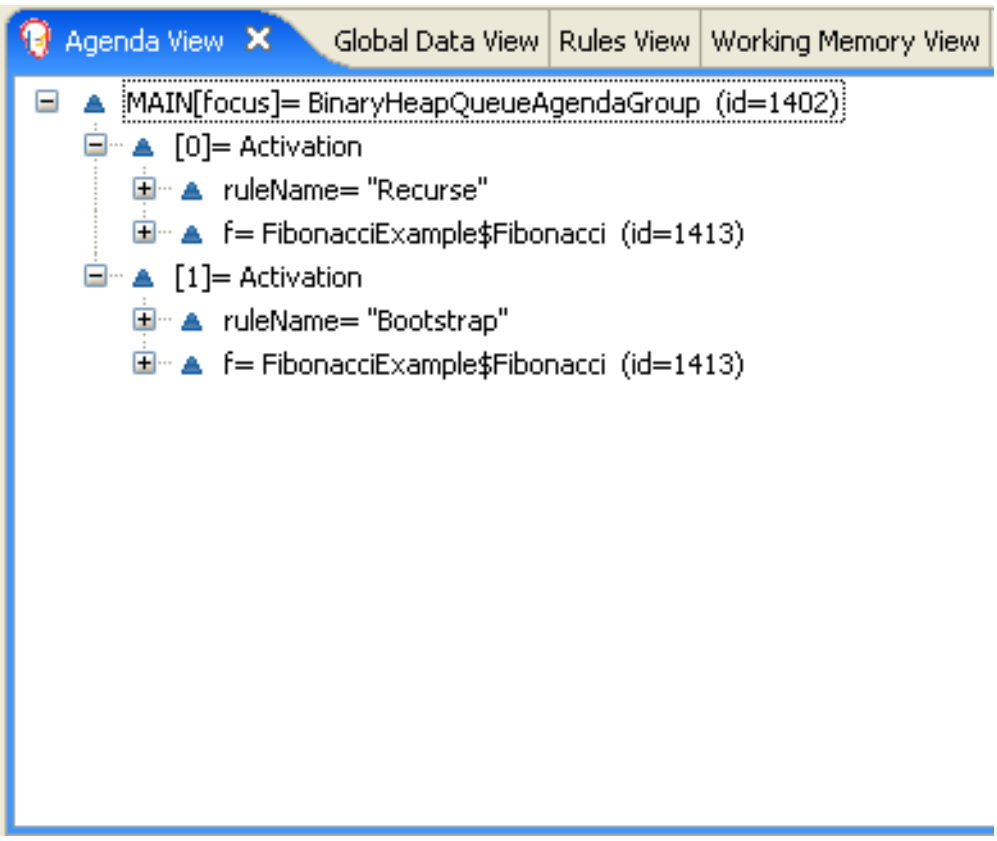
sequence フィールドが **2** の **Fibonacci** オブジェクトがアサートされると、**"Bootstrap"** ルールが一致し、**"Recurse"** ルールとともにアクティベートされます。フィールド **sequence** には複数の制約があり、**1** または **2** と同等かをテストしている点に注目してください。

ルール "Bootstrap"

```
rule "Bootstrap"
  when
    f : Fibonacci( sequence == 1 || == 2, value == -1 ) // multi-restriction
  then
    modify ( f ){ value = 1 };
    System.out.println( f.sequence + " == " + f.value );
end
```

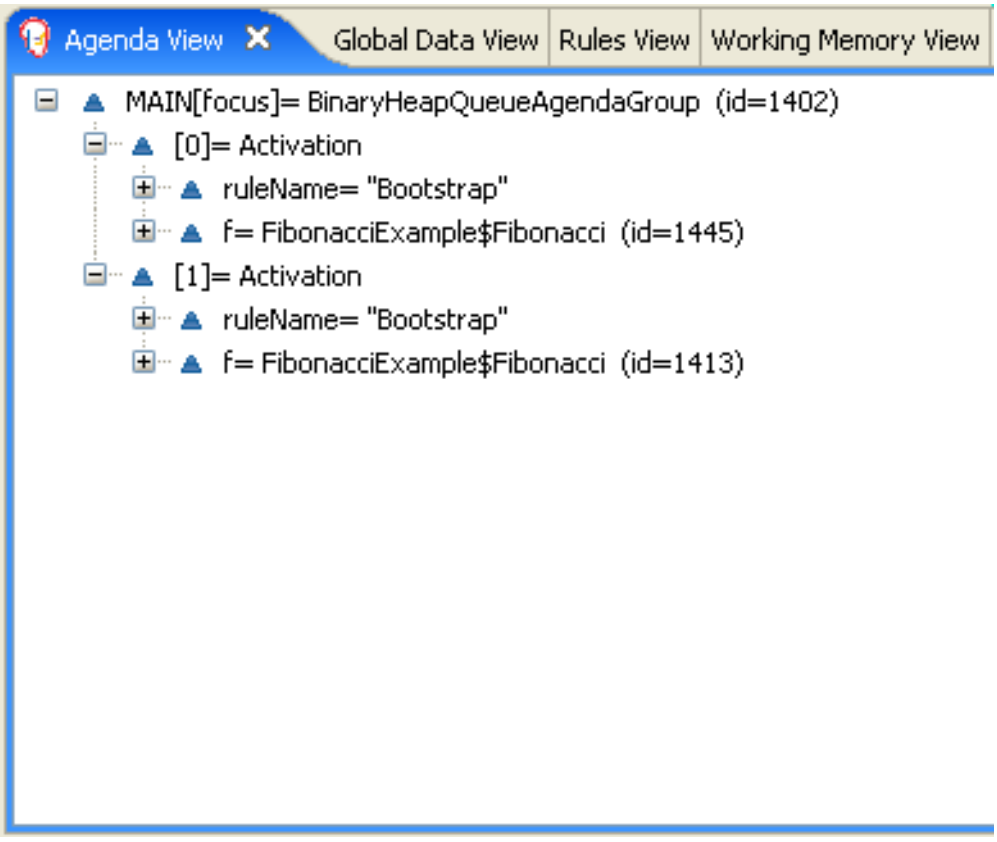
IDE で **Agenda View** を使用して、デシジョンエンジンアジェンダの状態を調査できます。**"Recurse"** の顕著性の値のほうが高いため、**"Bootstrap"** ルールは実行していません。

図21.8 アジェンダビュー 1 でのルール "Recurse" および "Bootstrap"



sequence が **1** の **Fibonacci** オブジェクトがアサートされると、**"Bootstrap"** ルールが再度一致し、このルールに含まれる 2 つのルールがアクティベートされます。**sequence** が **1** の **Fibonacci** オブジェクトが存在すると、すぐに **not** 条件要素でルールが一致しなくなるため、**"Recurse"** ルールの照合やアクティベーションはされません。

図21.9 アジェンダビュー 2 でのルール "Recurse" および "Bootstrap"



"Bootstrap" ルールは、**sequence** が **1** と **2** のオブジェクトの値を **1** に設定します。値が **-1** でない **Fibonacci** オブジェクトが 2 つあるため、**"Calculate"** ルールの照合が可能になります。

この例のある時点で、ワーキングメモリーに 50 近くの **Fibonacci** オブジェクトが存在します。3 つ選択してそれぞれを乗算し、順番に各値を計算する必要があります。フィールドの制約なしに、ルールで 3 つの Fibonacci パターンを使用してクラス積候補を絞り込む場合に、考えられる組み合わせとして 50x49x48 通りあり、約 12 万 5000 のルールを実行できるにもかかわらず、その大半が誤っていることになります。

"Calculate" ルールは、フィールドの制約を使用して正しい順番にフィボナッチパターンを 3 つ評価します。この手法は **cross-product matching** と呼ばれます。

最初のパターンでは、値が **!= -1** の **Fibonacci** オブジェクトを検索して、このパターンとフィールド両方をバインドします。2 番目の **Fibonacci** オブジェクトが実行する内容は同じですが、別のフィールド制約を追加して、シーケンスが **f1** にバインドされている **Fibonacci** オブジェクトより 1 つ大きくなるようにします。このルールが初めて実行されると、シーケンスが **1** と **2** にだけ、値 **1** が割り当てられていることが分かります。また、この 2 つの制約で、**f1** がシーケンス **1** を参照し、**f2** がシーケンス **2** を参照するようにします。

最後のパターンでは、値が **-1** と等しく、シーケンスが **f2** よりも大きい **Fibonacci** オブジェクトを検索します。

フィボナッチの例のこの時点で、3 つの **Fibonacci** オブジェクトが利用可能なクロス積から正しく選択され、**f3** にバインドされている 3 番目の **Fibonacci** オブジェクトの値を計算できます。

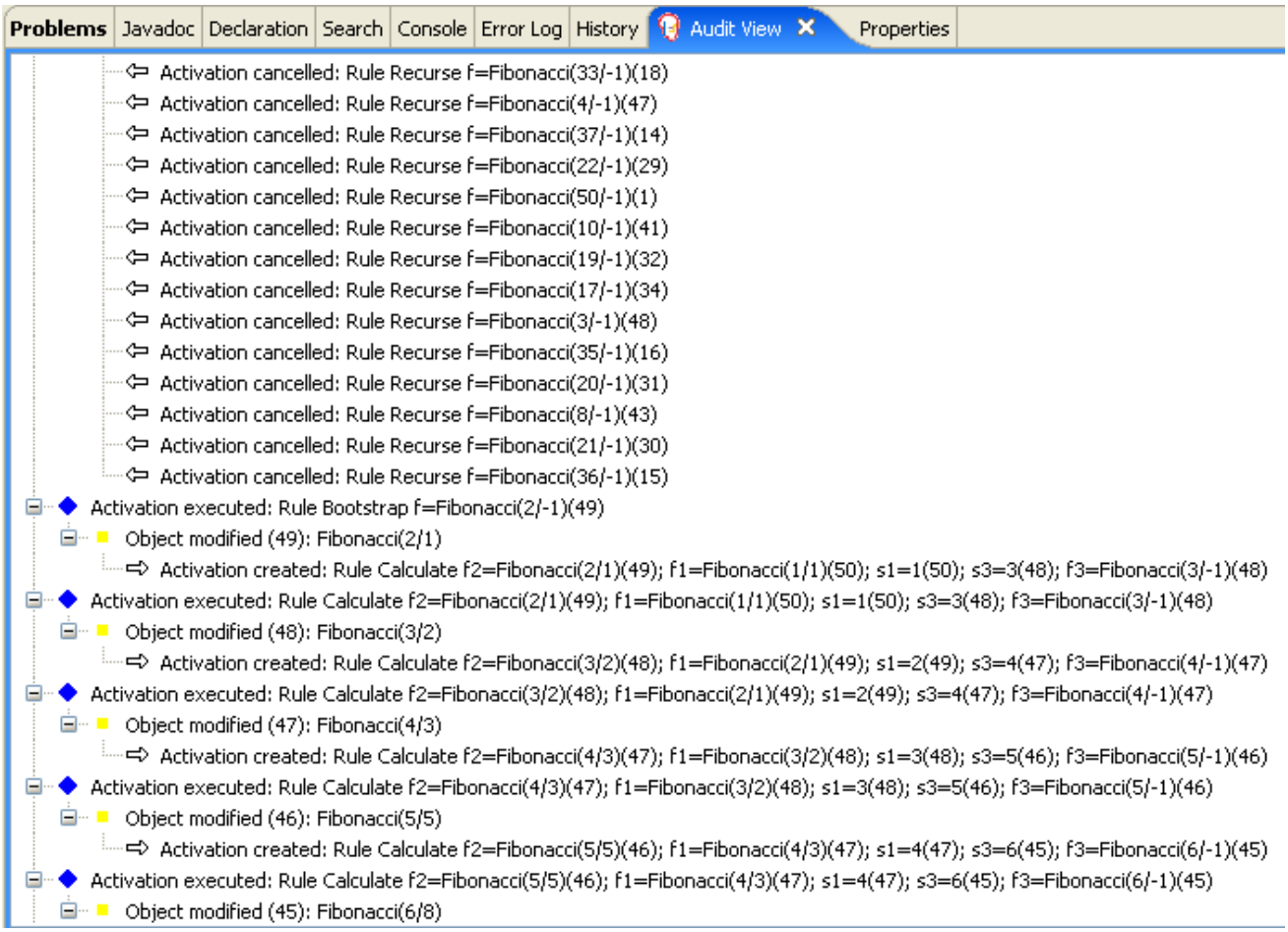
ルール "Calculate"

```
rule "Calculate"
  when
    // Bind f1 and s1.
    f1 : Fibonacci( s1 : sequence, value != -1 )
    // Bind f2 and v2, refer to bound variable s1.
    f2 : Fibonacci( sequence == (s1 + 1), v2 : value != -1 )
    // Bind f3 and s3, alternative reference of f2.sequence.
    f3 : Fibonacci( s3 : sequence == (f2.sequence + 1 ), value == -1 )
  then
    // Note the various referencing techniques.
    modify ( f3 ) { value = f1.value + v2 };
    System.out.println( s3 + " == " + f3.value );
end
```

modify ステートメントにより、**f3** にバインドされた **Fibonacci** オブジェクトの値が更新されます。つまり、値が **-1** 以外の **Fibonacci** オブジェクトが新たに存在するということで、**"Calculate"** ルールにより、再度合致があるか検索して次のフィボナッチ番号を算出することができます。

IDE のデバッグビューまたは **監査ビュー** では、最後の **"Bootstrap"** ルールが実行されることで **Fibonacci** オブジェクトが変更され、**"Calculate"** ルールに合致し、次に、別の **Fibonacci** オブジェクトが変更され、この **"Calculate"** ルールに再度合致できていることが分かります。このプロセスは、すべての **Fibonacci** オブジェクトに値が設定されるまで継続されます。

図21.10 監査ビューのルール



21.5. 価格設定のデシジョン例 (デシジョンテーブル)

価格設定のデシジョンセットの例では、スプレッドシートのデシジョンテーブルを使用して、DRL ファイルに直接ではなく、表形式で保険料金の価格を計算する方法を説明します。

以下は価格設定の例の概要です。

- **名前**: **decisiontable**
- **Main クラス**: (**src/main/java** の場合)
  org.drools.examples.decisiontable.PricingRuleDTExample
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: **org.drools.examples.decisiontable.ExamplePolicyPricing.xls**
  (**src/main/resources** 内)
- **目的**: スプレッドシートのデシジョンテーブルを使用してルールを定義する方法を示します。

スプレッドシートのデシジョンテーブルは、表形式でビジネスルールを定義する XLS 形式または XLSX 形式のスプレッドシートです。スプレッドシートのデシジョンテーブルは、スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにアップロードしたりできます。スプレッドシートの各行がルールになり、各列が条件、アクション、または別

のルール属性になります。最初に Red Hat Process Automation Manager でデシジョンテーブルを作成してアップロードします。次に、その他のすべてのルールアセットと同じように、定義したルールを Drools Rule Language (DRL) ルールにコンパイルします。

この価格設定の例では、特定タイプの保険を申請するドライバーに対して基本価格と割引を計算するビジネスルールセットを提供します。ドライバーの年齢と履歴、およびポリシータイプはすべて、基本保険料の計算に役立ち、追加のルールは、ドライバーが適格となる可能性のある潜在的な割引を計算します。

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.decisiontable.PricingRuleDTExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

```
Cheapest possible
BASE PRICE IS: 120
DISCOUNT IS: 20
```

この例を実行するコードは、標準の実行パターンに準拠しています。ルールが読み込まれ、ファクトが挿入されて、ステートレス KIE セッションが作成されます。この例における違いは、DRL ファイルや他のソースではなく、**ExamplePolicyPricing.xls** ファイルでルールが定義されるという点です。このスプレッドシートファイルは、テンプレートと DRL ルールを使用してデシジョンエンジンに読み込まれます。

スプレッドシートのデシジョンテーブルの設定

ExamplePolicyPricing.xls スプレッドシートには、以下の 2 つのデシジョンテーブルが含まれています。

- **Base pricing rules**
- **Promotional discount rules**

この例のスプレッドシートで分かるように、デシジョンテーブルの作成にはスプレッドシートの最初のタブしか使用できませんが、単一のタブ内に複数のテーブルが作成できます。デシジョンテーブルは必ずしもトップダウンの論理に従うものではなく、ルールとなるデータを補足する手段です。ルールの評価は、ルールエンジンのすべての通常のメカニックが適用されるため、必ずしも特定の順序で行われるわけではありません。このために、スプレッドシートの同一タブ内に複数のデシジョンテーブルが作成可能となります。

デシジョンテーブルは、対応するルールテンプレートファイルである **BasePricing.drt** と **PromotionalPricing.drt** で実行されます。これらのテンプレートファイルはテンプレートパラメーターによってデシジョンテーブルを参照し、デシジョンテーブルの条件およびアクションの各種ヘッダーを直接参照します。

BasePricing.drt ルールテンプレートファイル

```
template header
age[]
profile
priorClaims
policyType
base
reason

package org.drools.examples.decisiontable;
```

```
template "Pricing bracket"
age
policyType
base

rule "Pricing bracket_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}
        , priorClaims == "@{priorClaims}"
        , locationRiskProfile == "@{profile}"
    )
    policy: Policy(type == "@{policyType}")
  then
    policy.setBasePrice(@{base});
    System.out.println("@{reason}");
end
end template
```

PromotionalPricing.drt ルールテンプレートファイル

```
template header
age[]
priorClaims
policyType
discount

package org.drools.examples.decisiontable;

template "discounts"
age
priorClaims
policyType
discount

rule "Discounts_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}, priorClaims == "@{priorClaims}")
    policy: Policy(type == "@{policyType}")
  then
    policy.applyDiscount(@{discount});
end
end template
```

ルールは、KIE セッション **DTableWithTemplateKB** の **kmodule.xml** 参照によって実行されます。これは **ExamplePolicyPricing.xls** スプレッドシートを特定して参照するもので、ルールの実行の成功には必要なものです。この実行方法により、ルールをスタンドアロンユニットとして実行したり (ここでの例) パッケージ化されたナレッジ JAR (KJAR) ファイルにルールを含めたりすることができるため、スプレッドシートはルール実行とともにパッケージ化されます。

kmodule.xml ファイルの以下のセクションは、ルールが正常に実行され、スプレッドシートが機能するために必要になります。

```
<kbase name="DecisionTableKB" packages="org.drools.examples.decisiontable">
    <ksession name="DecisionTableKS" type="stateless"/>
</kbase>
```

```
<kbase name="DTableWithTemplateKB" packages="org.drools.examples.decisiontable-template">
  <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
          template="org/drools/examples/decisiontable-template/BasePricing.drt"
          row="3" col="3"/>
  <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
          template="org/drools/examples/decisiontable-template/PromotionalPricing.drt"
          row="18" col="3"/>
  <ksession name="DTableWithTemplateKS"/>
</kbase>
```

ルールテンプレートファイルを使用したデシジョンテーブルの実行方法とは別に、**DecisionTableConfiguration** オブジェクトを使用して、**DecisionTableInputType.xls** のような入力スプレッドシートを入力タイプとして指定することもできます。

```
DecisionTableConfiguration dtableconfiguration =
  KnowledgeBuilderFactory.newDecisionTableConfiguration();
  dtableconfiguration.setInputType( DecisionTableInputType.XLS );

  KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

  Resource xlsRes = ResourceFactory.newClassPathResource( "ExamplePolicyPricing.xls",
                                          getClass() );
  kbuilder.add( xlsRes,
          ResourceType.DTABLE,
          dtableconfiguration );
```

価格設定の例では以下の 2 つのファクトタイプを使用します。

- **Driver**
- **Policy**

この例では、これらのファクトのデフォルト値をそれぞれの Java クラス **Driver.java** と **Policy.java** に設定します。この **Driver** は 30 歳で、これまでに保険の請求をしたことがなく、現在のリスクプロファイルが **LOW** となっています。申請している **Policy** は **COMPREHENSIVE** です。

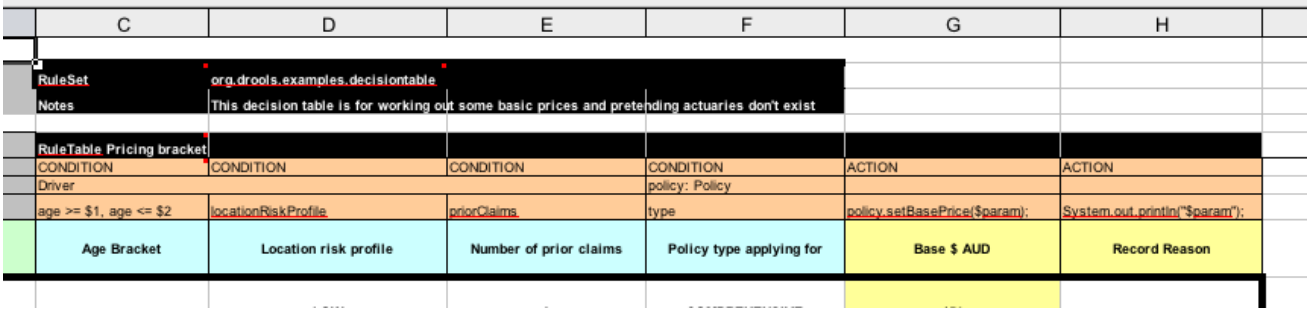
デシジョンテーブルでは、各行は異なるルールと見なされ、各列は条件またはアクションとみなされます。実行時にアジェンダがクリアされなければ、デシジョンテーブルの各行が評価されます。

デシジョンテーブルのスプレッドシート (XLS または XLSX) には、ルールデータを定義する以下の 2 つの主要なエリアが必要です。

- **RuleSet** エリア
- **RuleTable** エリア

RuleSet 領域では、ルールセット名、ユニバーサルルール属性など、(このスプレッドシートだけでなく) すべてのルールをパッケージ全体に、グローバルに適用する要素を定義します。**RuleTable** 領域では、実際のルール (行) と、指定したルールセットのルールテーブルを設定する条件、アクション、その他のルール属性 (列) を定義します。デシジョンテーブルのスプレッドシートには複数の **RuleTable** エリアを追加できますが、**RuleSet** エリアは 1 つのみとなります。

図21.11 デシジョンテーブルの設定



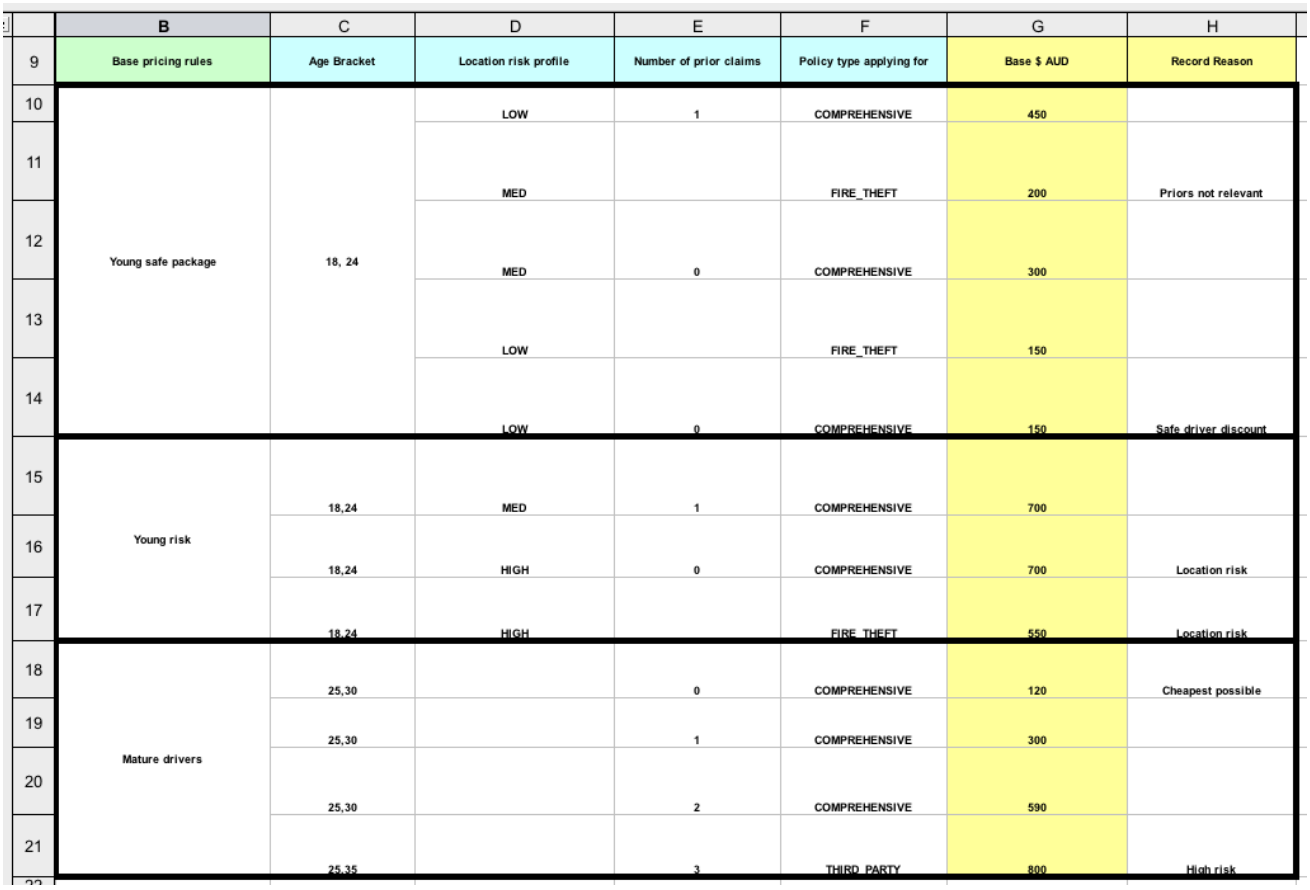
	C	D	E	F	G	H
	RuleSet	org.drools.examples.decisiontable				
	Notes	This decision table is for working out some basic prices and pretending actuaries don't exist				
	RuleTable Pricing bracket					
	CONDITION	CONDITION	CONDITION	CONDITION	ACTION	ACTION
	Driver			policy: Policy		
	age >= $1, age <= $2	locationRiskProfile	priorClaims	type	policy.setBasePrice($param);	System.out.println("$param");
	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason

RuleTable エリアでは、ルール属性を適用するオブジェクトも定義します。この例では、**Driver** と **Policy**、さらにオブジェクトの制限です。たとえば、**Driver** オブジェクトの制限では、**Age Bracket** 列が **age >= $1, age <= $2** と定義されています。ここでのコンマ区切りの範囲は、**18,24** などのテーブルの列値で定義されます。

Base pricing rules

価格設定例での **Base pricing rules** デシジョンテーブルでは、ドライバーの年齢、リスクプロファイル、請求数、ポリシータイプを評価し、これらの条件をベースにしたポリシーの基本価格を算定します。

図21.12 基本価格の計算



	B	C	D	E	F	G	H
9	Base pricing rules	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason
10	Young safe package	18, 24	LOW	1	COMPREHENSIVE	450	
11			MED		FIRE_THEFT	200	Priors not relevant
12			MED	0	COMPREHENSIVE	300	
13			LOW		FIRE_THEFT	150	
14			LOW	0	COMPREHENSIVE	150	Safe driver discount
15	Young risk	18,24	MED	1	COMPREHENSIVE	700	
16		18,24	HIGH	0	COMPREHENSIVE	700	Location risk
17		18,24	HIGH		FIRE_THEFT	550	Location risk
18	Mature drivers	25,30		0	COMPREHENSIVE	120	Cheapest possible
19		25,30		1	COMPREHENSIVE	300	
20		25,30		2	COMPREHENSIVE	590	
21		25,35		3	THIRD_PARTY	800	High risk

Driver 属性は、以下のテーブル列で定義されます。

- **Age Bracket**: この年齢層には、ドライバー年齢の条件範囲を定義する条件 **age >=$1, age <=$2** の定義があります。この条件列では **$1 and $2** を使用しており、スプレッドシートではコンマ区切りになります。ここでの値入力は **18,24** や **18, 24** の形式となり、両方ともビジネスルールの実行で機能する形式です。

- **Location risk profile**: リスクプロファイルは、この例のプログラムでは常に **LOW** として渡す文字列です。ただし、**MED** または **HIGH** に変更することが可能です。

- **Number of prior claims**: これまでの請求数は整数で定義し、アクションをトリガーするには条件列のものと同一である必要があります。この値は範囲ではなく、完全一致のみになります。

デシジョンテーブルの **Policy** は、ルールの条件とアクションの両方で使用され、属性は以下のテーブル列で定義されます。

- **Policy type applying for**: ポリシータイプは文字列として渡される条件で、適用する以下のいずれかのポリシータイプ (**COMPREHENSIVE**、**FIRE_THEFT**、または **THIRD_PARTY**) を定義します。

- **Base $ AUD**: **basePrice** は **ACTION** として定義され、これはこの値に対応するスプレッドシートのセルをベースに制限 **policy.setBasePrice($param);** で価格を設定します。このデシジョンテーブルの対応する DRL ルールを実行する際には、ファクトに合致する true 条件でルールの **then** 部分がこのアクションステートメントを実行し、基本価格を対応する値に設定します。

- **Record Reason**: ルールが正常に実行されると、アクションは出力メッセージを **System.out** コンソールに生成し、どのルールが適用されたかが反映されます。これは後でアプリケーションにキャプチャーされ、出力されます。

この例では、左側の最初の列でルールをカテゴリー分けしています。この列は注釈目的で、ルール実行には影響がありません。

Promotional discount rules

価格設定例での **Promotional discount rules** デシジョンテーブルでは、ドライバーの年齢、請求数、ポリシータイプを評価し、ポリシー価格の割引を算定します。

図21.13 割引計算

29	Promotional discount rules	Age Bracket	Number of prior claims	Policy type applying for	Discount %
30	Rewards for safe drivers	18,24	0	COMPREHENSIVE	1
31		18,24	0	FIRE_THEFT	2
32		25,30	1	COMPREHENSIVE	5
33		25,30	2	COMPREHENSIVE	1
34		25,30	0	COMPREHENSIVE	20

このデシジョンテーブルには、ドライバーに適用可能な割引条件が含まれています。基本価格の算定と同様に、このテーブルではドライバーの **Age**、**Number of prior claims**、および **Policy type applying for** を評価して、適用する **Discount %** 率を判定します。たとえば、ドライバーは 30 歳であり、請求履歴がなく、**COMPREHENSIVE** ポリシーを申請している場合は、**20** パーセントの割引率が導き出されます。

21.6. ペットショップの例のデシジョン (アジェンダグループ、グローバル変数、コールバック、および GUI 統合)

ペットショップの例のデシジョンセットでは、ルールでのアジェンダグループとグローバル変数の使用方法と、Red Hat Process Automation Manager ルールとグラフィカルユーザーインターフェイス (GUI) (この場合は、Swing ベースのデスクトップアプリケーション) の統合方法が分かります。また、この例では、コールバックを使用して実行中のデシジョンエンジンと通信し、ランタイム時に加えられたワーキングメモリー内の変更をもとに GUI を更新する方法を例示しています。

以下は、ペットショップの例の概要です。

- **名前**: **petstore**
- Main **クラス**: (**src/main/java** 内の) **org.drools.examples.petstore.PetStoreExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.petstore.PetStore.drl**
- **目的**: ルールアジェンダグループ、グローバル変数、コールバック、および GUI 統合を例示します。

ペットショップの例では、**PetStoreExample.java** クラス例を使用して (Swing イベントを処理する複数のクラスに加え)、以下のクラスを主に定義しています。

- **Petstore** には **main()** メソッドが含まれます。
- **PetStoreUI** は Swing ベースの GUI を作成して表示します。このクラスには複数の小さいクラスが含まれており、マウスボタンのクリックなど、さまざまな GUI イベントに主に対応します。
- **TableModel** には表データが含まれています。このクラスは基本的に、Swing クラス **AbstractTableModel** を拡張する JavaBean です。
- **CheckoutCallback** により、GUI がルールと対話できるようになります。
- **Ordershow** は購入するアイテムを保持します。
- **Purchase** には、注文の詳細と、購入する製品が保存されます。
- **Product** は、販売可能な商品と価格の詳細を含む JavaBean です。

この例の Java コードはほぼ、プレーンな JavaBean か Swing ベースとなっています。Swing コンポーネントの詳細は、Creating a GUI with JFC/Swing の Java チュートリアルを参照してください。

ペットショップの例でのルール実行動作
他の例のデシジョンセットではファクトがすぐにアサートされて実行されるのに対し、ペットショップの例では、ユーザーの対話をもとに他のファクトが収集されるまでルールが実行します。このルールでは、コンストラクターで作成される **PetStoreUI** オブジェクトを使用してルールを実行し、**Vector** オブジェクトの **stock** を受けれ入れて商品を収集します。次に、この例では、以前の読み込まれたルールベースを含む **CheckoutCallback** クラスのインスタンスを使用します。

ペットショップの KIE コンテナーおよびファクト実行の設定

```
// KieServices is the factory for all KIE services.
KieServices ks = KieServices.Factory.get();

// Create a KIE container on the class path.
KieContainer kc = ks.getKieClasspathContainer();

// Create the stock.
Vector<Product> stock = new Vector<Product>();
stock.add( new Product( "Gold Fish", 5 ) );
stock.add( new Product( "Fish Tank", 25 ) );
```

```
stock.add( new Product( "Fish Food", 2 ) );

// A callback is responsible for populating the working memory and for firing all rules.
PetStoreUI ui = new PetStoreUI( stock,
                                new CheckoutCallback( kc ) );
ui.createAndShowGUI();
```

ルールを実行する Java コードは **CheckoutCallBack.checkout()** メソッドに含まれます。このメソッドは、ユーザーが UI で **チェックアウト** をクリックするとトリガーされます。

CheckoutCallBack.checkout() からのルール実行

```
public String checkout(JFrame frame, List<Product> items) {
    Order order = new Order();

    // Iterate through list and add to cart.
    for ( Product p: items ) {
        order.addItem( new Purchase( order, p ) );
    }

    // Add the JFrame to the ApplicationData to allow for user interaction.

    // From the KIE container, a KIE session is created based on
    // its definition and configuration in the META-INF/kmodule.xml file.
    KieSession ksession = kcontainer.newKieSession("PetStoreKS");

    ksession.setGlobal( "frame", frame );
    ksession.setGlobal( "textArea", this.output );

    ksession.insert( new Product( "Gold Fish", 5 ) );
    ksession.insert( new Product( "Fish Tank", 25 ) );
    ksession.insert( new Product( "Fish Food", 2 ) );

    ksession.insert( new Product( "Fish Food Sample", 0 ) );

    ksession.insert( order );

    // Execute rules.
    ksession.fireAllRules();

    // Return the state of the cart
    return order.toString();
}
```

このコード例では、2 つの要素を **CheckoutCallBack.checkout()** メソッドに渡します。1 つ目の要素は、GUI の一番下にある出力テキストのフレームを囲む Swing コンポーネント **JFrame** のハンドルです。2 つ目の要素は注文アイテムのリストで、GUI の右上のセクションにある **Table** エリアからの情報を保存する **TableModel** から取得します。

for ループは GUI からの注文アイテム一覧を JavaBean **Order** に変換します。これは、**PetStoreExample.java** ファイルにも含まれています。

今回の例では、データはすべて Swing コンポーネントに含まれており、ユーザーが UI の **チェックアウト** をクリックしない限り実行しないため、ルールはステートレスの KIE セッションで実行します。ユーザーが **チェックアウト** をクリックするたびに、リストの内容を Swing **TableModel** から KIE セッショ

ンのワーキングメモリーに移動し、**ksession.fireAllRules()** メソッドで実行します。

このコード内には、**KieSession** への呼び出しが 9 個あります。1 つ目は、**KieContainer** から新しい **KieSession** を作成します (この例では、**main()** メソッドの **CheckoutCallBack** クラスから **KieContainer** に渡されます)。次の 2 つの呼び出しは、ルールでグローバル変数として保持されるオブジェクト を 2 つ渡します (メッセージの書き込みに使用する Swing テキストエリアと Swing フレーム)。より多くの商品の情報を **KieSession** と注文リストに入力します。最後の呼び出しは、標準の **fireAllRules()** です。

ペットショップのルールファイルのインポート、グローバル変数、Java 関数

PetStore.drl ファイルには、さまざまな Java クラスをルールで利用できるように、標準のパッケージとインポートステートメントが含まれています。このルールファイルには、**frame**、**textArea** などのように、ルール内で使用する **グローバル変数** が含まれています。グローバル変数では、Swing コンポーネント **JFrame** と、**setGlobal()** メソッドを呼び出した Java コードにより以前に渡された **JTextArea** コンポーネントへの参照を保持します。ルールが実行するとすぐに失効するルールの標準変数とは異なり、グローバル変数は KIE セッションの有効期間中この値を保持します。これは、このグローバル変数の内容が、後続のすべてのルールで評価できることを意味します。

PetStore.drl パッケージ、インポート、およびグローバル変数

```
package org.drools.examples;

import org.kie.api.runtime.KieRuntime;
import org.drools.examples.petstore.PetStoreExample.Order;
import org.drools.examples.petstore.PetStoreExample.Purchase;
import org.drools.examples.petstore.PetStoreExample.Product;
import java.util.ArrayList;
import javax.swing.JOptionPane;

import javax.swing.JFrame;

global JFrame frame
global javax.swing.JTextArea textArea
```

PetStore.drl ファイルには、このファイル内のルールが使用する関数が 2 つも含まれています。

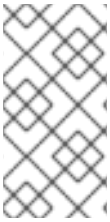
PetStore.drl Java 関数

```
function void doCheckout(JFrame frame, KieRuntime krt) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to checkout?",
                                             "",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

       if (n == 0) {
           krt.getAgenda().getAgendaGroup( "checkout" ).setFocus();
       }
}
```

```
function boolean requireTank(JFrame frame, KieRuntime krt, Order order, Product fishTank, int total)
{
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                        "Would you like to buy a tank for your " + total + " fish?",
                                        "Purchase Suggestion",
                                        JOptionPane.YES_NO_OPTION,
                                        JOptionPane.QUESTION_MESSAGE,
                                        null,
                                        options,
                                        options[0]);

        System.out.print( "SUGGESTION: Would you like to buy a tank for your "
                            + total + " fish? - " );

        if (n == 0) {
            Purchase purchase = new Purchase( order, fishTank );
            krt.insert( purchase );
            order.addItem( purchase );
            System.out.println( "Yes" );
        } else {
           System.out.println( "No" );
        }
        return true;
}
```

この 2 つの関数は以下のアクションを実行します。

- **doCheckout()** は、チェックアウトするかどうかユーザーに尋ねるダイアログボックスを表示します。チェックアウトする場合は、フォーカスが **checkout** アジェンダグループに設定され、そのグループのルールを (今後) 実行できるようにします。
- **requireTank()** は、水槽を購入するかどうかを確認するダイアログを表示します。購入する場合は、新しい水槽の **Product** がワーキングメモリーの注文リストに追加されます。



注記

この例では、効率化を図るため、すべてのルールと関数が同じルールファイルで実行しています。実稼働環境では、通常、ルールと関数を別のファイルに分けるか、静的な Java メソッドを構築して、**import function my.package.name.hello** などのインポート関数を使用し、ファイルをインポートします。

アジェンダグループを使用したペットショップルール

ペットショップの例のルールはほぼ、アジェンダグループを使用してルールの実行を制御しています。アジェンダグループを使用すると、デシジョンエンジンアジェンダを分割し、ルールのグループの実行を、詳細にわたり制御できるようになります。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

最初は、ワーキングメモリーは、アジェンダグループ **MAIN** にフォーカスを当てます。アジェンダグループのルールは、グループがこのフォーカスを受けた場合のみ実行されます。**setFocus()** メソッドか、**auto-focus** ルール属性を使用してフォーカスを設定できます。**auto-focus** 属性を使用すると、

ルールが一致してアクティベートされた場合のみ、ルールにアジェンダグループのフォーカスが自動的に当てられます。

ペットショップの例では、ルールに以下のアジェンダグループを使用します。

- **"init"**
- **"evaluate"**
- **"show items"**
- **"checkout"**

たとえば、同じルール **"Explode Cart"** は **"init"** のアジェンダグループを使用して、ショッピングカードのアイテムを起動して KIE セッションのワーキングメモリーに挿入するオプションが提供されるようにします。

ルール "Explode Cart"

```
// Insert each item in the shopping cart into the working memory.
rule "Explode Cart"
    agenda-group "init"
    auto-focus true
    salience 10
  when
    $order : Order( grossTotal == -1 )
    $item : Purchase() from $order.items
  then
    insert( $item );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "show items" ).setFocus();
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "evaluate" ).setFocus();
end
```

このルールは、**grossTotal** がまだ計算されていない全注文に対して照合が行われます。購入アイテムごとに、順番に実行がループされます。

ルールは、アジェンダグループに関連する以下の機能を使用します。

- **agenda-group "init"** はアジェンダグループの名前を定義します。この例では、グループにはルールが 1 つしかありません。ただし、Java コードもルール結果もこのグループにフォーカスされていないため、**auto-focus** の属性により、ルールが実行されるかが決まります。
- このルールはアジェンダグループで唯一のルールですが、**auto-focus true** を使用して、**fireAllRules()** が Java コードから呼び出されると、必ず実行されるようにします。
- **kcontext….setFocus()** は、**"show items"** および **"evaluate"** アジェンダグループにフォーカスを設定し、ルールを実行できるようにします。実際には、すべての項目をその順序でループしてメモリーに挿入し、各挿入の後に他のルールを実行します。

"show items" アジェンダグループには **"Show Items"** というルールだけが含まれます。KIE セッションのワーキングメモリーに現在含まれる注文で購入があるたびに、このルールを使用して、ルールファイルに定義した **textArea** 変数をもとに、GUI の下の部分にあるテキストエリアに詳細がロギングされます。

ルール "Show Items"

```
rule "Show Items"
    agenda-group "show items"
  when
    $order : Order()
    $p : Purchase( order == $order )
  then
   textArea.append( $p.product + "\n");
end
```

また、**"evaluate"** アジェンダグループにより、**"Explode Cart"** ルールからフォーカスを取得します。このアジェンダグループには、2 つのルール (**"Free Fish Food Sample"** と **"Suggest Tank"**) が含まれます。この順番で実行されます。

ルール "Free Fish Food Sample"

```
// Free fish food sample when users buy a goldfish if they did not already buy
// fish food and do not already have a fish food sample.
rule "Free Fish Food Sample"
    agenda-group "evaluate" (1)
  when
    $order : Order()
    not ( $p : Product( name == "Fish Food") && Purchase( product == $p ) ) (2)
    not ( $p : Product( name == "Fish Food Sample") && Purchase( product == $p ) ) (3)
    exists ( $p : Product( name == "Gold Fish") && Purchase( product == $p ) ) (4)
    $fishFoodSample : Product( name == "Fish Food Sample" );
  then
    System.out.println( "Adding free Fish Food Sample to cart" );
    purchase = new Purchase($order, $fishFoodSample);
    insert( purchase );
    $order.addItem( purchase );
end
```

ルール **"Free Fish Food Sample"** は、以下の条件がすべて該当する場合のみ実行されます。

1. アジェンダグループ **"evaluate"** がルール実行で評価されている
2. ユーザーが魚の餌をまだ持っていない
3. ユーザーが無料の魚の餌サンプルをまだ持っていない
4. ユーザーが金魚を注文している

この注文ファクトが上記の要件すべてを満たす場合は、新しい商品 (Fish Food Sample) が作成され、ワーキングメモリーの注文に追加されます。

ルール "Suggest Tank"

```
// Suggest a fish tank if users buy more than five goldfish and
// do not already have a tank.
rule "Suggest Tank"
    agenda-group "evaluate"
  when
    $order : Order()
    not ( $p : Product( name == "Fish Tank") && Purchase( product == $p ) ) (1)
```

```
    ArrayList( $total : size > 5 ) from collect( Purchase( product.name == "Gold Fish" ) ) (2)
    $fishTank : Product( name == "Fish Tank" )
  then
    requireTank(frame, kcontext.getKieRuntime(), $order, $fishTank, $total);
end
```

ルール **"Suggest Tank"** は以下の条件がすべて該当する場合のみ実行されます。

1. ユーザーが水槽を注文していない
2. ユーザーが 6 匹以上注文した

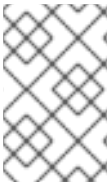
このルールが実行すると、ルールファイルに定義されている **requireTank()** 関数が呼び出されます。この関数により、水槽を購入するかどうかを尋ねるダイアログが表示されます。購入する場合は、新しい水槽の **Product** がワーキングメモリーの注文リストに追加されます。ルールが **requireTank()** 関数を呼び出した場合は、このルールを使用して、関数に Swing GUI のハンドルが含まれるように、**frame** のグローバル変数を渡します。

ペットショップの例の **"do checkout"** ルールにはアジェンダグループや **when** 条件がないため、ルールは常に実行され、デフォルトの **MAIN** のアジェンダグループの一部とみなされます。

ルール "do checkout"

```
rule "do checkout"
  when
  then
    doCheckout(frame, kcontext.getKieRuntime());
end
```

このルールが実行されると、ルールファイルで定義されている **doCheckout()** 関数を呼び出します。この関数により、チェックアウトするかどうかをユーザーに尋ねるダイアログボックスが表示されます。チェックアウトする場合は、フォーカスが **checkout** アジェンダグループに設定され、そのグループのルールを (今後) 実行できるようにします。このルールで **doCheckout()** 関数を呼び出し、この変数に Swing GUI のハンドルが含まれるように **frame** グローバル変数を渡します。



注記

この例では、結果が想定どおりに実行されない場合のトラブルシューティングの方法を例示します。ルールの **when** ステートメントから条件を削除して、**then** ステートメントのアクションをテストし、アクションが正しく実行されることを検証します。

"checkout" アジェンダグループには、注文のチェックアウト処理、および割引の適用の 3 つのルール (**"Gross Total"**、**"Apply 5% Discount"**、および **"Apply 10% Discount"**) が含まれています。

ルール "Gross Total"、"Apply 5% Discount"、および "Apply 10% Discount"

```
rule "Gross Total"
    agenda-group "checkout"
  when
    $order : Order( grossTotal == -1)
    Number( total : doubleValue ) from accumulate( Purchase( $price : product.price ),
                                                   sum( $price ) )
  then
    modify( $order ) { grossTotal = total }
```

```
    textArea.append( "\ngross total=" + total + "\n" );
end

rule "Apply 5% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 10 && < 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.95;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end

rule "Apply 10% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.90;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end
```

ユーザーがまだ総計を算出していない場合には、**Gross Total** で、商品の価格を累積して合計を出し、この合計を KIE セッションに渡して、**textArea** のグローバル変数を使用し、Swing **JTextArea** で合計を表示します。

総計が **10** から **20** (通貨単位) の場合は、**"Apply 5% Discount"** ルールで割引合計を計算し、KIE セッションに追加して、テキストエリアに表示します。

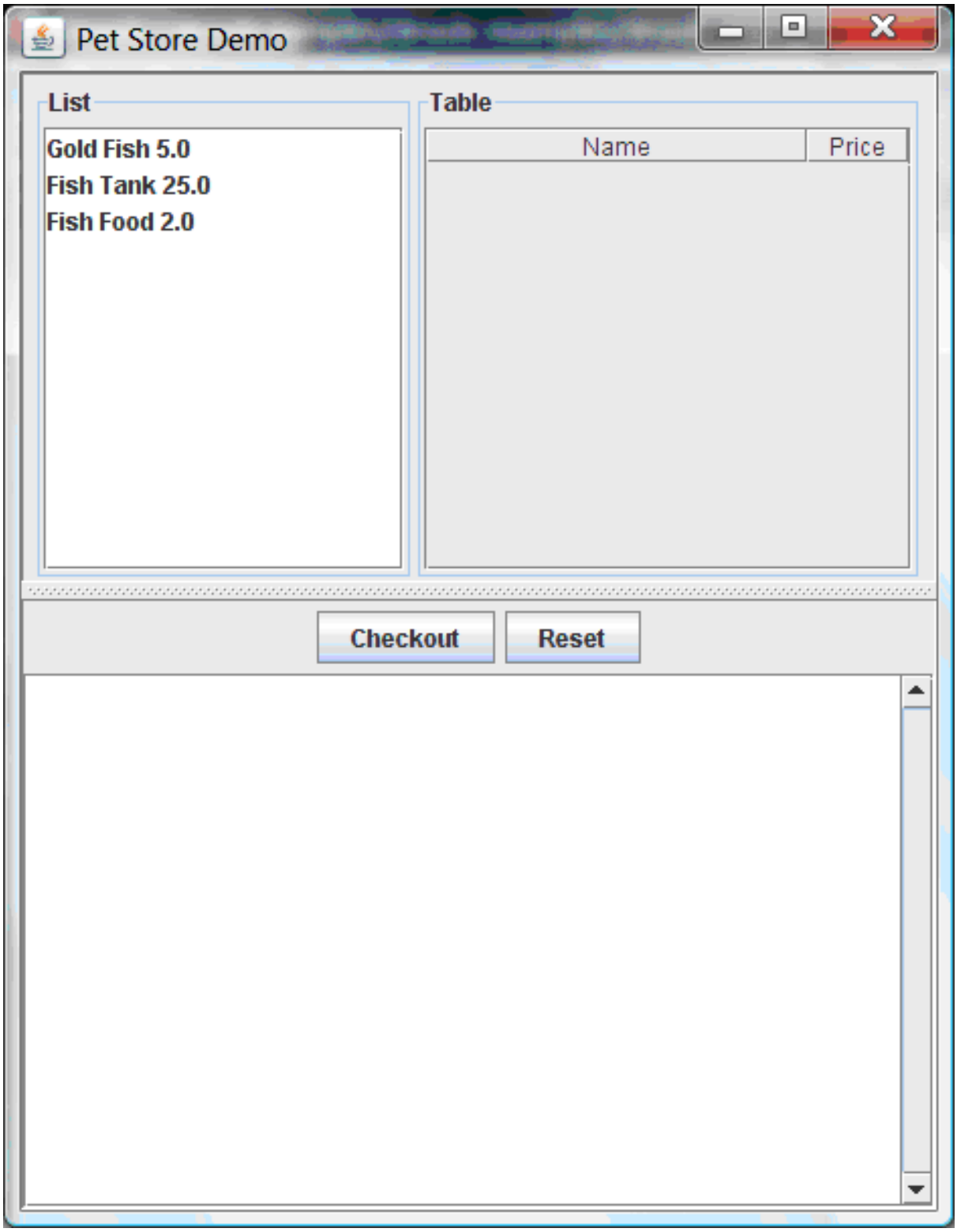
総計が **20** 未満の場合は、**"Apply 10% Discount"** ルールで割引合計を計算し、KIE セッションに追加して、テキストエリアに表示します。

ペットショップ例の実行

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.petstore.PetStoreExample** クラスを Java アプリケーションとして実行し、ペットショップの例を実行します。

ペットショップの例を実行すると、**Pet Store Demo** GUI ウィンドウが表示されます。このウィンドウでは、購入可能な商品 (左上)、選択済み商品の空白のリスト (右上)、**チェックアウト** および **リセット** ボタン (真ん中)、空白のシステムメッセージエリア (下) が表示されます。

図21.14 起動後のペットショップ例の GUI

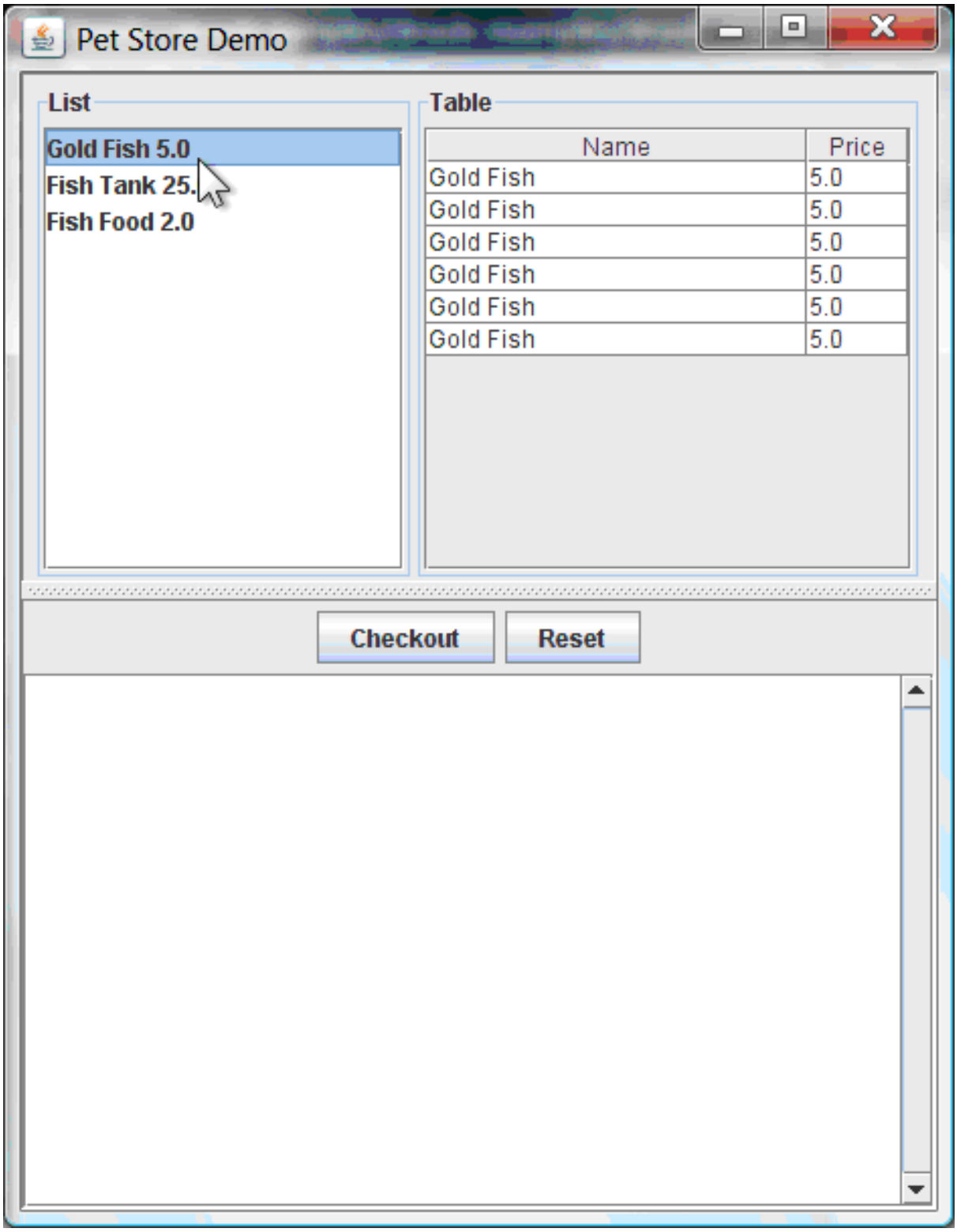


この例では、以下のイベントが発生して、この実行動作を確立します。

1. **main()** メソッドがルールベースの実行と読み込みを終えていますが、ルールは実行していない。今のところ、これが、実行されたルールに関連する唯一のコードになります。
2. 新しい **PetStoreUI** オブジェクトが作成され、後で使用できるようにルールベースにハンドルを渡している。
3. さまざまな Swing コンポーネントが関数を実行し、最初の UI 画面が表示され、ユーザーの入力を待っている。

リストからさまざまな商品をクリックして、UI 設定をチェックできます。

図21.15 ペットショップ例の GUI のチェック



ルールコードはまだ実行されていません。UI は Swing コードを使用してユーザーによるマウスクリックを検出し、選択済みの商品を **TableModel** オブジェクトに追加して、UI の右上隅に表示します。この例では、Model-View-Controller 設計パターンを紹介しています。

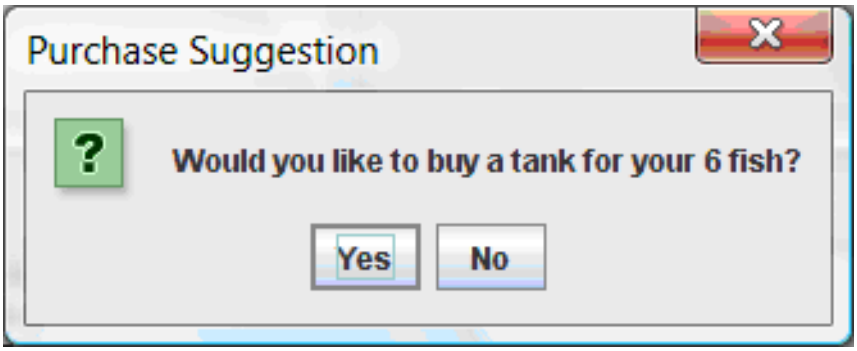
チェックアウト をクリックすると、ルールが以下の方法で実行されます。

1. Swing クラスは **Checkout** がクリックされるまで待機して、(最終的に) **CheckOutCallBack.checkout()** メソッドを呼び出します。これにより、**TableModel** オブジェ

クト (UI の右上隅) から KIE セッションのワーキングメモリーにデータを挿入します。その後に、メソッドによりルールが実行されます。

2. **"Explode Cart"** ルールは、**auto-focus** 属性を **true** に設定して最初に実行します。このルールは、カートの商品をすべて順にループしていき、商品がワーキングメモリーに含まれていることを確認し、アジェンダグループ **"show Items"** と **"evaluate"** に実行するオプションを提供します。このグループのルールは、カートのコンテンツをテキストエリア (UI の下) に追加して、魚の餌を無料で受け取る資格があるかどうかを評価し、また水槽購入の有無を尋ねるかどうかを決定します。

図21.16 水槽の資格



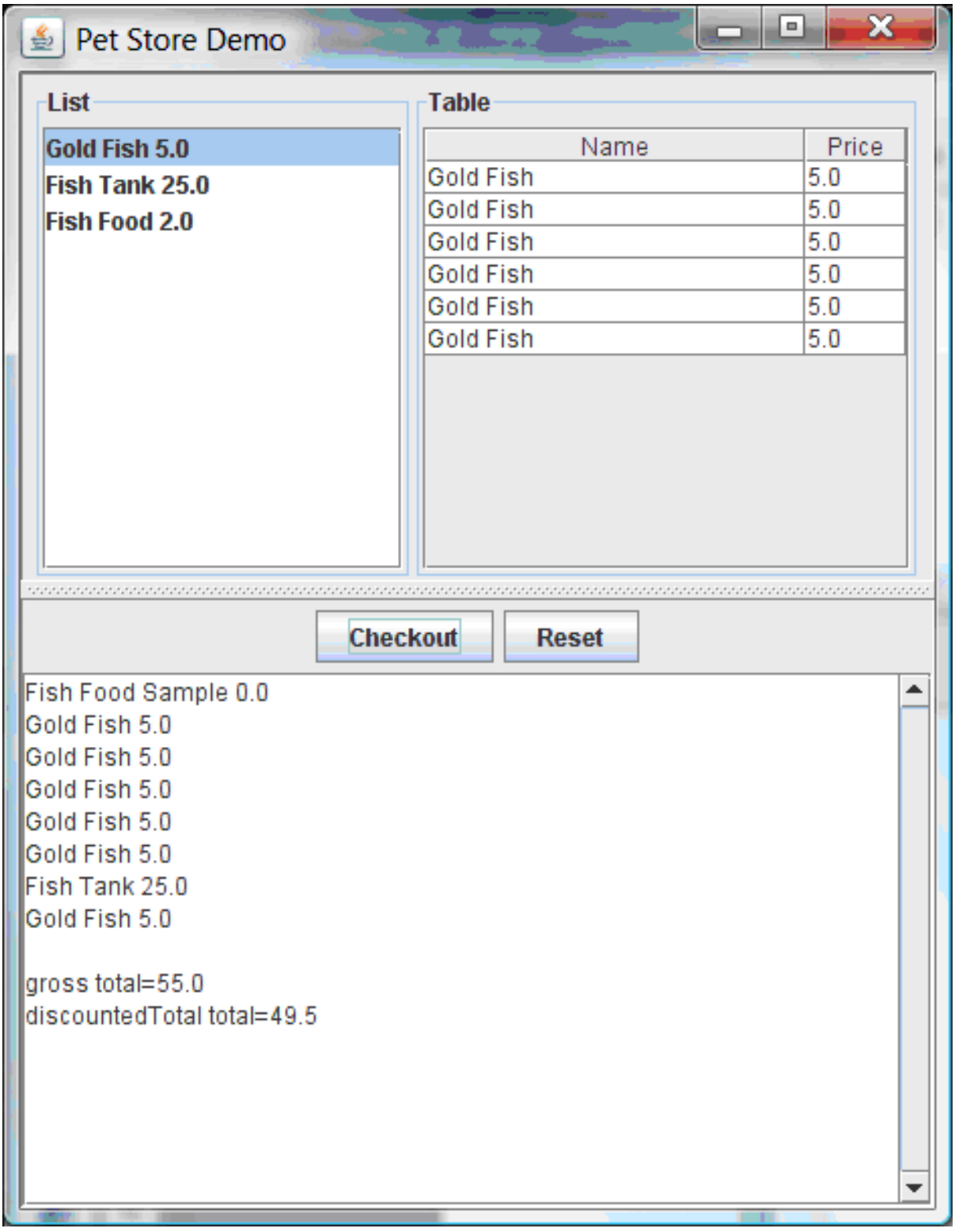
3. 現在、他のアジェンダグループにフォーカスが当たっておらず、**"do checkout"** ルールは、デフォルトの **MAIN** アジェンダグループに含まれているため、次に実行されます。このルールは常に **doCheckout()** 関数を呼び出し、この関数によりチェックアウトをするかどうかが尋ねられます。

4. **doCheckout()** 関数は、フォーカスを **"checkout"** アジェンダグループに設定し、そのグループ内のルールに、実行するオプションを提供します。

5. **"checkout"** アジェンダグループ内のルールは、カート内の内容を表示し、適切な割引を適用します。

6. Swing は、別の商品の選択 (およびもう一度ルールを実行) または GUI の終了のいずれかのユーザー入力を待ちます。

図21.17 全ルールが実行された後のペットショップ例の GUI



IDE コンソールでイベントのこのフローを例示するには、他の **System.out** 呼び出しを追加します。

IDE コンソールの System.out 出力

```
Adding free Fish Food Sample to cart
SUGGESTION: Would you like to buy a tank for your 6 fish? - Yes
```

21.7. 誠実な政治家の例のデシジョン (真理維持および顕著性)

誠実な政治家のデシジョンセットの例では、論理挿入を使用した真理維持の概念およびルールでの顕著性の使用方法を説明します。

以下は、誠実な政治家の例の概要です。

- **名前**: **honestpolitician**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.honestpolitician.HonestPoliticianExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.honestpolitician.HonestPolitician.drl**
- **目的**: ファクトの論理挿入をもとにした真理維持の概念およびルールでの顕著性の使用方法を紹介します。

誠実な政治家例の前提として基本的に、ステートメントが True の場合にのみ、オブジェクトが存在できます。**insertLogical()** メソッドを使用して、ルールの結果により、オブジェクトを論理的に挿入します。つまり、論理的に挿入されたルールが True の状態であれば、オブジェクトは KIE セッションのワーキングメモリー内に留まります。ルールが True でなくなると、オブジェクトは自動的に取り消されます。

この例では、ルールを実行することで、企業による政治家の買収が原因で、政治家グループが誠実から不誠実に変わります。各政治家が評価されるにつれ、最初は honesty 属性を **true** に設定して開始しますが、ルールが実行すると政治家は誠実ではなくなります。状態が誠実から不誠実に切り替わると、ワーキングメモリーから削除されます。ルールの顕著性により、顕著性が定義されているルールをどのように優先付けするかを、デシジョンエンジンに通知します。通知しないと、デフォルトの顕著性である **0** が使用されます。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。

Politician クラスおよび Hope クラス

この例の **Politician** クラス例は、誠実な政治家として設定されています。**Politician** クラスは、文字列アイテム **name** とブール値アイテム **honest** で設定されています。

Politician クラス

```
public class Politician {
    private String name;
    private boolean honest;
    ...
}
```

Hope クラスは、**Hope** オブジェクトが存在するかどうかを判断します。このクラスには意味を持つメンバーは存在しませんが、社会に希望がある限り、ワーキングメモリーに存在します。

Hope クラス

```
public class Hope {

    public Hope() {
```

```
    }
  }
```

政治家の誠実性に関するルール定義

誠実な政治家の例では、ワーキングメモリーに最低でも 1 名誠実な政治家が存在する場合は、**"We have an honest Politician"** ルールで論理的に新しい **Hope** オブジェクトを挿入します。すべての政治家が不誠実になると、**Hope** オブジェクトは自動的に取り除かれます。このルールでは、**salience** 属性の値が **10** となっており、他のルールより先に実行されます。理由は、この時点では **"Hope is Dead"** ルールが True となっているためです。

ルール "We have an honest politician"

```
rule "We have an honest Politician"
    salience 10
  when
    exists( Politician( honest == true ) )
  then
    insertLogical( new Hope() );
end
```

Hope オブジェクトが存在すると、すぐに **"Hope Lives"** ルールが一致して実行されます。**"Corrupt the Honest"** ルールよりも優先されるように、このルールにも **salience** 値を **10** に指定しています。

ルール "Hope Lives"

```
rule "Hope Lives"
    salience 10
  when
    exists( Hope() )
  then
    System.out.println("Hurrah!!! Democracy Lives");
end
```

最初は、誠実な政治家が 4 人いるため、このルールには 4 つのアクティベーションが存在し、すべてが競合しています。各ルールが順番に実行し、政治家が誠実でなくなるように、企業により各政治家を買収させていきます。政治家 4 人が全員買収されたら、プロパティーが **honest == true** の政治家はいなくなります。**"We have an honest Politician"** のルールは True でなくなり、論理的に挿入されるオブジェクト (最後に実行された **new Hope()** による) は自動的に取り除かれます。

ルール "Corrupt the Honest"

```
rule "Corrupt the Honest"
  when
    politician : Politician( honest == true )
    exists( Hope() )
  then
    System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );
    modify ( politician ) { honest = false };
end
```

真理維持システムにより **Hope** オブジェクトが自動的に取り除かれると、**Hope** に適用された条件付き要素 **not** は True でなくなり、 **"Hope is Dead"** ルールが一致して実行されます。

ルール "Hope is Dead"

```
rule "Hope is Dead"
  when
    not( Hope() )
  then
    System.out.println( "We are all Doomed!!! Democracy is Dead" );
end
```

実行と監査証跡
HonestPoliticianExample.java クラスでは、honest の状態が **true** に設定されている政治家 4 人が挿入され、定義したビジネスルールに対して評価されます。

HonestPoliticianExample.java クラスの実行

```
public static void execute( KieContainer kc ) {
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        ksession.fireAllRules();

        ksession.dispose();
    }
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.honestpolitician.HonestPoliticianExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの実行出力

```
Hurrah!!! Democracy Lives
I'm an evil corporation and I have corrupted President of Umpa Lumpa
I'm an evil corporation and I have corrupted Prime Minster of Cheeseland
I'm an evil corporation and I have corrupted Tsar of Pringapopaloo
I'm an evil corporation and I have corrupted Omnipotence Om
We are all Doomed!!! Democracy is Dead
```

この出力では、democracy lives に誠実な政治家が最低でも 1 人いることが分かります。ただし、各政治家は企業に買収されているため、全政治家が不誠実になり、民主性がなくなります。

この例の実行フローをさらに理解するために、**HonestPoliticianExample.java** クラスを変更し、**DebugRuleRuntimeEventListener** リスナーと監査ロガーを追加して実行の詳細を表示することができます。

監査ロガーを含む HonestPoliticianExample.java クラス

```
package org.drools.examples.honestpolitician;

import org.kie.api.KieServices;
import org.kie.api.event.rule.DebugAgendaEventListener; (1)
import org.kie.api.event.rule.DebugRuleRuntimeEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class HonestPoliticianExample {

    /**
     * @param args
     */
    public static void main(final String[] args) {
     KieServices ks = KieServices.Factory.get(); (2)
     //ks = KieServices.Factory.get();
        KieContainer kc = KieServices.Factory.get().getKieClasspathContainer();
        System.out.println(kc.verify().getMessages().toString());
        //execute( kc );
        execute( ks, kc); (3)
    }

    public static void execute( KieServices ks, KieContainer kc ) { (4)
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        // The application can also setup listeners (5)
        ksession.addEventListener( new DebugAgendaEventListener() );
        ksession.addEventListener( new DebugRuleRuntimeEventListener() );

        // Set up a file-based audit logger.
        ks.getLoggers().newFileLogger( ksession, "./target/honestpolitician" ); (6)

        ksession.fireAllRules();

        ksession.dispose();
    }

}
```

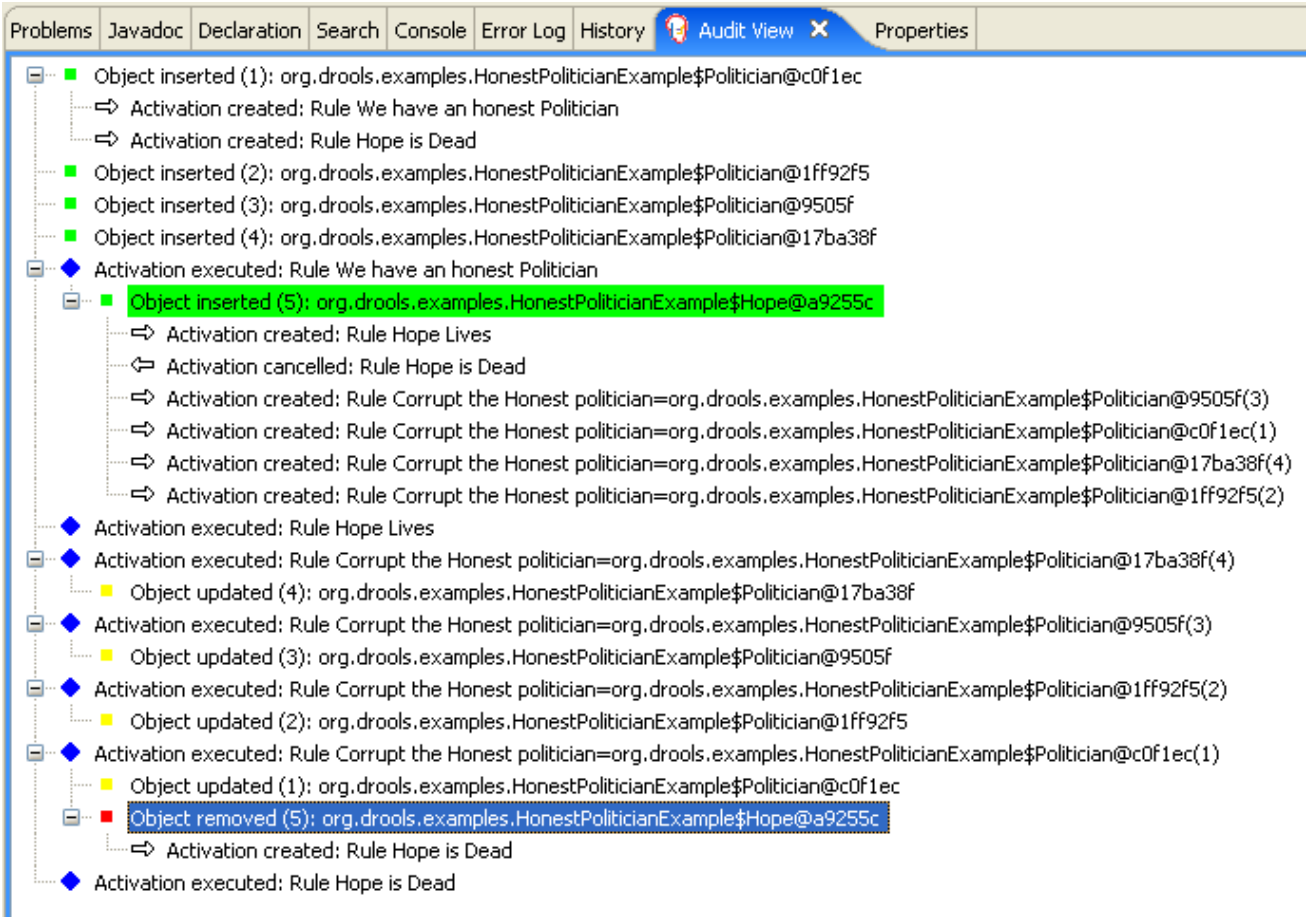
(1) **DebugAgendaEventListener** と **DebugRuleRuntimeEventListener** を処理するインポートにパッケージを追加します。

2. この監査ログは **KieContainer** レベルでは利用できないため、**KieServices Factory** 要素および **ks** 要素を作成してログを生成します。
3. **execute** メソッドを変更して **KieServices** と **KieContainer** の両方を使用します。
4. **execute** メソッドを変更して **KieContainer** に加えて **KieServices** で渡します。
5. リスナーを作成します。
6. ルールの実行後にデバッグビュー、**監査ビュー**、または IDE に渡すことが可能なログを構築します。

ログ機能を変更して誠実な政治家のサンプルを実行すると、**target/honestpolitician.log** から IDE デバッグビュー、または利用可能な場合には **監査ビュー** (IDE の一部では **Window → Show View**) に、監査ログファイルを読み込むことができます。

この例では、**監査ビュー** では、クラスやルールのサンプルで定義されているように、実行、挿入、取り消しのフローが示されています。

図21.18 誠実な政治家の例での監査ビュー



最初の政治家が挿入されると、2 つのアクティベーションが発生します。**"We have an honest Politician"** のルールは、**exists** の条件付き要素を使用するため、最初に挿入された政治家に対してのみ一度だけアクティベートされます。この条件付き要素は、政治家が最低でも 1 人挿入されると一致します。**Hope** オブジェクトがまだ挿入されていないため、ルール **"Hope is Dead"** もこの時点でアクティベートになります。**"We have an honest Politician"** ルールは、**"Hope is Dead"** ルールより、**salience** の値が高いため先に実行され、**Hope** オブジェクト (緑にハイライト) を挿入します。**Hope** オブジェクトを挿入すると、ルール **"Hope Lives"** が有効になり、ルール **"Hope is Dead"**

が無効になります。この挿入により、挿入された誠実な各政治家に対して **"Corrupt the Honest"** ルールがアクティベートになります。**"Hope Lives"** のルールが実行して、**"Hurrah!!!Democracy Lives"** が出力されます。

次に、政治家ごとに **"Corrupt the Honest"** ルールを実行して **"I'm an evil corporation and I have corrupted X"** と出力します。**X** は政治家の名前で、その政治家の誠実値が **false** に変更になります。最後の誠実な政治家が買収されると、真理維持システム (青でハイライト) により **Hope** が自動的に取り消されます。緑でハイライトされたエリアは、現在選択されている青のハイライトエリアの出元です。**Hope** ファクトが取り消されると、**"Hope is dead"** ルールが実行して **"We are all Doomed!!!Democracy is Dead"** が出力されます。

21.8. 数独例のデシジョン (複雑なパターン一致、コールバック、および GUI 統合)

人気の数字パズルに基づく数独の例の決定セットは、Red Hat Process Automation Manager のルールを使用して、さまざまな制約に基づいて大きな潜在的なソリューションを空間でソリューションを見つける方法を示しています。この例では、Red Hat Process Automation Manager ルールをグラフィカルユーザーインターフェイス (GUI) (この場合は Swing ベースのデスクトップアプリケーション) に統合する方法と、コールバックを使用して実行中の意思決定エンジンと対話し、ランタイム時に加えられた作業メモリー内の変更をもとに GUI を更新する方法を例示しています。

以下は数独の例の概要です。

- **名前**: **sudoku**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.sudoku.SudokuExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.sudoku.*.drl**
- **目的**: 複雑なパターン一致、問題解決、コールバック、および GUI 統合を例示します。

数独は、ロジックベースの数字配置パズルです。目的は、各列、各行、および各 3x3 ゾーンに 1 から 9 の数字が一度だけ含まれるように 9x9 のグリッドを埋めることです。パズルセッターでは、グリッド内の一部だけ記入されており、上記の制約ですべての空白を埋めるのがパズルの回答者のタスクです。

問題解決の一般的なストラテジーとして、新しい番号の挿入時に、特定の 3x3 ゾーン、行、および列で同じ番号がないことを確認します。この数独例のデシジョンセットでは、Red Hat Process Automation Manager ルールを使用して、さまざまな難易度の数独パズルを解き、無効なエントリーが含まれ、不備のあるパズルの解決を試みます。

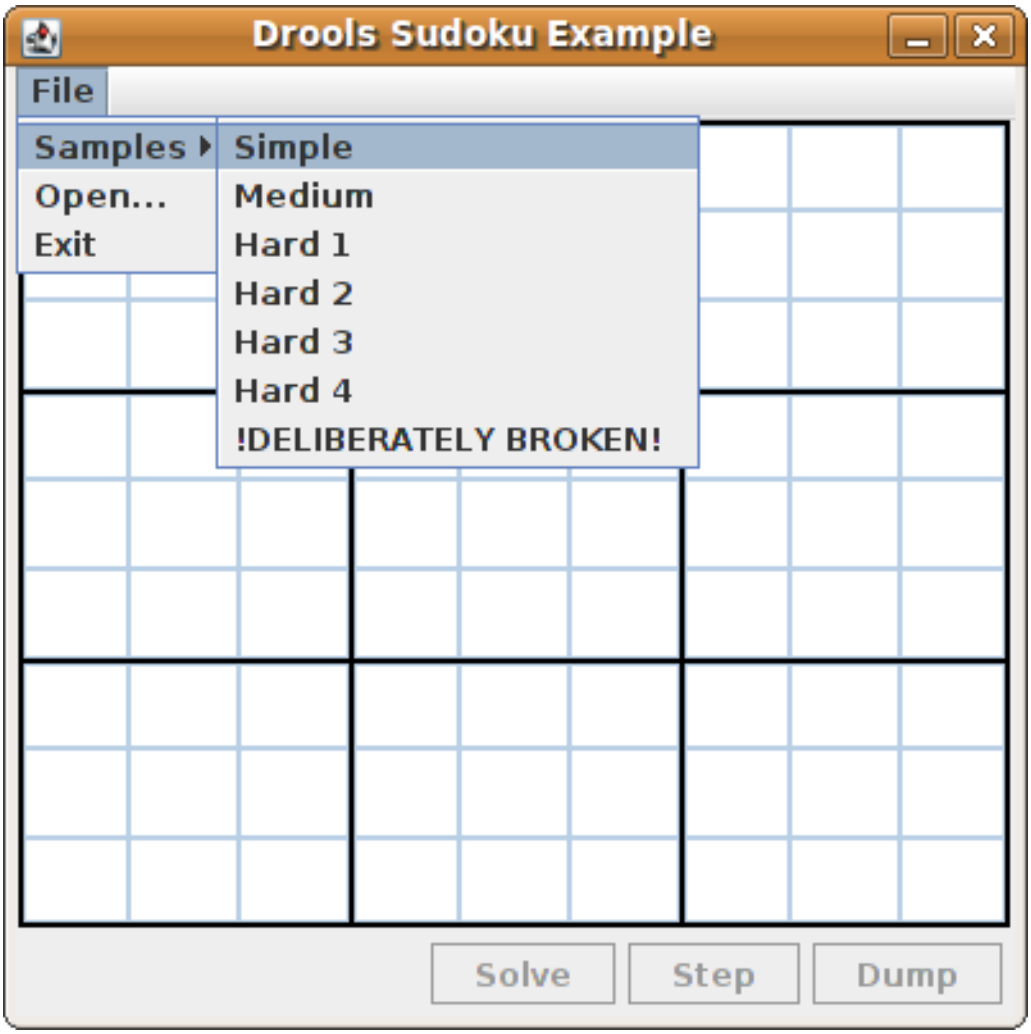
数独例の実行および対話

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.sudoku.SudokuExample** クラスを Java アプリケーションとして実行し、数独の例を実行します。

数独の例を実行すると、GUI ウィンドウ **Drools Sudoku Example** が表示されます。このウィンドウには空のグリッドが含まれていますが、プログラムには内部に保存されたさまざまなグリッドが含まれ、読み込んで解決できます。

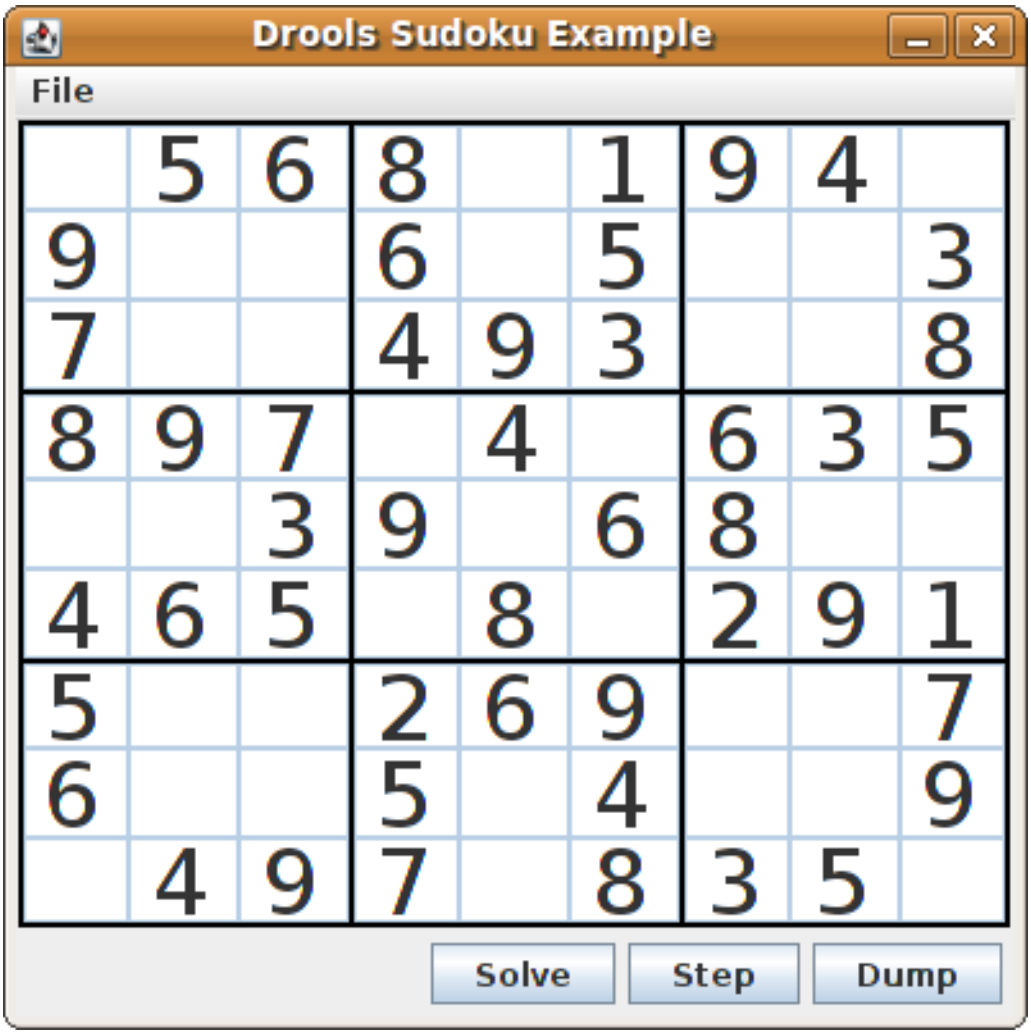
File → Samples → Simple をクリックして、例の 1 つを読み込みます。グリッドが読み込まれるまで、すべてのボタンが無効になっている点に注目してください。

図21.19 起動後の数独例の GUI



Simple サンプルを読み込むと、パズルの最初の状態に合わせて、グリッドが埋められます。

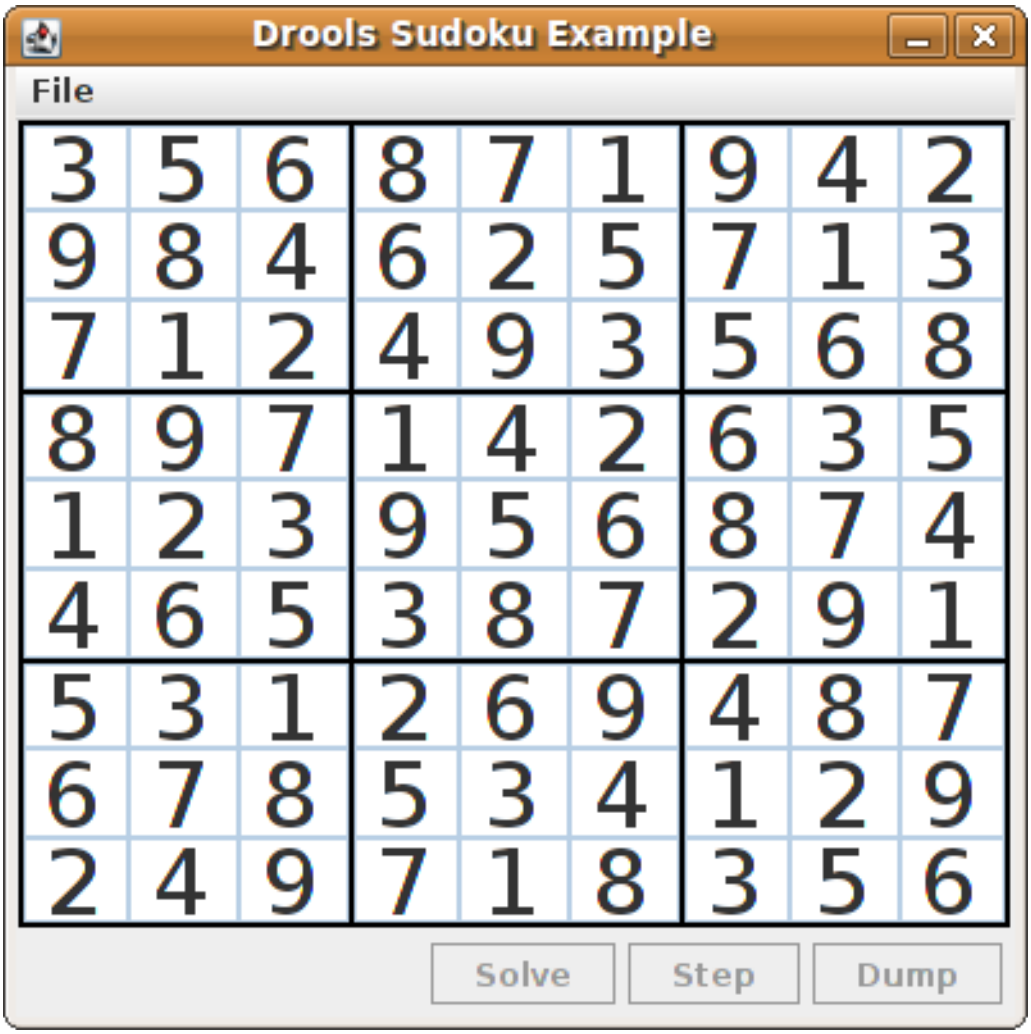
図21.20 Simple サンプルを読み込んだ後の数独例の GUI



以下のオプションから選択します。

- Solve をクリックして、数独の例に定義されているルールを実行し、残りの値を埋めていき、このボタンを再度無効にします。

図21.21 Simple サンプルの解決



- Step をクリックして、ルールセットに含まれる次の数字を表示します。IDE のコンソールウィンドウでは、解決手順を実行するルールに関する情報が詳細に表示されます。

IDE コンソールでの手順実行の出力

```
single 8 at [0,1]
column elimination due to [1,2]: remove 9 from [4,2]
hidden single 9 at [1,2]
row elimination due to [2,8]: remove 7 from [2,4]
remove 6 from [3,8] due to naked pair at [3,2] and [3,7]
hidden pair in row at [4,6] and [4,4]
```

- Dump をクリックしてグリッドの状態を表示します。セルには、解決済みの値か、残りの候補値が表示されます。

IDE コンソールでのダンプ実行の出力

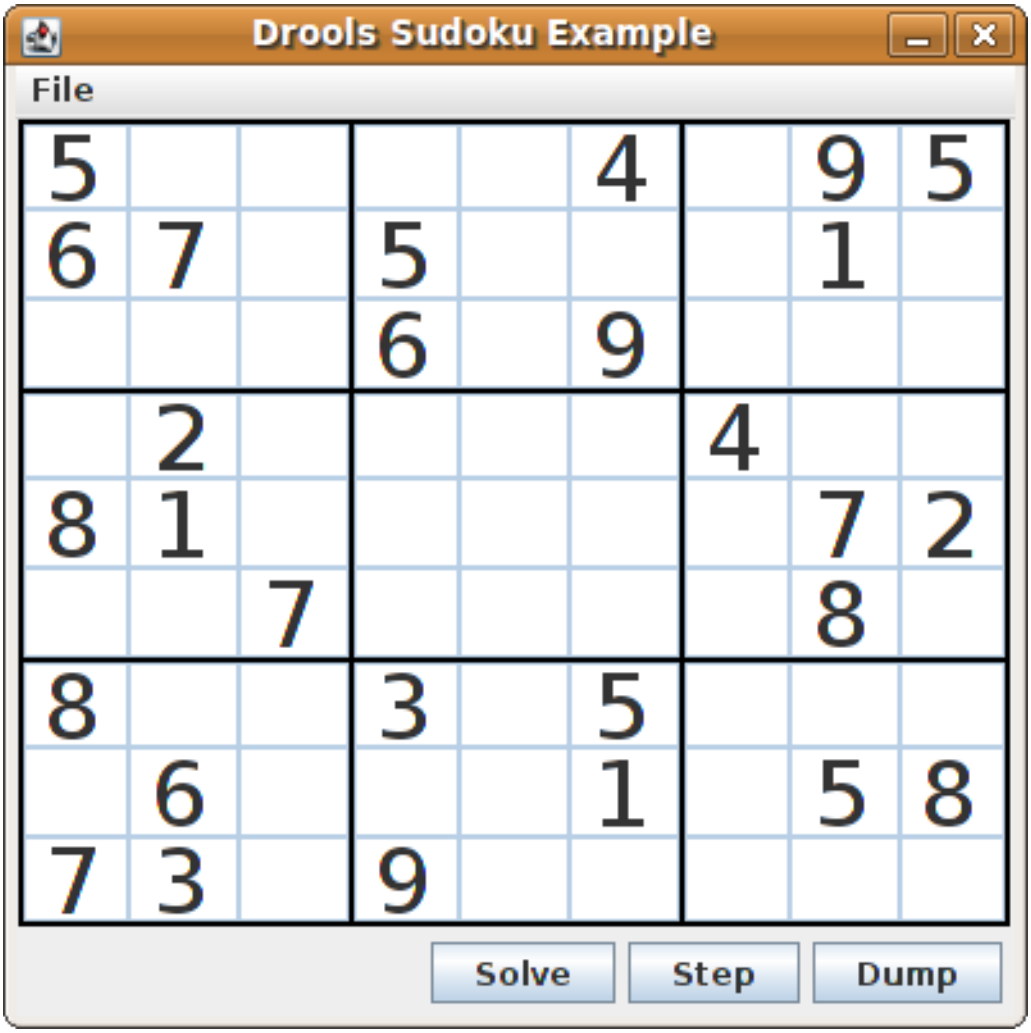
```
      Col: 0     Col: 1     Col: 2     Col: 3     Col: 4     Col: 5     Col: 6     Col: 7     Col: 8
Row 0:  123456789  --- 5 ---  --- 6 ---  --- 8 ---  123456789  --- 1 ---  --- 9 ---  --- 4 ---
123456789
Row 1:  --- 9 ---  123456789  123456789  --- 6 ---  123456789  --- 5 ---  123456789
123456789  --- 3 ---
Row 2:  --- 7 ---  123456789  123456789  --- 4 ---  --- 9 ---  --- 3 ---  123456789  123456789
--- 8 ---
```

```
Row 3:  --- 8 ---  --- 9 ---  --- 7 ---  123456789  --- 4 ---  123456789  --- 6 ---  --- 3 ---  --- 5 ---
Row 4:  123456789  123456789  --- 3 ---  --- 9 ---  123456789  --- 6 ---  --- 8 ---  123456789
123456789
Row 5:  --- 4 ---  --- 6 ---  --- 5 ---  123456789  --- 8 ---  123456789  --- 2 ---  --- 9 ---  --- 1 ---
Row 6:  --- 5 ---  123456789  123456789  --- 2 ---  --- 6 ---  --- 9 ---  123456789  123456789
--- 7 ---
Row 7:  --- 6 ---  123456789  123456789  --- 5 ---  123456789  --- 4 ---  123456789
123456789  --- 9 ---
Row 8:  123456789  --- 4 ---  --- 9 ---  --- 7 ---  123456789  --- 8 ---  --- 3 ---  --- 5 ---
123456789
```

数独の例には、不備のあるサンプルファイルが意図的に含められています。このファイルは、例で定義したルールを使用して解決できます。

File → Samples → !DELIBERATELY BROKEN! をクリックして、不備のあるサンプルを読み込みます。グリッドは、最初の行に **5** の値を 2 回表示できないにもかかわらず表示されるなど、問題が含まれた状態で表示されます。

図21.22 不備のある数独例の最初の状態



Solve をクリックしてこの無効なグリッドに解決ルールを適用します。数独の例に含まれる関連の解決ルールにより、サンプルの問題が検出され、できる限りパズルを解決します。このプロセスでは、すべてを完了させず、空白のセルをいくつか残します。

解決ルールのアクティビティーが IDE コンソールウィンドウに表示されます。

不備のあるサンプルでの問題検出

```
cell [0,8]: 5 has a duplicate in row 0
cell [0,0]: 5 has a duplicate in row 0
cell [6,0]: 8 has a duplicate in col 0
cell [4,0]: 8 has a duplicate in col 0
Validation complete.
```

図21.23 不備のあるサンプルの解決試行

Hard のラベルの付いた数独サンプルファイルはより複雑で、解決ルールを使用しても解決できない可能性があります。解決をしようとして失敗した場合は、IDE コンソールウィンドウに表示されます。

解決不可の Hard サンプル

```
Validation complete.
...
Sorry - can't solve this grid.
```

不備のあるサンプルを解決するためのルールでは、セルの候補となりえる値をもとにした標準の解決手法を実装します。たとえば、セットに値が 1 つ含まれる場合は、これが値になります。セルが 9 個あるグループの 1 つに値が 1 度挿入された場合に、ルールを使用して、特定のセルに対する値を持ち、タイプが **Setting** のファクトを挿入します。このファクトにより、そのセルが含まれるグループにある他のすべてのセルからこの値が削除され、この値が取り消されます。

この例の他のルールで、セルに入力可能な値を減らしていきます。**"naked pair"**、**"hidden pair in row"**、**"hidden pair in column"**、および **"hidden pair in square"** のルールでは、候補の絞り込みはできますが、回答を得ることはできません。**"X-wings in rows"**、"`X-wings in columns"`、**"intersection**

removal row"、および **"intersection removal column"** のルールは、より高度な絞り込みを実行します。

数独例のクラス

org.drools.examples.sudoku.swing パッケージには、以下のように、数独パズルのフレームワークを実装する主なクラスセットが含まれます。

- **SudokuGridModel** は、9x9 グリッドの **Cell** オブジェクトとして数独パズルを格納するために実装可能なインターフェイスを定義しています。
- **SudokuGridView** クラスは Swing コンポーネントで、**SudokuGridModel** クラス実装の視覚化が可能です。
- **SudokuGridEvent** クラスおよび **SudokuGridListener** クラスは、モデルとビューの間のステータスの変化をやり取りするために使用します。セルの値が解決または変更すると、イベントが実行します。
- **SudokuGridSamples** クラスは、デモ目的に一部入力されている数独パズルを複数提供します。

注記

このパッケージには、Red Hat Process Automation Manager ライブラリーの依存関係は含まれません。

org.drools.examples.sudoku パッケージには、以下のように、基本的な **Cell** オブジェクトと各種アグリゲーションを実装する主なクラスセットが含まれます。

- **CellRow**、**CellCol**、および **CellSqr** のサブタイプを含む **CellFile** クラス。これはすべて、**CellGroup** クラスのサブタイプになります。
- **Cell** と **CellGroup** は **SetOfNine** のサブクラスで、**Set<Integer>** 型の **free** プロパティーを提供します。**Cell** クラスは、個別の候補セットを表します。**CellGroup** は、セルの全候補セットの統合 (割り当ての必要がある数値セット) です。
  数独の例には、81 個の **Cell** と 27 個の **CellGroup** オブジェクト、**Cell** プロパティーの **cellRow**、**cellCol**、および **cellSqr** が提供するリンク、**CellGroup** プロパティー **cells** (**Cell** オブジェクトリスト) が提供するリストが含まれます。これらのコンポーネントを使用して、セルに値を割り当てたり、候補セットから値を取り除いたりできるように、特定の状態を検出するルールを記述できます。
- **Setting** クラスを使用して、値の割り当てに伴うオペレーションをトリガーします。**Setting** ファクトは、整合性の取れない中間の状態に対して反応しないように、新しい状況を検出する全ルールに配置して使用します。
- **Stepping** クラスは、優先順位が低いルールに使用して、**"Step"** が予期なく中断された場合に緊急停止を行います。この動作は、プログラムでパズルを解決できないということです。
- Main クラス **org.drools.examples.sudoku.SudokuExample** は、全コンポーネントを統合する Java アプリケーションを実装します。

数独の検証ルール (validate.drl)

数独の例の **validate.drl** ファイルには、セルグループで数が重複している状況を検出する検証ルールが含まれます。このグループは、**"validate"** アジェンダグループに統合され、ユーザーがパズルを読み込むと、明示的にルールをアクティベートできます。

"duplicate in cell ..." の 3 つのルールの **when** 条件はすべて以下の方法で機能します。

- このルールの最初の条件で、割り当てられた値でセルを特定します。
- このルールの 2 番目の条件では、3 つのセルグループのどれかを所属先にプルします。
- 最終条件は、ルールに従い、最初のセル、同じ行、列、または四角に入る値と同じセル (上記のセル以外) を検索します。

ルール "duplicate in cell ..."

```
rule "duplicate in cell row"
  when
    $c: Cell( $v: value != null )
    $cr: CellRow( cells contains $c )
    exists Cell( this != $c, value == $v, cellRow == $cr )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in row " + $cr.getNumber() );
end

rule "duplicate in cell col"
  when
    $c: Cell( $v: value != null )
    $cc: CellCol( cells contains $c )
    exists Cell( this != $c, value == $v, cellCol == $cc )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in col " + $cc.getNumber() );
end

rule "duplicate in cell sqr"
  when
    $c: Cell( $v: value != null )
    $cs: CellSqr( cells contains $c )
    exists Cell( this != $c, value == $v, cellSqr == $cs )
  then
    System.out.println( "cell " + $c.toString() + " has duplicate in its square of nine." );
end
```

ルール **"terminate group"** は最後に実行されます。このルールは、メッセージを出力して、シーケンスを停止します。

ルール "terminate group"

```
rule "terminate group"
    salience -100
  when
  then
    System.out.println( "Validation complete." );
    drools.halt();
end
```

数独の解決ルール (sudoku.drl)
数独の例の **sudoku.drl** ファイルには、3 種類のルールタイプが含まれます。1 つ目のグループは、セルへの数値の割り当てを処理して、2 つ目は実行可能な割り当てを検出して、3 つ目は候補セットからの値を削除します。

"set a value"、**"eliminate a value from Cell"**、および **"retract setting"** のルールは、**Setting** オブジェ

クトの有無により左右されます。最初のルールは、セルへの割り当てと、3 つのセルグループの **free** セットから値を削除する操作を処理します。また、ゼロの場合は、このグループでカウンターが 1 つ減り、**fireUntilHalt()** を呼び出した Java アプリケーションに制御を戻します。

"eliminate a value from Cell" ルールの目的は、新たに割り当てられたセルに関連する全セルの候補リストを絞り込むことです。最後に、すべての除外が完了したら、**"retract setting"** ルールにより、トリガーされている **Setting** ファクトを取り消します。

ルール "set a value"、"eliminate a value from a Cell"、および "retract setting"

```
// A Setting object is inserted to define the value of a Cell.
// Rule for updating the cell and all cell groups that contain it
rule "set a value"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // A matching Cell, with no value set
    $c: Cell( rowNo == $rn, colNo == $cn, value == null,
              $cr: cellRow, $cc: cellCol, $cs: cellSqr )

    // Count down
    $ctr: Counter( $count: count )
  then
    // Modify the Cell by setting its value.
    modify( $c ){ setValue( $v ) }
    // System.out.println( "set cell " + $c.toString() );
    modify( $cr ){ blockValue( $v ) }
    modify( $cc ){ blockValue( $v ) }
    modify( $cs ){ blockValue( $v ) }
    modify( $ctr ){ setCount( $count - 1 ) }
end

// Rule for removing a value from all cells that are siblings
// in one of the three cell groups
rule "eliminate a value from Cell"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    Cell( rowNo == $rn, colNo == $cn, value == $v, $exCells: exCells )

    // For all Cells that are associated with the updated cell
    $c: Cell( free contains $v ) from $exCells
  then
    // System.out.println( "clear " + $v + " from cell " + $c.posAsString()  );
    // Modify a related Cell by blocking the assigned value.
    modify( $c ){ blockValue( $v ) }
end

// Rule for eliminating the Setting fact
rule "retract setting"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )
```

```
    // The matching Cell, with the value already set
    $c: Cell( rowNo == $rn, colNo == $cn, value == $v )

    // This is the negation of the last pattern in the previous rule.
    // Now the Setting fact can be safely retracted.
    not( $x: Cell( free contains $v )
         and
         Cell( this == $c, exCells contains $x ) )
  then
    // System.out.println( "done setting cell " + $c.toString() );
    // Discard the Setter fact.
    delete( $s );
    // Sudoku.sudoku.consistencyCheck();
end
```

解決ルールを 2 つ使用して、セルに数字を割り当てることができる状況を検出します。**"single"** のルールは、**Cell** に、数字が 1 つだけの候補セットが含まれる場合に実行します。**"hidden single"** ルールは、候補が 1 つだけのセルが存在しない場合に実行しますが、セルに候補が含まれる場合は、セルが所属する 3 つのグループの 1 つに含まれるその他のすべてのセルに、この候補が存在しないということです。いずれのルールも **Setting** ファクトを作成して、挿入します。

ルール "single" および "hidden single"

```
// Detect a set of candidate values with cardinality 1 for some Cell.
// This is the value to be set.
rule "single"
  when
    // Currently no setting underway
    not Setting()

    // One element in the "free" set
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount == 1 )
  then
    Integer i = $c.getFreeValue();
    if (explain) System.out.println( "single " + i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, i ) );
end

// Detect a set of candidate values with a value that is the only one
// in one of its groups. This is the value to be set.
rule "hidden single"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Some integer
    $i: Integer()

    // The "free" set contains this number
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount > 1, free contains $i )

    // A cell group contains this cell $c.
    $cg: CellGroup( cells contains $c )
```

```
    // No other cell from that group contains $i.
    not ( Cell( this != $c, free contains $i ) from $cg.getCells() )
  then
    if (explain) System.out.println( "hidden single " + $i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, $i ) );
end
```

最大グループからのルール (個別または 2 ~ 3 のグループ単位) は、数独パズルを手作業で解決するのに使用する、さまざまな解決手法を実装します。

"naked pair" ルールは、グループの 2 つのセルで、全く同じ候補セットでサイズ **2** のものを検出します。これらの 2 つの値は、対象グループにあるその他のすべての候補セットから削除できます。

ルール "naked pair"

```
// A "naked pair" is two cells in some cell group with their sets of
// permissible values being equal with cardinality 2. These two values
// can be removed from all other candidate lists in the group.
rule "naked pair"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // One cell with two candidates
    $c1: Cell( freeCount == 2, $f1: free, $r1: cellRow, $rn1: rowNo, $cn1: colNo, $b1: cellSqr )

    // The containing cell group
    $cg: CellGroup( freeCount > 2, cells contains $c1 )

    // Another cell with two candidates, not the one we already have
    $c2: Cell( this != $c1, free == $f1 /*** , rowNo >= $rn1, colNo >= $cn1 ***/ ) from $cg.cells

    // Get one of the "naked pair".
    Integer( $v: intValue ) from $c1.getFree()

    // Get some other cell with a candidate equal to one from the pair.
    $c3: Cell( this != $c1 && != $c2, freeCount > 1, free contains $v ) from $cg.cells
  then
    if (explain) System.out.println( "remove " + $v + " from " + $c3.posAsString() + " due to naked pair
at " + $c1.posAsString() + " and " + $c2.posAsString() );
    // Remove the value.
    modify( $c3 ){ blockValue( $v ) }
end
```

3 つのルールの **"hidden pair in ..."** 関数は、ルール **"naked pair"** と同じように機能します。ルールはグループの 2 つのセルで 2 つの数字を検出します。どの値もこのグループの他のセルには入りません。つまり、他の候補はすべて、隠れたペアを持つ 2 つのセルから削除します。

ルール "hidden pair in ..."

```
// If two cells within the same cell group contain candidate sets with more than
// two values, with two values being in both of them but in none of the other
// cells, then we have a "hidden pair". We can remove all other candidates from
```

```
// these two cells.
rule "hidden pair in row"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Establish a pair of Integer facts.
    $i1: Integer()
    $i2: Integer( this > $i1 )

    // Look for a Cell with these two among its candidates. (The upper bound on
    // the number of candidates avoids a lot of useless work during startup.)
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellRow: cellRow )

    // Get another one from the same row, with the same pair among its candidates.
    $c2: Cell( this != $c1, cellRow == $cellRow, freeCount > 2, free contains $i1 && contains $i2 )

    // Ascertain that no other cell in the group has one of these two values.
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellRow.getCells() )
  then
    if( explain) System.out.println( "hidden pair in row at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    // Set the candidate lists of these two Cells to the "hidden pair".
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellCol: cellCol )
    $c2: Cell( this != $c1, cellCol == $cellCol, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellCol.getCells() )
  then
    if (explain) System.out.println( "hidden pair in column at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in square"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
            $cellSqr: cellSqr )
```

```
    $c2: Cell( this != $c1, cellSqr == $cellSqr, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellSqr.getCells() )
  then
    if (explain) System.out.println( "hidden pair in square " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end
```

2 つのルールは行と列で **"X-wings"** を処理します。2 つの異なる行 (または列) で、ある値を入力できるセルが 2 つしかなく、これらの候補が同じ列 (または行) に入る場合に、この列 (または行) のこの値に対する他の候補は除外できます。これらのルールの 1 つに含まれるパターンシーケンスに従うと、**same**、**only** などの用語で都合よく表現されている条件は、適切な制約が付けられたパターンになるか、**not** のプリフィックスが付きます。

ルール "X-wings in ..."

```
rule "X-wings in rows"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
                $ra: cellRow, $rano: rowNo,         $c1: cellCol,        $c1no: colNo )
    $cb1: Cell( freeCount > 1, free contains $i,
                $rb: cellRow, $rbno: rowNo > $rano,      cellCol == $c1 )
    not( Cell( this != $ca1 && != $cb1, free contains $i ) from $c1.getCells() )

    $ca2: Cell( freeCount > 1, free contains $i,
                cellRow == $ra, $c2: cellCol,       $c2no: colNo > $c1no )
    $cb2: Cell( freeCount > 1, free contains $i,
                cellRow == $rb,      cellCol == $c2 )
    not( Cell( this != $ca2 && != $cb2, free contains $i ) from $c2.getCells() )

    $cx: Cell( rowNo == $rano || == $rbno, colNo != $c1no && != $c2no,
               freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in rows " +
          $ca1.posAsString() + " - " + $cb1.posAsString() +
          $ca2.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "X-wings in columns"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
                $c1: cellCol, $c1no: colNo,         $ra: cellRow,        $rano: rowNo )
    $ca2: Cell( freeCount > 1, free contains $i,
                $c2: cellCol, $c2no: colNo > $c1no,      cellRow == $ra )
```

```
    not( Cell( this != $ca1 && != $ca2, free contains $i ) from $ra.getCells() )

    $cb1: Cell( freeCount > 1, free contains $i,
                cellCol == $c1, $rb: cellRow,  $rbno: rowNo > $rano )
    $cb2: Cell( freeCount > 1, free contains $i,
                cellCol == $c2,       cellRow == $rb )
    not( Cell( this != $cb1 && != $cb2, free contains $i ) from $rb.getCells() )

    $cx: Cell( colNo == $c1no || == $c2no, rowNo != $rano && != $rbno,
               freeCount > 1, free contains $i )
  then
    if (explain) {
        System.out.println( "X-wing with " + $i + " in columns " +
            $ca1.posAsString() + " - " + $ca2.posAsString() +
            $cb1.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString()  );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

この 2 つのルール **intersection removal …**は、1 つの行または 1 つの列のいずれかで、1 つの四角の中で一部の数字が制限されたことに基づきます。これは、この番号が行または列の 2 つまたは 3 つのセルのいずれかにある必要があり、グループの他のすべてのセルの候補セットから削除できることを意味します。このパターンは、発生制限を確立して、同じセルファイルの中、かつ四角の外のセルそれぞれに対して実行されます。

ルール "intersection removal ..."

```
rule "intersection removal column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cc: cellCol )
    // Does not occur in another cell of the same square and a different column
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellCol != $cc )

    // A cell exists in the same column and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellCol == $cc, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "column elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "intersection removal row"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
```

```
    $c: Cell( free contains $i, $cs: cellSqr, $cr: cellRow )
    // Does not occur in another cell of the same square and a different row.
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellRow != $cr )

    // A cell exists in the same row and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellRow == $cr, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "row elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

これらのルールは、すべてではありませんが、多くの数独パズルでは十分です。非常に難度の高いグリッドを解決するには、ルールセットにさらに複雑なルールが必要です。(最終的には、パズルは試行錯誤でしか解決できません)。

21.9. CONWAY の GAME OF LIFE 例のデシジョン (ルールフローグループおよび GUI 統合)

John Conway による有名なセルオートマトン (CA: Cellular automation) をベースにした Conway の Game of Life 例のデシジョンセットは、ルールでルールフローグループを使用してルール実行を制御する方法を例示します。またこの例は、Red Hat Process Automation Manager ルールをグラフィカルユーザーインターフェイス (GUI) と統合する方法も例示しています。今回は、Conway の Game of Life を Swing ベースで実装しています。

以下は、Conway の Game of Life の例の概要です。

- **名前**: **conway**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.conway.ConwayRuleFlowGroupRun**、**org.drools.examples.conway.ConwayAgendaGroupRun**
- **モジュール**: **droolsjbpm-integration-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.conway.*.drl**
- **目的**: ルールフローグループと GUI 統合を例示します。

注記

Conway の Game of Life の例は、Red Hat Process Automation Manager に含まれる他のデシジョンセットの例の多くとは異なり、Red Hat カスタマーポータル から取得する **Red Hat Process Automation Manager 7.11.0 Source Distribution**の ~/**rhpam-7.11.0-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples** におかれています。

Conway の Game of Life では、初期設定または定義済みのプロパティーで高度なパターンを作成して、初期状態からどのように進化していくかを観察することで、ユーザーはゲームと対話します。ゲームの目的は、世代ごとに人口の成長を表示します。各世代は、すべてのセル (細胞) が同時に進化していき、

前の世代をもとにして生み出されます。

以下の基本的なルールで、次の世代がどのようになるかを制御していきます。

- 生きているセルの近傍に、生きているセルが 2 個未満の場合は、孤独で死んでしまう。
- 生きているセルの近傍に、生きているセルが 4 個以上ある場合は、過密で死んでしまう。
- 死亡したセルの近傍に、生きているセルがちょうど 3 つある場合には、このセルは生き返る。

この基準のいずれも満たさないセルは、そのまま次の世代に残ります。

Conway の Game of Life の例は、**ruleflow-group** 属性が含まれる Red Hat Process Automation Manager ルールで、ゲームに実装されているパターンを定義します。この例には、アジェンダグループを使用して同じ動作を行うデシジョンセットのバージョンも含まれています。アジェンダグループは、デシジョンエンジンアジェンダのパーティションを作成して、ルールのグループを実行制御できるようにします。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

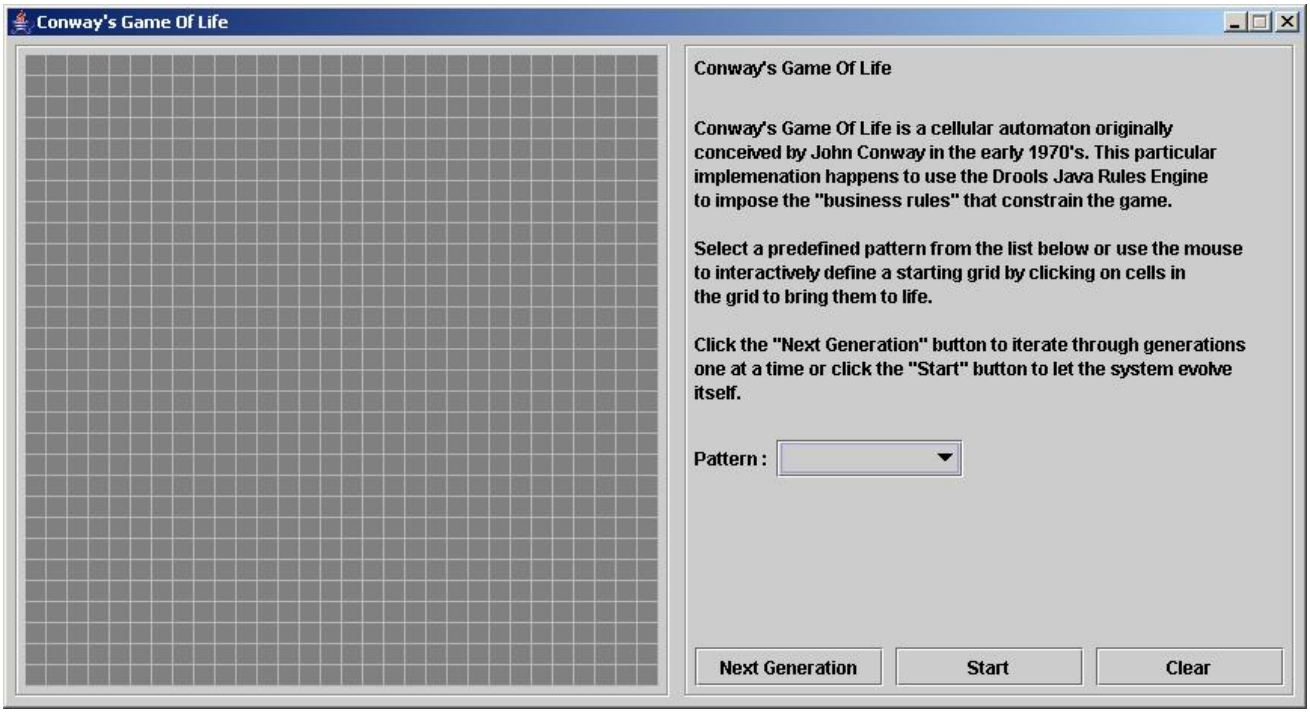
この概要では、Conway の例でアジェンダグループを使用したバージョンは触れません。アジェンダグループの詳細情報は、特にアジェンダグループについて対応している Red Hat Process Automation Manager 例のデシジョンセットを参照してください。

Conway 例の実行および対話

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.conway.ConwayRuleFlowGroupRun** クラスを Java アプリケーションとして実行し、Conway の例を実行します。

Conway の例を実行すると、**Conway's Game of Life** GUI ウィンドウが表示されます。このウィンドウには、空のグリッドまたはアリーナが含まれており、ここで生命のシミュレーションが行われます。システムにまだ生きているセルが含まれていないため、グリッドは最初は空白です。

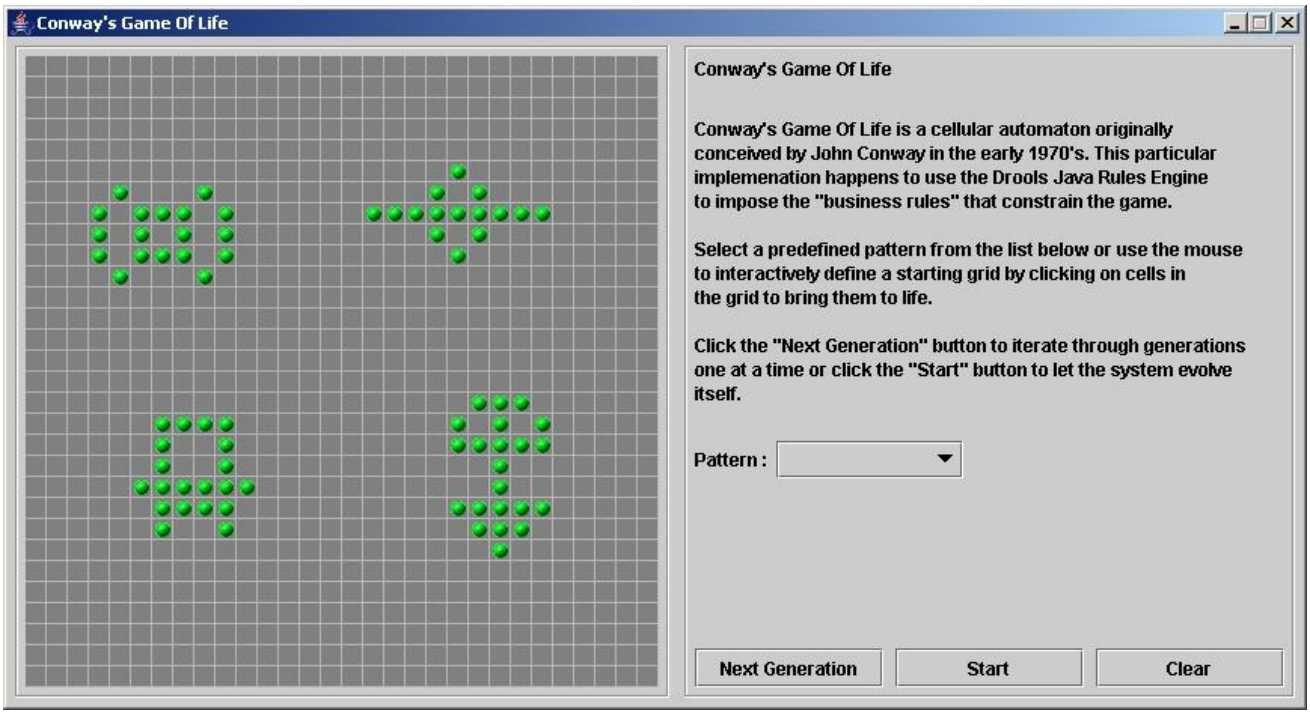
図21.24 起動後の Conway 例の GUI



パターン のドロップダウンメニューから事前定義済みのパターンを選択して、**次の世代** をクリックし、各人口の世代をクリックしていきます。セルは生きているか、死んでいるかのどちらかで、生きて

いるセルには緑のボールが含まれます。最初のパターンから人口が進化するにつれ、ゲームのルールをもとに、セルが近傍のセルに合わせて、生存するか、死亡していきます。

図21.25 Conway の例の世代進化



近傍には、上下左右のセルだけでなく対角線上につながっているセルも含まれるため、各セルには合計 8 つの近傍があります。例外は、角のセルと 4 辺上にあるセルで、それぞれ順に近傍が 3 つだけと、5 つだけになります。

セルをクリックすることで手動で介入して、セルを作成することも、死亡させることもできます。

最初のパターンから自動的に進化を実行するには、**スタート** をクリックします。

ルールグループを使用する Conway 例のルール

ConwayRuleFlowGroupRun の例のルールは、ルールフローグループを使用して、ルール実行を制御します。ルールフローグループは、**ruleflow-group** ルール属性に関連付けられたルールのグループです。これらのルールは、このグループがアクティベートされたときにしか実行されません。グループ自体は、ルールフローの図の詳細がグループを表すノードに到達してからでないと、アクティブになりません。

Conway の例では、ルールに以下のルールフローグループを使用します。

- **"register neighbor"**
- **"evaluate"**
- **"calculate"**
- **"reset calculate"**
- **"birth"**
- **"kill"**
- **"kill all"**

Cell オブジェクトはすべて KIE セッションに挿入され、**"register neighbor"** ルールフローグループの

"register ..." ルールがルールフロー処理により実行できるようになります。4 つのルールが含まれるこのグループは、セル同士の Neighbor の関係と、北東、北、北西、西の近傍との **Neighbor** の関係を作り出します。

この関係は双方向で、他の 4 方向を処理します。各辺上のセルは、特別な対応は必要ありません。これらのセルは、近傍のセルがなければペアは作成されません。

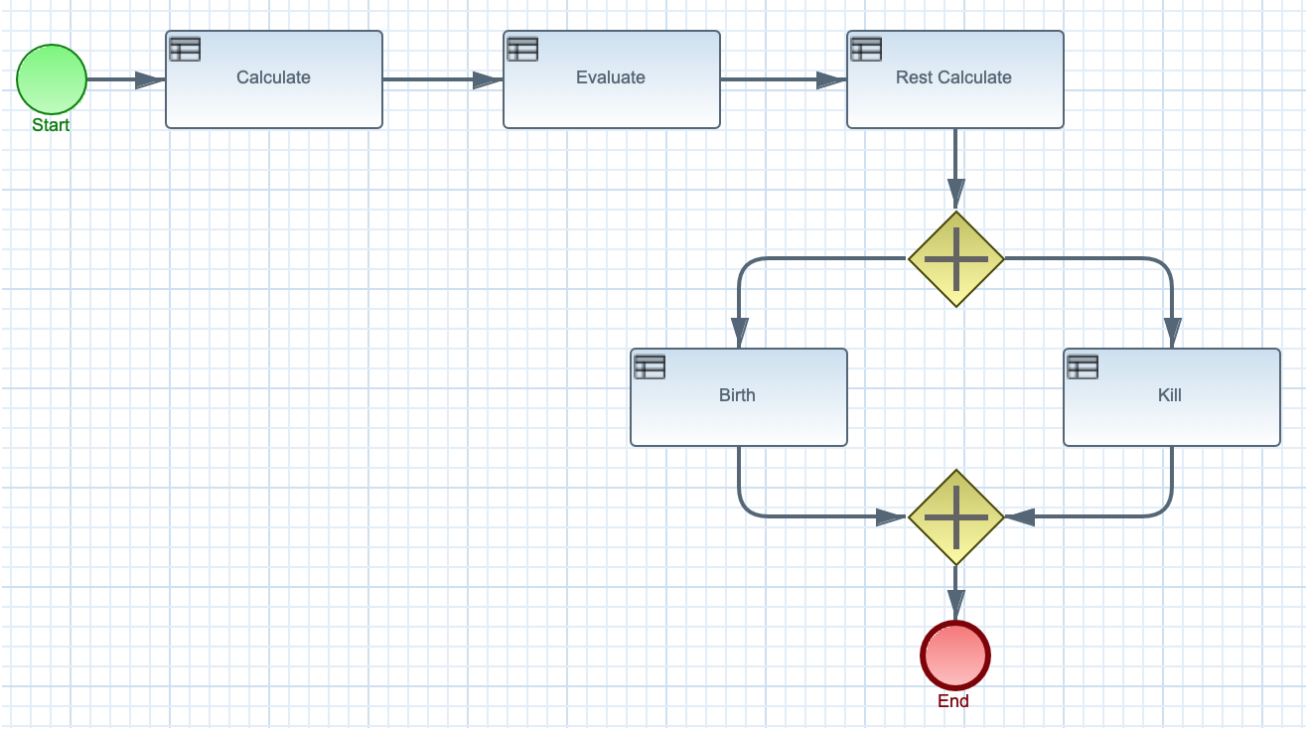
これらのルールに対して、すべてのアクティベーションが実行されるまで、全セルは、近傍の全セルと関係があります。

ルール "register ..."

```
rule "register north east"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northEast : Cell( row  == ($row - 1), col == ( $col + 1 ) )
  then
    insert( new Neighbor( $cell, $northEast ) );
    insert( new Neighbor( $northEast, $cell ) );
end

rule "register north"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $north : Cell( row  == ($row - 1), col == $col )
  then
    insert( new Neighbor( $cell, $north ) );
    insert( new Neighbor( $north, $cell ) );
end

rule "register north west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northWest : Cell( row  == ($row - 1), col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $northWest ) );
    insert( new Neighbor( $northWest, $cell ) );
end

rule "register west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $west : Cell( row  == $row, col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $west ) );
    insert( new Neighbor( $west, $cell ) );
end
```

全セルが挿入されたら、Java コードはグリッドにパターンを適用し、特定のセルを **Live** に設定します。次に、ユーザーが **スタート** または **次の世代** をクリックすると、**Generation** のルールフローが実行されます。このルールフローは、世代のサイクルごとにセルの変更をすべて管理します。

図21.26 世代のルールフロー



ルールフロープロセスは、実行可能なグループに **"evaluate"** ルールフローグループおよびアクティブなルールを追加します。このグループの **"Kill the ..."** と **"Give Birth"** ルールを使用して、細胞の誕生または死亡セルにゲームのルールを適用します。この例では、**phase** 属性を使用して、特定のルールグループで **Cell** オブジェクトの推論をトリガーします。通常は、フェーズはルールフロープロセス定義に含まれるルールフローグループに紐づけされています。

この例では、変更の適用前に評価を完全に完了しておく必要があるため、この時点では **Cell** オブジェクトの状態は変更されません。細胞の **phase** を **Phase.KILL** または **Phase.BIRTH** に適用し、後ほど **Cell** オブジェクトに適用されたアクションを制御するのに使用します。

ルール "Kill the ..." および "Give Birth"

```
rule "Kill The Lonely"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has fewer than 2 live neighbors.
    theCell: Cell( liveNeighbors < 2, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Kill The Overcrowded"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has more than 3 live neighbors.
    theCell: Cell( liveNeighbors > 3, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
```

```
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Give Birth"
    ruleflow-group "evaluate"
    no-loop
  when
    // A dead cell has 3 live neighbors.
    theCell: Cell( liveNeighbors == 3, cellState == CellState.DEAD,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        theCell.setPhase( Phase.BIRTH );
    }
end
```

グリッド内の全 **Cell** オブジェクトが評価されると、この例では **"reset calculate"** ルールを使用して **"calculate"** ルールフローグループのアクティベーションを消去します。次に、ルールフローグループがアクティベートされると、**"kill"** と **"birth"** のルールを有効にするルールフローに分岐を挿入します。これらのルールにより状態の変更が適用されます。

ルール "reset calculate"、"kill"、および "birth"

```
rule "reset calculate"
    ruleflow-group "reset calculate"
  when
  then
    WorkingMemory wm = drools.getWorkingMemory();
    wm.clearRuleFlowGroup( "calculate" );
end

rule "kill"
    ruleflow-group "kill"
    no-loop
  when
    theCell: Cell( phase == Phase.KILL )
  then
    modify( theCell ){
        setCellState( CellState.DEAD ),
        setPhase( Phase.DONE );
    }
end

rule "birth"
    ruleflow-group "birth"
    no-loop
  when
    theCell: Cell( phase == Phase.BIRTH )
  then
    modify( theCell ){
        setCellState( CellState.LIVE ),
        setPhase( Phase.DONE );
    }
end
```

この段階では、複数の **Cell** オブジェクトの状態が **LIVE** または **DEAD** のいずれかに変更されています。この例では、細胞が生存または死亡すると、**"Calculate ..."** ルールの **Neighbor** 関係を使用して、周辺のすべての細胞に繰り返し実行することで、**liveNeighbor** の数が増減します。数が変更した細胞は、**EVALUATE** フェーズに設定され、ルールフロー処理の評価段階の推論に含められるようにします。

生存数が判断され、全細胞に設定されると、ルールフロープロセスが終了します。ユーザーが最初に **Start** をクリックした場合は、その時点でデシジョンエンジンによりルールフローが再起動します。ユーザーが最初に **Next Generation** をクリックした場合は、ユーザーが別の世代を要求することができます。

ルール "Calculate ..."

```
rule "Calculate Live"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.LIVE )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() + 1 ),
        setPhase( Phase.EVALUATE );
    }
end

rule "Calculate Dead"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.DEAD )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() - 1 ),
        setPhase( Phase.EVALUATE );
    }
end
```

21.10. HOUSE OF DOOM 例のデシジョン (後向き連鎖および再帰)

House of Doom のデシジョンセットの例では、デシジョンエンジンが後向き連鎖と再帰を使用して、階層システムで定義した目的やサブゴールに到達する方法を説明します。

以下は House of Doom の例の概要です。

- **名前**: **backwardchaining**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.backwardchaining.HouseOfDoomMain**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション

- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.backwardchaining.BC-Example.drl**

- **目的**: 後向き連鎖と再帰を例示します。

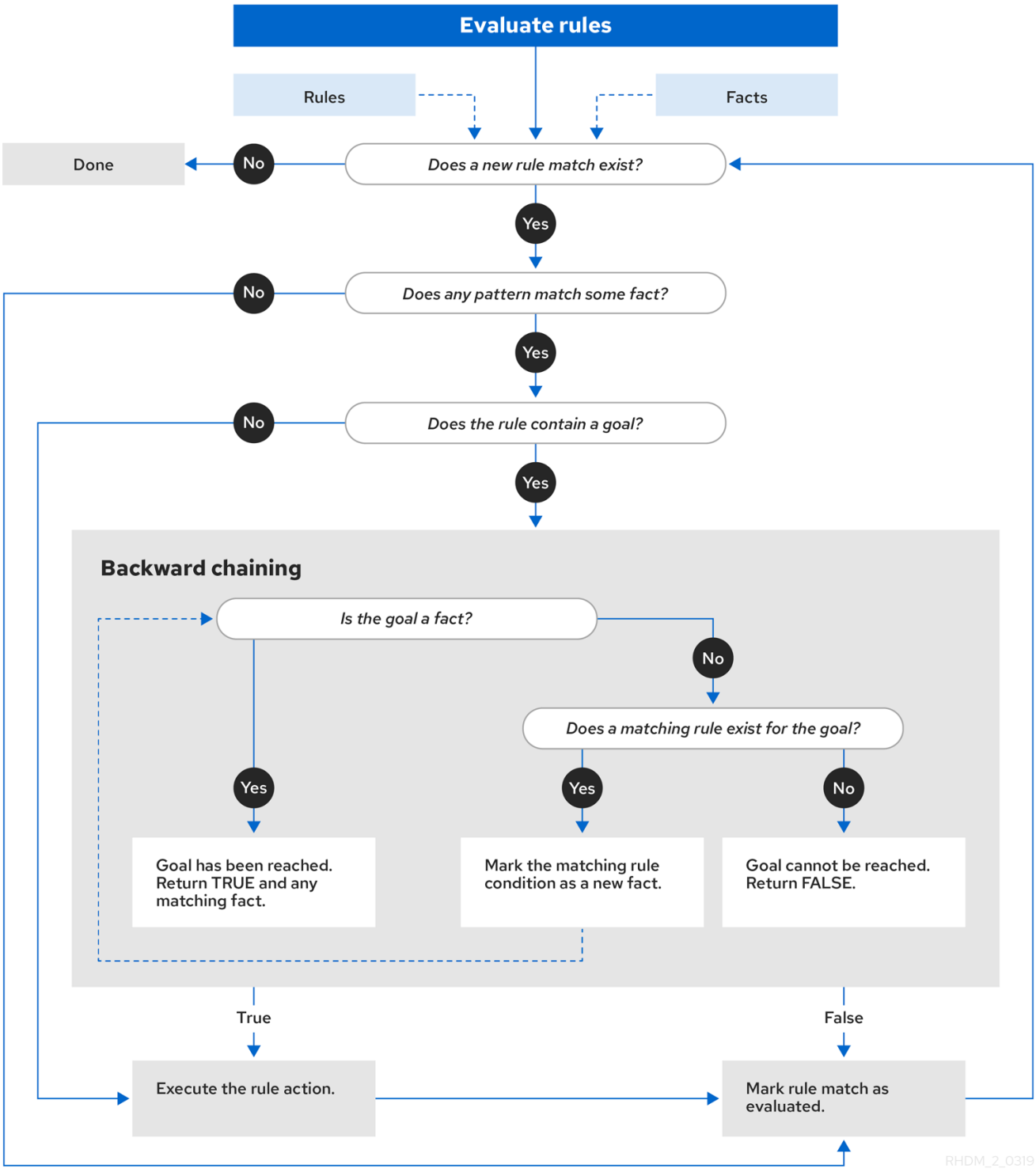
後向き連鎖のルールシステムは、通常再帰を使用して、デシジョンエンジンが満たそうとする結論から開始する目的駆動型のシステムです。システムが結論または目的に到達できない場合は、サブとなる目的、つまり、現在の目的の一部を完了する結論を検索します。システムは、最初の結論が満たされるか、すべてのサブとなる目的が満たされるまでこのプロセスを続行します。

反対に、前向き連鎖のルールシステムは、デシジョンエンジンのワーキングメモリーにあるファクトで開始して、そのファクトへの変更に反応するデータ駆動型のシステムです。オブジェクトがワーキングメモリーに挿入されると、その変更の結果として True となるルールの条件はすべて、アジェンダによって実行されるようにスケジュールされます。

Red Hat Process Automation Manager のデシジョンエンジンは、前向き連鎖と後向き連鎖の両方を使用してルールを評価します。

以下の図は、デシジョンエンジンが、ロジックフローで後向き連鎖のセグメントと、前向き連鎖全体を使用してルールを評価する方法を例示します。

図21.27 前向き連鎖と後向き連鎖を使用したルール評価のロジック



House of Doom の例は、さまざまなクエリータイプが含まれるルールを使用し、部屋の場所と家の中のアイテムを探し出します。**Location.java** のサンプルクラスには、この例で使用する **item** と **location** 要素が含まれます。**HouseOfDoomMain.java** のサンプルクラスで、家の該当の場所にアイテムまたは部屋を挿入して、ルールを実行します。

HouseOfDoomMain.java クラスでのアイテムと場所

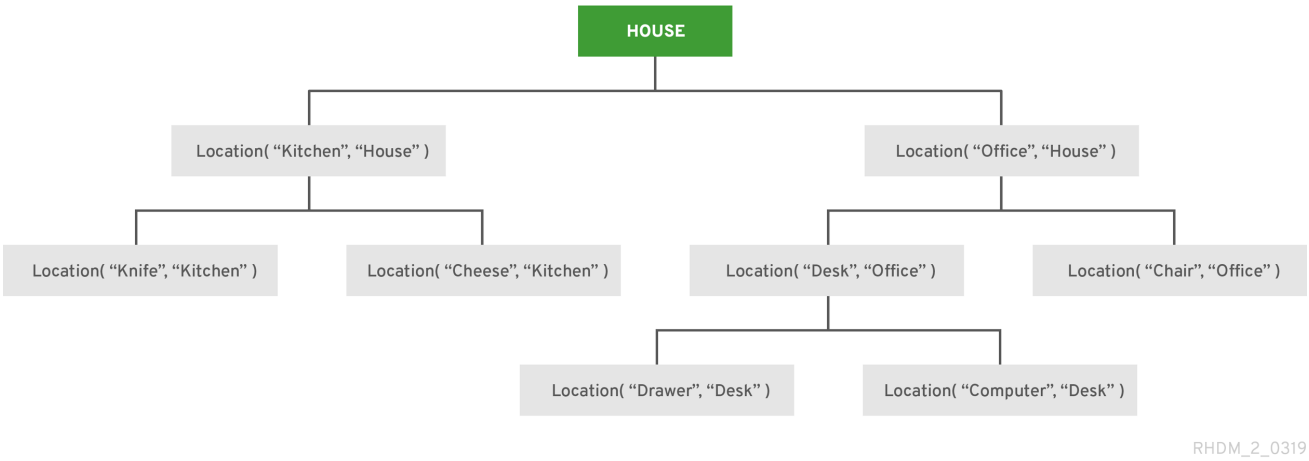
```
ksession.insert( new Location("Office", "House") );
ksession.insert( new Location("Kitchen", "House") );
ksession.insert( new Location("Knife", "Kitchen") );
ksession.insert( new Location("Cheese", "Kitchen") );
```

```
ksession.insert( new Location("Desk", "Office") );
ksession.insert( new Location("Chair", "Office") );
ksession.insert( new Location("Computer", "Desk") );
ksession.insert( new Location("Drawer", "Desk") );
```

ルールの例では、家の構造の中で全アイテムおよび部屋の場所を判断するのに、後向き連鎖と再帰を使用します。

以下の図は、House of Doom の構造と、その構造内のアイテムと部屋を示しています。

図21.28 House of Doom の構造



この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.backwardchaining.HouseOfDoomMain** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの実行出力

```
go1
Office is in the House
---
go2
Drawer is in the House
---
go3
---
Key is in the Office
---
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
---
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
```

```
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

この例のルールはすべて実行し、家の中の全アイテムの場所を検出して、出力でそれぞれの場所を出力します。

再帰クエリーおよび関連のルール
再帰クエリーは、要素間の関係におけるデータ構造階層を使用して繰り返し検索を行います。

House of Doom の例では、**BC-Example.drl** ファイルに、この例のルールの大半が使用する **isContainedIn** クエリーが含まれており、家のデータ構造を再帰的に評価して、デシジョンエンジンに挿入するデータがないかを確認します。

BC-Example.drl の再帰クエリー

```
query isContainedIn( String x, String y )
  Location( x, y; )
  or
  ( Location( z, y; ) and isContainedIn( x, z; ) )
end
```

"go" のルールは、システムに挿入する文字列をすべて出力し、アイテムをどのように導入し、**"go1"** ルールが **isContainedIn** クエリーを呼び出すかを判断します。

ルール "go" および "go1"

```
rule "go" salience 10
  when
    $s : String()
  then
    System.out.println( $s );
end

rule "go1"
  when
    String( this == "go1" )
    isContainedIn("Office", "House"; )
  then
    System.out.println( "Office is in the House" );
end
```

この例は、**"go1"** 文字列をデシジョンエンジンに挿入して、**"go1"** ルールを有効化し、**House** の場所にある **Office** アイテムを検出します。

文字列の挿入とルールの実行

```
ksession.insert( "go1" );
ksession.fireAllRules();
```

IDE コンソールでの ルール "go1" の出力

```
go1
Office is in the House
```

推移閉包ルール
推移閉包は、階層構造の複数レベルであり、上層にある親要素に含まれる要素間の関係です。

"go2" ルールは、**Drawer** と **House** の推移閉包の関係を特定します。**Drawer** は、**House** の中の、**Office** の中の、**Desk** の中にあります。

```
rule "go2"
  when
    String( this == "go2" )
    isContainedIn("Drawer", "House"; )
  then
    System.out.println( "Drawer is in the House" );
end
```

この例は、**"go2"** 文字列をデシジョンエンジンに挿入して、**"go2"** ルールを有効化し、最終的に **House** の場所に含まれる **Drawer** アイテムを検出します。

文字列の挿入とルールの実行

```
ksession.insert( "go2" );
ksession.fireAllRules();
```

IDE コンソールのルール "go2" の出力

```
go2
Drawer is in the House
```

デシジョンエンジンは、以下のロジックをもとにこの結果を判断します。

1. クエリーは再帰的に、家の中の複数レベルを検索して、**Drawer** と **House** の間の推移閉包を検出します。
2. **Drawer** は **House** に直接含まれないため、**Location(x, y;)** を使用する代わりに、このクエリーは **(z, y;)** の値を使用します。
3. **z** の引数は現在バインドされておらず、値が指定されていないため、引数に含まれるものはすべて返されます。
4. **y** の引数は現在、**House** にバインドされているため、**z** は **Office** と **Kitchen** を返します。

5. クエリーは、**Office** からの情報を収集して、**Drawer** が **Office** に含まれているかを再帰的にチェックします。これらのパラメーターに対して、クエリーの行 **isContainedIn(x, z;)** が呼び出されます。

6. **Office** に直接含まれる **Drawer** が存在しないため、一致するものはありません。

7. **z** のバインドがない場合、このクエリーでは **Office** 内のデータが返され、**z == Desk** と判断されます。

```
isContainedIn(x==drawer, z==desk)
```

8. **isContainedIn** クエリーは再帰的に 3 回検索し、3 回目に、このクエリーにより **Desk** の中に **Drawer** があることが検出されます。

```
Location(x==drawer, y==desk)
```

9. 最初の場所で上記が一致した後に、このクエリーにより再帰的に構造を上方向に検索し、**Drawer** が **Desk** の中に、**Desk** が **Office** の中に、**Office** が **House** の中にあることを判断します。このように、**Drawer** は **House** の中にあるため、このルールは満たされます。

リアクティブクエリールール

リアクティブクエリーでは、データ構造の階層を検索して、要素間に関係があるかを確認し、構造内の要素が変更されると動的に更新されます。

"go3" ルールは、リアクティブクエリーとして機能し、推移閉包により、新しいアイテム **Key** が **Office** に含まれるかどうかを検出します (**Office** の中の **Drawer** の中の **Key** など)。

ルール "go3"

```
rule "go3"
  when
    String( this == "go3" )
    isContainedIn("Key", "Office"; )
  then
    System.out.println( "Key is in the Office" );
end
```

この例は、**"go3"** 文字列をデシジョンエンジンに挿入して、**"go3"** ルールを有効にします。最初は、**Key** が家の構造に存在するため、このルールは満たされず、出力が生成されません。

文字列の挿入とルールの実行

```
ksession.insert( "go3" );
ksession.fireAllRules();
```

IDE コンソールのルール "go3" の出力 (条件を満たさない)

```
go3
```

この例では、**Office** の中にある **Drawer** の場所に、新しいアイテム **Key** を挿入します。この変更で、**"go3"** ルールの推移閉包が満たされ、それに合わせて出力が生成されます。

新規アイテムの場所の挿入とルールの実行

```
ksession.insert( new Location("Key", "Drawer") );
ksession.fireAllRules();
```

IDE コンソールのルール "go3" の出力 (条件を満たす)

```
Key is in the Office
```

またこの変更で、クエリーにより、後に続く再帰検索に含まれるよう、この構造に別のレベルが追加されます。

ルールにバインドなしの引数が含まれたクエリー
バインドなしの引数が 1 つ以上あるクエリーでは、クエリーの定義済み (バインドされている) 引数に含まれる未定義 (バインドされていない) のアイテムをすべて返します。クエリー内の引数でバインドされているものがない場合、クエリーはクエリーの範囲内のアイテムをすべて返します。

"go4" ルールは、バインドされている引数を使用して、**Office** 内の特定のアイテムを検索するのではなく、バインドされていない引数 **thing** を使用して、バインドされている引数 **Office** 内の全アイテムを検索します。

ルール "go4"

```
rule "go4"
  when
    String( this == "go4" )
    isContainedIn(thing, "Office"; )
  then
    System.out.println( thing + "is in the Office" );
end
```

この例では **"go4"** 文字列をデシジョンエンジンに挿入して、**"go4"** ルールをアクティベートし、**Office** の全アイテムを返します。

文字列の挿入とルールの実行

```
ksession.insert( "go4" );
ksession.fireAllRules();
```

IDE コンソールのルール "go4" の出力

```
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
```

"go5" ルールは、バインドされていない引数 **thing** と **location** を使用して、**House** の全データ構造の中に含まれる全アイテムとその場所を検索します。

ルール "go5"

```
rule "go5"
  when
```

```
    String( this == "go5" )
    isContainedIn(thing, location; )
  then
    System.out.println(thing + " is in " + location );
end
```

この例は **"go5"** 文字列をデシジョンエンジンに挿入して、**"go5"** ルールをアクティベートし、**House** データ構造に含まれる全アイテムとその場所を返します。

文字列の挿入とルールの実行

```
ksession.insert( "go5" );
ksession.fireAllRules();
```

IDE コンソールのルール "go5" の出力

```
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

第22章 DRL 使用時のパフォーマンスチューニングに関する考慮点

以下の主要な概念または推奨のプラクティスを使用すると、DRL ルールとデシジョンエンジンのパフォーマンス最適化に役立ちます。本セクションではこの概念についてまとめており、随時、他のドキュメントを相互参照して詳細を説明します。本セクションは、Red Hat Process Automation Manager の新しいリリースで、必要に応じて拡張または変更します。

パターンの制約のプロパティーおよび値は左から右方向に定義する

DRL パターンの制約では、ファクトプロパティー名は演算子の左側に、値 (定数または変数) は右側に配置されるようにします。プロパティー名は常に、インデックスの値ではなく、キーでなければなりません。たとえば、**Person("John" == firstName)** ではなく **Person(firstName == "John")** のように指定します。制約のプロパティーと値を右から左に定義すると、デシジョンエンジンのパフォーマンスが低下する可能性があります。

DRL パターンおよび制約の詳細については、「DRL のルール条件 (WHEN)」 を参照してください。

パターン制約では他の演算子よりも等価演算子タイプをできるだけ使用する

デシジョンエンジンは、ビジネスルールロジックの定義に使用可能な DRL 演算子タイプを多数サポートしていますが、等価演算子 **==** の評価を最も効率的に実行します。実用的な場合には、他の演算子ではなく、この演算子を使用してください。たとえば、パターン (**Person(firstName == "John")**) は **Person(firstName != "OtherName")** よりも効率的に評価されます。等価演算子だけを使用すると実用的ではない場合があるため、DRL 演算子を使用するときはビジネスロジックの要件とオプションをすべて検討してください。

最も制限の厳しいルールの条件を先にリストする

ルールに複数の条件がある場合には最も制限の厳しい条件から順にリストしてください。こうすることで、最も制限の厳しい条件が満たされていない場合は、デシジョンエンジンがすべての条件セットを評価せずに済むようになります。

たとえば、以下の条件は、航空券とホテルをあわせて予約した旅行者には割引を適用する旅行予約ルールの一部です。このシナリオでは、ホテルを予約するお客様がこの割引を受けるために航空券をあわせて予約することはほぼないため、ホテルの条件はほぼ満たされず、このルールは実行されません。そのため、必要のない航空券の条件を頻繁に評価しないため、1 つ目の条件の順番のほうがより効率的です。これは、1 つ目の条件では、ホテルの条件が満たされていない場合に不要かつ頻繁に、航空券の条件をデシジョンエンジンが評価せずに済むためです。

優先条件の順番: ホテルと航空券

```
when
  $h:hotel() // Rarely booked
  $f:flight()
```

効率の悪い条件の順番: 航空券とホテル

```
when
  $f:flight()
  $h:hotel() // Rarely booked
```

DRL パターンおよび制約の詳細については、「DRL のルール条件 (WHEN)」 を参照してください。

from 句を過剰に使用する、サイズの大きいオブジェクトコレクションで反復を回避する

以下の例のように、サイズの大きいオブジェクトコレクションで反復を行う DRL ルールでは **from** 要素の使用を回避してください。

from 句を使用した条件の例

```
when
  $c: Company()
  $e : Employee ( salary > 100000.00) from $c.employees
```

このような場合には、デシジョンエンジンはルールの条件が評価されるたびにサイズの大きいグラフを反復するので、ルール評価の妨げになります。

他の手段として以下の例のように、サイズの大きいグラフが含まれるオブジェクトを追加してデシジョンエンジンが頻繁に反復作業を行うのではなく、コレクションを KIE セッションに直接追加して、条件内でコレクションを結合します。

from 句なしの条件の例

```
when
  $c: Company();
  Employee (salary > 100000.00, company == $c)
```

この例では、デシジョンエンジンは 1 回だけリストを反復して、ルールの評価を効率化します。

from 要素または他の DRL 条件要素に関する情報は、「DRL でサポートされるルール条件要素 (キーワード)」 を参照してください。

ロギングのデバッグには、System.out.println ステートメントの代わりに、デシジョンエンジンのイベントリスナーをルール内で使用する

ルールのデバッグやコンソール出力に、**System.out.println** ステートメントをルールアクションで使用できますが、多数のルールに対してこれを行うと、ルール評価の妨げになります。別の効率的な方法として、できるだけ内蔵のデシジョンエンジンイベントリスナーを使用してください。このイベントリスナーが貴社の要件に満たない場合には、Logback、Apache Commons Logging、Apache Log4j など、デシジョンエンジンがサポートするシステムロギングユーティリティーを使用してください。
サポート対象のデシジョンエンジンのイベントリスナーおよびロギングユーティリティーに関する詳細は、**Red Hat Process Automation Manager のデシジョンエンジン** を参照してください。

drools-metric モジュールを使用して、ルール内の障害物を特定する

drools-metric モジュールを使用すると、特に多くのルールを処理する場合に、遅いルールを特定できます。**drools-metric** モジュールは、デシジョンエンジンのパフォーマンスの分析にも役立ちます。**drools-metric** モジュールは、実稼働環境で使用するためのものではないことに注意してください。ただし、テスト環境で分析を実行することはできます。
drools-metric を使用してデシジョンエンジンのパフォーマンスを分析するには、次の例に示すように、プロジェクトの依存関係に **drools-metric** を追加し、**org.drools.metric.util.MetricLogUtils** のトレースログを有効にします。

drools-metric のプロジェクト依存関係の例

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-metric</artifactId>
</dependency>
```

logback.xml 設定ファイルの例

```
<configuration>
  <logger name="org.drools.metric.util.MetricLogUtils" level="trace"/>
```

```
  ...
<configuration>
```

また、システムプロパティー **drools.metric.logger.enabled** を **true** に設定して、**MetricLogUtils** を有効にします。必要に応じて、**drools.metric.logger.threshold** システムプロパティーを設定することにより、メトリックログのマイクロ秒しきい値を変更できます。

注記

しきい値を超えるノードの実行のみがログに記録されます。デフォルト値は **500** です。

設定が完了すると、次の例に示すように、ルールの実行によりログが生成されます。

ルール実行出力の例

```
TRACE [JoinNode(6) - [ClassObjectType class=com.sample.Order]], evalCount:1000,
elapsedMicro:5962
TRACE [JoinNode(7) - [ClassObjectType class=com.sample.Order]], evalCount:100000,
elapsedMicro:95553
TRACE [ AccumulateNode(8) ], evalCount:4999500, elapsedMicro:2172836
TRACE [EvalConditionNode(9)]:
cond=com.sample.Rule_Collect_expensive_orders_combination930932360Eval1Invoker@ee2a6922
], evalCount:49500, elapsedMicro:18787
```

この例には、次の主要なパラメーターが含まれています。

- **evalCount** は、ノードの実行中に挿入されたファクトに対する制約評価の数です。
- **elapsedMicro** は、ノード実行の経過時間 (マイクロ秒単位) です。

未処理の **evalCount** または **elapsedMicro** ログが見つかった場合は、ノード名を **ReteDumper.dumpAssociatedRulesRete()** 出力と関連付けて、ノードに関連付けられているルールを識別します。

ReteDumper の使用例

```
ReteDumper.dumpAssociatedRulesRete(kbase);
```

ReteDumper の出力例

```
[ AccumulateNode(8) ] : [Collect expensive orders combination]
...
```

第23章 次のステップ

- テストシナリオを使用したデシジョンサービスのテスト
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート IV. ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成

ビジネス分析者またはビジネスルールの開発者は、ガイド付きデシジョンテーブルを使用して、ウィザード主導のテーブル形式でビジネスルールを定義することができます。このルールは Drools Rule Language (DRL) に組み込まれ、プロジェクトのデシジョンサービスの中心となります。

注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

前提条件

- ガイド付きデシジョンテーブルのスペースおよびプロジェクトが Business Central に作成されている。各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は **デシジョンサービスのスタートガイド** を参照してください。

第24章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表24.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第25章 ガイド付きデシジョンテーブル

ガイド付きデシジョンテーブルは、デシジョンテーブルのスプレッドシートに代わる方法で、ウィザードを用いて表形式でビジネスルールを定義します。ガイド付きデシジョンテーブルでは、プロジェクトで指定したデータオブジェクトをもとに、Business Central の UI ベースのウィザードに従ってルール属性、メタデータ、条件、およびアクションを定義します。ガイド付きデシジョンテーブルを作成すると、定義したルールは、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。

ガイド付きデシジョンテーブルに関連するすべてのデータオブジェクトは、ガイド付きデシジョンテーブルと同じプロジェクトパッケージに存在する必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとガイド付きデシジョンテーブルの作成後、ガイド付きデシジョンテーブルデザイナーの **Data Objects** タブを使用して、必要なデータオブジェクトがすべてリストされていることを検証したり、**新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

第26章 データオブジェクト

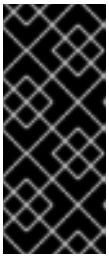
データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

26.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

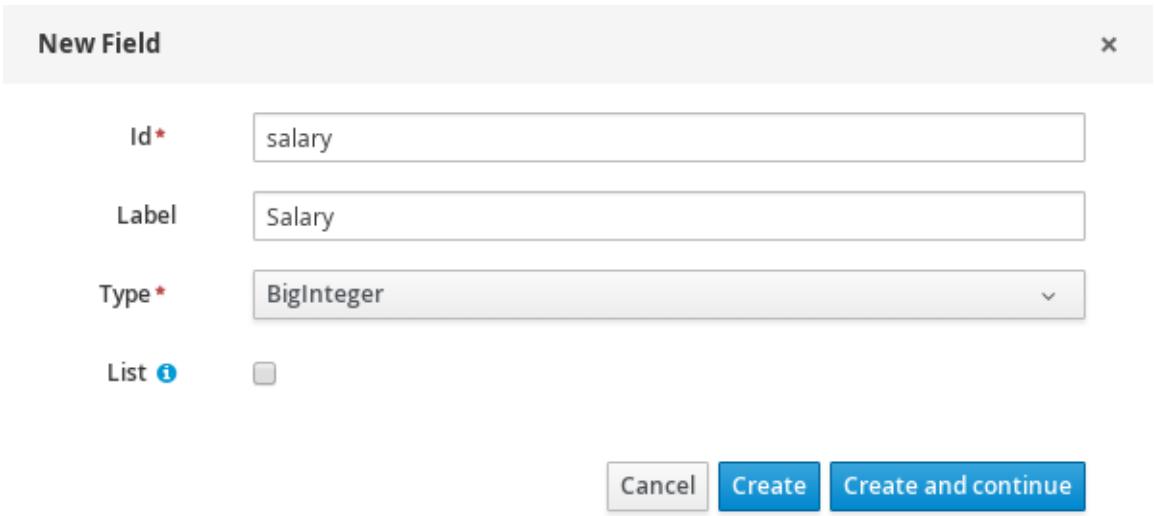


別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図26.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

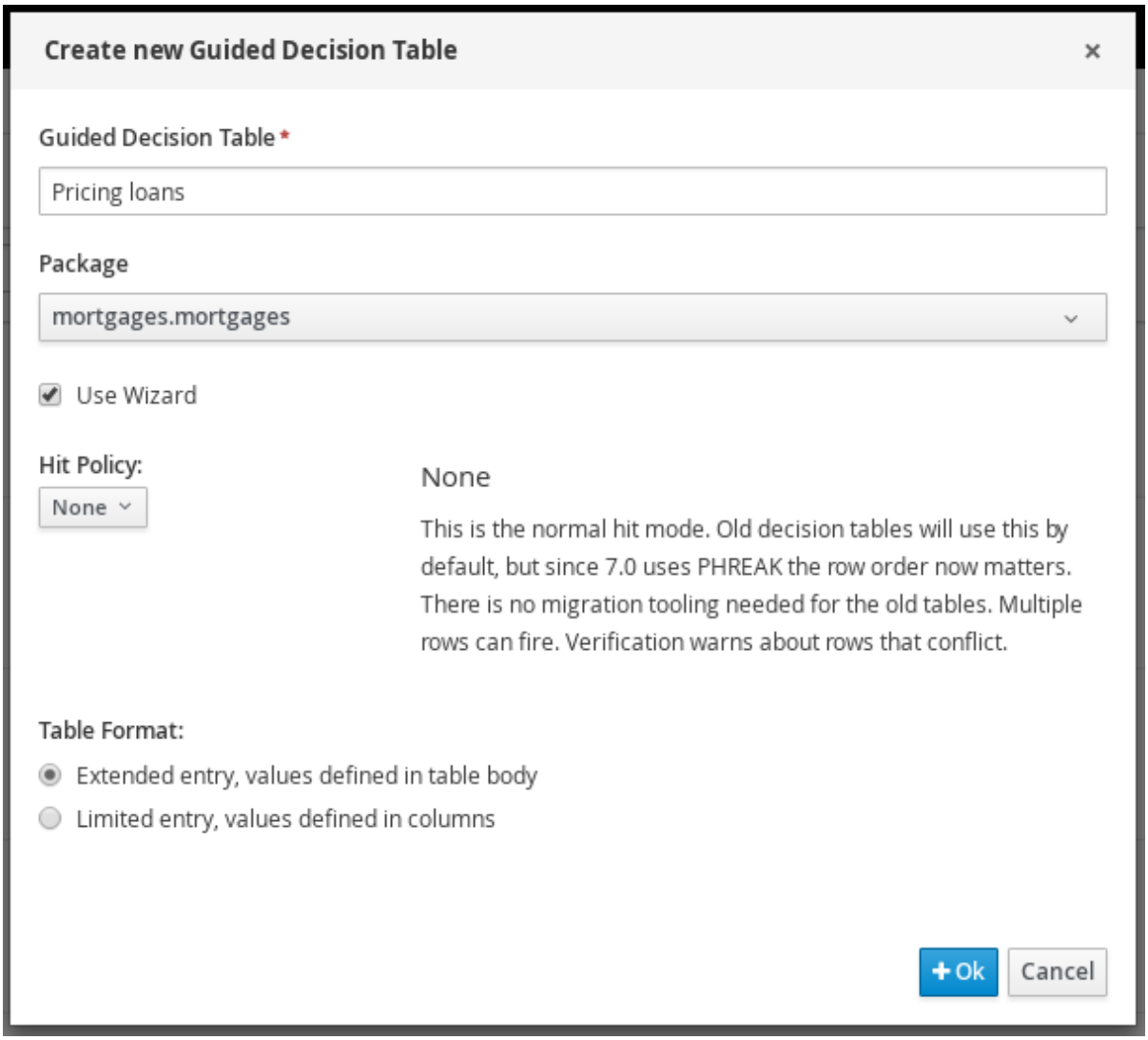
第27章 ガイド付きデシジョンテーブルの作成

ガイド付きデシジョンテーブルを使用して、ルール属性、メタデータ、条件、およびアクションを表形式で定義し、ビジネスルールプロジェクトに追加できます。

手順

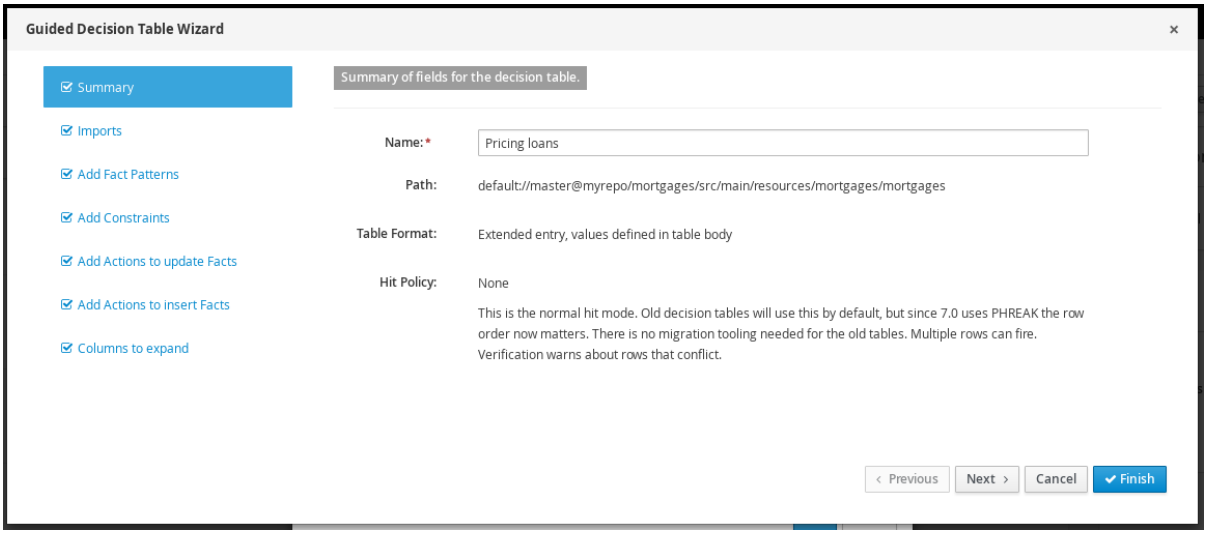
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Decision Table** をクリックします。
3. **ガイド付きデシジョンテーブル** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
4. **Use Wizard** を選択して、ウィザードでテーブルの設定を終わらせるか、このオプションは選択せずにテーブル作成を終わりにし、ガイド付きデシジョンテーブルデザイナーで残りの設定を行います。
5. テーブルのルール行が準拠するヒットポリシーを選択します。詳細は、*28章 ガイド付きデシジョンテーブルのヒットポリシー* を参照してください。
6. テーブルの形式を、**拡張エントリー** と **制限エントリー** から選択します。詳細は、「ガイド付きデシジョンテーブルの種類」 を参照してください。
7. **OK** をクリックして設定を完了します。**Use Wizard** をクリックすると、ガイド付きデシジョンテーブルが表示されます。**Use Wizard** オプションを選択しないとこのプロンプトは表示されず、テーブルデザイナーが表示されます。
   たとえば、以下のウィザード設定は、ローン申請のデシジョンサービスで使用するガイド付きデシジョンテーブルです。

図27.1 ガイド付きデシジョンテーブルの作成



8. ウィザードを使用する場合は、利用可能なインポート、ファクトパターン、制約、およびアクションを追加して、テーブルの列を拡張するかどうかを選択します。**Finish** をクリックしてウィザードを閉じ、テーブルデザイナーを表示します。

図27.2 ガイド付きデシジョンテーブルのウィザード



ガイド付きデシジョンテーブルデザイナーで、列および行を追加または編集し、最終調整を行います。

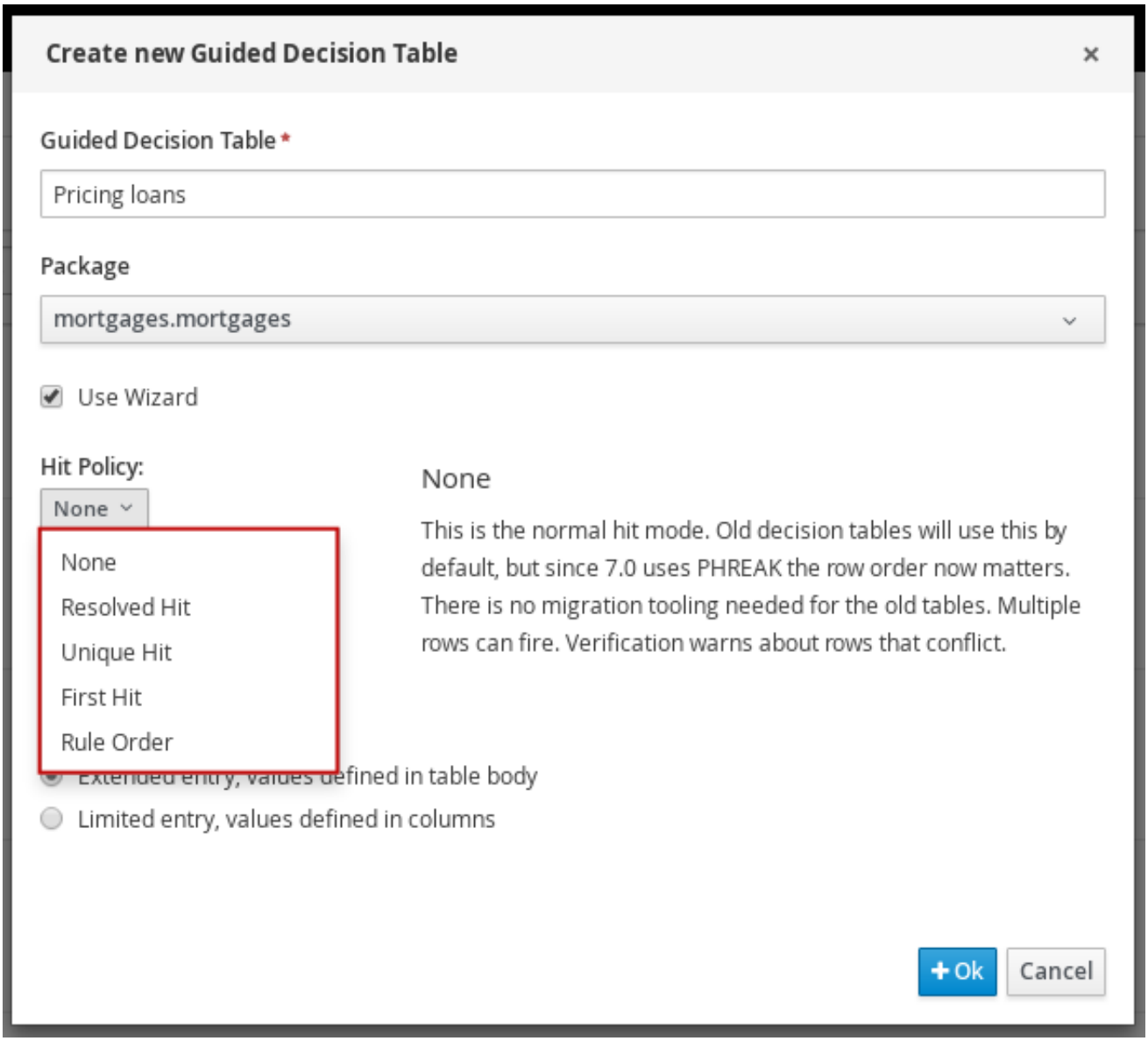
第28章 ガイド付きデシジョンテーブルのヒットポリシー

ヒットポリシーは、ガイド付きデシジョンテーブルのルール (行) を適用する順番 (上から下、優先順位を指定した順など) を指定します。

次のヒットポリシーが利用できます。

- **None:** (デフォルトのヒットポリシー) 複数の行を実行できます。競合している行については検証により警告されます。(ガイド付きデシジョンテーブル以外のスプレッドシート使用して) アップロードされているデシジョンテーブルには、このヒットポリシーが適用されます。
- **Resolved Hit:** 指定されている優先順位に従って同時に実行できる行は 1 つだけです。リストされている順序は無視されます (たとえば、行 10 の優先順位を行 5 よりも高くできます)。したがって、視覚的な分かりやすさを基準に行の順番を決め、それとは別に優先順位を指定することができます。
- **Unique Hit:** 同時に実行できる行は 1 つだけです。一致する条件は重複しないようにする必要があります。複数の行が実行すると、開発時に検証により警告が生成されます。
- **First Hit:** 同時に実行できる行は 1 つだけです。テーブルの順番に従って、上から下に実行します。
- **Rule Order:** 複数の行を実行できます。行の競合は期待されるため、報告されません。

図28.1 利用可能なヒットポリシー



28.1. ヒットポリシーの例: 映画観賞券の割引価格を定めるデシジョンテーブル

以下の表は、顧客の年齢、学生かどうか、軍隊経験の有無、またはこの 3 つの全条件をもとにして映画観賞券の割引価格を求める時に使用するデシジョンテーブル例の一部です。

表28.1 映画観賞券で利用可能な割引価格を定めたデシジョンテーブル例

行番号	割引の種類	割引価格
1	高齢者 (60 歳以上)	10%
2	学生	10%
3	軍隊経験	10%

以下の例では、最終的に適用される割引合計は、テーブルに指定したヒットポリシーによって変わります。

- **None/Rule Order:** ヒットポリシー **None** および **Rule Order** では、適用可能なルールがすべて組み込まれ、それぞれの顧客に適用される割引が積み重ねられます。
例: 現在学生で、軍隊経験がある高齢者には、3 つの割引がすべて適用され、合計 30% の割引になります。

  重要な相違点: **None** では、複数の行が適用されると警告が作成されます。**Rule Order** では、この警告が作成されません。

- **First Hit/Resolved Hit:** ヒットポリシー **First Hit** ポリシーおよび **Resolved Hit** ポリシーでは、割引の中から 1 つだけが適用されます。
First Hit の場合は、リストの中で最初に満たされた割引が適用され、その他の割引は適用されません。

  例: 現在学生で、軍隊経験がある高齢者には、テーブルに最初にリストされている高齢者割引 10% だけが適用されます。

  Resolved Hit の場合は、テーブルの修正が必要になります。テーブルで優先順位の例外を割り当てた割引が、リストされた順番にかかわらず最初に適用されます。この例外を割り当てるには、割引 (行) の優先順位を指定する新しい列を追加します。

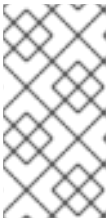
  例: 軍隊経験者向けの割引の優先順位が、高齢者向けまたは学生向けの割引よりも高く設定されている場合、現在学生で、軍隊経験がある高齢者には、軍隊経験者の割引 10% が適用されます (高齢者向けまたは学生向け割引は適用されません)。

  Resolved Hit ポリシーを適用して修正したデシジョンテーブルをご覧ください。

表28.2 Resolved Hit ポリシーを適用して修正したデシジョンテーブル

行番号	割引の種類	優先する行	割引価格
1	高齢者 (60 歳以上)		10%
2	学生		10%
3	軍隊経験	1	10%

この修正したテーブルでは、軍人経験者向けの割引が事実上の行 1 になるため、高齢者向けと学生向けの割引よりも優先され、その後でその他の割引が追加されます。優先順位は、行 1 に対する優先だけを指定する必要があり、行 1 および行 2 の両方に指定する必要はありません。この変更により、行のヒット順は、3 → 1 → 2 となり、さらに行が増えた場合はこの後に続きます。



注記

ここで変更した行の順番は、軍隊経験者向けの割引を行 1 に移動して **First Hit** ポリシーをテーブルに適用した場合と同じになります。ただし、ルールを特定の順番で並べ (アルファベット順)、ルールの適用順を変更したい場合は、**Resolved Hit** ポリシーが便利です。

重要な相違点: **First Hit** を使用すると、ルールの適用順はリストした順番に固定されます。**Resolved Hit** を使用すると、指定された優先順位の例外を除いて、リストされた順にルールが適用されます。

- **Unique Hit:** テーブルの修正が必要になります。**Unique Hit** ポリシーを使用する場合は、一度

に複数の行を適用できないように行を作成する必要があります。ただし、ルールを 1 つまたは複数適用するかどうかは、行ごとに指定できます。この方法では、**Unique Hit** ポリシーを使用した場合に重複の警告が表示されないように、デシジョンテーブルの粒度をより細かくできます。
Unique Hit ポリシーを適用して修正したデシジョンテーブルをご覧ください。

表28.3 Unique Hit ポリシーを適用して修正デシジョンテーブル

行番号	高齢者 (65 歳以上)	学生	軍隊経験	割引価格
1	はい	いいえ	いいえ	10%
2	いいえ	はい	いいえ	10%
3	いいえ	いいえ	はい	10%
4	はい	はい	いいえ	20%
5	はい	いいえ	はい	20%
6	いいえ	はい	はい	20%
7	はい	はい	はい	30%

この修正したテーブルでは、各行が一意で、重複はできず、1 つまたは複数の割引が適用されます。

28.1.1. ガイド付きデシジョンテーブルの種類

Red Hat Process Automation Manager では、拡張エントリーテーブルと制限エントリーテーブルの 2 種類のデシジョンテーブルに対応します。

- **拡張エントリー:** 拡張エントリーのデシジョンテーブルの列定義には、パターン、フィールド、演算子を指定します。値は指定しません。値またはステート (state) は、デシジョンテーブルの本体に保持されます。

Pricing loans

#	Description	application : LoanApplication				ome : IncomeSou	application		
		amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

- **制限エントリー:** 制限エントリーのデシジョンテーブルの列定義には、パターン、フィールド、演算子に加えて、値を指定します。デシジョンテーブルの本体には、デシジョンテーブルのステート (state) をブール値で指定します。正の値 (チェックボックスを選択した場合) はその列が適用されます。負の値 (チェックボックスを選択しない場合) はその列が適用されないことを示しています。

Credit rating						
#	Description	CR = AA	CR = OK	CR = Sub prime	Approve	Decline
1		☑	☐	☐	☑	☐
2		☐	☑	☐	☑	☐
3		☐	☐	☑	☐	☑

第29章 ガイド付きデシジョンテーブルへの列の追加

ガイド付きデシジョンテーブルを作成したら、ガイド付きデシジョンテーブルデザイナーで、さまざまなタイプの列を定義して追加できます。

前提条件

- ファクトやフィールドなど、列パラメーターに使用されるデータオブジェクトが、ガイド付きデシジョンテーブルと同じパッケージに作成されている。もしくは、ガイド付きデシジョンテーブルデザイナーの **Data Objects → New item** で、別のパッケージからインポートされている。

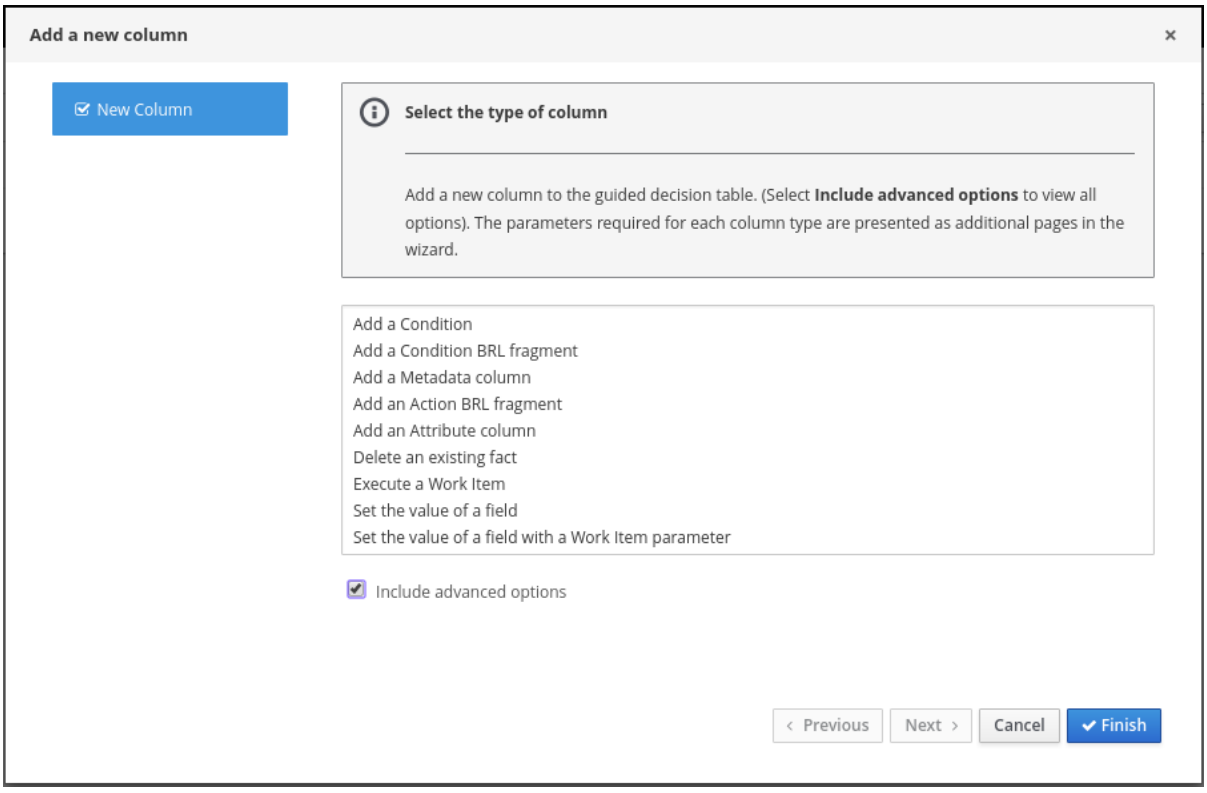
この列パラメーターの説明は 30章 *ガイド付きデシジョンテーブルの列の種類* の必須の列パラメーターを参照してください。

データオブジェクトの作成の詳細は 「データオブジェクトの作成」 を参照してください。

手順

1. ガイド付きデシジョンテーブルデザイナーで、**Columns → Insert Column** をクリックします。
2. **Include advanced options** をクリックして、列の全オプションを表示します。

   図29.1 列の追加



3. 追加する列の種類を選択して **Next** をクリックし、ウィザードの手順に従って、列を追加するのに必要なデータを指定します。
   列の各種類と、設定に必要なパラメーターは 30章 *ガイド付きデシジョンテーブルの列の種類* を参照してください。
4. **Finish** をクリックして、設定した列を追加します。

列を追加したら、関連するルール行を列に追加し、デシジョンテーブルを完了します。詳細は、34章 *ガイド付きデシジョンテーブルで行の追加およびルールの定義* を参照してください。

以下は、ローン申請のデシジョンサービスのデシジョンテーブル例です。

図29.2 完成したガイド付きデシジョンテーブルの例

Pricing loans

#	Description	application : LoanApplication				ome : IncomeSou	application		
		amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

第30章 ガイド付きデシジョンテーブルの列の種類

ガイド付きデシジョンテーブルの **Add a new column** ウィザードは、次の列オプションを提供します。(**Include advanced options** を選択して、すべてのオプションを表示します)。

- Add a Condition (条件の追加)
- Add a Condition BRL fragment (条件 BRL フラグメントの追加)
- Add a Metadata column (メタデータ列の追加)
- Add an Action BRL fragment (アクション BRL フラグメントの追加)
- Add an Attribute column (属性列の追加)
- Delete an existing fact (既存ファクトの削除)
- Execute a Work Item (作業アイテムの実行)
- Set the value of a field (フィールド値の設定)
- Set the value of a field with a Work Item result (作業アイテムの結果でフィールド値の設定)

Add a new column ウィザードで必要な列タイプとパラメーターは、以下のセクションで説明します。

重要: 列パラメーターに必要なデータオブジェクト

ファクトパターン、フィールドなど、このセクションで説明するいくつかの列パラメーターは、ガイド付きデシジョンテーブルと同じパッケージにすでに定義されているデータオブジェクトだけで設定されるドロップダウンオプションを提供します。パッケージで利用可能なデータオブジェクトは、Project Explorer の **Data Objects** パネル、およびガイド付きデシジョンテーブルデザイナーの **Data Objects** タブに一覧表示されます。必要に応じてパッケージにデータオブジェクトを追加したり、ガイド付きデシジョンテーブルデザイナーの **Data Objects** → **New item** で、別のパッケージからインポートしたりできます。データオブジェクトの作成の詳細は 「データオブジェクトの作成」 を参照してください。

30.1. "ADD A CONDITION (条件の追加)"

条件はファクトパターンを表し、ルールの左側 (WHEN) に定義されます。この列オプションで、特定のフィールド値を使用して、データオブジェクトが存在するかどうかを確認し、ルールのアクション (THEN) 部分に影響を及ぼす条件列を 1 つ以上定義します。条件テーブルでファクトにバインディングを定義したり、以前定義したものを選択できます。パターンを無効にすることもできます。

ルール条件の例

```
when
  $i : IncomeSource( type == "Asset" ) // Binds the IncomeSource object to the $i variable
then
  ...
end
```

```
when
  not IncomeSource( type == "Asset" ) // Negates matching pattern
then
```

```
  ...
end
```

バインディングを指定したら、フィールド制約を定義できます。同じファクトパターンのバインディングを使用して列を 2 つ以上定義すると、フィールド制約は、同じパターンに定義された複合フィールド制約になります。1 つのモデルクラスに複数のバインディングを定義すると、それぞれのバインディングは、そのルールの条件 (WHEN) で異なるモデルクラスになります。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルの条件に使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト (詳細は 必要なデータオブジェクト の注記を参照) と、指定するモデルクラスバインディングの組み合わせとなります(例: **LoanApplication [application]** または **IncomeSource [income]** で、括弧の部分は特定のファクトタイプに対するバインディング)。
- **Entry point:** 可能な場合は、ファクトパターンのエントリーポイントを定義します。エントリーポイントは、指定するとファクトがデシジョンエンジンに組み込まれるゲートまたはストリームです。(例: **Application Stream**、**Credit Check Stream**)。
- **Calculation type:** 以下の計算タイプの中から 1 つ選択します。
  - **Literal value:** 演算子を使用して、セルの値とフィールドを比較します。
  - **Formula:** セルの表現を評価して、フィールドと比較します。
  - **Predicate:** フィールドは必要ありません。表現を **true** または **false** で評価します。
- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **データオブジェクト** パネルのファクトファイルに定義されています(例: **LoanApplication** ファクトタイプ内の **amount** フィールドまたは **lengthYears** フィールド)。
- **Binding (optional):** 必要に応じて、以前選択したフィールドのバインディングを定義します。(例: **LoanApplication [application]** パターン、**amount** フィールド、および **equal to** 演算子に対してバインディングを **$amount** に設定すると、終了条件は **application : LoanAppplication($amount : amount == [value])** になります)。
- **Operator:** 事前に指定したファクトパターンおよびフィールドに適用する演算子を選択します。
- **Value list (任意):** コンマおよび空白文字で区切った値オプションのリストを入力して、ルールの条件 (WHEN) 部分のテーブル入力データを制限します。この値リストを定義すると、値はその列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを 1 つだけ選択できます。(例: これらの 3 つのオプションのみを指定する **Monday, Wednesday, Friday**)。
- **Default value (任意):** 事前定義した値オプションのいずれかを、新しい列のセルに自動的に表示するデフォルト値として選択します。デフォルト値が指定されていないと、テーブルのセルはデフォルトでは空欄となります。また、Project Explorer の **Enumeration Definitions** パネルにリストした、プロジェクトに事前に設定したデータの列挙からデフォルト値を選択できます (列挙は、**Menu → Design → Projects → [select project] → Add Asset → Enumeration** から作成できます)。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

30.1.1. 条件列セルに any other 値を追加

ガイド付きデシジョンテーブルにおける単純な条件列では、以下のパラメーターを設定した場合に、列のテーブルセルに **any other** 値を適用できます。

- 条件列の **Calculation type** を **Literal value** に設定している。
- **Operator** を、等価演算子 **==** または非等価演算子 **!=** に設定している。

any other 値は、テーブルにすでに定義されているルールに明示的に定義している値以外を設定できるルールを有効にします。DRL ソースでは、**any other** は **not in** と表記されます。

any other に使用される not in が設定されたルール条件の例

```
when
  IncomeSource( type not in ("Asset", "Job") )
  ...
then
  ...
end
```

手順

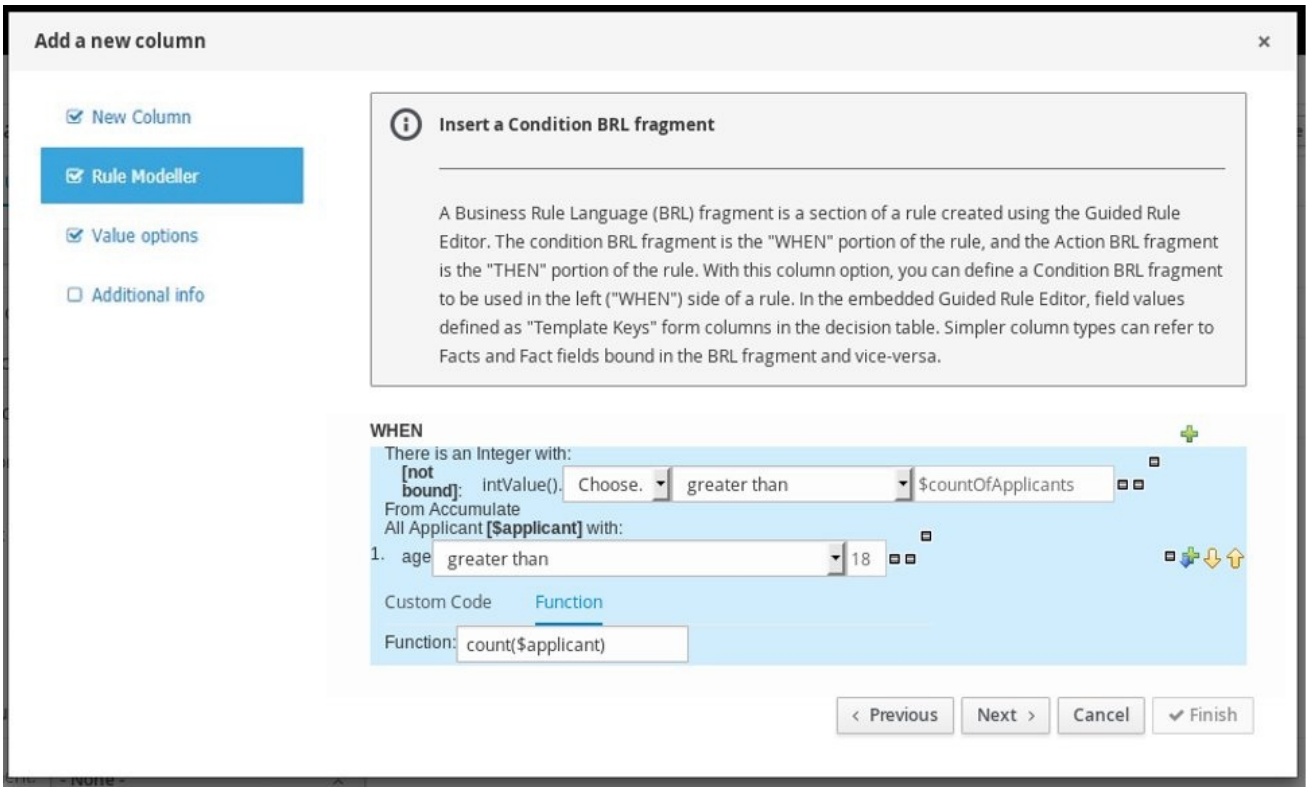
1. **==** 演算子または **!=** 演算子を使用する条件列のセルを選択します。
2. テーブルデザイナーの右上のツールバーで、**Edit → Insert "any other" value** をクリックします。

30.2. "ADD A CONDITION BRL FRAGMENT (条件 BRL フラグメントの追加)"

BRL (Business Rule Language) フラグメントは、ガイド付きルールデザイナーを使用して作成したルールのセクションです。条件 BRL フラグメントはルールの WHEN 部分で、action BRL fragment (アクション BRL フラグメント) はルールの THEN の部分です。この列オプションを使用して、ルールの左側 (WHEN) 部分で使用する条件 BRL フラグメントを定義できます。BRL フラグメントでバインドされているファクトおよびファクトのフィールドは簡単な列タイプで参照でき、これらのファクトおよびファクトのフィールドから列タイプの参照も可能です。

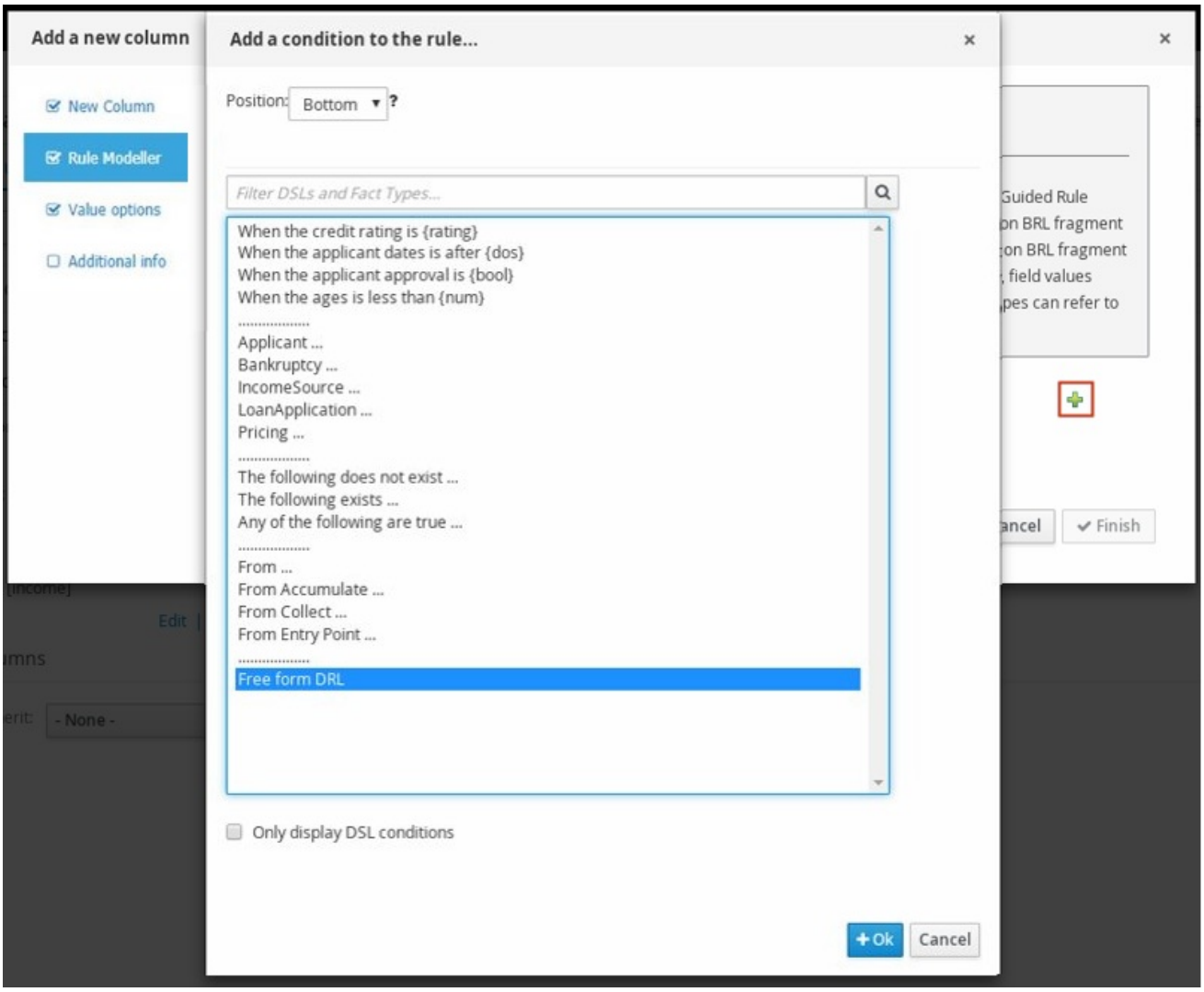
以下の例は、ローン申請の条件 BRL フラグメントです。

図30.1 組み込みガイド付きルールデザイナーを使用する条件 BRL フラグメントの追加



条件オプションのリストから **Free form DRL** を選択して、組み込みガイド付きルールデザイナーを使用せずに条件 BRL フラグメントを定義します。

図30.2 Free form DRL を使用する条件 BRL フラグメントの追加



テンプレートキー

条件 BRL フラグメントにフィールドを追加すると、値オプションの 1 つが (**Literal** または **Formula** ではなく) **テンプレートキー** になります。テンプレートキーはプレースホルダー変数で、ガイド付きデシジョンテーブルを作成し、別の列に各テンプレートキー値を指定すると、指定した値に入れ替えられます。**Value options** ページでテンプレートキーのデフォルト値を指定できます。デシジョンテーブルの Literal 値および Formula 値は静的ですが、テンプレートキーの値は必要に応じて修正できます。

組み込みのガイド付きルールデザイナーでは、**Template key** フィールドオプションを選択し、エディターに **$key** 書式で値を入力し、テンプレートのキー値をフィールドに追加できます。たとえば、**$age** は、デシジョンテーブルに **$age** 列を作成します。

Free form DRL では、**@{key}** の書式でテンプレートのキー値をファクトに追加できます。たとえば、**Person(age > @{age})** にすると、デシジョンテーブルに **$age** 列が作成されます。

テンプレートキーを使用して追加した新しい列のデータ型は String です。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Rule Modeller:** ルールの条件 BRL フラグメント ("WHEN" 部分) を定義します。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

30.3. "ADD A METADATA COLUMN (メタデータ列の追加)"

この列オプションを使用して、デシジョンテーブルでメタデータ要素を列として定義できます。各列は、DRL ルールの通常のメタデータアノテーションを表します。デフォルトでは、メタデータ列は非表示になっています。列を表示するには、ガイド付きデシジョンテーブルデザイナーで **Edit Columns** をクリックし、**Hide column** チェックボックスの選択を解除します。

必須の列パラメーター

この列タイプを設定する **Add a new column** ウィザードには、以下のパラメーターが必要です。

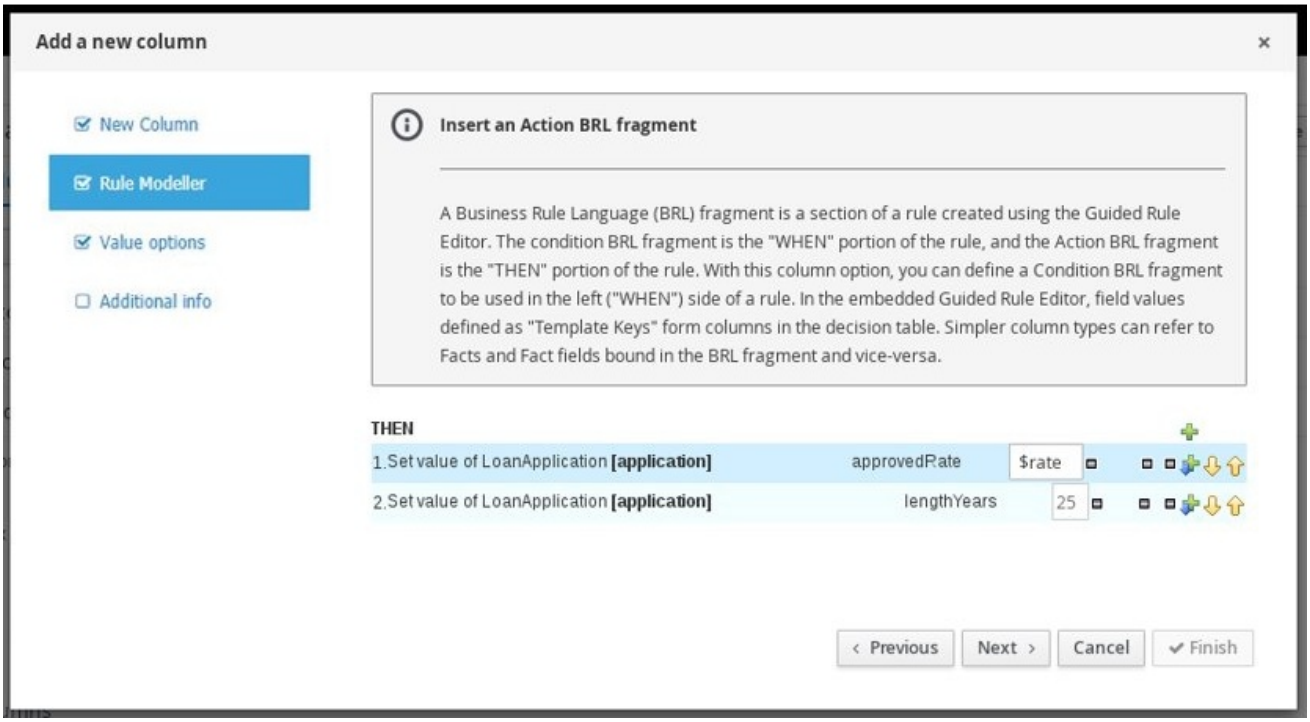
- **Metadata:** Java 変数書式のメタデータ項目の名前を入力します (つまり、空白文字または特殊文字を使用することはできません)。

30.4. "ADD AN ACTION BRL FRAGMENT (アクション BRL フラグメントの追加)"

BRL (Business Rule Language) フラグメントは、ガイド付きルールデザイナーを使用して作成したルールのセクションです。条件 BRL フラグメント はルールの WHEN 部分で、action BRL fragment (アクション BRL フラグメント) はルールの THEN の部分です。この列オプションを使用すると、ルールの右側 (THEN) で使用するアクション BRL フラグメントを定義できます。BRL フラグメントでバインドされているファクトおよびファクトのフィールドは簡単な列タイプで参照でき、これらのファクトおよびファクトのフィールドから列タイプの参照も可能です。

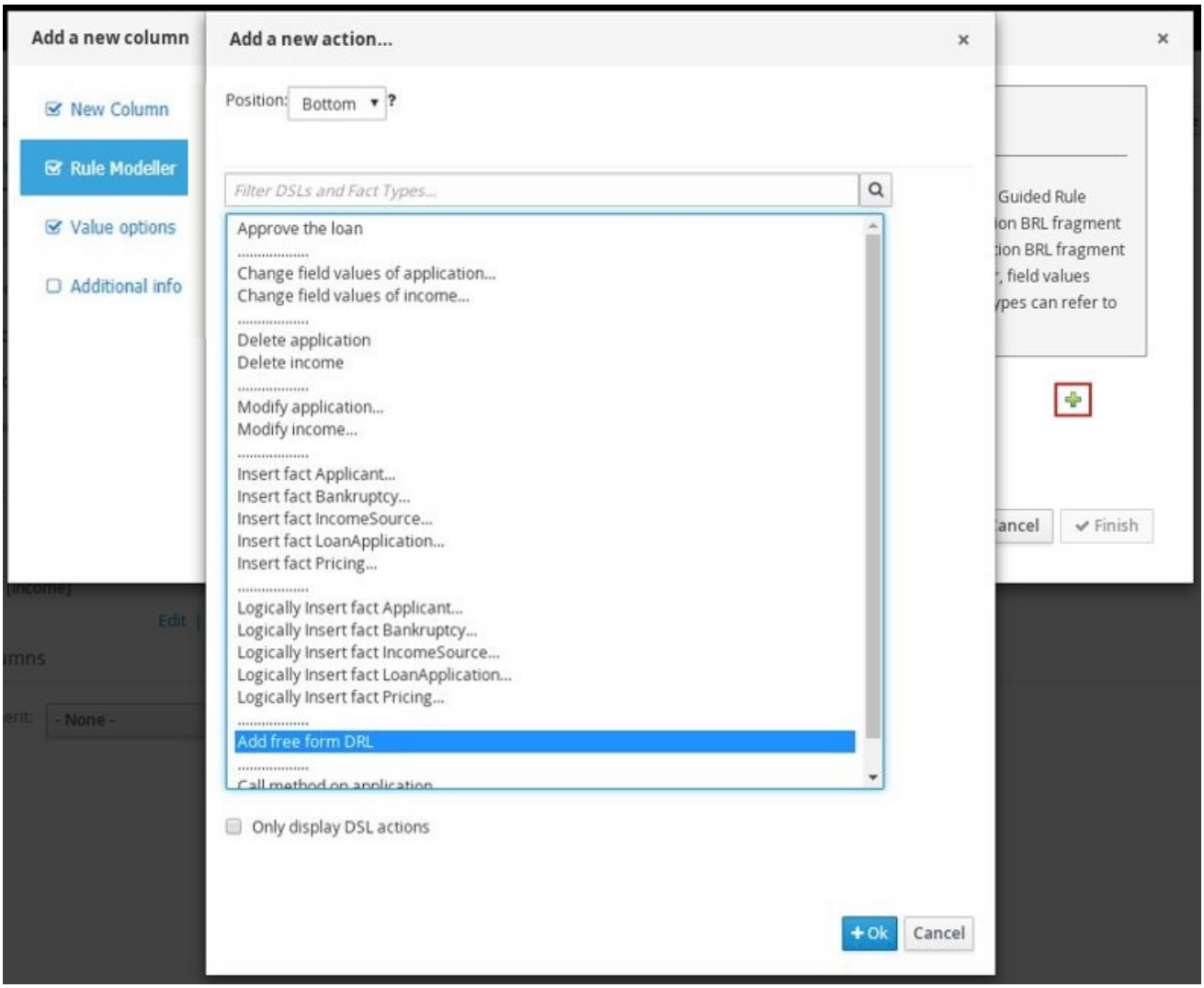
以下の例は、ローン申請のアクション BRL フラグメントです。

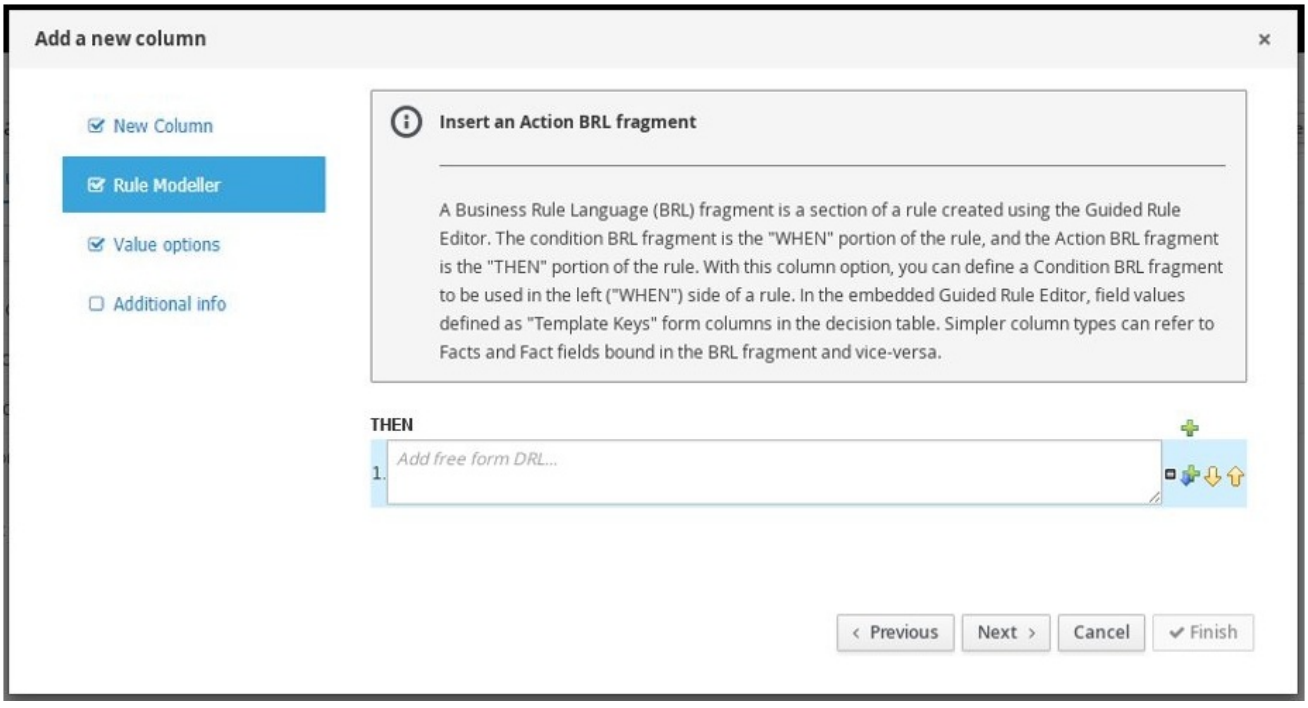
図30.3 組み込みガイド付きルールデザイナーを使用するアクション BRL フラグメントの追加

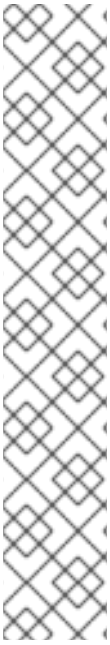


アクションオプションのリストから **Add free form DRL** を選択して、組み込みガイド付きルールデザイナーを使用しないアクション BRL フラグメントを定義します。

図30.4 Free form DRL を使用するアクション BRL フラグメントの追加







テンプレートキー

アクション BRL フラグメントにフィールドを追加すると、値オプションの 1 つが (**Literal** または **Formula** ではなく) **テンプレートキー** となります。テンプレートキーはプレースホルダー変数で、ガイド付きデシジョンテーブルを作成し、別の列に各テンプレートキー値を指定すると、指定した値に入れ替えられます。**Value options** ページでテンプレートキーのデフォルト値を指定できます。デシジョンテーブルの Literal 値および Formula 値は静的ですが、テンプレートキーの値は必要に応じて修正できます。

組み込みのガイド付きルールデザイナーでは、**Template key** フィールドオプションを選択し、エディターに **$key** 書式で値を入力し、テンプレートのキー値をフィールドに追加できます。たとえば、**$age** は、デシジョンテーブルに **$age** 列を作成します。

Free form DRL では、**@{key}** の書式でテンプレートのキー値をファクトに追加できます。たとえば、**Person(age > @{age})** にすると、デシジョンテーブルに **$age** 列が作成されます。

テンプレートキーを使用して追加した新しい列のデータ型は String です。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Rule Modeller:** ルールのアクション BRL フラグメントの定義 (THEN 部分)
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

30.5. "ADD AN ATTRIBUTE COLUMN (属性列の追加)"

この列オプションを使用して、Saliance、Enabled、Date-Effective などの DRL ルール属性を表現する属性列を 1 つ以上追加できます。

たとえば、以下のガイド付きデシジョンテーブルでは **salience** 属性を使用してルールの優先度を、**enabled** 属性を使用して評価のルールを有効化または無効化します。**salience** の値が大きいルールが最初に評価され、**enabled** 属性が指定されたルールは、チェックボックスが選択されている場合にのみ、評価されます。

図30.5 評価の動きを定義する salience 属性および enabled 属性が含まれるルールの例

Pricing loans							
#	Description	salience	enabled	application : LoanApplication			
				amount min	amount max	period	deposit max
1		100	☑	131000	200000	30	20000
2			☑	10000	100000	20	2000
3			☐	100001	130000	20	3000

ルール属性を含むルールソースの例

```
rule "Row 1 Pricing loans"
  salience 100
  enabled true
  when
    ...
```

```
  then
    ...
end
...
rule "Row 3 Pricing loans"
  enabled false
  when
    ...
  then
    ...
end
```

各属性の説明について、ウィザードのリストから属性を選択します。

ヒットポリシーおよび属性

デシジョンテーブルに定義したヒットポリシーに応じて、ヒットポリシーが内部的に使用されているため、一部の属性を無効にできます。たとえば、**Resolved Hit** ポリシーをこのテーブルに割り当てて、テーブルに指定した優先順位に従って行 (ルール) を適用すると、Salience 属性が使用されなくなります。Salience 属性は、定義した salience (優先順位) 値に従って、ルールの優先順位をエスカレーションし、値は、テーブルの **Resolved Hit** ポリシーによって上書きされるためです。

必須の列パラメーター

この列タイプを設定する **Add a new column** ウィザードには、以下のパラメーターが必要です。

- **属性:** 列に適用する属性を選択します。

30.6. "DELETE AN EXISTING FACT (既存ファクトの削除)"

この列のオプションを使用して、ファクトパターンとしてテーブルに以前追加したファクトを削除するアクションを実装できます。この列を作成すると、ファクトタイプは、その列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを 1 つだけ選択できます。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

30.7. "EXECUTE A WORK ITEM (作業アイテムの実行)"

この列オプションを使用すると、Business Central で事前定義された作業アイテム定義に基づいて、作業アイテムハンドラーを実行できます。(作業アイテムは、**Menu → Design → Projects → [select project] → Add Asset → Work Item definition** から作成できます)。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Work Item:** 事前設定した作業アイテムのリストから選択します。
- **Header (説明):** 列にヘッダーテキストを追加します。

- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

30.8. "SET THE VALUE OF A FIELD (フィールド値の設定)"

この列オプションを使用して、ルールの THEN 部分に事前にバインドしたファクトにフィールドの値を設定するアクションを実装できます。その他のルールを再度アクティブにするように修正した値をデシジョンエンジンに通知するオプションがあります。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルの条件または条件 BRL フラグメントに使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト (詳細は 必要なデータオブジェクト の注記を参照) と、指定するモデルクラスバインディングの組み合わせとなります(例: **LoanApplication [application]** または **IncomeSource [income]** で、括弧の部分は特定のファクトタイプに対するバインディング)。

- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **データオブジェクト** パネルのファクトファイルに定義されています(例: **LoanApplication** ファクトタイプ内の **amount** フィールドまたは **lengthYears** フィールド)。

- **Value list (任意):** 値オプションをコンマおよび空白文字で区切ったリストを入力して、ルールのアクション (THEN) 部分に対するテーブルの入力データを制限します。この値リストを定義すると、値はその列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを 1 つだけ選択できます。(リストの例: **Accepted, Declined, Pending**)。

- **Default value (任意):** 事前定義した値オプションのいずれかを、新しい列のセルに自動的に表示するデフォルト値として選択します。デフォルト値が指定されていないと、テーブルのセルはデフォルトでは空欄となります。また、Project Explorer の **Enumeration Definitions** パネルにリストした、プロジェクトに事前に設定したデータの列挙からデフォルト値を選択できます (列挙は、**Menu → Design → Projects → [select project] → Add Asset → Enumeration** から作成できます)。

- **Header (説明):** 列にヘッダーテキストを追加します。

- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

- **Logically insert (論理的な挿入):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列に現在使用されていない場合に表示されます (次のフィールドの説明を参照)。ファクトパターンをデシジョンエンジンに論理的に挿入する場合はこれを選択し、定期的に挿入する場合は選択を解除します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくなると、ファクトは自動的に取り消されます。

- **Update engine with changes (変更でエンジンを更新):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの列に使用されている場合に表示されます。修正したフィールド値を使用してデシジョンエンジンを更新する場合はこれを選択し、デシジョンエンジンを更新しない場合は選択を解除します。

30.9. "SET THE VALUE OF A FIELD WITH A WORK ITEM RESULT (作業アイテム結果でフィールド値の設定)"

この列オプションを使用して、ルールの THEN 部分の作業アイテムハンドラーの結果値に、事前定義し

たファクトフィールドの値を設定するアクションを実装できます。作業アイテムは、フィールドをリターンパラメーターに設定するために、同じデータタイプの結果パラメーターをバインドされたファクト上のフィールドとして定義しなければなりません。(作業アイテムは、**Menu → Design → Projects → [select project] → Add Asset → Work Item definition** から作成できます)。

この列オプションを作成するためには、テーブルに **Execute a Work Item (作業アイテムの実行)** 列が作られている必要があります。

必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルに使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト (詳細は 必要なデータオブジェクト の注記を参照) と、指定するモデルクラスバインディングの組み合わせとなります(例: **LoanApplication [application]** または **IncomeSource [income]** で、括弧の部分は特定のファクトタイプに対するバインディング)。

- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **データオブジェクト** パネルのファクトファイルに定義されています(例: **LoanApplication** ファクトタイプ内の **amount** フィールドまたは **lengthYears** フィールド)。

- **Work Item:** 事前設定した作業アイテムのリストから選択します。(作業アイテムは、return パラメーターにそのフィールドを設定するために、バインドしたファクトにフィールドと同じデータ型の result パラメーターを設定する必要があります)。

- **Header (説明):** 列にヘッダーテキストを追加します。

- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

- **Logically insert (論理的な挿入):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列に現在使用されていない場合に表示されます (次のフィールドの説明を参照)。ファクトパターンをデシジョンエンジンに論理的に挿入する場合はこれを選択し、定期的に挿入する場合は選択を解除します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくなると、ファクトは自動的に取り消されます。

- **Update engine with changes (変更でエンジンを更新):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの列に使用されている場合に表示されます。修正したフィールド値を使用してデシジョンエンジンを更新する場合はこれを選択し、デシジョンエンジンを更新しない場合は選択を解除します。

第31章 ガイド付きデシジョンテーブルのルール名列の表示

必要に応じて、ガイド付きデシジョンテーブルで **Rule Name** 列を表示できます。

手順

1. ガイド付きデシジョンテーブルで、**Columns** をクリックします。
2. **Show rule name column**のチェックボックスを選択します。
3. **Finish** をクリックして保存します。

デフォルトのルール名形式は **Row (row_number)(table_name)** です。**Source** には、ルール名を指定しない場合にはデフォルト値が含まれます。ガイド付きデシジョンテーブルでは、**Rule Name** の列にルール名を追加して、デフォルト値を上書きできます。

第32章 ガイド付きデシジョンテーブルの列の値の並び替え

ガイド付きデシジョンテーブルで作成した列の値を並べ替えることができます。

前提条件

- ガイド付きデシジョンテーブルに必要な列が作成されている。

手順

1. 昇順でソートする列のヘッダーをダブルクリックします。
2. 同じ列の値を降順で並び替えるには、コラムヘッダーを再度ダブルクリックします。

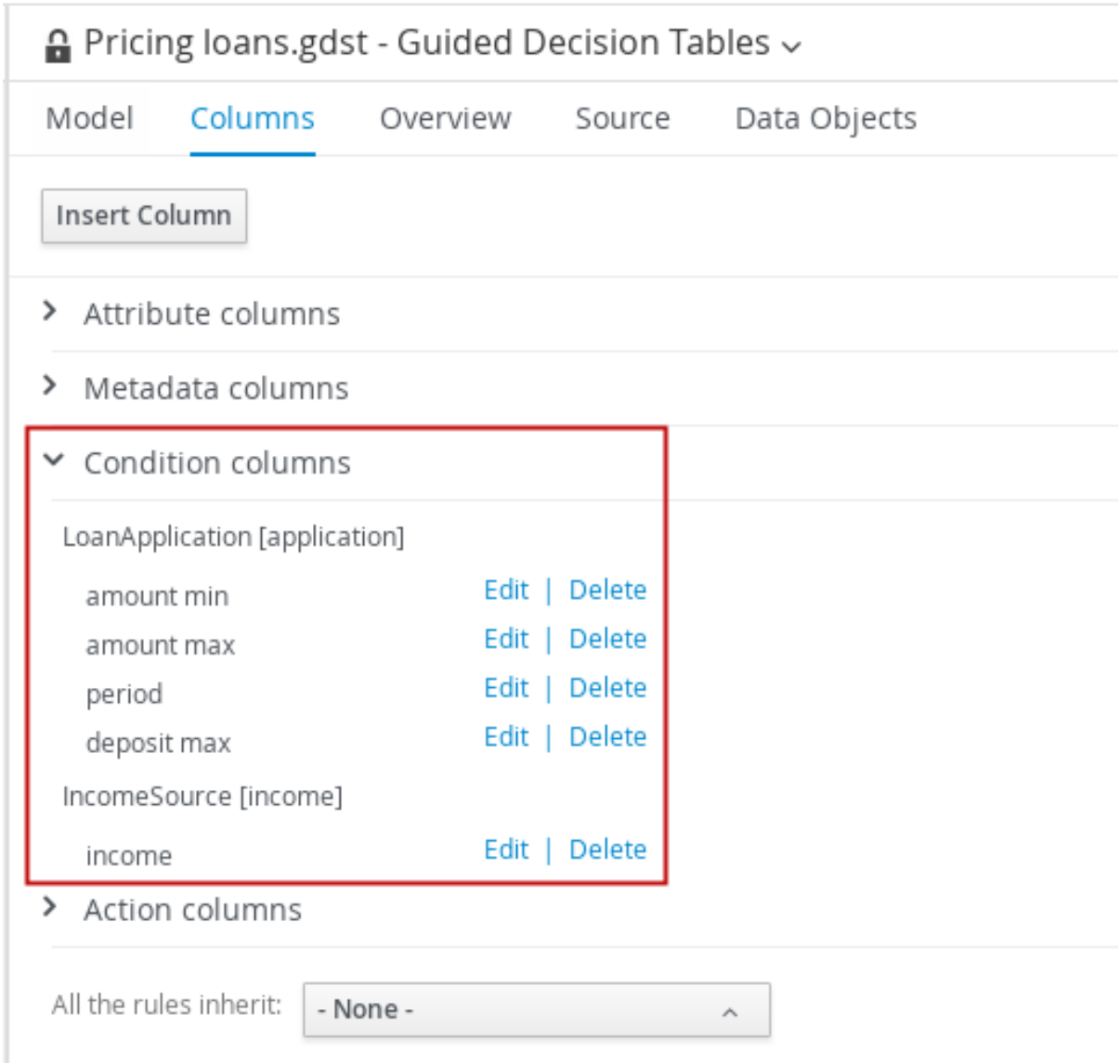
第33章 ガイド付きデシジョンテーブルで列の編集または削除

ガイド付きデシジョンテーブルデザイナーに作成した列は、いつでも編集または削除できます。

手順

1. ガイド付きデシジョンテーブルで、**Columns** をクリックします。
2. 適切なセクションを展開し、列名の隣にある **Edit** または **Delete** をクリックします。

図33.1 列の編集または削除



注記

既存のアクション列が、条件列と同じパターン一致パラメーターを使用している場合は、条件列を削除できません。

3. 列を変更したら、ウィザードの **Finish** をクリックして保存します。

第34章 ガイド付きデシジョンテーブルで行の追加およびルールの定義

ガイド付きデシジョンテーブルで列を作成したら、ガイド付きデシジョンテーブルデザイナーに行を追加してルールを定義します。

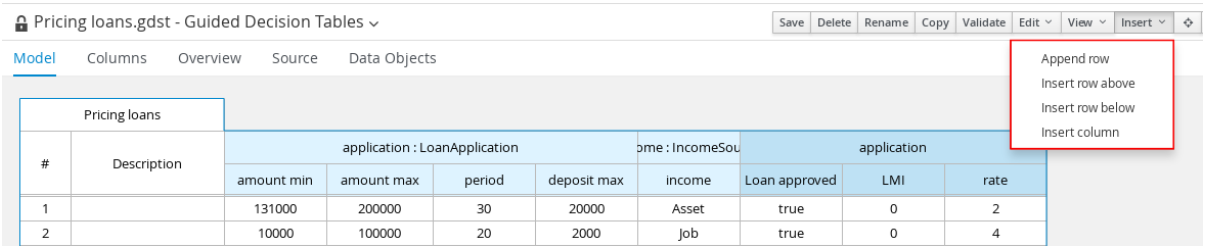
前提条件

- *29章 ガイド付きデシジョンテーブルへの列の追加* の手順に従って、ガイド付きデシジョンテーブルの列が追加されている。

手順

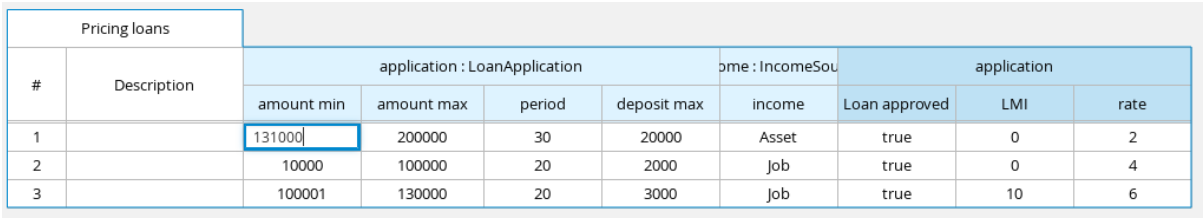
1. ガイド付きデシジョンテーブルデザイナーで、**Insert → Append row**、またはいずれかの **Insert row** オプションをクリックします。(**Insert column** をクリックして列ウィザードを開き、新しい列を定義することもできます)。

図34.1 行の追加



2. 各セルをダブルクリックしてデータを入力します。入力する値が決まっている場合は、セルのドロップダウンオプションから選択します。

図34.2 各セルに入力データの入力



3. ガイド付きデシジョンテーブルですべてのデータ行を定義したら、ガイド付きデシジョンテーブルデザイナーの右上のツールバーで **Validate** をクリックして、テーブルの妥当性を確認します。テーブルの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、テーブルの全コンポーネントを見直し、エラーが表示されなくなるまでテーブルの妥当性確認を行います。

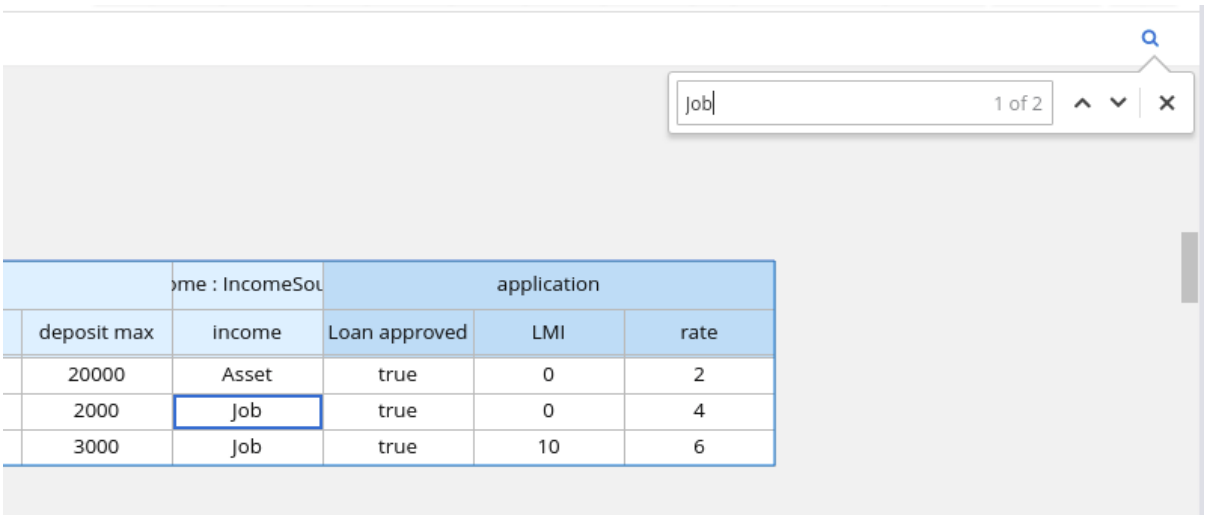
注記

ガイド付きデシジョンテーブルには、リアルタイム検証および妥当性確認がありますが、最善の結果を確実に得るために、完了したデシジョンテーブルの妥当性確認を手動で行うことができます。

4. テーブルデザイナーで **Save** をクリックして、変更を保存します。
ガイド付きのデシジョンテーブルのコンテンツを定義した後に、ガイド付きデシジョンテーブルに表示されているテキストを検索するには、ガイド付きデシジョンテーブルデザイナーの右上隅の検索バーを使用します。検索機能は、特に多数の値が指定された、複雑なガイド付きデ

シジョンテーブルで有用です。

図34.3 ガイド付きデシジョンテーブルのコンテンツ検索



第35章 ルールアセットのドロップダウンリストの列挙定義

Business Central での列挙定義では、ガイド付きルール、ガイド付きルールテンプレート、ガイド付きデシジョンテーブルの条件やアクションのフィールドで使用可能な値を指定します。列挙定義には、ルールアセットで該当するフィールドのドロップダウンリストとして表示される対応値一覧に対する **fact.field** マッピングが含まれています。列挙定義と同じファクトとフィールドをベースにしたフィールドを選択すると、定義した値のドロップダウンリストが表示されます。

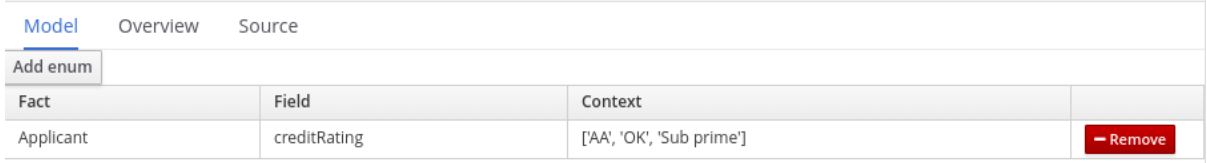
列挙は、Business Central または Red Hat Process Automation Manager プロジェクトの DRL ソースで定義できます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Enumerataion** をクリックします。
3. 分かりやすい **Enumeration** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトと適切なルールアセットが割り当てられているか、これから割り当てるパッケージと同じでなければなりません。
4. **Ok** をクリックして列挙を作成します。
Project Explorer の **Enumeration Definitions** パネルに、新しい列挙が追加されました。
5. 列挙デザイナーの **Model** タブで、**Add enum** をクリックし、以下の列挙値を定義します。
   - **Fact**: この列挙を関連付けるプロジェクトの同じパッケージ内に、既存のデータオブジェクトを指定します。**Project Explorer** で **Data Objects** パネルを開き、利用可能なデータオブジェクトを表示するか、必要に応じて新規アセットとして適切なデータオブジェクトを作成します。
   - **Field**: **Fact** 用に選択したデータオブジェクトの一部として定義した既存のフィールド ID を指定します。**Project Explorer** で **Data Objects** パネルを開き、適切なデータオブジェクトを選択して、利用可能な **Identifier** オプションの一覧を表示します。必要に応じて、データオブジェクトに関連する ID を作成してください。
   - **Context**: **Fact** と **Field** の定義にマッピングする **['string1','string2','string3']** 形式または **[integer1,integer2,integer3]** 形式の値一覧を定義します。これらの値は、ルールアセットの適切なフィールドに、ドロップダウンリストとして表示されます。

   たとえば、以下の列挙は、ローン申請デシジョンサービスの申請者でクレジットスコアに使用するドロップダウンの値を定義します。

図35.1 Business Central での申請者のクレジットスコアの列挙例



DRL ソースの申請者のクレジットスコアの列挙例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

以下の例では、プロジェクトと同じパッケージ内にあり、**Applicant** データオブジェクトと **creditRating** フィールドを使用するガイド付きルール、ガイド付きルールテンプレート、またはガイド付きデシジョンテーブルであれば、設定値がドロップダウンオプションとして利用できます。

図35.2 ガイド付きルールまたはガイド付きルールテンプレートでの列挙ドロップダウンオプション例

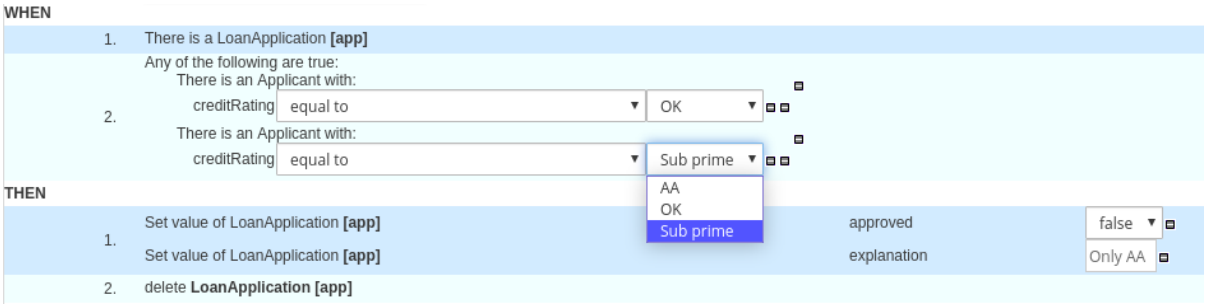
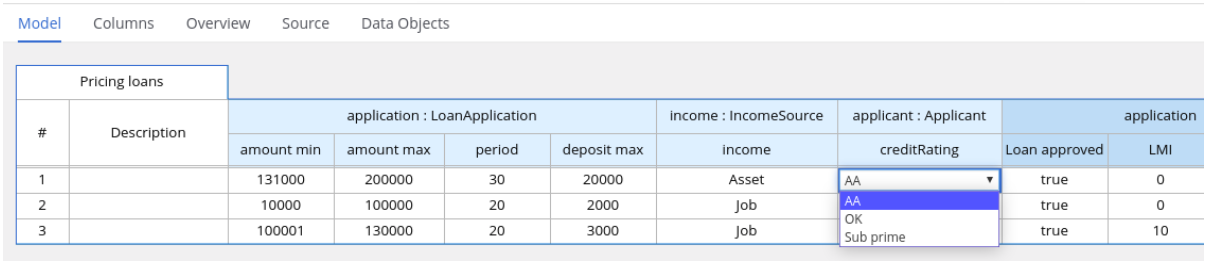


図35.3 ガイド付きデシジョンテーブルでの列挙ドロップダウンオプション例



35.1. ルールアセットの詳細列挙オプション

Red Hat Process Automation Manager プロジェクトの列挙定義を使用した詳細ユースケースについては、列挙を定義する時に、以下の拡張オプションの使用を検討してください。

Business Central の値との DRL の値におけるマッピング

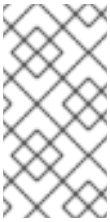
列挙値を DRL ソースに表示されるものとは異なる方法で、または完全に Business Central インターフェイスに表示する場合は、列挙定義値の形式 **'fact.field' : ['sourceValue1=UIValue1','sourceValue2=UIValue2', ...]** のマッピングを使用します。
たとえば、ローンの状態に関する以下の列挙定義では、**A** または **D** のオプションを DRL ファイルで使用しますが、Business Central では **Approved** または **Declined** のオプションが表示されます。

```
'Loan.status' : ['A=Approved','D=Declined']
```

列挙値の依存関係

選択した値を 1 つのドロップダウンリストにまとめて、後続のドロップダウンリストで利用可能なオプションを判断する場合は、列挙定義で **'fact.fieldB[fieldA=value1]' : ['value2', 'value3', ...]** の形式を使用します。
たとえば、保険契約に関する以下の列挙定義では、**policyType** フィールドに **Home** または **Car** の値を使用できます。ユーザーが選択する保険契約タイプにより、利用できる契約 **coverage** のフィールドオプションが決まります。

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注記

列挙依存関係は、ルールの条件およびアクションをまたいで適用されません。たとえば、保険契約のユースケースでは、ルール条件で選択した契約をもとに、ルールアクションで利用可能な補償オプションが決定されるわけではありません (該当する場合)。

列挙の外部データソース

列挙定義で直接、値を定義するのではなく、外部のデータソースから列挙値の一覧を取得する場合は、プロジェクトのクラスパスで、文字列の **java.util.List** リストを返すヘルパークラスを追加します。列挙定義で、値を指定する代わりに、外部の値を取得するように設定したヘルパークラスを特定します。
たとえば、ローン申請者の地域に関する以下の列挙定義では、**'Applicant.region' : ['country1', 'country2', ...]** の形式で明示的に申請者の地域を定義するのではなく、外部で定義した値の一覧を返すヘルパークラスを使用します。

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

この例では、**DataHelper** クラスに、文字列の一覧を返す **getListOfRegions()** メソッドが含まれます。列挙は、ルールアセットの関連フィールドのドロップダウンリストに、読み込まれます。

また、通常通り従属フィールドを特定して、ヘルパーへの呼び出しを引用符で囲むことで、ヘルパークラスから動的に、従属の列挙定義を読み込むこともできます。

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

リレーショナルデータベースなど、外部データソースから列挙データをすべて読み込む場合は、**Map<String, List<String>>** マッピングを返す Java クラスを実装できます。マップのキーは **fact.field** マッピングです。この値は、値の **java.util.List<String>** リストになります。

たとえば、以下の Java クラスでは、関連する列挙のローン申請者の地域を定義します。

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下の列挙定義は、この Java クラスの例に相関します。参照は Java クラスで定義されているため、列挙にはファクトまたはフィールド名への参照が含まれません。

```
=(new SampleDataSource()).loadData()
```

= 演算子を使用して、Business Central がヘルパークラスから全列挙データを読み込めるようにします。エディターで使用するように列挙定義を要求すると、ヘルパーメソッドが静的に評価されます。

注記

現在、Business Central では、ファクトおよびフィールド定義なしで列挙を定義することはできません。この方法で関連の Java クラスの列挙を定義するには、Red Hat Process Automation Manager プロジェクトで DRL ソースを使用します。

第36章 ガイド付きデシジョンテーブルのリアルタイム検証および妥当性確認

Business Central は、ガイド付きデシジョンテーブルにリアルタイム検証および妥当性確認を提供し、テーブルのエラーがなくなったことを確認します。ガイド付きデシジョンテーブルは、各セルが変更するたびに妥当性が確認されます。ロジックに問題が検出されたら、エラー通知が表示され、問題を確認できます。

36.1. ガイド付きデシジョンテーブルの問題の種類

検証および妥当性確認の機能は、以下のタイプの問題を検出します。

冗長性 (Redundancy)

冗長性は、デシジョンテーブルの 2 つの行で、同じファクトセットの同じ結果を実行する際に生じます。たとえば、顧客の誕生日をチェックして誕生日割引を提供する行が 2 つあると、割引は 2 倍になる可能性があります。

包含 (Subsumption)

包含は冗長と似ていますが、2 つルールが同じ結果を実行し、1 つのルールがもう 1 つのルールのファクトのサブセットに対して実行する場合に生じます。たとえば、以下の 2 つのルールを見てみましょう。

- **when Person age > 10 then Increase Counter**
- **when Person age > 20 then Increase Counter**

この場合、対象者の年齢が 15 歳の場合はルールが 1 つだけ実行しますが、対象者の年齢が 20 歳を超えてる場合は 2 つのルールが実行します。これにより、実行時に冗長性の場合と同様の問題が生じます。

競合 (Conflict)

2 つの類似した条件が異なる結果をもたらす場合は、競合する状況が発生します。デシジョンテーブルの 2 つ行の (ルール) または 2 つのセルの間で競合が発生する場合があります。
以下の例は、デシジョンテーブルの 2 つの行で競合が発生しているのを示しています。

- **when Deposit > 20000 then Approve Loan**
- **when Deposit > 20000 then Refuse Loan**

この場合は、ローンが承認されるかどうかについて確認することができません。

以下の例は、デシジョンテーブルの 2 つのセル間の競合を示します。

- **when Age > 25**
- **when Age < 25**

競合セルを持つ行は実行しません。

Unique Hit ポリシーの違反 (Broken Unique Hit Policy)

Unique Hit ポリシーをデシジョンテーブルに適用する際は、同時に 1 行しか実行できず、各列は一意であり、満たした条件が重複しないようにする必要があります。複数の行が実行した場合は、検証レポートが、違反したヒットポリシーを特定します。たとえば、価格の割引資格を決定するテーブルで、以下の条件を見てみましょう。

- when Is Student = true
- when Is Military = true

顧客が学生であり、軍隊に所属している場合は、両方の条件が適用され、**Unique Hit** ポリシーに違反します。したがって、この種のテーブルの行は、一度に複数のルールを実行できないように作成する必要があります。ヒットポリシーの詳細は、28章 *ガイド付きデシジョンテーブルのヒットポリシー* を参照してください。

欠陥 (Deficiency)

欠陥は競合と似ていますが、デシジョンテーブルのルールのロジックが未完成の場合に生じます。たとえば、欠陥がある以下の 2 つのルールを見てみましょう。

- when Age > 20 then Approve Loan
- when Deposit < 20000 then Refuse Loan

この 2 つのルールは、20 歳を超え、預金が 20000 より少ない場合に混乱が発生する場合があります。さらに制約を追加すると、競合を避けることができます。

列の欠落 (Missing Column)

列を削除したため、不完全または不正確なロジックが発生した場合は、ルールが正しく発生しません。これを検出し、不足している列に対処したり、ロジックを調整して、意図的に削除した条件またはアクションに依存しないようにすることができます。

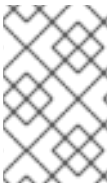
不完全な範囲 (Incomplete Ranges)

可能なフィールド値に対する制約がテーブルに追加されているにもかかわらず、可能な値がすべて定義されていない場合は、フィールド値の範囲が不完全です。検証レポートは、提供された不完全な範囲を特定します。たとえば、アプリケーションが承認されたかどうかをテーブルが確認した場合は、検証レポートにより、アプリケーションが承認されていない状況も処理できるようになります。

36.2. 通知の種類

検証および妥当性確認の機能では、3 種類の通知を使用します。

- Error: ガイド付きデシジョンテーブルが、ランタイム時に設計された通りに機能しなくなるような重大な問題があることを意味します。たとえば、競合はエラーとしてレポートされます。
- Warning: ガイド付きデシジョンテーブルが動作しなくなるような重要な問題ではありませんが、注意が必要です。たとえば、包含は警告として報告されます。
- Information: ガイド付きデシジョンテーブルの動作が止まる可能性の低い、軽度または中度の問題があり、注意が必要です。たとえば、列が不明な場合は、情報として報告されます。



注記

Business Central の確認および検証機能は、誤った変更の保存を防ぐものではありません。この機能は、編集時に問題のみをレポートするため、これらの問題を無視して変更を保存することができます。

36.3. ガイド付きデシジョンテーブルの検証および妥当性確認の無効化

Business Central のデシジョンテーブルの検証および妥当性確認機能は、デフォルトで有効になっています。この機能を使用すると、ガイド付きデシジョンテーブルの検証が容易になりますが、複雑なガイド付きデシジョンテーブルの場合は、この機能が原因でデシジョンエンジンのパフォーマンスが低下してしまう可能性があります。お使いの Red Hat Process Automation Manager ディストリビューションで **org.kie.verification.disable-dtable-realtime-verification** のシステムプロパティーを **true** に設定して、この機能を無効にできます。

手順

~/standalone-full.xml に移動して、以下のシステムプロパティーを追加します。

```
<property name="org.kie.verification.disable-dtable-realtime-verification" value="true"/>
```

たとえば、Red Hat JBoss EAP では、このシステムプロパティーを **$EAP_HOME/standalone/configuration/standalone-full.xml** に追加します。

第37章 スプレッドシート形式のデシジョンテーブルへの、ガイド付きデシジョンテーブルの変換

Business Central でガイド付きデシジョンテーブルを定義した後に、オフライン参照やファイル共有向けに、ガイド付きデシジョンテーブルを XLS スプレッドシート形式のデシジョンテーブルファイルに変換できます。ガイド付きデシジョンテーブルを変換するには、拡張エントリーのガイド付きデシジョンテーブルを使用する必要があります。変換ツールは、制限エントリーのガイド付きデシジョンテーブルをサポートしていません。

スプレッドシートのデシジョンテーブルの詳細は、**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成** を参照してください。

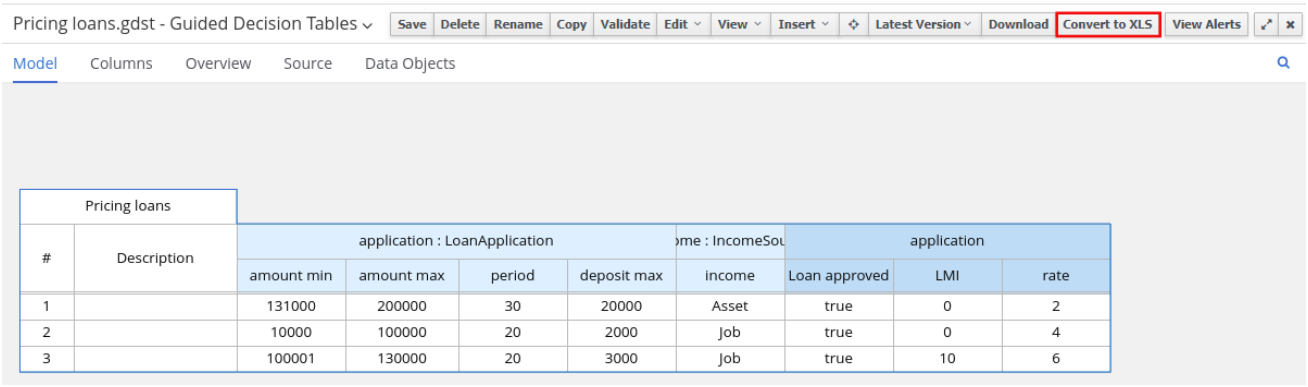
警告

ガイド付きデシジョンテーブルとスプレッドシートのデシジョンテーブルは、テーブル形式が異なり、サポート対象機能も異なります。別のデシジョンテーブル形式 (例: ヒットポリシー) に変換する場合には、サポート対象機能が両者で異なる分は変更されるか、失われることになります。

手順

Business Central で、変換するガイド付きデシジョンテーブルに移動して、デシジョンテーブルデザイナーの右上隅で **Convert to XLS** をクリックします。

図37.1 アップロードしたデシジョンテーブルの変換



変換後に、変換したデシジョンテーブルはプロジェクト内のスプレッドシート形式のデシジョンテーブルアセットとして利用でき、ダウンロードしてオフラインで参照できるようになります。

第38章 ルールの実行

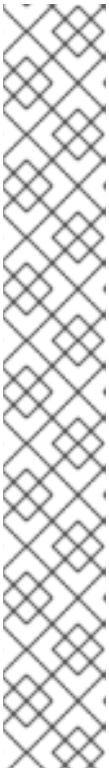
ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントのオプションに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。



注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合は、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれていることを確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、Business Central 外に Maven または Java プロジェクトが作成されていない場合は作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例は、その他の DRL ルールの作成および実行方法 を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。

- **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルに読み込みます。
- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (任意) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける Red Hat Process Automation Manager 7.11 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

5. モデルクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
   Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイルの両方に表示されます。

   ```
   <dependency>
     <groupId>com.sample</groupId>
     <artifactId>Person</artifactId>
     <version>1.0.0</version>
   </dependency>
   ```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

   ```
   org.slf4j.simpleLogger.defaultLogLevel=debug
   ```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースを読み込む **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
   たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
```

```
    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

KIE Server でのルールの実行

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);
```

```
        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert Person into the session:
        KieCommands kieCommands = KieServices.Factory.get().getCommands();
        List<Command> commandList = new ArrayList<Command>();
        commandList.add(kieCommands.newInsert(p, "personReturnId"));

        // Fire all rules:
        commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
        BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

        // Use rule services client to send request:
        RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
        ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
        System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
}
```

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリーにおける) Java の実行例

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

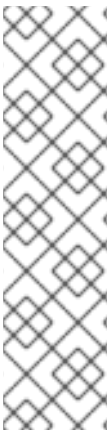
9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

第39章 次のステップ

- テストシナリオを使用したデシジョンサービスのテスト
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート V. スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成

ビジネスアナリストまたはビジネスルールの開発者は、スプレッドシートのデシジョンテーブル内に、テーブル形式でビジネスルールを定義し、Business Central のプロジェクトにスプレッドシートをアップロードできます。このルールは Drools Rule Language (DRL) に組み込まれ、プロジェクトのデシジョンサービスの中心となります。



注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

前提条件

- デシジョンテーブルのスペースおよびプロジェクトが Business Central に作成されている。各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は **デシジョンサービスのスタートガイド** を参照してください。

第40章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表40.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	**ガイド付きルールを使用したデシジョンサービスの作成**
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	**ガイド付きルールテンプレートを使用したデシジョンサービスの作成**
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	**DRL ルールを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第41章 スプレッドシートのデシジョンテーブル

スプレッドシートのデシジョンテーブルは、表形式でビジネスルールを定義する XLS 形式または XLSX 形式のスプレッドシートです。スプレッドシートのデシジョンテーブルは、スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにアップロードしたりできます。スプレッドシートの各行がルールになり、各列が条件、アクション、または別のルール属性になります。スプレッドシートのデシジョンテーブルを作成してアップロードした後に、その他のすべてのルールアセットと同じように、定義したルールを Drools Rule Language (DRL) ルールにコンパイルします。

スプレッドシートのデシジョンテーブルに関連するデータオブジェクトはすべて、スプレッドシートのデシジョンテーブルと同じプロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。その他のパッケージの既存アセットは、デシジョンテーブルを使用してインポートできます。

第42章 データオブジェクト

データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

42.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

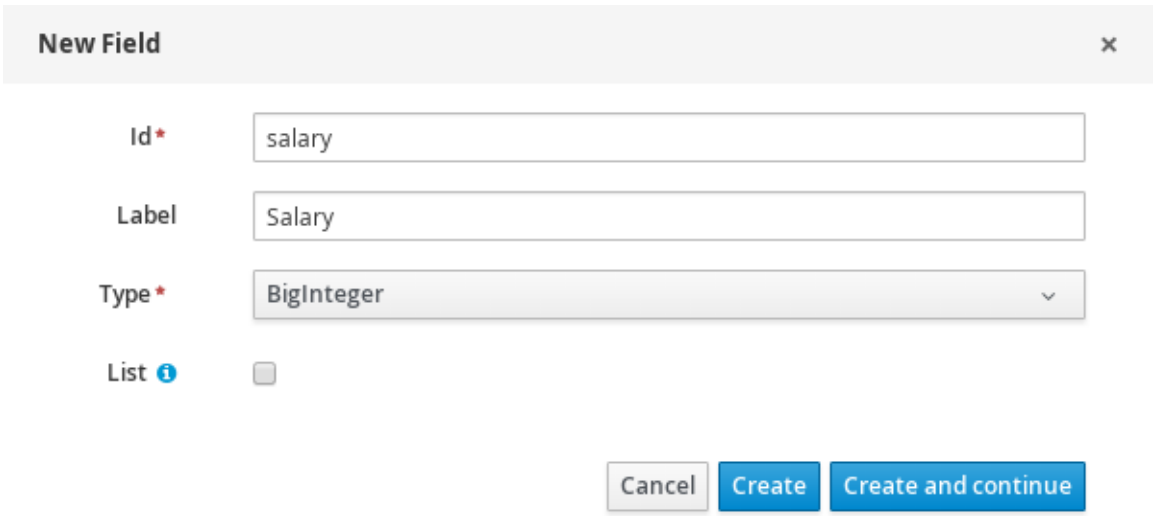
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

別のパッケージからのデータオブジェクトのインポート

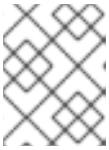
別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図42.1データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。



注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

第43章 デシジョンテーブルのユースケース

オンラインショッピングのサイトでは、注文したアイテムの配送料金が一覧表示されます。このサイトでは、以下の条件で配送料金が無料となります。

- 注文したアイテム数が 4 点以上、合計金額が 300 ドル以上。
- 配送の種類で標準が選択されている (購入日から 4 - 5 営業日に配送)。

この条件を適用した配送料金は以下のようになります。

表43.1 注文金額が 300 ドル未満の場合

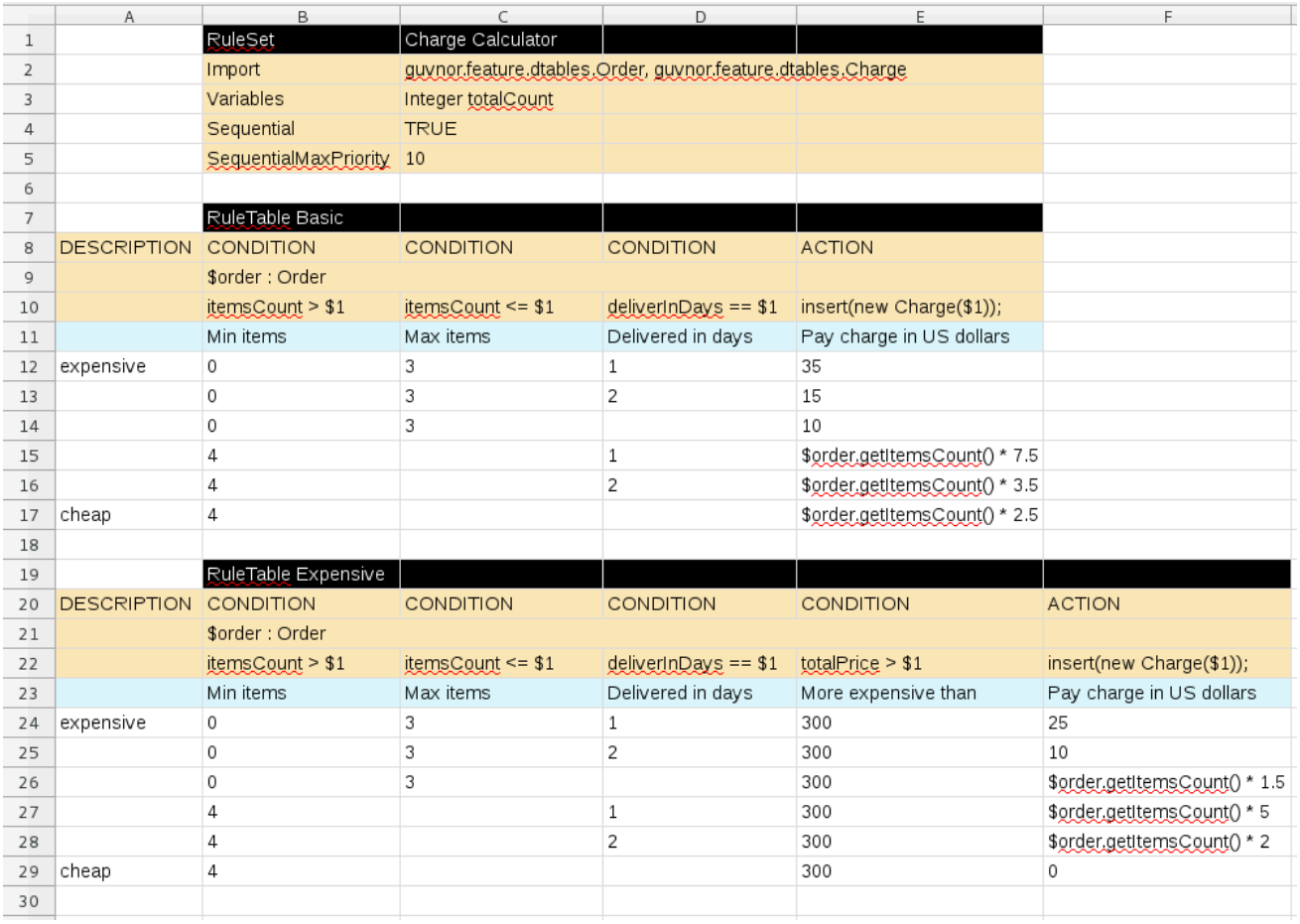
アイテム数	配送日	配送料金 (米ドル) (N はアイテム数)
3 点以下	翌日	35
	2 日後	15
	標準	10
4 点以上	翌日	N*7.50
	2 日後	N*3.50
	標準	N*2.50

表43.2 注文金額が 300 ドルを超える場合

アイテム数	配送日	配送料金 (米ドル) (N はアイテム数)
3 点以下	翌日	25
	2 日後	10
	標準	N*1.50
4 点以上	翌日	N*5
	2 日後	N*2
	標準	無料

この条件と料金は、以下のサンプルスプレッドシートのデシジョンテーブルで紹介しています。

図43.1 配送料金のデシジョンテーブル



	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

この例が示すように、デシジョンテーブルを Business Central にアップロードするには、テーブルが XLS 形式または XLSX 形式のスプレッドシートで、構造と構文の要件に準拠する必要があります。詳細は、*44章 スプレッドシートのデシジョンテーブルの定義* を参照してください。

第44章 スプレッドシートのデシジョンテーブルの定義

スプレッドシート形式のデシジョンテーブル (XLS または XLSX) には、ルールデータを定義する 2 つの重要な領域、**RuleSet** 領域と **RuleTable** 領域が必要です。**RuleSet** 領域では、ルールセット名、ユニバーサルルール属性など、(このスプレッドシートだけでなく) すべてのルールをパッケージ全体に、グローバルに適用する要素を定義します。**RuleTable** 領域では、実際のルール (行) と、指定したルールセットのルールテーブルを設定する条件、アクション、その他のルール属性 (列) を定義します。スプレッドシート形式のデシジョンテーブルには **RuleTable** 領域を複数追加できますが、**RuleSet** 領域は 1 つだけとなります。

重要

通常は、デシジョンテーブルのスプレッドシートを 1 つだけアップロードする必要があります。これには、Business Central の 1 つのルールパッケージに必要なすべての **RuleTable** 定義が含まれます。異なるパッケージに複数のデシジョンテーブルのスプレッドシートをアップロードすることはできますが、同じパッケージに複数のスプレッドシートをアップロードすると、**RuleSet** 属性または **RuleTable** 属性が競合するコンパイルエラーが発生する可能性があるため、これは推奨されません。

デシジョンテーブルを定義する際は、以下のサンプルスプレッドシートを参照してください。

図44.1配送料金のスプレッドシートデシジョンテーブル例

	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

手順

1. 新しいスプレッドシート (XLS または XLSX) の 2 列目または 3 列目 (サンプルの行 1) のセルに、**RuleSet** とラベルを付けます。左の列は、(任意で) 記述的メタデータに使用します。

2. 右隣のセルに、**RuleSet** の名前を入力します。このルールセットには、ルールパッケージに定義する **RuleTable** ルールがすべて含まれます。

3. **RuleSet** セルの下に、そのパッケージ内のすべてのルールテーブルにグローバルに適用するルール属性 (セルごとに 1 つ) を定義します。右のセルに属性値を指定します。たとえば、ラベルを **Import** にして、その右隣のセルに、その他のパッケージからデシジョンテーブルのパッケージにインポートするデータオブジェクトを指定します (形式は **package.name.object.name**)。サポートされるセルのラベルと値については、「RuleSet の定義」 を参照してください。

4. **RuleSet** セルと同じ列で、**RuleSet** 領域の何行か下の新しいセルに、ラベル **RuleTable** を入力し (サンプルの行 7)、テーブル名も同じセルに入力します。この名前は、区別のために追加したルールの番号とともに、このルールテーブルに指定した全ルールの名前の最初の部分として使用されます。この自動命名ルールは、**NAME** 属性列を追加すると上書きできます。

5. その下の 4 行には、必要に応じて以下の要素を定義します (サンプルの行 8-11)。

   - **ルール属性:** 条件、アクション、またはその他の属性。サポートされるセルのラベルと値については 「RuleTable の定義」 を参照してください。

   - **オブジェクトタイプ:** ルール属性が適用されるデータオブジェクト。同じオブジェクトタイプを複数の列に適用する場合は、複数のセルでオブジェクトタイプを繰り返すのではなく、(サンプルのデシジョンテーブルに示されるように) 複数のオブジェクトセルを 1 つのセルに結合します。オブジェクトタイプを結合すると、結合範囲の下にあるすべての列が、1 つのパターンに指定する一連の制約になり、1 度に 1 つのファクトに一致するようになります。オブジェクトを別の列で繰り返し使用すると、列ごとに別のパターンを作成でき、異なるファクトや同じファクトを一致させることができます。

   - **制約:** オブジェクトタイプの制約。

   - **列ラベル:** (任意) 見やすくするために説明を入力する列のラベル。使用しない場合は空白にします。

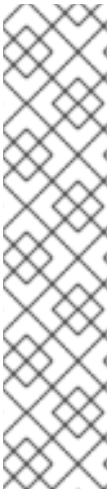
   

   注記

   オブジェクトタイプと制約セルの両方を追加する代わりに、オブジェクトタイプのセルを空のままにし、対応する制約セルに完全式を追加します。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

6. 必要なルール属性 (列) をすべて定義したら、必要に応じて各列の各行に値を入力してルールを生成します (サンプルの行 12-17)。データのないセルは無視されます (条件やアクションなどが適用されません)。
   デシジョンテーブルのスプレッドシートにさらにルールテーブルを追加する場合は、前のテーブルの最後の行の後に 1 行空け、前のテーブルの **RuleTable** セルと **RuleSet** セルと同じ列のセルに、別の **RuleTable** のラベルを付け、このセクションの手順を繰り返して新しいテーブルを作成します (サンプルの行 19-29)。

7. XLS または XLSX のスプレッドシートを保存して終了します。



注記

デフォルトでは、Business Central にスプレッドシートをアップロードすると、スプレッドシートワークブックの最初のワークシートだけがデシジョンテーブルとして処理されます。**RuleTable** 名とともに使用する各 **RuleSet** の名前は、同じパッケージの全デシジョンテーブルファイルで一意にする必要があります。

複数のワークシートを含むデシジョンテーブルを処理する場合には、スプレッドシートワークブックと同じ名前の **.properties** ファイルを作成します。**.properties** ファイルには、ワークシート名を CSV (コンマ区切りの値) 形式で指定したプロパティーシートを含める必要があります。以下の例を示します。

```
sheets=Sheet1,Sheet2
```

Business Central でデシジョンテーブルをアップロードすると、以下の例のように、サンプルスプレッドシートのルールが DRL ルールとして表示されます。

```
//row 12
rule "Basic_12"
salience 10
  when
    $order : Order( itemsCount > 0, itemsCount <= 3, deliverInDays == 1 )
  then
    insert( new Charge( 35 ) );
end
```

セルの値に使用するホワイトスペースの有効化

デフォルトでは、デシジョンテーブルのセルの値の前後にあるホワイトスペースのは、デシジョンエンジンがデシジョンテーブルを処理する前に削除されます。セルの値の前後に意図的にホワイトスペースのを保持するには、Red Hat Process Automation Manager のディストリビューションで **drools.trimCellsInDTable** システムプロパティーを **false** に設定します。

たとえば、Red Hat Process Automation Manager と Red Hat JBoss EAP を併用するには、以下のシステムプロパティーを **$EAP_HOME/standalone/configuration/standalone-full.xml** ファイルに追加してください。

```
<property name="drools.trimCellsInDTable" value="false"/>
```

Java アプリケーションに埋め込まれたデシジョンエンジンを使用する場合は、以下のコマンドでシステムプロパティーを追加してください。

```
java -jar yourApplication.jar -Ddrools.trimCellsInDTable=false
```

44.1. RULESET の定義

デシジョンテーブルの **RuleSet** 領域のエントリーは、(そのスプレッドシートだけでなく) パッケージのすべてのルールに適用される DRL 制約およびルール属性を定義します。エントリーは、セルのペア (最初のセルにラベル、その右隣のセルに値) が縦方向に積み上げられます。デシジョンテーブルのスプレッドシートには、**RuleSet** 領域が 1 つだけあります。

以下の表は、**RuleSet** 定義でサポートされるラベルと値を示しています。

表44.1 サポートされる RuleSet の定義

ラベル	値	使用法
RuleSet	生成した DRL ファイルのパッケージ名。任意。デフォルトは **rule_table** です。	最初のエントリーになります。
Sequential	**true** または **false**。**true** の場合は、ルールを上から適用する優先順位を使用します。	任意。1つまで指定可能。1つまで指定可能。省略すると、適用順は指定されません。
SequentialMaxPriority	整数値	任意。1つまで指定可能。1つまで指定可能。順次モードでこのオプションを使用して、優先順位の開始値を設定します。省略した場合のデフォルト値は 65535 です。
SequentialMinPriority	整数値	任意。1つまで指定可能。順次モードでこのオプションを使用して、優先順位の最低値に違反していないかどうかを確認します。省略した場合のデフォルト値は 0 です。
EscapeQuotes	**true** または **false**。**true** の場合は、引用符がエスケープされ、DRL にそのまま表示されます。	任意。1つまで指定可能。1回まで指定可能。省略すると、引用符がエスケープされます。
Import	別のパッケージからインポートする、コンマ区切りの Java クラスのリスト。	任意。繰り返して使用可能。
Variables	DRL グローバルの宣言 (型に変数名が続く)。グローバル定義が複数になる場合は、コンマで区切る必要があります。	任意。繰り返して使用可能。
関数	DRL 構文に準拠している1つまたは複数の関数定義。	任意。繰り返して使用可能。
Queries	DRL 構文に準拠している1つまたは複数のクエリー定義。	任意。繰り返して使用可能。
Declare	DRL 構文に準拠している1つまたは複数の宣言型。	任意。繰り返して使用可能。
Unit	このデシジョンテーブルから生成されたルールが属するルールユニットです。	任意。1つまで指定可能。省略すると、ルールはユニットに属しません。

ラベル	値	使用法
Dialect	**Java** または **mvel**。デシジョンテーブルのアクションで使用される方言。	任意。1つまで指定可能。省略した場合には、**java** が課されます。

警告

Microsoft Office、LibreOffice、および OpenOffice で二重引用符のエンコード方法が異なり、コンパイルエラーが発生する場合があります。たとえば、**“A"** は失敗しますが、**"A"** は成功します。

44.2. RULETABLE の定義

デシジョンテーブルの **RuleTable** 領域のエントリーは、そのルールテーブルのルールに対する条件、アクション、その他のルール属性を定義します。デシジョンテーブルのスプレッドシートには、**RuleTable** 領域を複数追加できます。

以下の表は、**RuleTable** 定義でサポートされるラベル (列ヘッダー) および値を示しています。列ヘッダーには、指定のラベル、またはこの表に記載されている文字で始まるカスタムラベルのいずれかを使用できます。

表44.2 サポートされる RuleTable の定義

ラベル	またはカスタムラベルの頭文字	値	使用法
NAME	N	その行で生成したルールの名前を提供します。デフォルトは、**RuleTable** タグと行番号に続くテキストから作成されます。	最大1列。
DESCRIPTION	I	生成したルールのコメントになります。	最大1列。
CONDITION	C	条件内のパターンに制約を構築するコードスニペットおよび補間値。	ルールテーブルごとに最低1つ。
ACTION	A	ルールの結果に対するアクションを構築するコードスニペットおよび補間値。	ルールテーブルごとに最低1つ。
METADATA	@	ルールに対するメタデータエントリーを構築するコードスニペットおよび補間値。	任意。列の数。

以下のセクションでは、条件、アクション、メタデータの列のセルデータがどのように使用されるかを説明します。

条件

CONDITION ヘッダーの列では、連続した行のセルが、条件要素になります。

- **1つ下のセル: CONDITION** のすぐ下のセルのテキストは、ルール条件のパターンを進化させ、次の行のスニペットを制約として使用します。そのセルを、隣接した1つ以上のセルと結合した場合は、複数の制約を持った1つのパターンが作成されます。すべての制約が結合して括弧で囲まれ、このセルのテキストに追加されます。
  このセルを空にすると、以下のセルのコードスニペットが自動的に有効な条件要素となります。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

  パターンのテキストの前に別のパターンを記述すれば、制約を使用しないパターンを追加できます。中が空の括弧は追加することも省くこともできます。パターンに **from** 句を追加することもできます。

  パターンを **eval** で終了すると、コードスニペットが **eval** の後の括弧の中にブール表現を生成します。

  @watch アノテーションを使用してパターンを終了できます。これは、パターンが反応するプロパティーをカスタマイズするために使用されます。

- **2つ下のセル: CONDITION** の2つ下のセルのテキストは、1つ下のセルのオブジェクト参照の制約として処理されます。このセルのコードスニペットは、その列のさらに下にあるセルから値が補間されます。下のセルからの値と **==** を使用した比較で設定される制約を作成する場合は、フィールドセレクターだけで十分です。フィールドセレクターのみを使用し、**==** の比較を追加せずに条件を使用する場合は、記号 **?** で条件を終了する必要があります。その他の比較演算子は、スニペットの最後に指定する必要があり、値は下のセルから追加されます。その他のすべての制約形式については、**$param** シンボルを使用して、セルの内容を追加する場所を指定する必要があります。**$1**、**$2** などのシンボルを使用し、下のセルにコンマで区切った値を指定すれば、複数の値を挿入することもできます。**$1**、**$2** などをコンマで区切らないでください。テーブルの処理に失敗します。
  パターン **forall($delimiter){$snippet}** に従ってテキストを展開する場合は、その下の各セルで、コンマ区切りの値に対して **$snippet** がそれぞれ1回ずつ使用されます。**$** シンボルの場所に値が挿入され、指定した **$delimiter** で結合します。**forall** 構文は、他のテキストで囲むことができるのに注意してください。

  1つ下のセルにオブジェクトが含まれている場合は、そのセルから条件要素に完全なコードスニペットが追加されます。括弧のペアと、(結合したセルのパターンに複数の制約が追加されている場合は) 区切り文字のコンマが自動的に提供されます。1つ下のセルが空の場合は、このセルのコードスニペットが自動的に有効な条件要素となります。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

- **3つ下のセル: CONDITION** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。

- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補間データとして提供されます。セルが空の場合は、このルールで制約や条件が省略されます。

アクション

ACTION ヘッダーの列では、連続した行のセルが、アクション命令文になります。

- **1つ下のセル: ACTION** ヘッダーの1つ下のセルのテキストは任意です。テキストがある場合は、オブジェクト参照として解釈されます。

- **2つ下のセル: ACTION** の2つ下のセルのテキストは、その列のさらに下にあるセルの値が補完されるコードスニペットです。挿入が1つの場合は、**$param** シンボルを使用して、セルの内容を追加する場所を指定します。**$1**、**$2** などのシンボルを使用し、下のセルにコンマで区切った値を指定すれば、複数の値を挿入することもできます。**$1**、**$2** などをコンマで区切らないでください。テーブルの処理に失敗します。
  テキストにマーカーシンボルがない場合は、補完なしでメソッドが呼び出されます。このとき、下の行で空セル以外のエントリーを使用して、命令文を追加します。**forall** 構文がサポートされます。

  1つ下のセルにオブジェクトが含まれている場合は、そのセルのテキスト、ピリオド、2つ下のセルのテキスト、終わりを示すセミコロンが1列に並べられ、アクション命令文として追加されるメソッドコールとなります。1つ下のセルが空の場合は、このセルのコードスニペットが自動的に有効なアクション要素となります。

- **3つ下のセル: ACTION** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。

- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補間データとして提供されます。セルが空の場合は、このルールで制約や条件が省略されます。

メタデータ

ヘッダーが **METADATA** の列では、連続した行のセルが、生成されるルールのメタデータアノテーションになります。

- **1つ下のセル: METADATA** の1つ下のセルのテキストは無視されます。

- **2つ下のセル: METADATA** の2つ下のセルのテキストは、ルール行のセルの値を使用して補完されます。メタデータのマーカー文字 @ がプリフィックスとして自動的に追加されるため、このセルのテキストに追加する必要はありません。

- **3つ下のセル: METADATA** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。

- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補間データとして提供されます。セルが空の場合は、このルールでメタデータアノテーションが省略されます。

44.3. RULESET 定義または RULETABLE 定義におけるその他のルール属性

RuleSet 領域および **RuleTable** 領域は、**PRIORITY**、**NO-LOOP** などのラベルおよび値もサポートします。**RuleSet** 領域に指定したルール属性は、(そのスプレッドシートだけでなく) 同じパッケージにあるすべてのルールアセットに影響します。**RuleTable** 領域に指定したルール属性は、そのルールテーブル内のルールにのみ影響します。各ルール属性は、**RuleSet** 領域で一度、**RuleTable** 領域で一度使用できます。同じ属性を、スプレッドシートの **RuleSet** 領域および **RuleTable** 領域の両方で使用した場合は、**RuleTable** が優先され、**RuleSet** 領域の属性が上書きされます。

以下の表は、その他の **RuleSet** 定義または **RuleTable** 定義でサポートされるラベル (列ヘッダー) および値を示します。列ヘッダーには、指定のラベル、またはこの表に記載されている文字で始まるカスタムラベルのいずれかを使用できます。

表44.3 RuleSet 定義または RuleTable 定義におけるその他のルール属性

ラベル	またはカスタムラベルの頭文字	値
PRIORITY	P	ルールの優先順位 (**salience**) の値を定義する整数。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。これは、**Sequential** フラグで上書きされます。 例: **PRIORITY 10**
DATE-EFFECTIVE	V	日付定義および時間定義を含む文字列。現在の日時が **DATE-EFFECTIVE** 属性よりも後の場合は、このルールがアクティブになります。 例: **DATE-EFFECTIVE "4-Sep-2018"**
DATE-EXPIRES	Z	日付定義および時間定義を含む文字列。現在日時が **DATE-EXPIRES** 属性よりも後になると、このルールをアクティブにすることはできません。 例: **DATE-EXPIRES "4-Oct-2018"**
NO-LOOP	U	ブール値。このオプションを **true** に設定すると、以前一致した条件がこのルールにより再トリガーとなる場合に、このルールを再度アクティブにする (ループする) ことができません 。 例: **NO-LOOP true**
AGENDA-GROUP	G	ルールを割り当てるアジェンダグループを指定する文字列。アジェンダグループを使用すると、アジェンダをパーティションで区切り、ルールのグループに対する実行をさらに制御できます。フォーカスを取得したアジェンダグループのルールだけがアクティブになります。 例: **AGENDA-GROUP "GroupName"**
ACTIVATION-GROUP	X	ルールを割り当てるアクティベーション (または XOR) グループを指定する文字列。アクティベーショングループでは、1 つのルールのみをアクティブ化できます。最初のルールが実行すると、アクティベーショングループの中で、アクティベーションが保留中のルールはすべてキャンセルされます。 例: **ACTIVATION-GROUP "GroupName"**
DURATION	D	ルールの条件が一致している場合に、ルールがアクティブになってからの時間をミリ秒で定義する長整数値。 例: **DURATION 10000**

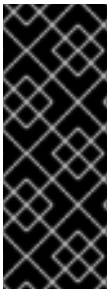
ラベル	またはカスタムラベルの頭文字	値
TIMER	T	ルールのスケジュールに対する **int** (間隔) または **cron** タイマー定義を指定する文字列。 例: **TIMER "*/5 * * * *"** (5 分ごと)
CALENDAR	E	ルールのスケジュールを指定する Quartz カレンダーの定義。 例: **CALENDAR "* * 0-7,18-23 ?* *"** (営業時間外を除く)
AUTO-FOCUS	F	アジェンダグループ内のルールにのみ適用可能なブール値。このオプションが **true** に設定されている場合は、次にルールがアクティブになったときに、そのルールが割り当てられたアジェンダグループにフォーカスが自動的に指定されます。 例: **AUTO-FOCUS true**
LOCK-ON-ACTIVE	L	ルールフローグループまたはアジェンダグループ内のルールにのみ適用可能なブール値。このオプションを **true** に設定すると、次回、ルールのルールフローグループがアクティブになるか、ルールのアジェンダグループがフォーカスを受けると、(ルールフローグループがアクティブでなくなるか、アジェンダグループがフォーカスを失うまで) ルールをアクティブにすることができません。これは、**no-loop** 属性を強力にしたものです。なぜなら、一致するルールのアクティベーションが、(ルールそのものによるものだけでなく) 更新元にかかわらず破棄されるためです。この属性は、ファクトを修正するルールが多数あり、ルールの再一致と再発行を希望しない計算ルールに適しています。 例: **LOCK-ON-ACTIVE true**
RULEFLOW-GROUP	R	ルールフローグループを指定する文字列。ルールフローグループで、関連するルールフローによってそのグループがアクティブになった場合に限りルールを発行できます。 例: **RULEFLOW-GROUP "GroupName"**

図44.2 属性列が含まれるデシジョンテーブルのサンプルスプレッドシート

	A	B	C	D	E	F	G	H
1		RuleSet	Charge Calculator					
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge					
3		Variables	Integer totalCount					
4		Sequential	TRUE					
5		SequentialMaxPriority	10					
6								
7		RuleTable Basic						
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION			
9		$order : Order						
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));			
11		Min items	Max items	Delivered in days	Pay charge in US dollars			
12	expensive	0	3	1	35			
13		0	3	2	15			
14		0	3		10			
15		4		1	$order.getItemsCount() * 7.5			
16		4		2	$order.getItemsCount() * 3.5			
17	cheap	4			$order.getItemsCount() * 2.5			
18								
19		RuleTable Expensive						
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	RULEFLOW-GROUP	NOLOOP	ACTION
21		$order : Order						
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1			insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than			Pay charge in US dollars
24	expensive	0	3	1	300		TRUE	25
25		0	3	2	300		TRUE	10
26		0	3		300		TRUE	$order.getItemsCount() * 1.5
27		4		1	300	discount assessment		$order.getItemsCount() * 5
28		4		2	300	discount assessment		$order.getItemsCount() * 2
29	cheap	4			300	discount assessment		0
30								

第45章 スプレッドシート形式のデシジョンテーブルの BUSINESS CENTRAL へのアップロード

外部の XLS ファイルまたは XLSX スプレッドシート形式のデシジョンテーブルを定義した後に、Business Central のプロジェクトに、このスプレッドシートファイルをアップロードできます。



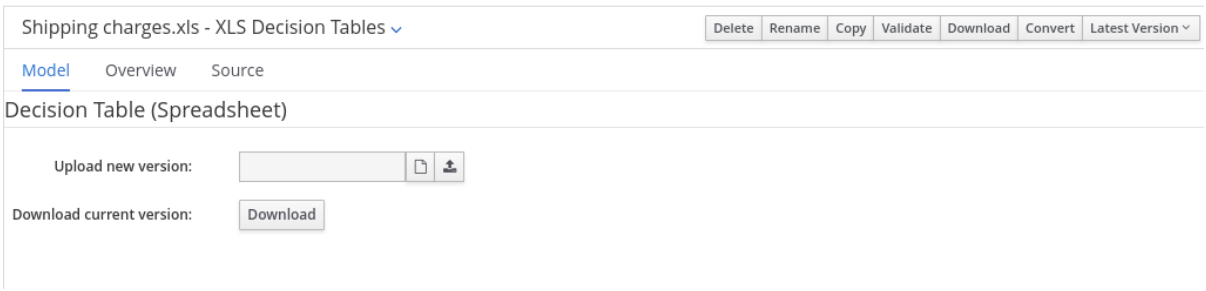
重要

通常は、デシジョンテーブルのスプレッドシートを 1 つだけアップロードする必要があります。これには、Business Central の 1 つのルールパッケージに必要なすべての **RuleTable** 定義が含まれます。異なるパッケージに複数のデシジョンテーブルのスプレッドシートをアップロードすることはできますが、同じパッケージに複数のスプレッドシートをアップロードすると、**RuleSet** 属性または **RuleTable** 属性が競合するコンパイルエラーが発生する可能性があるため、これは推奨されません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Decision Table (Spreadsheet)** をクリックします。
3. 参考となる **デシジョンテーブル** 名を入力し、適切な **パッケージ** を選択します。
4. **Choose File** アイコンをクリックして、スプレッドシートを選択します。**OK** をクリックしてアップロードします。
5. デシジョンテーブルデザイナーの右上のツールバーで **Validate** をクリックして、テーブルを検証します。テーブル検証に失敗した場合は、XLS ファイルまたは XLSX ファイルを開いて、構文エラーに対処します。構文のヘルプは *44章 スプレッドシートのデシジョンテーブルの定義* を参照してください。
   デシジョンテーブルの新しいバージョンのアップロード、または現行バージョンのダウンロードが可能です。

図45.1 アップロードしたデシジョンテーブルのオプション



第46章 BUSINESS CENTRAL にアップロードしたスプレッドシート形式のデシジョンテーブルの、ガイド付きデシジョンテーブルへの変換

XLS または XLSX のスプレッドシート形式のデシジョンテーブルファイルを Business Central のプロジェクトへアップロードした後、デシジョンテーブルを Business Central で直接変更できるガイド付きデシジョンテーブルに変換することができます。

ガイド付きデシジョンテーブルの詳細は、**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成** を参照してください。

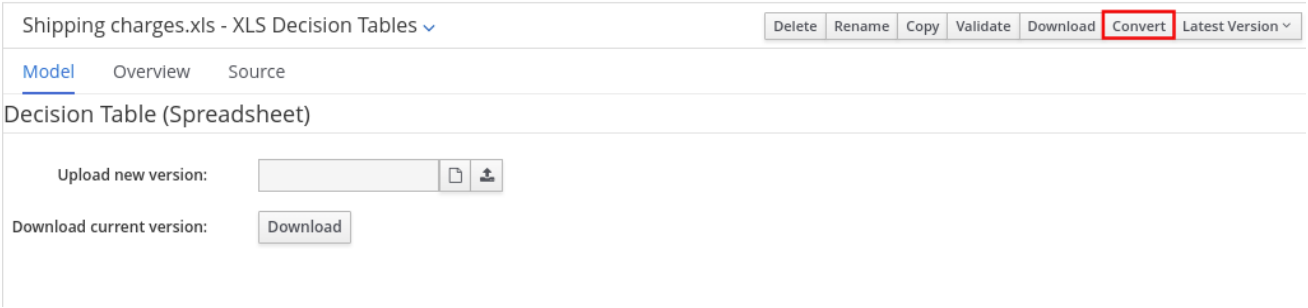
警告

ガイド付きデシジョンテーブルとスプレッドシートのデシジョンテーブルは、デシジョンテーブル形式が異なり、サポート対象機能も異なります。別のデシジョンテーブル形式に変換する場合には、サポート対象機能が両者で異なる分は変更されるか、失われることになります。

手順

Business Central で、変換するアップロードされたデシジョンテーブルアセットに移動して、デシジョンテーブルデザイナーの右上にあるツールバーで **Convert** をクリックします。

図46.1 アップロードしたデシジョンテーブルの変換



変換後のデシジョンテーブルは、Business Central で直接変更可能なプロジェクト内のガイド付きデシジョンテーブルアセットとして利用できるようになります。

第47章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントのオプションに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合は、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれていることを確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、Business Central 外に Maven または Java プロジェクトが作成されていない場合は作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例は、その他の DRL ルールの作成および実行方法 を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。

- **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルに読み込みます。
- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (任意) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける Red Hat Process Automation Manager 7.11 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

5. モデルクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイルの両方に表示されます。

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースを読み込む **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
```

```
    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

KIE Server でのルールの実行

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);
```

```
        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert Person into the session:
        KieCommands kieCommands = KieServices.Factory.get().getCommands();
        List<Command> commandList = new ArrayList<Command>();
        commandList.add(kieCommands.newInsert(p, "personReturnId"));

        // Fire all rules:
        commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
        BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

        // Use rule services client to send request:
        RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
        ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
        System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
}
```

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリーにおける) Java の実行例

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

第48章 次のステップ

- テストシナリオを使用したデシジョンサービスのテスト
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート VI. ガイド付きルールを使用したデシジョンサービスの作成

ビジネス分析者またはビジネスルールの開発者は、Business Central でガイド付きルールデザイナーを使用してビジネスルールを定義できます。このガイド付きルールは Drools Rule Language (DRL) に組み込まれ、プロジェクトのデシジョンサービスの中心となります。

注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

前提条件

- ガイド付きルールの領域およびプロジェクトが Business Central に作成されている。各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は **デシジョンサービスのスタートガイド** を参照してください。

第49章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表49.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第50章 ガイド付きルール

ガイド付きルールは、ルール作成のプロセスを提供する、Business Central の UI ベースのガイド付きルールデザイナーで作成するビジネスルールです。ガイド付きルールデザイナーを使用すると、ルールを定義するデータオブジェクトに基づいて、可能なインプットにフィールドおよびオプションを提供します。定義したガイド付きルールは、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。

ガイド付きルールに関連するすべてのデータオブジェクトは、ガイド付きルールと同じプロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとガイド付きルールを作成したら、ガイド付きルールデザイナーの **Data Objects** タブから、必要なデータオブジェクトがすべてリストされていることを検証したり、**新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

第51章 データオブジェクト

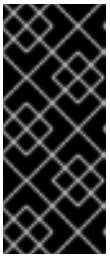
データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

51.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

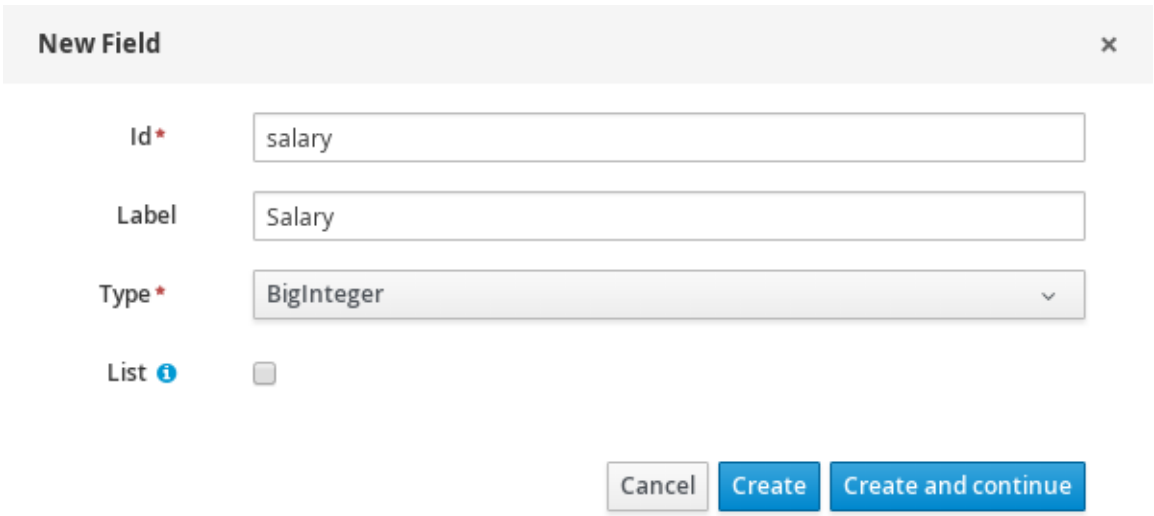


別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図51.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

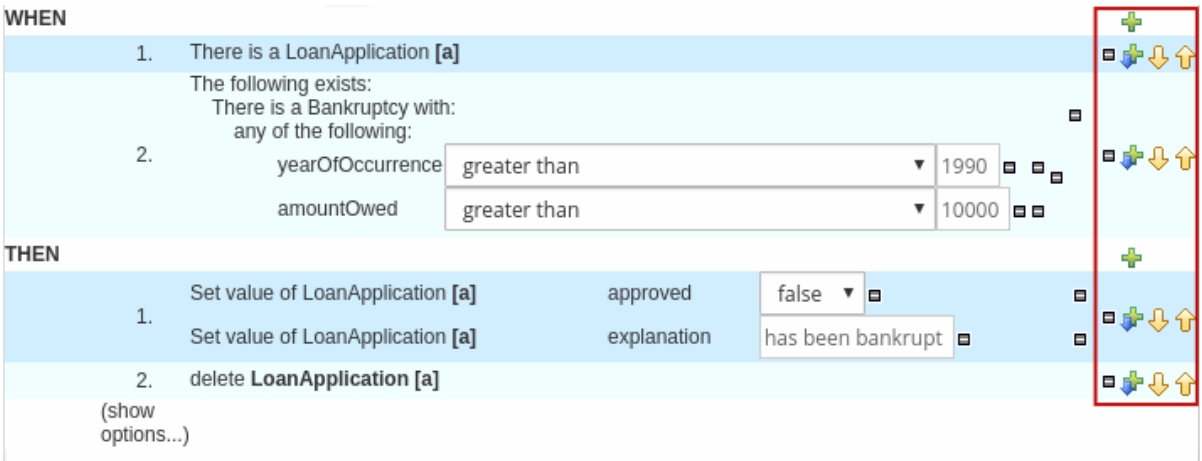
第52章 ガイド付きルールの作成

ガイド付きルールを使用すると、そのルールに関連するデータオブジェクトに基づいて、構造化フォーマットでビジネスルールを定義できるようになります。プロジェクトに個別にルールを作成および定義することができます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule** をクリックします。
3. 参考となる **ガイド付きルール** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
   ドメイン固有言語 (DSL) アセットがプロジェクトに定義されている場合は、**Show declared DSL sentences** を選択することもできます。この DSL アセットは、ガイド付きルールデザイナーで定義する条件およびアクションに使用できるオブジェクトです。
4. **OK** をクリックして、ルールアセットを作成します。
   新しいガイド付きルールが、**Project Explorer** の **Guided Rules** パネルに追加されます。**Show declared DSL sentences** オプションを選択している場合は **Guided Rules (with DSL)** パネルに追加されます。
5. **Data Objects** タブをクリックして、ルールに必要なデータオブジェクトがすべてリストされていることを確認します。リストされていない場合は、**New item** をクリックして、他のパッケージからデータオブジェクトをインポートするか、パッケージに データオブジェクトを作成 します。
6. データオブジェクトをすべて配置したら、ガイド付きルールデザイナーの **Model** タブに戻り、ウィンドウの右側のボタンから、利用可能なデータオブジェクトに、ルールの **WHEN** (条件) セクションおよび **THEN** (アクション) セクションを追加して定義します。

図52.1 ガイド付きルールデザイナー



ルールの **WHEN** 部分は、アクションを実行するのに必要な条件が含まれます。たとえば、銀行のローン申し込みに年齢制限 (21 歳以上) が必要な場合、**Underage** ルールの **WHEN** 条件は **Age | less than | 21** となります。

ルールの **THEN** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、ローンの申込者が 21 歳に満たない場合は、**THEN** アクションの **approved** が **false** になり、年齢が基準に達していないためローンの申し込みが承認されません。

例外を設定して、より複雑なルールを指定することもできます。たとえば、申込者の年齢が達していなくても、保護者の承認があれば承認されるようにすることもできます。この場合は、guarantor データオブジェクトを作成またはインポートして、そのフィールドをガイド付きルールに追加します。

7. ルールのコンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ガイド付きルールの妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。

8. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

52.1. ガイド付きルールへの WHEN 条件の追加

ルールの **WHEN** 部分は、アクションを実行するのに必要な条件が含まれます。たとえば、銀行のローン申し込みに年齢制限 (21 歳以上) が必要な場合、**Underage** ルールの **WHEN** 条件は **Age | less than | 21** となります。ルールをいつ、どのように適用するかを決定するために、単純または複雑な条件を設定できます。

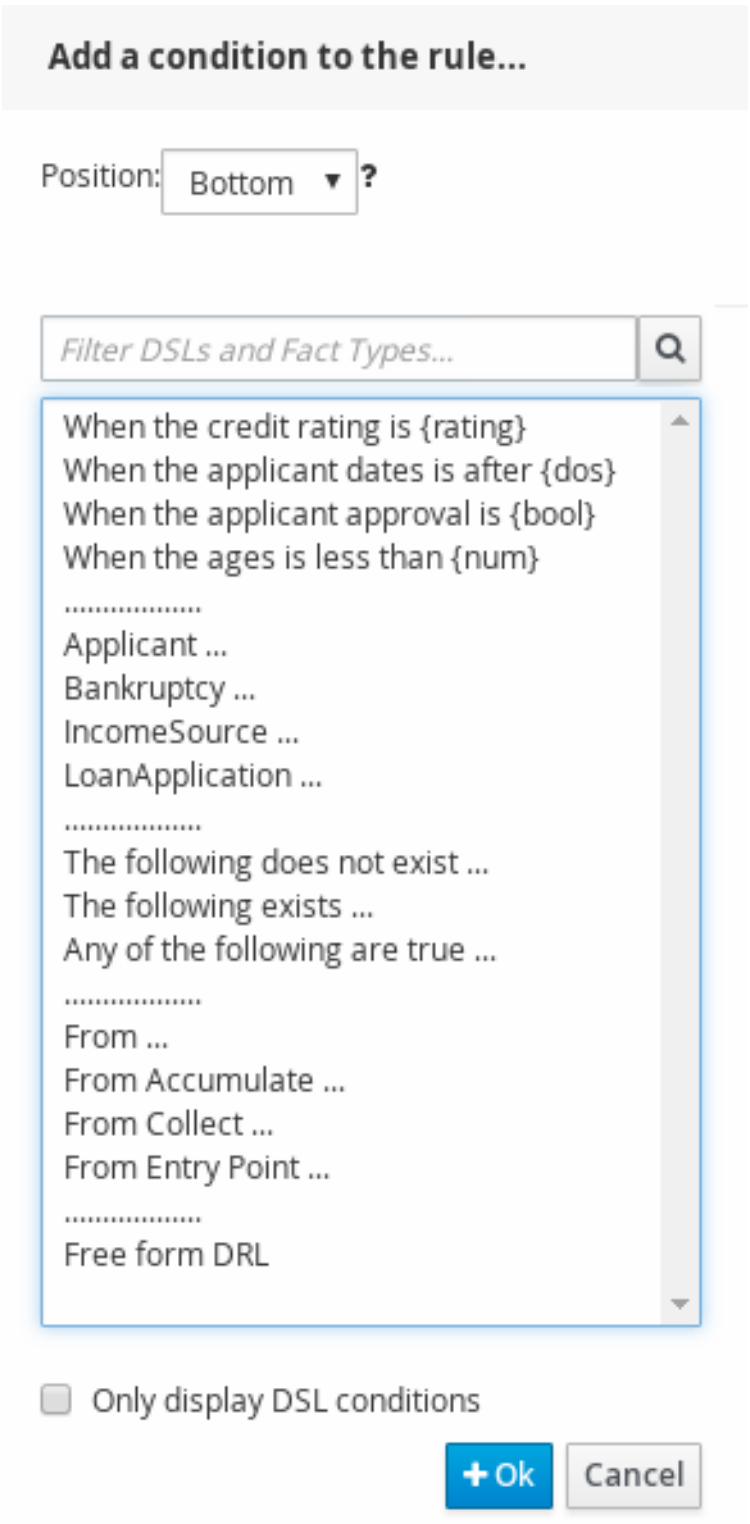
前提条件

- ルールに必要なデータオブジェクトはすべて作成、またはインポートされており、ガイド付きルールデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールデザイナーで、**WHEN** セクションの右側のプラスアイコン () をクリックします。
利用可能な条件要素が追加された **Add a condition to the rule**ウィンドウが開きます。

図52.2 ルールへの条件の追加



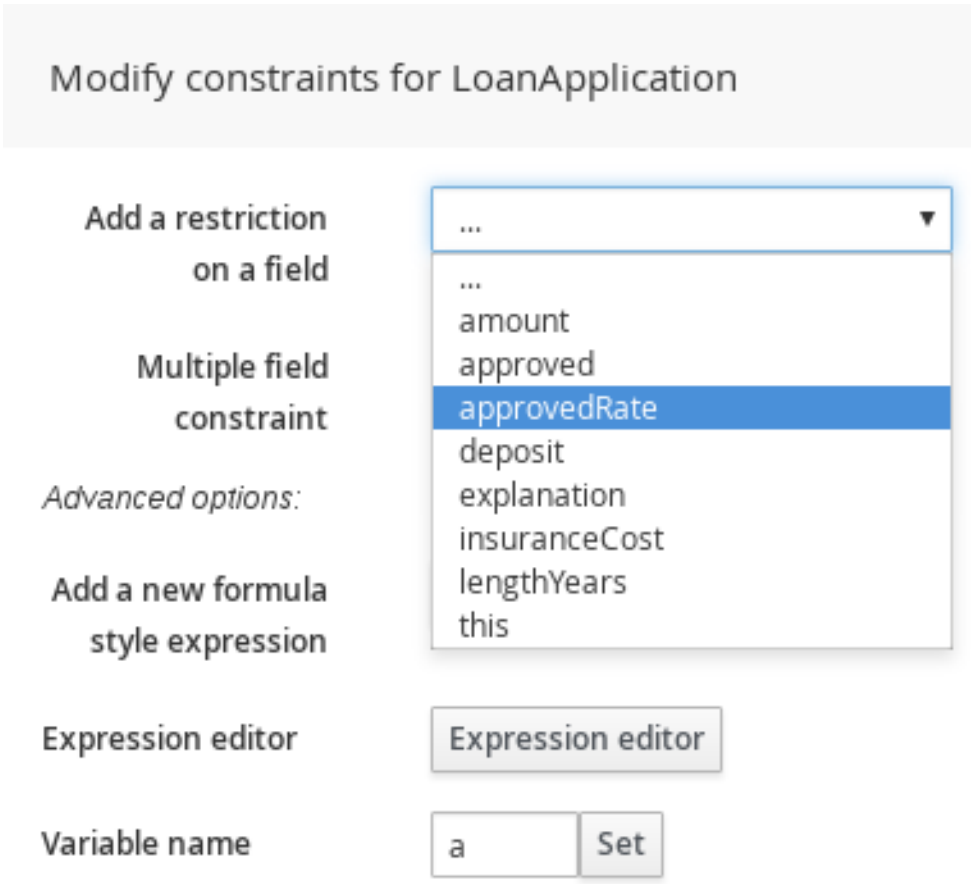
このリストには、ガイド付きルールデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクト (このガイド付きルールを作成したときに **Show declared DSL sentences** を選択した場合) と、以下の標準オプションが含まれます。

- **The following does not exist:** 存在すべきでないファクトと制約を指定します。
- **The following exists:** 存在すべきファクトと制約を指定します。このオプションは、最初に一致したものだけが適用され、その後一致するものは無視されます。
- **Any of the following are true:** true であるファクトと制約をリストします。

- From: ルールの **From** 条件要素を定義します。
- From Accumulate: ルールの **Accumulate** 条件要素を定義します。
- From Collect: ルールの **Collect** 条件要素を定義します。
- From Entry Point: パターンの **Entry Point** を定義します。
- **Free form DRL:** ガイド付きルールデザイナーを使用せずに条件要素を自由に定義できる free form DRL フィールドを挿入します。

2. 条件要素 (**LoanApplication** など) を選択し、**OK** をクリックします。
3. ガイド付きルールデザイナーで条件要素をクリックし、**Modify constraints for LoanApplication** ウィンドウで、フィールドへの制限の追加、複数のフィールド制約の適用、新しい数式表現の追加、式エディターの適用、または変数名の設定を行います。

図52.3 条件の変更



注記

変数名を使用すると、ガイド付きルールの別の設定でファクトまたはフィールドを指定できます。たとえば、**LoanApplication** の変数を **a** とし、倒産の根拠になっている申し込みを指定する **Bankruptcy** 制約で **a** を参照します。

```
a : LoanApplication()
Bankruptcy( application == a ).
```

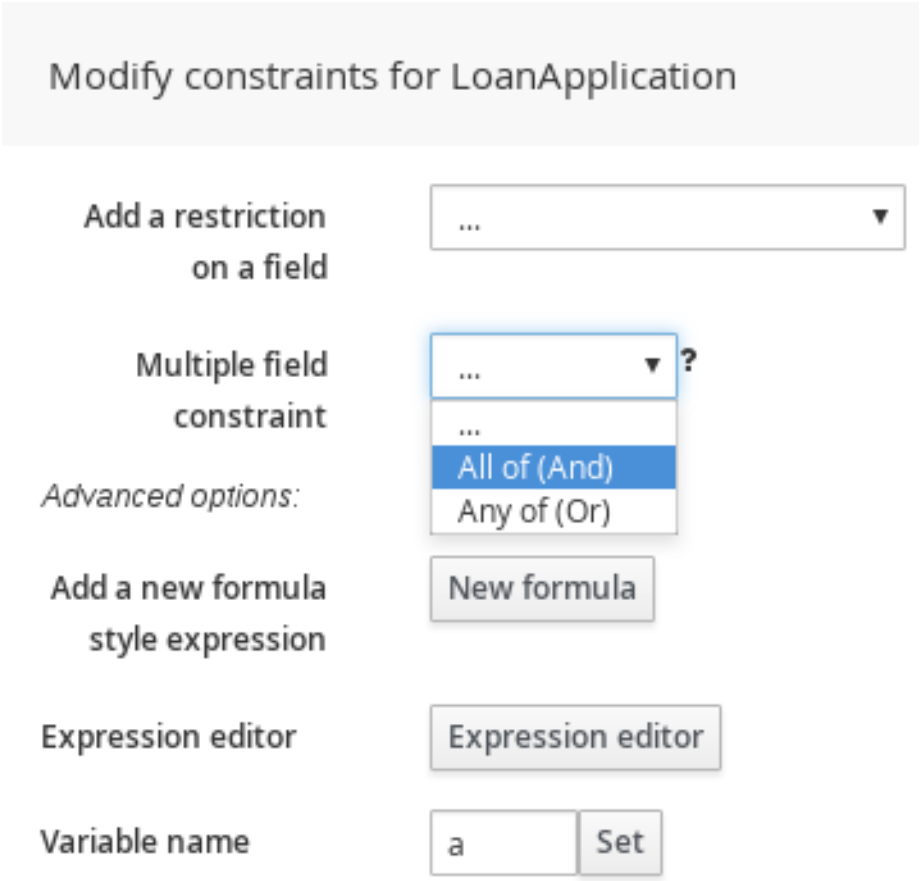
制約を選択したら、ウィンドウが自動的に閉じます。

4. 追加した制約の隣にあるドロップダウンメニューから、制限の演算子 (**greater than** など) を選択します。

5. 編集アイコン ( ) をクリックして、フィールド値を定義します。フィールド値はリテラル値、式、または完全な MVEL 表現にすることができます。

6. フィールド制約を複数適用するには、条件をクリックし、**Modify constraints for LoanApplication** ウィンドウで、**Multiple field constraint** ドロップダウンメニューから **All of(And)** または **Any of(Or)** を選択します。

図52.4 複数のフィールド制約の追加



7. ガイド付きルールデザイナーで制約をクリックして、フィールド値をさらに定義します。

8. ルールの条件コンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ガイド付きルール条件の妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。

9. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

52.2. ガイド付きルールに THEN アクションの追加

ルールの **WHEN** 条件が一致した場合に実行するアクションがルールの **THEN** 部分に含まれます。たとえば、ローンの申込者が 21 歳に満たない場合は、**THEN** アクションにより **approved** が **false** になり、年齢が達していないためローンの申し込みが承認されません。ルールをいつ、どのように適用するかを決定するために、単純または複雑な条件を設定できます。

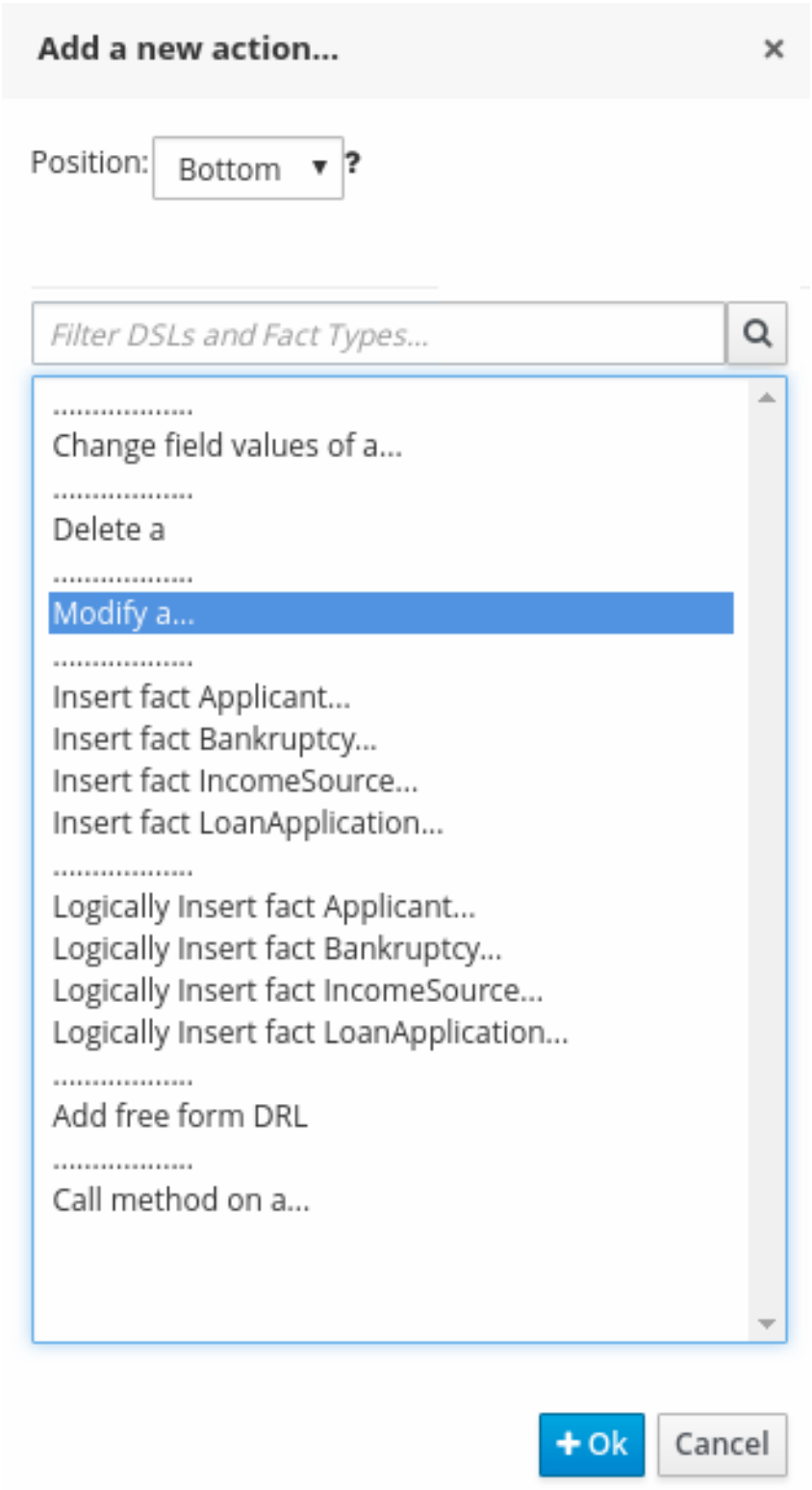
前提条件

- ルールに必要なデータオブジェクトはすべて作成、またはインポートされており、ガイド付きルールデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールデザイナーで、**THEN** セクションの右側のプラスアイコン ( ) をクリックします。
利用可能なアクション要素が追加された **Add a new action** ウィンドウが開きます。

図52.5 ルールへのアクションの追加

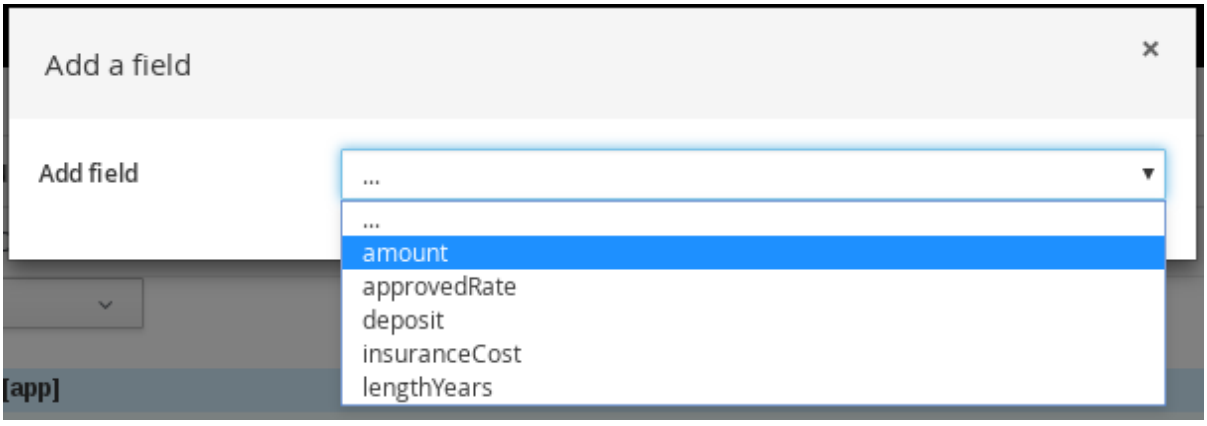


このリストには、ガイド付きルールデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクト (ガイド付きルールを作成したときに **Show declared DSL sentences** を選択した場合) に基づいた挿入と修正のオプションが含まれます。

- **Change field values of:** (**LoanApplication** などの) ファクトにフィールドの値を設定します。この変更はデシジョンエンジンには通知されません。
- **Delete:** ファクトを削除します。
- **Modify:** ファクトに対して修正するフィールドを指定します。この変更はデシジョンエンジンには通知されません。
- **Insert fact:** ファクトを挿入し、ファクトの結果フィールドと値を定義します。
- **Logically Insert fact (ファクトの論理的な挿入):** ファクトをデシジョンエンジンに論理的に挿入し、ファクトに対してフィールドと値を定義します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくなると、ファクトは自動的に取り消されます。
- **Add free form DRL:** ガイド付きルールデザイナーを使用せずに条件要素を自由に定義できる free form DRL フィールドを挿入します。
- **Call method on:** 別のファクトからメソッドを呼び出します。

2. アクション要素 (**Modify** など) を選択し、**OK** をクリックします。
3. ガイド付きルールデザイナーでアクション要素をクリックし、**Add a field** ウィンドウを使用してフィールドを選択します。

図52.6 フィールドの追加



フィールドを選択したら、ウィンドウが自動的に閉じます。

4. 編集アイコン ( ) をクリックして、フィールド値を定義します。このフィールド値は、リテラル値または式にすることができます。
5. ルールのアクションコンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ルールのアクションの妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールの全コンポーネントを見直し、エラーが表示されなくなるまでルールの妥当性確認を行います。
6. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

52.3. ルールアセットのドロップダウンリストの列挙定義

Business Central での列挙定義では、ガイド付きルール、ガイド付きルールテンプレート、ガイド付きデシジョンテーブルの条件やアクションのフィールドで使用可能な値を指定します。列挙定義には、ルールアセットで該当するフィールドのドロップダウンリストとして表示される対応値一覧に対する **fact.field** マッピングが含まれています。列挙定義と同じファクトとフィールドをベースにしたフィールドを選択すると、定義した値のドロップダウンリストが表示されます。

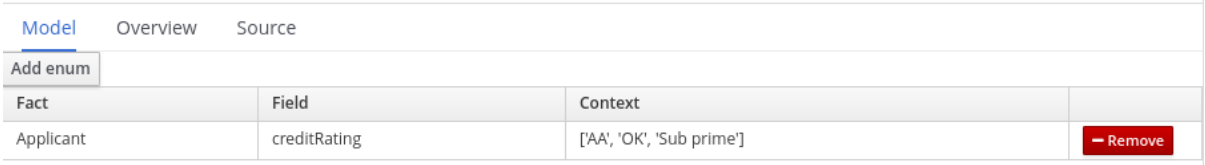
列挙は、Business Central または Red Hat Process Automation Manager プロジェクトの DRL ソースで定義できます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Enumerataion** をクリックします。
3. 分かりやすい **Enumeration** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトと適切なルールアセットが割り当てられているか、これから割り当てるパッケージと同じでなければなりません。
4. **Ok** をクリックして列挙を作成します。
Project Explorer の **Enumeration Definitions** パネルに、新しい列挙が追加されました。
5. 列挙デザイナーの **Model** タブで、**Add enum** をクリックし、以下の列挙値を定義します。
   - **Fact**: この列挙を関連付けるプロジェクトの同じパッケージ内に、既存のデータオブジェクトを指定します。**Project Explorer** で **Data Objects** パネルを開き、利用可能なデータオブジェクトを表示するか、必要に応じて新規アセットとして適切なデータオブジェクトを作成します。
   - **Field**: **Fact** 用に選択したデータオブジェクトの一部として定義した既存のフィールド ID を指定します。**Project Explorer** で **Data Objects** パネルを開き、適切なデータオブジェクトを選択して、利用可能な **Identifier** オプションの一覧を表示します。必要に応じて、データオブジェクトに関連する ID を作成してください。
   - **Context**: **Fact** と **Field** の定義にマッピングする **['string1','string2','string3']** 形式または **[integer1,integer2,integer3]** 形式の値一覧を定義します。これらの値は、ルールアセットの適切なフィールドに、ドロップダウンリストとして表示されます。

   たとえば、以下の列挙は、ローン申請デシジョンサービスの申請者でクレジットスコアに使用するドロップダウンの値を定義します。

図52.7 Business Central での申請者のクレジットスコアの列挙例



DRL ソースの申請者のクレジットスコアの列挙例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

以下の例では、プロジェクトと同じパッケージ内にあり、**Applicant** データオブジェクトと **creditRating** フィールドを使用するガイド付きルール、ガイド付きルールテンプレート、またはガイド付きデシジョンテーブルであれば、設定値がドロップダウンオプションとして利用できます。

図52.8 ガイド付きルールまたはガイド付きルールテンプレートでの列挙ドロップダウンオプション例

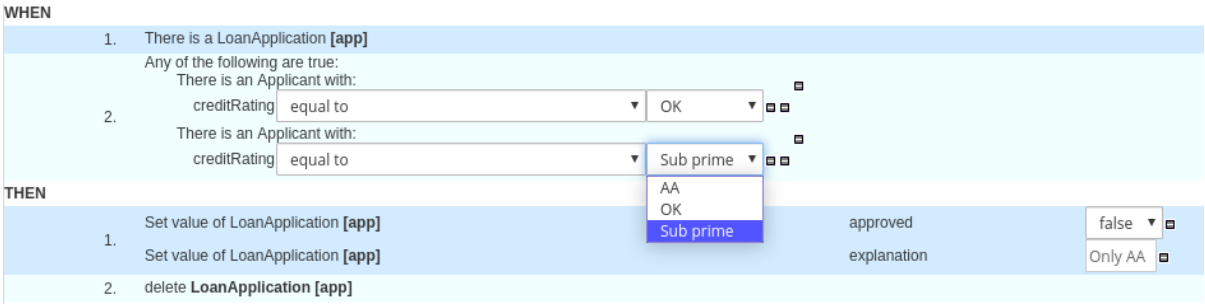
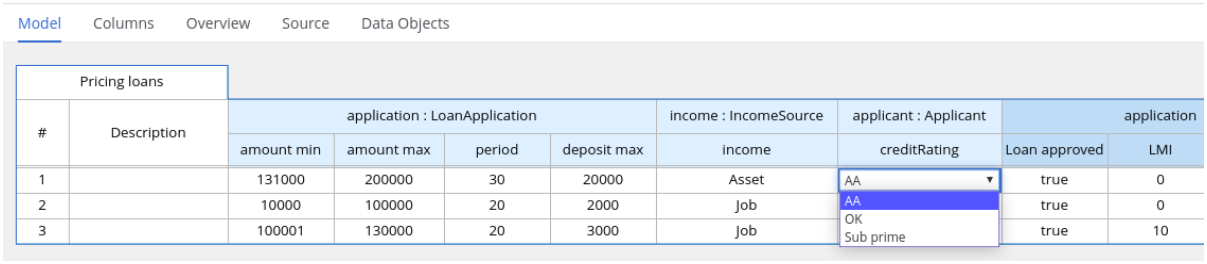


図52.9 ガイド付きデシジョンテーブルでの列挙ドロップダウンオプション例



52.3.1. ルールアセットの詳細列挙オプション

Red Hat Process Automation Manager プロジェクトの列挙定義を使用した詳細ユースケースについては、列挙を定義する時に、以下の拡張オプションの使用を検討してください。

Business Central の値との DRL の値におけるマッピング

列挙値を DRL ソースに表示されるものとは異なる方法で、または完全に Business Central インターフェイスに表示する場合は、列挙定義値の形式 **'fact.field' : ['sourceValue1=UIValue1','sourceValue2=UIValue2', ...]** のマッピングを使用します。
たとえば、ローンの状態に関する以下の列挙定義では、**A** または **D** のオプションを DRL ファイルで使用しますが、Business Central では **Approved** または **Declined** のオプションが表示されます。

```
'Loan.status' : ['A=Approved','D=Declined']
```

列挙値の依存関係

選択した値を 1 つのドロップダウンリストにまとめて、後続のドロップダウンリストで利用可能なオプションを判断する場合は、列挙定義で **'fact.fieldB[fieldA=value1]' : ['value2', 'value3', ...]** の形式を使用します。
たとえば、保険契約に関する以下の列挙定義では、**policyType** フィールドに **Home** または **Car** の値を使用できます。ユーザーが選択する保険契約タイプにより、利用できる契約 **coverage** のフィールドオプションが決まります。

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```

注記

列挙依存関係は、ルールの条件およびアクションをまたいで適用されません。たとえば、保険契約のユースケースでは、ルール条件で選択した契約をもとに、ルールアクションで利用可能な補償オプションが決定されるわけではありません (該当する場合)。

列挙の外部データソース

列挙定義で直接、値を定義するのではなく、外部のデータソースから列挙値の一覧を取得する場合は、プロジェクトのクラスパスで、文字列の **java.util.List** リストを返すヘルパークラスを追加します。列挙定義で、値を指定する代わりに、外部の値を取得するように設定したヘルパークラスを特定します。
たとえば、ローン申請者の地域に関する以下の列挙定義では、**'Applicant.region' : ['country1', 'country2', ...]** の形式で明示的に申請者の地域を定義するのではなく、外部で定義した値の一覧を返すヘルパークラスを使用します。

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

この例では、**DataHelper** クラスに、文字列の一覧を返す **getListOfRegions()** メソッドが含まれます。列挙は、ルールアセットの関連フィールドのドロップダウンリストに、読み込まれます。

また、通常通り従属フィールドを特定して、ヘルパーへの呼び出しを引用符で囲むことで、ヘルパークラスから動的に、従属の列挙定義を読み込むこともできます。

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

リレーショナルデータベースなど、外部データソースから列挙データをすべて読み込む場合は、**Map<String, List<String>>** マッピングを返す Java クラスを実装できます。マップのキーは **fact.field** マッピングです。この値は、値の **java.util.List<String>** リストになります。

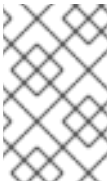
たとえば、以下の Java クラスでは、関連する列挙のローン申請者の地域を定義します。

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下の列挙定義は、この Java クラスの例に相関します。参照は Java クラスで定義されているため、列挙にはファクトまたはフィールド名への参照が含まれません。

```
=(new SampleDataSource()).loadData()
```

= 演算子を使用して、Business Central がヘルパークラスから全列挙データを読み込めるようにします。エディターで使用するように列挙定義を要求すると、ヘルパーメソッドが静的に評価されます。



注記

現在、Business Central では、ファクトおよびフィールド定義なしで列挙を定義することはできません。この方法で関連の Java クラスの列挙を定義するには、Red Hat Process Automation Manager プロジェクトで DRL ソースを使用します。

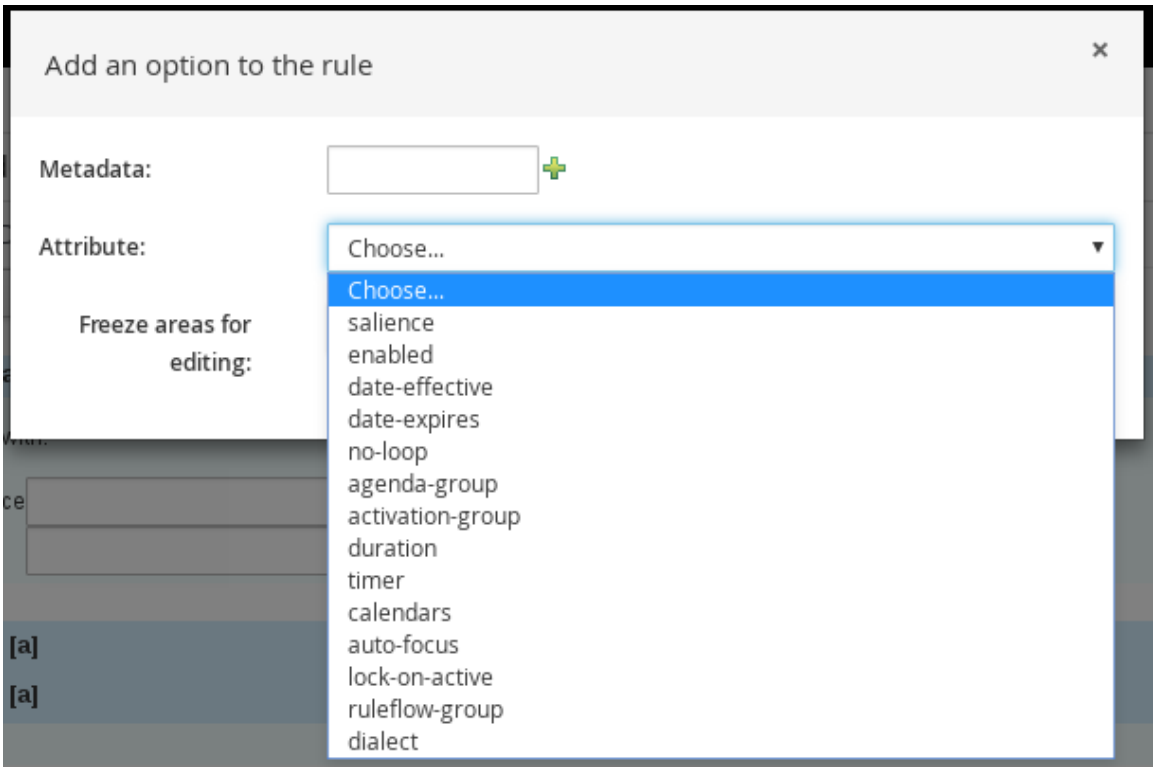
52.4. その他のルールオプションの追加

ルールデザイナーを使用してルールにメタデータを追加し、追加のルール属性 (**salience**、**no-loop** など) を定義し、条件またはアクションの変更を制限するために、ルールの領域を凍結します。

手順

1. ルールデザイナーの **THEN** セクションの下にある **(show options...)** をクリックします。
2. ウィンドウの右側にあるプラスアイコン ( ) をクリックして、オプションを追加します。
3. ルールに追加するオプションを選択します。
   - **Metadata:** メタデータのラベルを入力し、プラスアイコン ( ) をクリックします。次に、ルールデザイナーに提供されるフィールドに必要なデータを入力します。
   - **Attribute:** ルール属性のリストから選択します。次に、ルールデザイナーに表示されるフィールドまたはオプションに値を定義します。
   - **Freeze areas for editing (編集する領域を制限):**ルールデザイナーで修正する領域を制限する **条件** または **アクション** を選択します。

図52.10 ルールオプション



4. ルールデザイナーで **Save** をクリックして、設定した内容を保存します。

52.4.1. ルールの属性

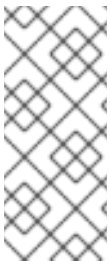
ルール属性は、ルールの動作を修正するビジネスルールを指定する追加設定です。

次の表では、ルールに割り当て可能な属性の名前と、対応する値を紹介します。

表52.1 ルールの属性

属性	値
salience	ルールの優先順位を定義する整数。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。 例: **salience 10**
enabled	ブール値。このオプションを選択すると、ルールが有効になります。このオプションを選択しないと、ルールは無効になります。 例: **enabled true**
date-effective	日付定義および時間定義を含む文字列。現在の日時が **date-effective** 属性よりも後の場合は、このルールがアクティブになります。 例: **date-effective "4-Sep-2018"**

属性	値
date-expires	日付定義および時間定義を含む文字列。現在日時が **date-expires** 属性よりも後になると、このルールをアクティブにすることはできません。 例: **date-expires "4-Oct-2018"**
no-loop	ブール値。このオプションが選択される場合は、ルールの結果が以前一致した条件を再度トリガーすると、ルールは再アクティブ化 (ループ) されません。条件を選択しないと、この状況でルールがループされます。 例: **no-loop true**
agenda-group	ルールを割り当てるアジェンダグループを指定する文字列。アジェンダグループを使用すると、アジェンダをパーティションで区切り、ルールのグループに対する実行をさらに制御できます。フォーカスを取得したアジェンダグループのルールだけがアクティブになります。 例: **agenda-group "GroupName"**
activation-group	ルールを割り当てるアクティベーション (または XOR) グループを指定する文字列。アクティベーショングループでは、1 つのルールのみをアクティブ化できます。最初のルールが実行すると、アクティベーショングループの中で、アクティベーションが保留中のルールはすべてキャンセルされます。 例: **activation-group "GroupName"**
duration	ルールの条件が一致している場合に、ルールがアクティブになってからの時間をミリ秒で定義する長整数値。 例: **duration 10000**
timer	ルールのスケジュールに対する **int** (間隔) または **cron** タイマー定義を指定する文字列。 例: **timer (cron:* 0/15 * * * ?)** (15 分ごと)
calendar	ルールのスケジュールを指定する Quartz カレンダーの定義。 例: **calendars "* * 0-7,18-23 ?* *"** (営業時間外を除く)
auto-focus	アジェンダグループ内のルールにのみ適用可能なブール値。このオプションが選択されている場合は、次にルールがアクティブになった場合に、そのルールが割り当てられたアジェンダグループにフォーカスが自動的に指定されます。 例: **auto-focus true**

属性	値
lock-on-active	ルールフローグループまたはアジェンダグループ内のルールにのみ適用可能なブール値。このオプションを選択すると、次回、ルールのルールフローグループがアクティブになるか、ルールのアジェンダグループがフォーカスを受け取ると、(ルールフローグループがアクティブでなくなるか、アジェンダグループがフォーカスを失うまで) ルールをアクティブにすることができません。これは、**no-loop** 属性を強力にしたものです。なぜなら、一致するルールのアクティベーションが、(ルールそのものによるものだけでなく) 更新元にかかわらず破棄されるためです。この属性は、ファクトを修正するルールが多数あり、ルールの再一致と再発行を希望しない計算ルールに適しています。 例: **lock-on-active true**
ruleflow-group	ルールフローグループを指定する文字列。ルールフローグループで、関連するルールフローによってそのグループがアクティブになった場合に限りルールを発行できます。 例: **ruleflow-group "GroupName"**
dialect	ルールのコード表記に使用される言語を指定する文字列 (**JAVA** または **MVEL**)。デフォルトでは、ルールは、パッケージレベルに指定されている方言を使用します。ここで指定した方言は、ルールのパッケージ方言設定を上書きします。 例: **dialect "JAVA"**  **注記** 実行可能モデルなしで Red Hat Process Automation Manager を使用する場合には、**dialect "JAVA"** ルールの結果は、Java 5 構文のみをサポートします。実行可能モデルに関する詳細は、Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ を参照してください。

第53章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントのオプションに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合は、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれていることを確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

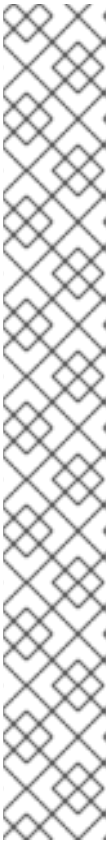
3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、Business Central 外に Maven または Java プロジェクトが作成されていない場合は作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例は、その他の DRL ルールの作成および実行方法 を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。

- **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルに読み込みます。
- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (任意) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける Red Hat Process Automation Manager 7.11 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。



注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

5. モデルクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
   Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイルの両方に表示されます。

   ```
   <dependency>
     <groupId>com.sample</groupId>
     <artifactId>Person</artifactId>
     <version>1.0.0</version>
   </dependency>
   ```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

   ```
   org.slf4j.simpleLogger.defaultLogLevel=debug
   ```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースを読み込む **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
   たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
```

```
    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

KIE Server でのルールの実行

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);
```

```
        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert Person into the session:
        KieCommands kieCommands = KieServices.Factory.get().getCommands();
        List<Command> commandList = new ArrayList<Command>();
        commandList.add(kieCommands.newInsert(p, "personReturnId"));

        // Fire all rules:
        commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
        BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

        // Use rule services client to send request:
        RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
        ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
        System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
}
```

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリーにおける) Java の実行例

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

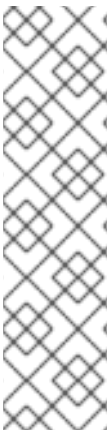
9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

第54章 次のステップ

- テストシナリオを使用したデシジョンサービスのテスト
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート VII. ガイド付きルールテンプレートを使用したデシジョンサービスの作成

ビジネス分析者またはビジネスルールの開発者は、Business Central でガイド付きルールテンプレートデザイナーを使用してビジネスルールテンプレートを定義できます。このガイド付きルールテンプレートは Drools Rule Language (DRL) に組み込まれ、プロジェクトのデシジョンサービスの中心となります。



注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.11 の DMN サポートに関する詳細は、以下の資料を参照してください。

- **デシジョンサービスのスタートガイド**(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- **DMN モデルを使用したデシジョンサービスの作成**(Red Hat Process Automation Manager における DMN のサポートおよび機能の概要)

前提条件

- ガイド付きルールテンプレートの領域およびプロジェクトが Business Central に作成されている。各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は **デシジョンサービスのスタートガイド** を参照してください。

第55章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、目的やニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表55.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメント
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 一部またはすべての意思決定要件グラフ (DRG: Decision Requirements Graph) を表すグラフィカルな意思決定要件ダイアグラム (DRD: Decision Requirements Diagram) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式表現 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性、および安定性のある意思決定フローの作成に最適である	Business Central または DMN 準拠のエディター	**DMN モデルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブルである • デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていないヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適である	Business Central	**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適である • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	**スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成**

アセット	主な特徴	オーサリングツール	ドキュメント
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルールである • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適である	Business Central	ガイド付きルールを使用したデシジョンサービスの作成
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造である • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの目的の基本となる) ルールテンプレートを作成するテンプレートのキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適である	Business Central	ガイド付きルールテンプレートを使用したデシジョンサービスの作成
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルールである • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細な DRL オプションを必要とするルールを作成するのに最適である • ルールを適切にコンパイルするための厳密な構文要件がある	Business Central または統合開発環境 (IDE)	DRL ルールを使用したデシジョンサービスの作成

アセット	主な特徴	オーサリングツール	ドキュメント
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	PMML モデルでのデシジョンサービスの作成

第56章 ガイド付きルールテンプレート

ガイド付きルールテンプレートは、別のデータテーブルに定義した実際の値と置き換えるプレースホルダー値 (テンプレートキー) を使用したビジネスルール構造です。そのテンプレートに対応するデータテーブルに定義する各行の結果がルールになります。ガイド付きルールテンプレートは、多くのルールで条件、アクションなどの属性が同じで、ファクトまたは制約の値が異なる場合に適しています。このような場合は、同じようなガイド付きルールを多数作成し、各ルールに値を定義する代わりに、各ルールに適用できるルール構造を持つガイド付きルールテンプレートを作成し、データテーブルに値だけを定義します。

ガイド付きルールテンプレートデザイナーは、ルールテンプレートを定義したデータオブジェクトと、テンプレートキー値を追加するデータテーブルに基づいて適切なテンプレート入力を行うフィールドとオプションを提供します。ガイド付きルールテンプレートを作成し、対応するデータテーブルに値を追加すると、定義したルールが、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。

ガイド付きルールテンプレートに関連するすべてのデータオブジェクトは、ガイド付きルールテンプレートと同じプロジェクトパッケージに置く必要があります。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとガイド付きルールテンプレートを作成したら、ガイド付きルールテンプレートデザイナーの **Data Objects** タブから、必要なデータオブジェクトがすべてリストされていることを検証したり、**新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

第57章 データオブジェクト

データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

57.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

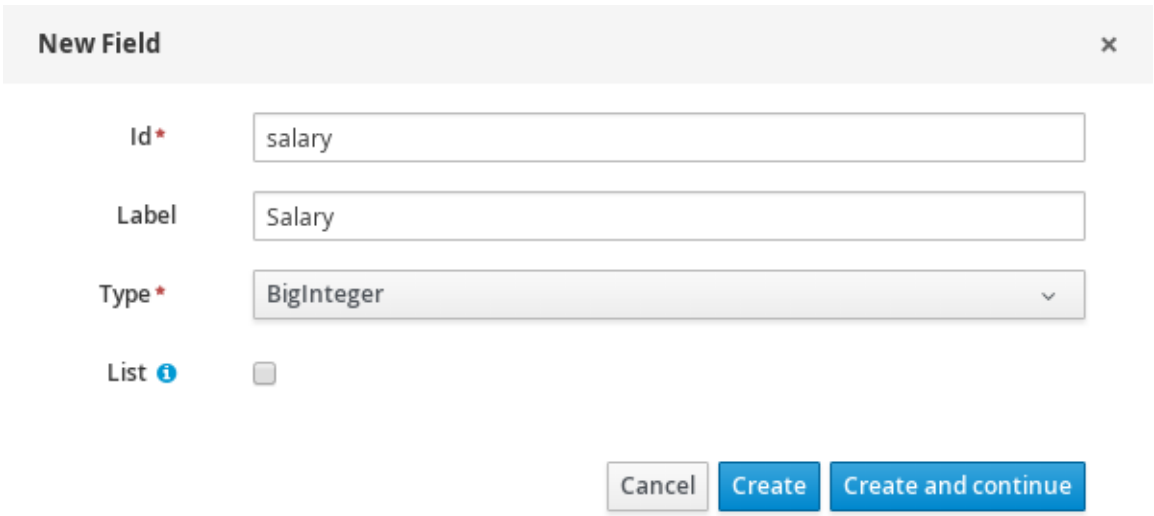
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図57.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

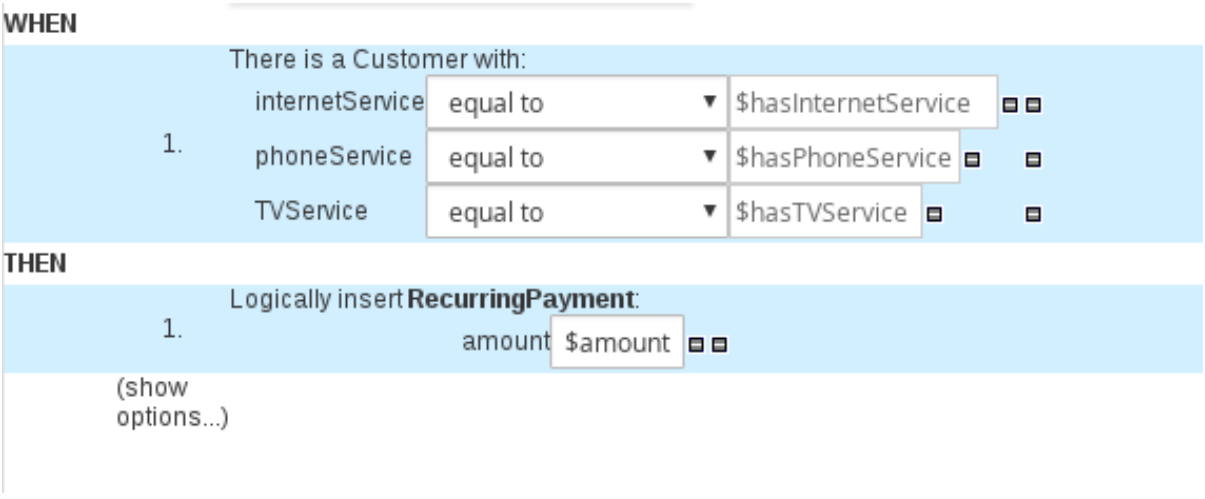
第58章 ガイド付きルールテンプレートの作成

ガイド付きルールテンプレートを使用して、データテーブルに定義した値に対応するプレースホルダー値 (テンプレートキー) を持つルール構造を定義できます。ガイド付きルールテンプレートは、構造が同じ多数のガイド付きルールセットを個別に定義するのに利用できる効率的な方法です。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Rule Template** をクリックします。
3. 参考となる **ガイド付きルールテンプレート** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
4. **OK** をクリックして、ルールテンプレートを作成します。
新しいガイド付きルールテンプレートが、**Project Explorer** の **Guided Rule Templates** パネルに追加されます。
5. **Data Objects** タブをクリックして、ルールに必要なデータオブジェクトがすべてリストされていることを確認します。リストされていない場合は、**New item** をクリックして、他のパッケージからデータオブジェクトをインポートするか、パッケージに データオブジェクトを作成 します。
6. データオブジェクトをすべて配置したら、**Model** タブに戻り、ウィンドウの右側のボタンから、利用可能なデータオブジェクトに、ルールテンプレートの **WHEN** (条件) セクションおよび **THEN** (アクション) セクションを追加して定義します。ルールごとに変更するフィールド値については、ルールデザイナーで **$key** 形式、または (DRL を使用している場合は) free form DRL の **@{key}** 形式のテンプレートキーを使用します。

図58.1 ガイド付きルールテンプレートの例



テンプレートキーに関する注記

テンプレートキーは、ガイド付きルールテンプレートの基本となるものです。テンプレートキーは、同じテンプレートから異なるルールを生成するために、対応するデータテーブルで定義する実際の値と入れ替えるテンプレートのフィールド値を有効にするものです。**リテラル** や **式** などの値を使用することもできますが、その値は、そのテンプレートに基づいたすべてのルールのルール構造の一部となります。ただし、ルール間で異なる値については、特定のキーを使用した **テンプレートキー** フィールドタイプを使用します。ガイド付きルールテンプレートでテンプレートキーを使用しない場合、対応するデータテーブルはテンプレートデザイナーに生成されず、テンプレートは、個々のガイド付きルールとして機能します。

ルールテンプレートの **WHEN** 部分は、アクションを実行するのに必要な条件が含まれます。たとえば、電話会社が、顧客が契約したサービス内容 (インターネット、電話、およびテレビ番組) に基づいて利用代金を請求する場合、**WHEN** 条件の 1 つは **internetService | equal to | $hasInternetService** となります。テンプレートキー **$hasInternetService** は、そのテンプレートのデータテーブルに定義した実際のブール値 (**true** または **false**) に置き換えられます。

ルールテンプレートの **THEN** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、顧客がインターネットサービスだけを契約した場合は、**RecurringPayment** の **$amount** テンプレートキーを使用した **THEN** アクションに、データテーブルのインターネットサービス料金に定義した整数値が実際の月額に設定されます。

7. ルールのコンポーネントをすべて定義したら、ガイド付きルールテンプレートデザイナーで **Save** をクリックして、設定した内容を保存します。

58.1. ガイド付きルールテンプレートへの WHEN 条件の追加

ルールの **WHEN** 部分は、アクションを実行するのに必要な条件が含まれます。たとえば、電話会社が、顧客が契約したサービス内容 (インターネット、電話、およびテレビ番組) に基づいて利用代金を請求する場合、**WHEN** 条件の 1 つは **internetService | equal to | $hasInternetService** となります。テンプレートキー **$hasInternetService** は、そのテンプレートのデータテーブルに定義した実際のブール値 (**true** または **false**) に置き換えられます。

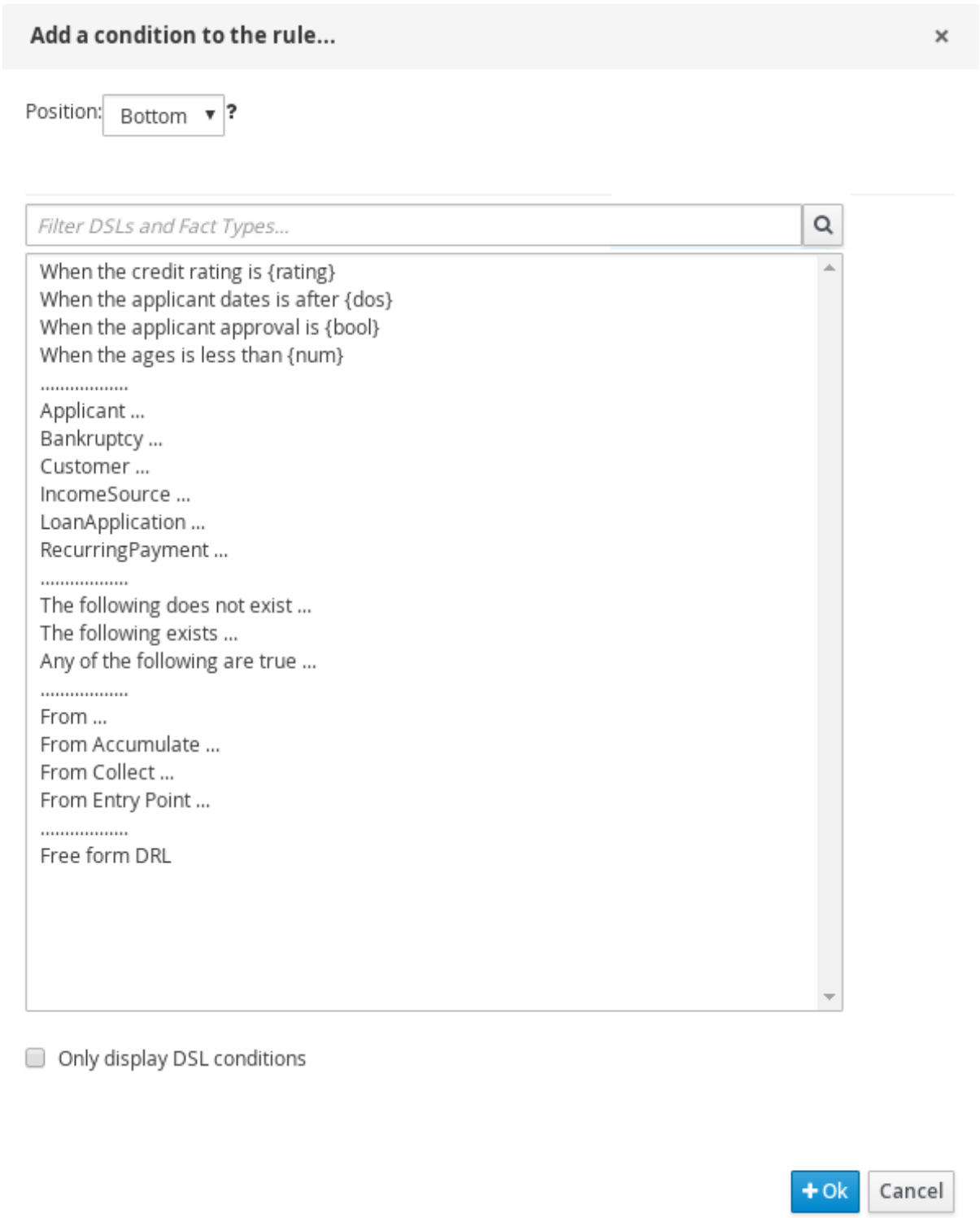
前提条件

- ルールに必要なデータオブジェクトがすべて作成、またはインポートされていて、ガイド付きルールテンプレートデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールテンプレートデザイナーで、**WHEN** セクションの右側のプラスアイコン ( ) をクリックします。
利用可能な条件要素が追加された **Add a condition to the rule**ウィンドウが開きます。

図58.2 ルールへの条件の追加



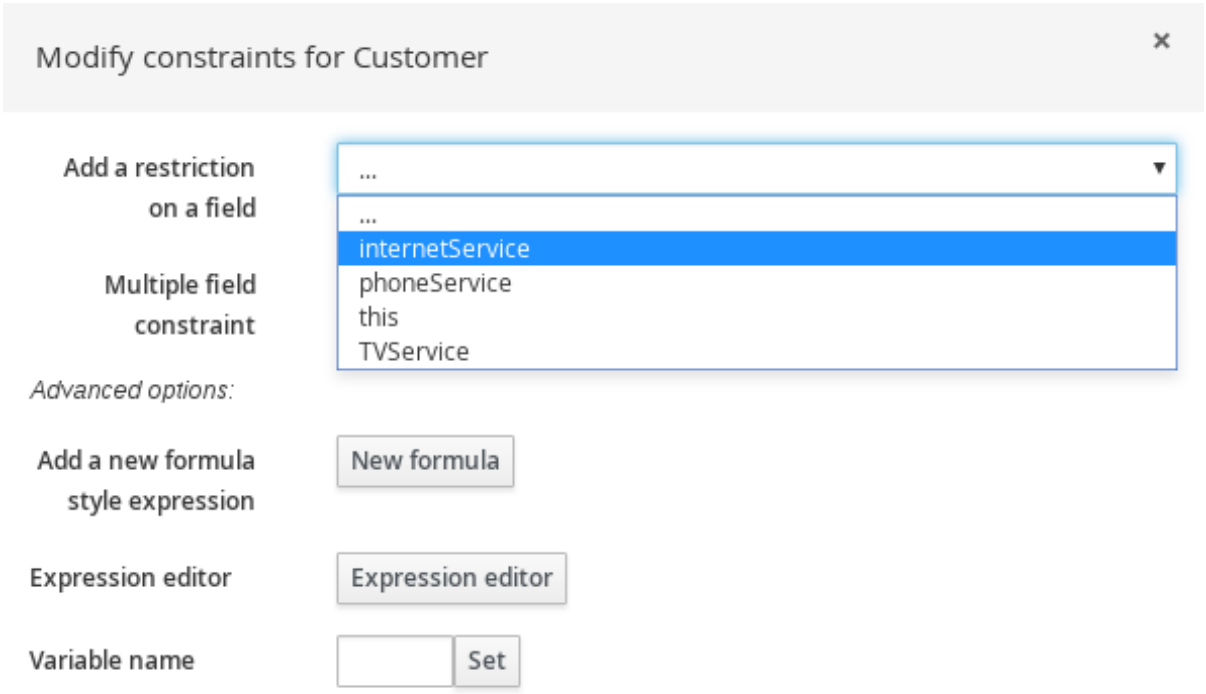
このリストには、ガイド付きルールテンプレートデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクトと、以下の標準オプションが含まれます。

- **The following does not exist:** 存在すべきでないファクトと制約を指定します。
- **The following exists:** 存在すべきファクトと制約を指定します。このオプションは、最初に一致したものだけが適用され、その後一致するものは無視されます。
- **Any of the following are true:** true であるファクトと制約をリストします。
- **From:** ルールの **From** 条件要素を定義します。

- **From Accumulate:** ルールの **Accumulate** 条件要素を定義します。
- **From Collect:** ルールの **Collect** 条件要素を定義します。
- **From Entry Point:** パターンの **Entry Point** を定義します。
- **Free form DRL:** ガイド付きルールデザイナーを使用せずに条件要素を自由に定義できる free form DRL フィールドを挿入します。free form DRL のテンプレートキーには、**@{key}** 形式を使用します。

2. 条件要素 (**Customer** など) を選択し、**OK** をクリックします。
3. ガイド付きルールテンプレートデザイナーで条件要素をクリックし、**Modify constraints for Customer** ウィンドウで、フィールドへの制限の追加、複数のフィールド制約の適用、新しい数式表現の追加、式エディターの適用、または変数名の設定を行います。

図58.3 条件の変更



注記

変数名を使用すると、ガイド付きルールの別の設定でファクトまたはフィールドを指定できます。たとえば、**Customer** の変数を **c** にし、**Customer** が **Applicant** であることを指定する別の **Applicant** 制約で、**c** を参照します。

```
c : Customer()
Applicant( this == c )
```

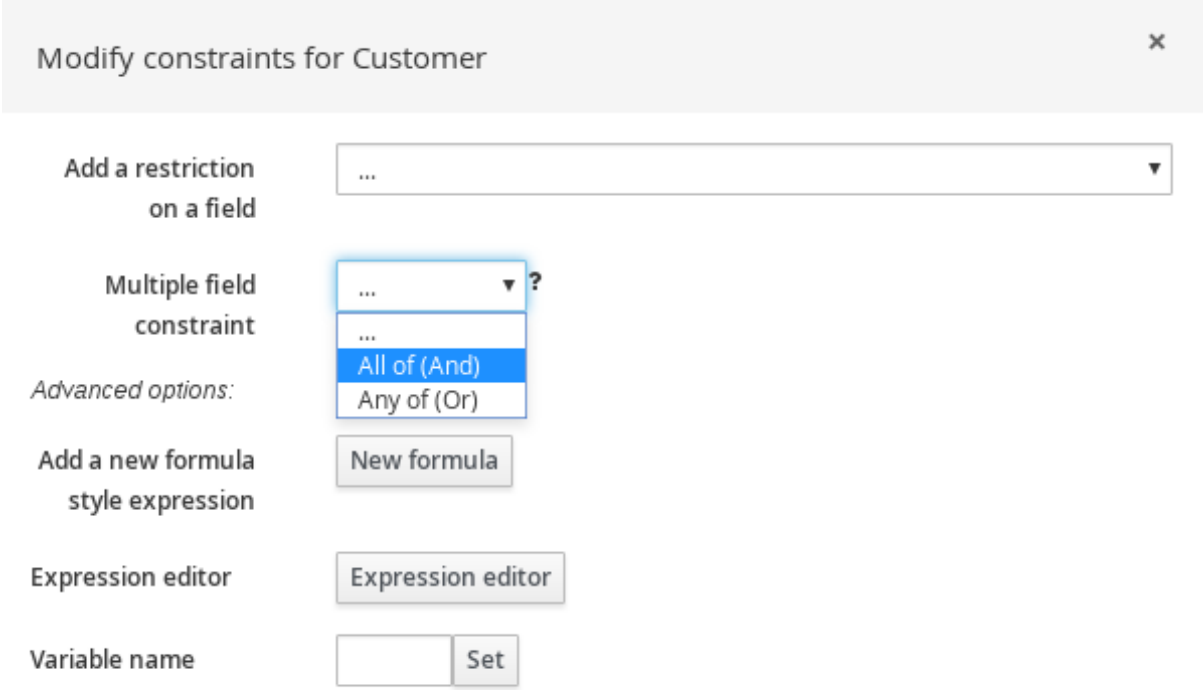
制約を選択したら、ウィンドウが自動的に閉じます。

4. 追加した制約の隣にあるドロップダウンメニューから、制限の演算子 (**greater than** など) を選択します。
5. 編集アイコン () をクリックして、フィールド値を定義します。

6. このテンプレートに基づいたルール間で値が異なる場合は、**テンプレートキー** を選択し、**$key** 形式でテンプレートキーを追加します。これにより、フィールド値を、対応するデータテーブルに定義する実際の値と入れ替えて、同じテンプレートから異なるルールを生成します。ルールテンプレートの中で、ルール間で変更しないフィールド値については、別の型の値を使用できます。

7. フィールド制約を複数適用するには、条件をクリックし、**Modify constraints for Customer** ウィンドウで、**Multiple field constraint** ドロップダウンメニューから **All of(And)** または **Any of(Or)** を選択します。

図58.4 複数のフィールド制約の追加



8. ガイド付きルールテンプレートデザイナーで制約をクリックして、フィールド値をさらに定義します。

9. 条件要素をすべて定義したら、ガイド付きルールテンプレートデザイナーで **Save** をクリックして、設定した内容を保存します。

58.2. ガイド付きルールテンプレートへの THEN アクションの追加

ルールテンプレートの **THEN** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、顧客がインターネットサービスだけを契約した場合は、**RecurringPayment** の **$amount** テンプレートキーを使用した **THEN** アクションに、データテーブルのインターネットサービス料金に定義した整数値が実際の月額に設定されます。

前提条件

- ルールに必要なデータオブジェクトがすべて作成、またはインポートされていて、ガイド付きルールテンプレートデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールテンプレートデザイナーで、**THEN** セクションの右側のプラスアイコン ( ) をクリックします。

利用可能なアクション要素が追加された **Add a new action** ウィンドウが開きます。

図58.5 ルールへのアクションの追加

このリストには、ガイド付きルールテンプレートデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクトに基づいた、挿入および修正のオプションが含まれます。

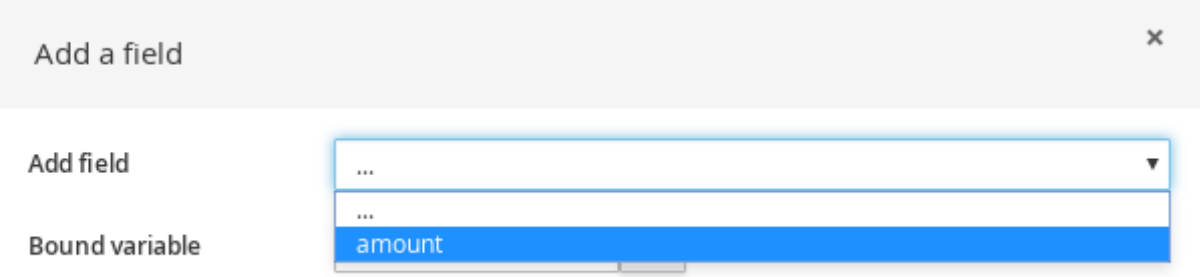
- **Insert fact:** ファクトを挿入し、ファクトの結果フィールドと値を定義します。
- **Logically Insert fact (ファクトの論理的な挿入):** ファクトをデシジョンエンジンに論理的に挿入し、ファクトに対してフィールドと値を定義します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定し

た挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくなると、ファクトは自動的に取り消されます。

- **Add free form DRL:** ガイド付きルールデザイナーを使用せずに条件要素を自由に定義できる free form DRL フィールドを挿入します。free form DRL のテンプレートキーには、**@{key}** 形式を使用します。

2. アクション要素 (**Logically Insert fact RecurringPayment** など) を選択し、**OK** をクリックします。
3. ガイド付きルールテンプレートデザイナーでアクション要素をクリックし、**Add a field** ウィンドウを使用してフィールドを選択します。

図58.6 フィールドの追加



フィールドを選択したら、ウィンドウが自動的に閉じます。

4. 編集アイコン ( ) をクリックして、フィールド値を定義します。
5. このテンプレートに基づいたルール間で値が異なる場合は、**テンプレートキー** を選択し、**$key** 形式でテンプレートキーを追加します。これにより、フィールド値を、対応するデータテーブルに定義する実際の値と入れ替えて、同じテンプレートから異なるルールを生成します。ルールテンプレートの中で、ルール間で変更しないフィールド値については、別の型の値を使用できます。
6. アクション要素をすべて定義したら、ガイド付きルールテンプレートで **Save** をクリックして、設定した内容を保存します。

58.3. ルールアセットのドロップダウンリストの列挙定義

Business Central での列挙定義では、ガイド付きルール、ガイド付きルールテンプレート、ガイド付きデシジョンテーブルの条件やアクションのフィールドで使用可能な値を指定します。列挙定義には、ルールアセットで該当するフィールドのドロップダウンリストとして表示される対応値一覧に対する **fact.field** マッピングが含まれています。列挙定義と同じファクトとフィールドをベースにしたフィールドを選択すると、定義した値のドロップダウンリストが表示されます。

列挙は、Business Central または Red Hat Process Automation Manager プロジェクトの DRL ソースで定義できます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Enumerataion** をクリックします。

3. 分かりやすい Enumeration 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトと適切なルールアセットが割り当てられているか、これから割り当てるパッケージと同じでなければなりません。

4. **Ok** をクリックして列挙を作成します。
Project Explorer の **Enumeration Definitions** パネルに、新しい列挙が追加されました。

5. 列挙デザイナーの **Model** タブで、**Add enum** をクリックし、以下の列挙値を定義します。

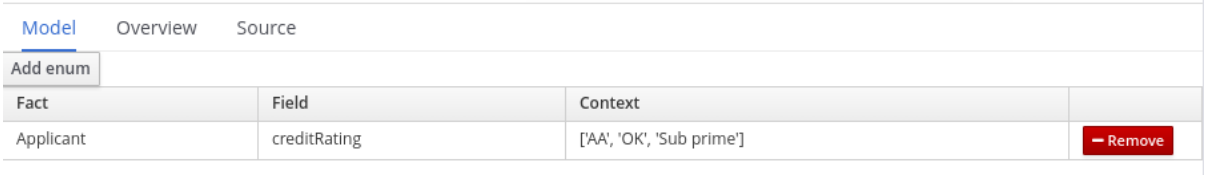
   - **Fact**: この列挙を関連付けるプロジェクトの同じパッケージ内に、既存のデータオブジェクトを指定します。**Project Explorer** で **Data Objects** パネルを開き、利用可能なデータオブジェクトを表示するか、必要に応じて新規アセットとして適切なデータオブジェクトを作成します。

   - **Field**: **Fact** 用に選択したデータオブジェクトの一部として定義した既存のフィールド ID を指定します。**Project Explorer** で **Data Objects** パネルを開き、適切なデータオブジェクトを選択して、利用可能な **Identifier** オプションの一覧を表示します。必要に応じて、データオブジェクトに関連する ID を作成してください。

   - **Context**: **Fact** と **Field** の定義にマッピングする **['string1','string2','string3']** 形式または **[integer1,integer2,integer3]** 形式の値一覧を定義します。これらの値は、ルールアセットの適切なフィールドに、ドロップダウンリストとして表示されます。

たとえば、以下の列挙は、ローン申請デシジョンサービスの申請者でクレジットスコアに使用するドロップダウンの値を定義します。

図58.7 Business Central での申請者のクレジットスコアの列挙例



DRL ソースの申請者のクレジットスコアの列挙例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

以下の例では、プロジェクトと同じパッケージ内にあり、**Applicant** データオブジェクトと **creditRating** フィールドを使用するガイド付きルール、ガイド付きルールテンプレート、またはガイド付きデシジョンテーブルであれば、設定値がドロップダウンオプションとして利用できます。

図58.8 ガイド付きルールまたはガイド付きルールテンプレートでの列挙ドロップダウンオプション例

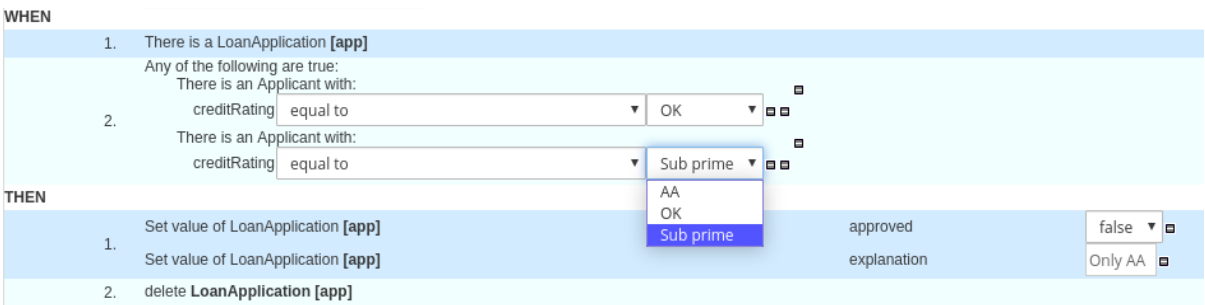
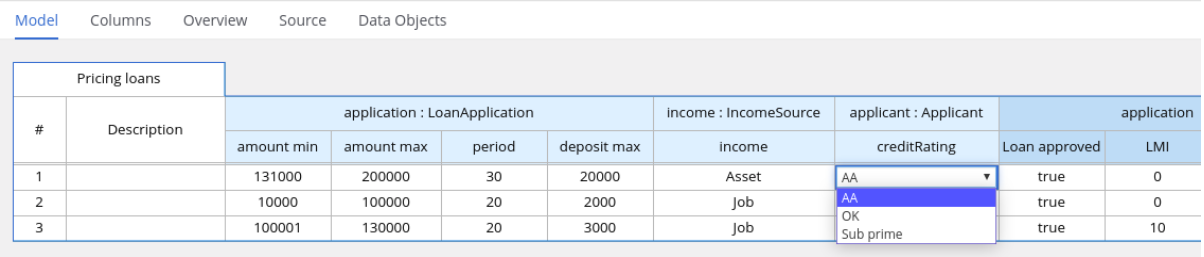


図58.9 ガイド付きデシジョンテーブルでの列挙ドロップダウンオプション例



58.3.1. ルールアセットの詳細列挙オプション

Red Hat Process Automation Manager プロジェクトの列挙定義を使用した詳細ユースケースについては、列挙を定義する時に、以下の拡張オプションの使用を検討してください。

Business Central の値との DRL の値におけるマッピング

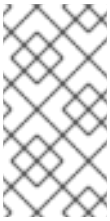
列挙値を DRL ソースに表示されるものとは異なる方法で、または完全に Business Central インターフェイスに表示する場合は、列挙定義値の形式 **'fact.field' : ['sourceValue1=UIValue1','sourceValue2=UIValue2', ...]** のマッピングを使用します。
たとえば、ローンの状態に関する以下の列挙定義では、**A** または **D** のオプションを DRL ファイルで使用しますが、Business Central では **Approved** または **Declined** のオプションが表示されます。

```
'Loan.status' : ['A=Approved','D=Declined']
```

列挙値の依存関係

選択した値を 1 つのドロップダウンリストにまとめて、後続のドロップダウンリストで利用可能なオプションを判断する場合は、列挙定義で **'fact.fieldB[fieldA=value1]' : ['value2', 'value3', ...]** の形式を使用します。
たとえば、保険契約に関する以下の列挙定義では、**policyType** フィールドに **Home** または **Car** の値を使用できます。ユーザーが選択する保険契約タイプにより、利用できる契約 **coverage** のフィールドオプションが決まります。

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注記

列挙依存関係は、ルールの条件およびアクションをまたいで適用されません。たとえば、保険契約のユースケースでは、ルール条件で選択した契約をもとに、ルールアクションで利用可能な補償オプションが決定されるわけではありません (該当する場合)。

列挙の外部データソース

列挙定義で直接、値を定義するのではなく、外部のデータソースから列挙値の一覧を取得する場合は、プロジェクトのクラスパスで、文字列の **java.util.List** リストを返すヘルパークラスを追加します。列挙定義で、値を指定する代わりに、外部の値を取得するように設定したヘルパークラスを特定します。
たとえば、ローン申請者の地域に関する以下の列挙定義では、**'Applicant.region' : ['country1', 'country2', ...]** の形式で明示的に申請者の地域を定義するのではなく、外部で定義した値の一覧を返すヘルパークラスを使用します。

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

この例では、**DataHelper** クラスに、文字列の一覧を返す **getListOfRegions()** メソッドが含まれます。列挙は、ルールアセットの関連フィールドのドロップダウンリストに、読み込まれます。

また、通常通り従属フィールドを特定して、ヘルパーへの呼び出しを引用符で囲むことで、ヘルパークラスから動的に、従属の列挙定義を読み込むこともできます。

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

リレーショナルデータベースなど、外部データソースから列挙データをすべて読み込む場合は、**Map<String, List<String>>** マッピングを返す Java クラスを実装できます。マップのキーは **fact.field** マッピングです。この値は、値の **java.util.List<String>** リストになります。

たとえば、以下の Java クラスでは、関連する列挙のローン申請者の地域を定義します。

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下の列挙定義は、この Java クラスの例に相関します。参照は Java クラスで定義されているため、列挙にはファクトまたはフィールド名への参照が含まれません。

```
=(new SampleDataSource()).loadData()
```

= 演算子を使用して、Business Central がヘルパークラスから全列挙データを読み込めるようにします。エディターで使用するように列挙定義を要求すると、ヘルパーメソッドが静的に評価されます。

注記

現在、Business Central では、ファクトおよびフィールド定義なしで列挙を定義することはできません。この方法で関連の Java クラスの列挙を定義するには、Red Hat Process Automation Manager プロジェクトで DRL ソースを使用します。

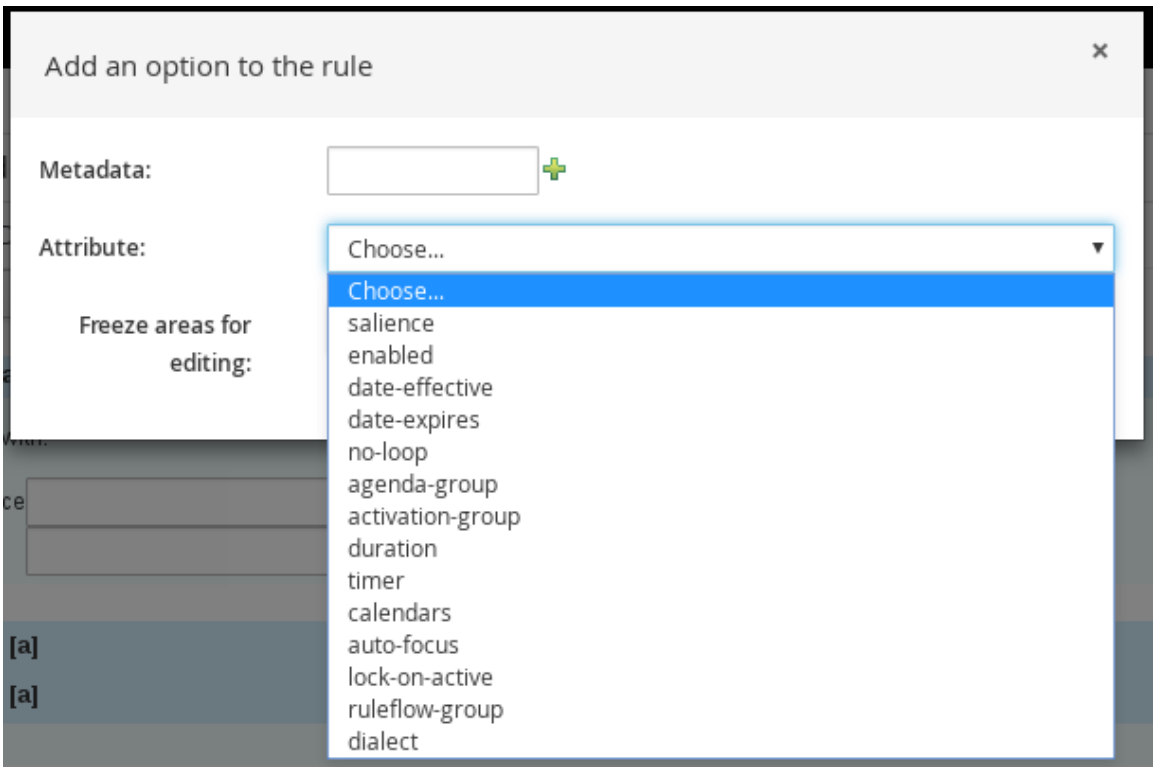
58.4. その他のルールオプションの追加

ルールデザイナーを使用してルールにメタデータを追加し、追加のルール属性 (**salience**、**no-loop** など) を定義し、条件またはアクションの変更を制限するために、ルールの領域を凍結します。

手順

1. ルールデザイナーの **THEN** セクションの下にある **(show options...)** をクリックします。
2. ウィンドウの右側にあるプラスアイコン ( ) をクリックして、オプションを追加します。
3. ルールに追加するオプションを選択します。
   - **Metadata:** メタデータのラベルを入力し、プラスアイコン ( ) をクリックします。次に、ルールデザイナーに提供されるフィールドに必要なデータを入力します。
   - **Attribute:** ルール属性のリストから選択します。次に、ルールデザイナーに表示されるフィールドまたはオプションに値を定義します。
   - **Freeze areas for editing (編集する領域を制限):**ルールデザイナーで修正する領域を制限する **条件** または **アクション** を選択します。

     図58.10 ルールオプション



4. ルールデザイナーで **Save** をクリックして、設定した内容を保存します。

58.4.1. ルールの属性

ルール属性は、ルールの動作を修正するビジネスルールを指定する追加設定です。

次の表では、ルールに割り当て可能な属性の名前と、対応する値を紹介します。

表58.1 ルールの属性

属性	値
salience	ルールの優先順位を定義する整数。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。 例: **salience 10**
enabled	ブール値。このオプションを選択すると、ルールが有効になります。このオプションを選択しないと、ルールは無効になります。 例: **enabled true**
date-effective	日付定義および時間定義を含む文字列。現在の日時が **date-effective** 属性よりも後の場合は、このルールがアクティブになります。 例: **date-effective "4-Sep-2018"**
date-expires	日付定義および時間定義を含む文字列。現在日時が **date-expires** 属性よりも後になると、このルールをアクティブにすることはできません。 例: **date-expires "4-Oct-2018"**
no-loop	ブール値。このオプションが選択される場合は、ルールの結果が以前一致した条件を再度トリガーすると、ルールは再アクティブ化 (ループ) されません。条件を選択しないと、この状況でルールがループされます。 例: **no-loop true**
agenda-group	ルールを割り当てるアジェンダグループを指定する文字列。アジェンダグループを使用すると、アジェンダをパーティションで区切り、ルールのグループに対する実行をさらに制御できます。フォーカスを取得したアジェンダグループのルールだけがアクティブになります。 例: **agenda-group "GroupName"**
activation-group	ルールを割り当てるアクティベーション (または XOR) グループを指定する文字列。アクティベーショングループでは、1 つのルールのみをアクティブ化できます。最初のルールが実行すると、アクティベーショングループの中で、アクティベーションが保留中のルールはすべてキャンセルされます。 例: **activation-group "GroupName"**
duration	ルールの条件が一致している場合に、ルールがアクティブになってからの時間をミリ秒で定義する長整数値。 例: **duration 10000**
timer	ルールのスケジュールに対する **int** (間隔) または **cron** タイマー定義を指定する文字列。 例: **timer (cron:* 0/15 * * * ?)** (15 分ごと)

属性	値
calendar	ルールのスケジュールを指定する Quartz カレンダーの定義。 例: **calendars "* * 0-7,18-23 ?* *"** (営業時間外を除く)
auto-focus	アジェンダグループ内のルールにのみ適用可能なブール値。このオプションが選択されている場合は、次にルールがアクティブになった場合に、そのルールが割り当てられたアジェンダグループにフォーカスが自動的に指定されます。 例: **auto-focus true**
lock-on-active	ルールフローグループまたはアジェンダグループ内のルールにのみ適用可能なブール値。このオプションを選択すると、次回、ルールのルールフローグループがアクティブになるか、ルールのアジェンダグループがフォーカスを受け取ると、(ルールフローグループがアクティブでなくなるか、アジェンダグループがフォーカスを失うまで) ルールをアクティブにすることができません。これは、**no-loop** 属性を強力にしたものです。なぜなら、一致するルールのアクティベーションが、(ルールそのものによるものだけでなく) 更新元にかかわらず破棄されるためです。この属性は、ファクトを修正するルールが多数あり、ルールの再一致と再発行を希望しない計算ルールに適しています。 例: **lock-on-active true**
ruleflow-group	ルールフローグループを指定する文字列。ルールフローグループで、関連するルールフローによってそのグループがアクティブになった場合に限りルールを発行できます。 例: **ruleflow-group "GroupName"**
dialect	ルールのコード表記に使用される言語を指定する文字列 (**JAVA** または **MVEL**)。デフォルトでは、ルールは、パッケージレベルに指定されている方言を使用します。ここで指定した方言は、ルールのパッケージ方言設定を上書きします。 例: **dialect "JAVA"**  **注記** 実行可能モデルなしで Red Hat Process Automation Manager を使用する場合には、**dialect "JAVA"** ルールの結果は、Java 5 構文のみをサポートします。実行可能モデルに関する詳細は、Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ を参照してください。

第59章 ガイド付きルールテンプレートのデータテーブルの定義

ガイド付きルールテンプレートを作成し、フィールド値にテンプレートキーを追加すると、ガイド付きルールテンプレートデザイナーの **Data** テーブルにデータテーブルが表示されます。データテーブルの各列は、ガイド付きルールテンプレートに追加したテンプレートキーに対応します。このテーブルを使用して、各テンプレートキーに値を 1 行ずつ定義します。そのテンプレートのデータテーブルに定義する値の各行の結果がルールになります。

手順

1. ガイド付きルールテンプレートデザイナーで、**Data** タブをクリックしてデータテーブルを表示します。データテーブルの各列は、ガイド付きルールテンプレートに追加したテンプレートキーに対応します。

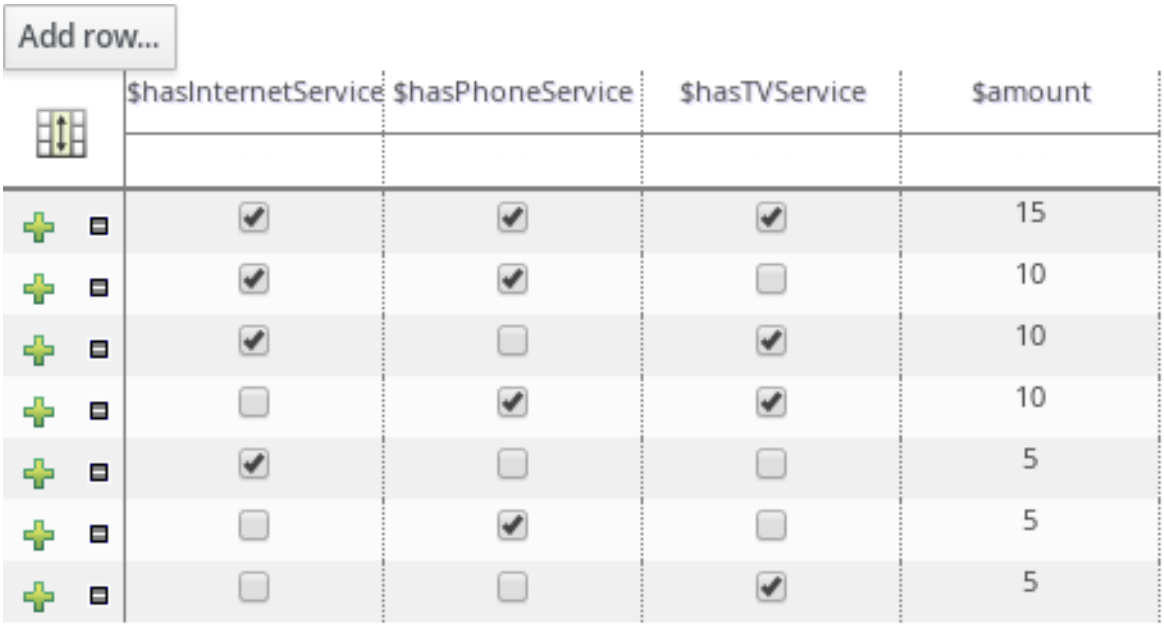
注記

ルールテンプレートにテンプレートキーを追加していないと、このデータテーブルは表示されず、テンプレートが正式なテンプレートとして機能しません。代わりに、個々のガイド付きルールとして機能します。このため、テンプレートキーは、ガイド付きルールテンプレートを作成するための基本となります。

2. **Add row** をクリックして、各テンプレートキー列にデータ値を定義して、ルール (行) を生成します。

3. 引き続き、行を追加して、生成する各ルールにデータ値を定義します。**Add row** をクリックするか、プラスアイコン ( ) をクリックして行を追加するか、マイナスアイコンをクリックして行を削除します。

図59.1 ガイド付きルールテンプレートのデータテーブルの例



DRL コードを表示するには、ガイド付きルールテンプレートデザイナーで **Source** タブをクリックします。

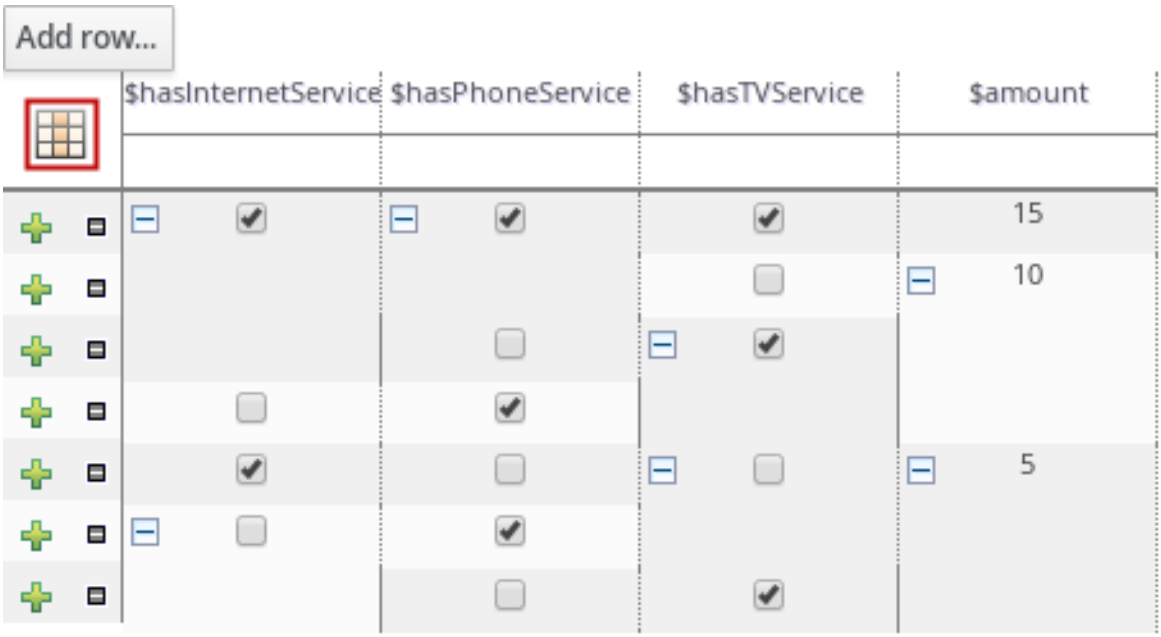
以下に例を示します。

```
rule "PaymentRules_6"
```

```
 when
  Customer( internetService == false ,
   phoneService == false ,
   TVService == true )
 then
  RecurringPayment fact0 = new RecurringPayment();
  fact0.setAmount( 5 );
  insertLogical( fact0 );
end

rule "PaymentRules_5"
 when
  Customer( internetService == false ,
   phoneService == true ,
   TVService == false )
 then
  RecurringPayment fact0 = new RecurringPayment();
  fact0.setAmount( 5 );
  insertLogical( fact0 );
end
...
//Other rules omitted for brevity.
```

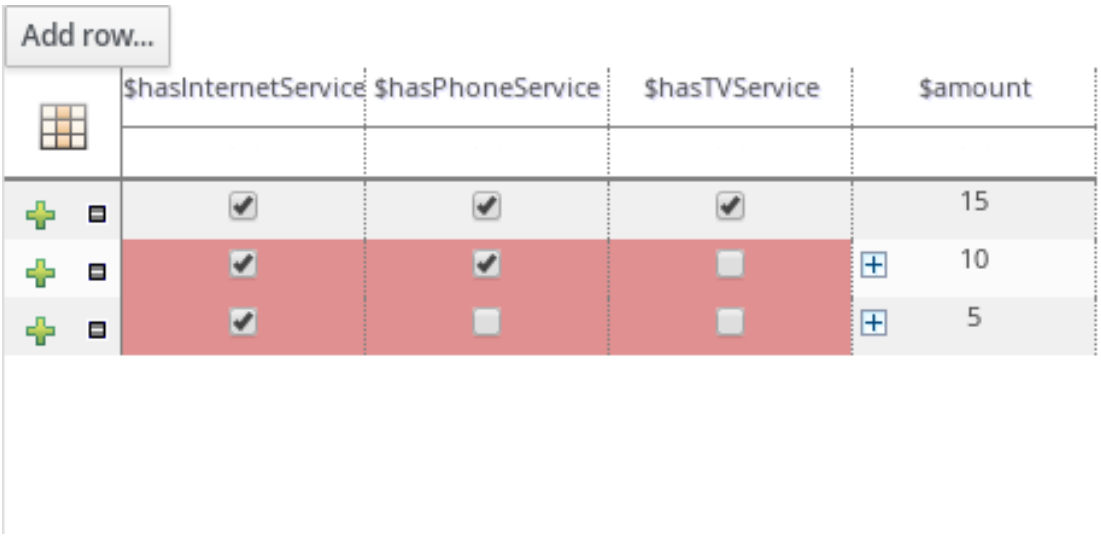
4. 必要に応じて、見やすくするために、データテーブルの左上にあるグリッドアイコンをクリックして、セルの結合をオンまたはオフに切り替えます。同じ列で、値が同じセルは、1つのセルに結合されます。

図59.2 データテーブルでセルの結合



次に、新たに結合した各セルの左上にある展開アイコンまたは折りたたみアイコン [+/-] を使用して、結合したセルに対応する行を折りたたんだり、折りたたんだ行を再展開したりします。

図59.3 結合したセルの折りたたみ



5. すべてのルールにテンプレートキーデータを定義し、必要に応じてテーブル表示を変更したら、ガイド付きルールテンプレートデザイナーの右上のツールバーで **Validate** をクリックして、ガイド付きルールテンプレートの妥当性を確認します。ルールテンプレートの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルールテンプレートの全コンポーネントと、データテーブルに定義したデータを見直し、エラーが表示されなくなるまでルールテンプレートの妥当性確認を行います。

6. ガイド付きルールテンプレートデザイナーで **Save** をクリックして、設定した内容を保存します。

第60章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントのオプションに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合は、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれていることを確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、Business Central 外に Maven または Java プロジェクトが作成されていない場合は作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例は、その他の DRL ルールの作成および実行方法 を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。

- **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルに読み込みます。
- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (任意) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける Red Hat Process Automation Manager 7.11 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.52.0.Final-redhat-00007</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation BOM (bill of materials) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.11.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

5. モデルクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイルの両方に表示されます。

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースを読み込む **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、および賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
```

```
    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

KIE Server でのルールの実行

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);
```

```
        // Set up the fact model:
        Person p = new Person();
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert Person into the session:
        KieCommands kieCommands = KieServices.Factory.get().getCommands();
        List<Command> commandList = new ArrayList<Command>();
        commandList.add(kieCommands.newInsert(p, "personReturnId"));

        // Fire all rules:
        commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
        BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

        // Use rule services client to send request:
        RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
        ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
        System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
}
```

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリーにおける) Java の実行例

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

第61章 次のステップ

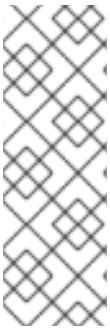
- テストシナリオを使用したデシジョンサービスのテスト
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート VIII. テストシナリオを使用したデシジョンサービスのテスト

ビジネス分析者またはビジネスルールの開発者は、Business Central でテストシナリオを使用して、プロジェクトをデプロイする前にデシジョンサービスをテストできます。DMN ベースおよびルールベースのデシジョンサービスが適切に、想定通りに機能することをテストできます。またデシジョンサービスは、プロジェクトの開発時にいつでもテストできます。

前提条件

- デシジョンサービスの領域およびプロジェクトが Business Central に作成されている。詳細は **デシジョンサービスのスタートガイド** を参照してください。
- ルールベースのデシジョンサービスに、ビジネスルールおよび関連するデータオブジェクトが定義されている。詳細は、**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成** を参照してください。
- DMN ベースのデシジョンサービスに、DMN デシジョンロジックとその関連のカスタムデータタイプが定義されている。詳細は **DMN モデルを使用したデシジョンサービスの作成** を参照してください。



注記

テストシナリオでは、ビジネスルールを設定するように定義したデータをテストできるため、ビジネスルールを先に定義しておくことは、テストシナリオにおける技術的な前提条件ではありません。ただし、先にルールを作成しておくと、テストシナリオでルール全体をテストすることができ、かつ意図するデシジョンサービスにシナリオがより近づくため便利です。DMN ベースの場合には、テストシナリオを使用することで、DMN デシジョンロジックとその関連のカスタムデータタイプがデシジョンサービス用に定義されます。

第62章 テストシナリオ

Red Hat Process Automation Manager のテストシナリオでは、ビジネスルールを実稼働環境にデプロイする前に、(ルールベースのテストシナリオの場合) ビジネスルールの機能とデータの妥当性、および (DMN ベースのテストシナリオの場合) DMN モデルを検証できます。このテストシナリオでは、プロジェクトのデータを使用して、指定した条件と、定義した 1 つ以上のビジネスルールで想定される結果を設定できます。シナリオを実行する際は、想定した結果と、ルールのインスタンスから実際に得られた結果を比較します。想定される結果が実際の結果と一致すると、テストは成功します。想定された結果が実際の結果と一致しないと、テストは失敗します。

Red Hat Process Automation Manager は現在、新規の **テストシナリオ** デザイナーと以前の **テストシナリオ (レガシー)** デザイナーの両方を含みます。デフォルトのデザイナー、新規のテストシナリオデザイナーで、ルールと DMN モデルのテストをサポートし、テストシナリオの全体的な使用感が改善されています。必要に応じて、レガシーのテストシナリオをそのまま使用することができますが、ルールベースのテストシナリオしかサポートされません。

重要

レガシーのテストシナリオデザイナーは、Red Hat Process Automation Manager バージョン 7.3.0 で非推奨になりました。このツールは、今後の Red Hat Process Automation Manager リリースで削除予定です。代わりに、新しいテストシナリオデザイナーを使用してください。

プロジェクトレベルや、特定のシナリオアセット内で利用可能なテストシナリオを実行するなど、複数の方法で定義済みのテストシナリオを実行できます。テストシナリオは独立しており、他のテストシナリオに影響を与えたり、テストシナリオを変更したりできません。テストシナリオは、Business Central のプロジェクト開発時にいつでも実行できます。テストシナリオを実行するために、デシジョンサービスをコンパイルまたはデプロイする必要はありません。

別のパッケージからのデータオブジェクトは、テストシナリオと同じプロジェクトパッケージにインポートできます。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとテストシナリオを作成したら、テストシナリオデザイナーの **Data Objects** タブを使用して、必要なデータオブジェクトがすべてリストされていることを検証するか、**アイテムを追加** して既存のデータオブジェクトをインポートします。

重要

テストシナリオのドキュメント全体で、**テストシナリオ** および **テストシナリオデザイナー** に関する言及はすべて、レガシーバージョンと明示的に記載がない限り、新規バージョンを対象としています。

第63章 データオブジェクト

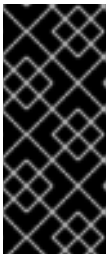
データオブジェクトは、作成するルールアセットの設定要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータタイプです。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

63.1. データオブジェクトの作成

次の手順は、データオブジェクトの作成の一般的な概要です。特定のビジネスアセットに固有のものではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

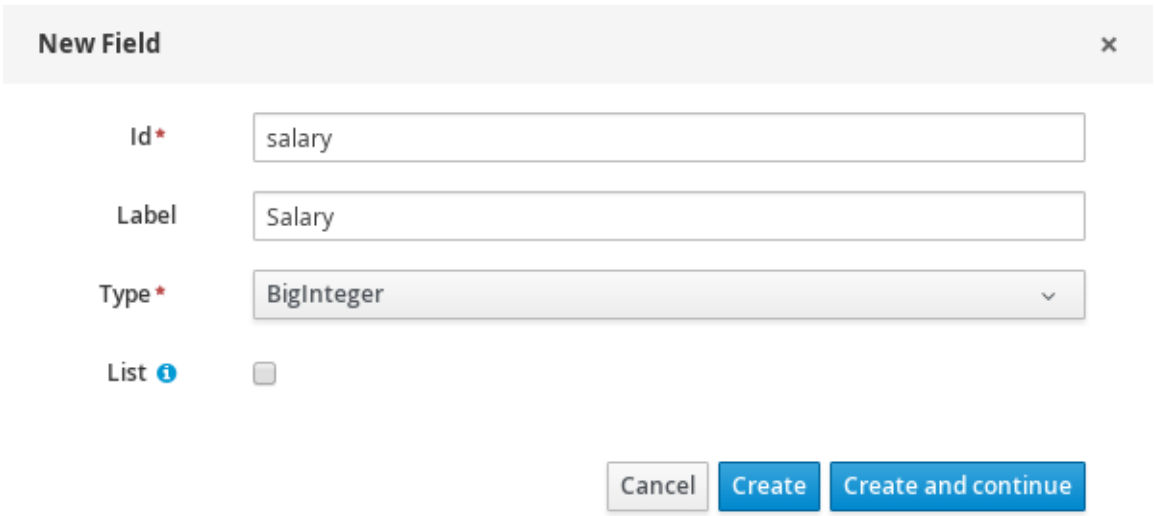


別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、および **Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータ型を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図63.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして、新しいフィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き作成できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

第64章 BUSINESS CENTRAL でのテストシナリオデザイナー

テストシナリオデザイナーは、テーブル形式のレイアウトを提供し、シナリオテンプレートと関連するテストケースをすべて定義できるようにします。デザイナーレイアウトはヘッダーと個別の行を持つテーブルで設定されています。ヘッダーは、**GIVEN** と **EXPECT** の行、インスタンスの行、対応のフィールドの行の 3 つで設定されます。ヘッダーは、テストシナリオテンプレートとしても知られており、個別の行はテストシナリオ定義と呼ばれます。

テストシナリオテンプレートまたはヘッダーは以下の 2 つの部分で設定されます。

- **GIVEN** データオブジェクトおよびそのフィールド: 入力情報を表現します。
- **EXPECT** データオブジェクトおよびそのフィールド: オブジェクトとフィールドを表現します。実際の値が指定の情報をもとにチェックされ、想定の結果を設定するのにも、この値を使用します。

テストシナリオの定義は、個別のテストケーステンプレートを表現します。

Project Explorer にはデザイナーの左パネルからアクセスできますが、右パネルからは **Settings**、**Test Tools**、**Scenario Cheatsheet**、**Test Report**、**Coverage Report** タブにアクセスできます。**Settings** タブにアクセスし、ルールベースおよび DMN ベースのテストシナリオのグローバル設定を表示し、編集できます。**Test Tools** を使用してデータオブジェクトマッピングを設定できます。**Scenario Cheatsheet** タブには参照として使用できるメモとチートシートが含まれています。**Test Report** タブにはテストの概要とシナリオステータスが表示されます。テストカバレッジ統計を表示するには、テストシナリオデザイナーの右側にある **Coverage Report** タブを使用します。

64.1. データオブジェクトのインポート

テストシナリオデザイナーは、テストシナリオと同じパッケージ内に配置されている全データオブジェクトを読み込みます。すべてのデータオブジェクトは、デザイナーの **Data Objects** タブから表示できます。読み込んだデータオブジェクトは、**Test Tools** パネルにも表示されます。

データオブジェクトが変更された場合に (新規データオブジェクトの作成時や、既存のデータオブジェクトの削除時など) デザイナーを終了して、開き直す必要があります。一覧からデータオブジェクトを選択して、フィールドとフィールドタイプを表示します。

テストシナリオとは異なるパッケージに配置されているデータオブジェクトを使用する場合には、先にそのデータオブジェクトをインポートする必要があります。以下の手順に従い、ルールベースのテストシナリオ用にデータオブジェクトをインポートしてください。

注記

DMN ベースのテストシナリオの作成中にデータオブジェクトをインポートできません。DMN ベーステストシナリオは、プロジェクトからのデータオブジェクトを使用せず、DMN ファイルで定義したカスタムのデータタイプを使用します。

手順

1. テストシナリオデザイナーの **Project Explorer** パネルに移動します。
2. **Test Scenario** からテストシナリオを選択します。
3. **Data Objects** タブを選択して、**New Item** をクリックします。
4. **Add import** ウィンドウで、ドロップダウンリストからデータオブジェクトを選択します。

5. **Ok** をクリックしてから **Save** をクリックします。
6. テストシナリオデザイナーを終了して再度開き、データオブジェクトリストから新しいデータオブジェクトを表示します。

64.2. テストシナリオのインポート

プロジェクトビューの **Asset** タブにある **Import Asset** ボタンを使用して、既存のテストシナリオをインポートできます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Asset** タブから **Import Asset** をクリックします。
3. **Create new Import Asset** ウィンドウで、以下を実行します。
   - インポートセットの名前を入力します。
   - **Package** ドロップダウンリストからパッケージを選択します。
   - **Please select a file to upload**から、**Choose File...** をクリックしてテストシナリオファイルを参照します。
4. ファイルを選択して **Open** をクリックします。
5. **Ok** をクリックすると、テストシナリオデザイナーでテストシナリオが開きます。

64.3. テストシナリオの保存

テストシナリオは、テストシナリオテンプレートの作成時や、テストシナリオの定義時にいつでも保存できます。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Save** をクリックします。
2. **Confirm Save** ウィンドウで、以下を実行します。
   a. テストシナリオに関するコメントを追加する場合は、**add a comment** をクリックします。
   b. もう一度 **Save** をクリックします。

テストシナリオが正常に保存されたことを示すメッセージが画面に表示されます。

64.4. テストシナリオのコピー

右上のツールバーの **Copy** ボタンを使用して、既存のテストシナリオを同じパッケージまたは別のパッケージにコピーできます。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Copy** をクリックします。

2. **Make a Copy** ウィンドウで、以下を実行します。
    a. **New Name** フィールドに名前を入力します。
    b. テストシナリオをコピーするパッケージを選択します。
    c. 必要に応じて、コメントを追加するには、**add a comment** をクリックします。
    d. **Make a Copy** をクリックします。

テストシナリオが正常にコピーされたことを示すメッセージが画面に表示されます。

64.5. テストシナリオのダウンロード

今後の参考に、またはバックアップとして、ローカルのマシンにテストシナリオのコピーをダウンロードできます。

手順

右上のテストシナリオデザイナーのツールバーで、**Download** をクリックします。

.scesim ファイルがローカルマシンにダウンロードされます。

64.6. テストシナリオのバージョン間の切り替え

Business Central には、テストシナリオのさまざまなバージョン間で切り替える機能があります。シナリオを保存するたびに、シナリオの新しいバージョンが **Latest Versions** に表示されます。この機能を使用するには、一度はテストシナリオファイルを保存しておく必要があります。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Latest Version** をクリックします。ファイルに複数バージョンがある場合は、ファイルの全バージョンが **Latest Version** に表示されます。
2. 作業するバージョンをクリックします。
テストシナリオの選択したバージョンがテストシナリオデザイナーで開きます。
3. デザイナーツールバーから **Restore** をクリックします。
4. **Confirm Restore** で以下を実行します。
    a. コメントを追加するには、**add a comment** をクリックします。
    b. **Restore** をクリックして確定します。

選択したバージョンが正常にデザイナーに再読込されたことを示すメッセージが画面に表示されます。

64.7. アラートパネルの表示または非表示

Alerts パネルは、テストシナリオデザイナーまたはプロジェクトビューの下部に表示されます。実行したテストに失敗した場合は、ビルド情報とエラーメッセージが表示されます。

手順

右上のデザイナーツールバーで、**Hide Alerts/View Alerts** をクリックして、レポートパネルを有効または無効にします。

64.8. コンテキストメニューのオプション

テストシナリオデザイナーには、コンテキストメニューオプションがあり、行と列の追加、削除、複製など、テーブルでの基本操作を実行できます。コンテキストメニューを使用するには、テーブル要素を右クリックする必要があります。メニューオプションは、選択するテーブル要素により異なります。

表64.1 コンテキストメニューのオプション

テーブル要素	セルのラベル	利用可能なコンテキストメニューオプション
Header	# & シナリオの説明	下に行を挿入
	GIVEN, EXPECT	左端に列を挿入、右端に列を挿入、下に行を挿入
	INSTANCE 1, INSTANCE 2, PROPERTY 1, PROPERTY 2	左に列を挿入、右に列を挿入、列を削除、インスタンスを複製、下に行を挿入
行	行番号、テストシナリオの説明、またはテストシナリオの定義を含むすべてのセル	上に行を挿入、下に行を挿入、行を複製、行を削除、シナリオを実行

表64.2 テーブルの操作の説明

テーブルの操作	説明
左端に列を挿入	(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションで) 新たに左端に列を挿入します。
右端に列を挿入	(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションで) 新たに右端に列を挿入します。
左に列を挿入	選択した列の左に新しい列を挿入します。新しい列は、(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションに) 選択した列と同じタイプを挿入します。
右に行を挿入	選択した列の右に新しい列を挿入します。新しい列は、(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションに) 選択した列と同じタイプを挿入します。
列を削除	選択した列を削除します。
上に行を挿入	選択した行の上に新しい行を挿入します。
下に行を挿入	選択した行の下に新しい行を挿入します。ヘッダーセルからの呼び出しの場合は、インデックス 1 の行を新たに挿入します。
行を複製	選択した行を複製します。

テーブルの操作	説明
インスタンスを複製	選択したインスタンスを複製します。
行を削除	選択した行を削除します。
シナリオを実行	単一のテストシナリオを実行します。

Insert column right コンテキストメニューオプションと **Insert column left** コンテキストメニューオプションの動作は異なります。

- 選択した列にタイプが定義されていない場合は、タイプなしで新しい列が追加されます。
- 選択した列にタイプが定義されていない場合は、親のインスタンスタイプを引き継いだ列または新しい空の列が作成されます。
  - アクションがインスタンスヘッダーから実行する場合は、タイプなしの列が新たに作成されます。
  - アクションがプロパティーのヘッダーから実行する場合は、親のインスタンスタイプを引き継いだ列が新たに作成されます。

64.9. テストシナリオのグローバル設定

テストシナリオデザイナーの右側にあるグローバル **Settings** タブを使用して、アセットの追加プロパティーを設定および変更できます。

64.9.1. ルールベースのテストシナリオのグローバル設定の設定

以下の手順に従って、ルールベースのテストシナリオのグローバル設定を表示および編集します。

手順

1. テストシナリオデザイナーの右側にある **Settings** タブをクリックして、属性を表示します。
2. **Settings** パネルで次の属性を設定します。
   - **Name**: デザイナーの右上のツールバーから **Rename** オプションを使用して、既存のテストシナリオの名前を変更できます。
   - **Type**: この属性は、ルールベースのテストシナリオであり、読み取り専用であることを指定します。
   - **Stateless Session**: KieSession がステートレスかどうかを指定するには、このチェックボックスを選択するか、選択解除します。

注記

現在の KieSession がステートレスで、チェックボックスが選択されていない場合は、テストに失敗します。

- **KieSession**: (任意) テストシナリオの KieSession を入力します。
- **RuleFlowGroup/AgendaGroup**: (任意) テストシナリオの RuleFlowGroup または AgendaGroup を入力します。

3. 任意: テスト実行後にプロジェクトレベルからシミュレーション全体をスキップするには、このチェックボックスを選択します。
4. **Save** をクリックします。

64.9.2. DMN ベースのテストシナリオのグローバル設定設定

以下の手順に従って、DMN ベースのテストシナリオのグローバル設定を表示および編集します。

手順

1. テストシナリオデザイナーの右側にある **Settings** タブをクリックして、属性を表示します。
2. **Settings** パネルで次の属性を設定します。
   - **Name**: デザイナーの右上のツールバーから **Rename** オプションを使用して、既存のテストシナリオの名前を変更できます。
   - **Type**: この属性は、DMN ベースのテストシナリオであり、読み取り専用であることを指定します。
   - **DMN model**: (オプション) テストシナリオ用の DMN モデルを入力します。
   - **DMN name**: これは DMN モデルの名前で、読み取り専用です。
   - **DMN namespace**: これは DMN モデルのデフォルトの名前空間で、読み取り専用です。
3. 任意: テスト実行後にプロジェクトレベルからシミュレーション全体をスキップするには、このチェックボックスを選択します。
4. **Save** をクリックします。

第65章 テストシナリオテンプレート

テストシナリオの定義を指定する前に、テストシナリオテンプレートを作成する必要があります。テストシナリオテーブルのヘッダーにより、各シナリオのテンプレートが定義されます。GIVEN と EXPECT の両セクションに、インスタンスタイプとプロパティーヘッダーを設定する必要があります。インスタンスヘッダーは、特定のデータオブジェクト (ファクト) にマッピングし、プロパティーヘッダーは対応するデータオブジェクトの特定のフィールドにマッピングします。

テストシナリオデザイナーを使用すると、ルールベースと DMN ベースの両方のテストシナリオのテストシナリオテンプレートを作成できます。

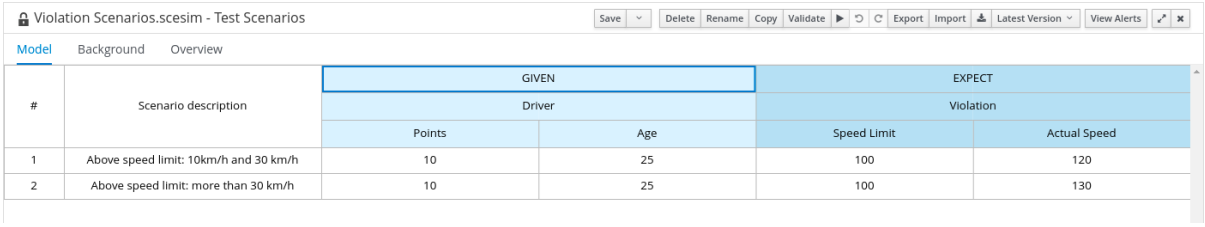
65.1. ルールベースシナリオのテストシナリオテンプレートの作成

以下の手順に従い、ルールベースのテストシナリオのテストシナリオテンプレートを作成してルールとデータを検証します。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、テストシナリオを作成するプロジェクトをクリックします。
2. **Add Asset → Test Scenario** の順にクリックします。
3. **テストシナリオ** 名を入力し、適切な **パッケージ** を選択します。選択するパッケージは、必要なデータオブジェクトとルールアセットが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
4. **Source type** として **RULE** を選択します。
5. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。
6. **GIVEN** 列ヘッダーをデータオブジェクトにマッピングするには、以下を実行します。

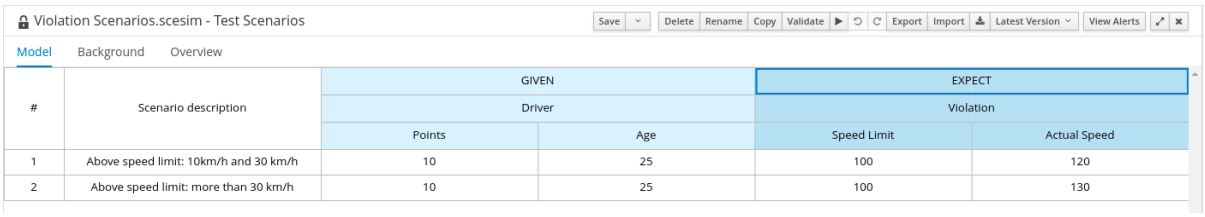
図65.1 テストシナリオの GIVEN ヘッダーセル



   a. **GIVEN** セクションのインスタンスヘッダーセルを選択します。
   b. **Test Tools** タブからデータオブジェクトを選択します。
   c. **Insert Data Object** をクリックします。

7. **EXPECT** 列ヘッダーをデータオブジェクトにマッピングするには、以下を実行します。

図65.2 テストシナリオの EXPECT ヘッダーセル



a. **EXPECT** セクションのインスタンスヘッダーセルをクリックします。

b. **Test Tools** タブからデータオブジェクトを選択します。

c. **Insert Data Object** をクリックします。

8. データオブジェクトをプロパティーセルにマッピングするには、以下を実行します。

a. インスタンスヘッダーセルまたはプロパティーヘッダーセルを選択します。

b. **Test Tools** タブからデータオブジェクトフィールドを選択します。

c. **Insert Data Object** をクリックします。

9. データオブジェクトのプロパティーをさらに挿入するには、プロパティーヘッダーを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。

10. テストシナリオの実行中に java メソッドをプロパティーセルに定義するには、以下を実行します。

a. インスタンスヘッダーセルまたはプロパティーヘッダーセルを選択します。

b. **Test Tools** タブからデータオブジェクトフィールドを選択します。

c. **Insert Data Object** をクリックします。

d. プリフィックスに **#** を指定した MVEL 式使用して、テストシナリオの実行に java メソッドを定義します。

e. データオブジェクトのプロパティーをさらに挿入するには、プロパティーヘッダーセルを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。

11. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。

ルールベースのシナリオの式の構文に関する詳細は、「ルールベースのテストシナリオの式構文」 を参照してください。

65.2. ルールベースのテストシナリオでのエイリアスの使用

テストシナリオデザイナーで、ヘッダーセルをデータオブジェクトにマッピングすると、データオブジェクトは **Test Tools** タブから削除されます。エイリアスを使用して、データオブジェクトを別のヘッダーセルに再マッピングできます。また、エイリアスを使用すると、テストシナリオで同じデータオブジェクトのインスタンスを複数指定できます。さらに、プロパティーエイリアスを作成して、使用するプロパティーの名前をテーブルで直接変更することもできます。

手順

Business Central のテストシナリオデザイナーで、ヘッダーセルをダブルクリックし、名前を手動で変更します。エイリアスの名前が一意であることを確認してください。

インスタンスが **Test Tools** タブのデータオブジェクトのリストに表示されます。

第66章 DMN ベーステストシナリオのテストテンプレート

Business Central は、DMN ベースの全テストシナリオアセットに対してテンプレートを生成し、この中に、関連の DMN モデルの指定のインプットやデシジョンがすべて含まれます。DMN モデルの入力ノードごとに、**GIVEN** 列が追加され、各デシジョンノードは **EXPECT** 列で表現されます。デフォルトのテンプレートは、必要に応じていつでも変更できます。また、DMN モデルの一部のみをテストするには、生成した列を削除することも、EXPECT から GIVEN セクションのデシジョンノードを移動することが可能です。

66.1. DMN ベースのテストシナリオのテストシナリオテンプレート作成

以下の手順に従い、DMN ベースのシナリオのテストシナリオテンプレートを作成し、DMN モデルを検証します。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、テストシナリオを作成するプロジェクトをクリックします。
2. **Add Asset → Test Scenario** の順にクリックします。
3. **テストシナリオ** 名を入力し、適切な **パッケージ** を選択します。
4. **Source type** で **DMN** を選択します。
5. **Choose DMN asset**オプションを使用して、既存の DMN アセットを選択します。
6. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。
テンプレートは自動的に生成され、ニーズに合わせてこのテンプレートを変更できます。
7. テストシナリオの実行中に java メソッドをプロパティーセルに定義するには、以下を実行します。
    a. インスタンスヘッダーセルまたはプロパティーヘッダーセルをクリックします。
    b. **Test Tools** タブからデータオブジェクトフィールドを選択します。
    c. **Insert Data Object** をクリックします。
    d. 式を使用して、テストシナリオの実行に java メソッドを定義します。
    e. データオブジェクトにプロパティーをさらに追加するには、プロパティーヘッダーセルを右クリックして、必要に応じて **Insert column right** または **Insert column left** を選択します。
8. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。

DMN ベースのシナリオの式の構文に関する詳細は、「DMN ベースのテストシナリオでの式の構文」を参照してください。

第67章 テストシナリオの定義

テストシナリオテンプレートの作成後、テストシナリオを定義する必要があります。テストシナリオテーブルの行では、個別のテストシナリオを定義します。テストシナリオには、一意のインデックス番号、説明、入力値セット (**Given** の値)、出力値セット (**Expect** の値) を定義します。

前提条件

- 選択したテストシナリオに、テストシナリオテンプレートが作成されている。

手順

1. テストシナリオデザイナーでテストシナリオを開きます。
2. テストシナリオの説明を入力して、行のセルに必要な値を入力します。
3. コンテキストメニューを使用して、必要に応じて行を追加または削除します。
   セルをダブルクリックしてインラインの編集を開始します。テスト評価で特定のセルを省略するには、そのセルを空白のままにします。

テストシナリオの定義したら、テストを実行できます。

第68章 テストシナリオでのバックグラウンドインスタンス

テストシナリオデザイナーでは、**Background** タブを使用して、ルールベースのテストシナリオと DMN ベースのテストシナリオのバックグラウンドデータを追加して設定できます。利用可能なデータオブジェクトに基づいて、テストシナリオ全体で共通となる GIVEN データを追加して定義できます。**Background** タブには、すべてのテストシナリオでデータを追加および共有できます。**Background** タブを使用して追加したデータは、**Model** タブデータで上書きできません。

たとえば、特定の人の **Age** が全テストシナリオ例で同じ値にする必要がある場合に、**Background** ページの **Age** の値を定義し、テストシナリオテーブルのテンプレートからその列を除外できます。このような場合に、**Age** は全テストシナリオで **25** に設定されます。

図68.1 Age の反復値を使用するテストシナリオ例

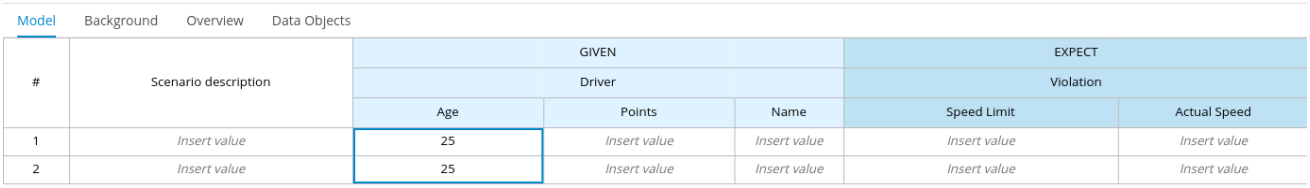


図68.2 Age の反復値のバックグラウンド定義例

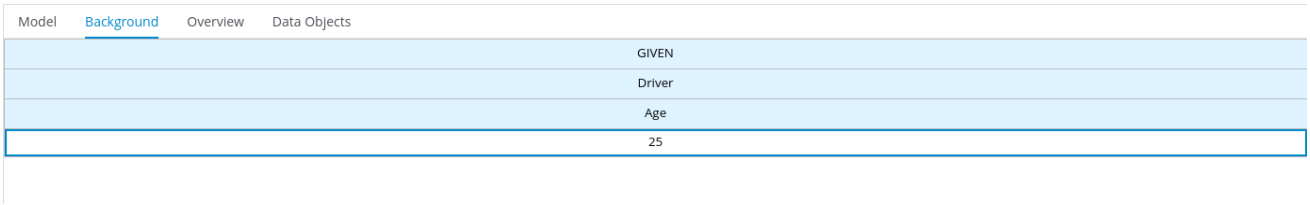
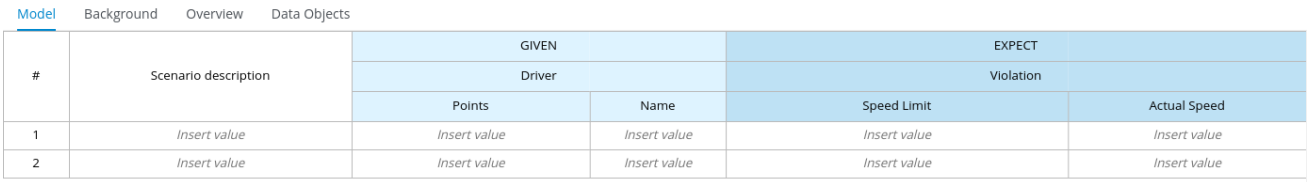


図68.3 除外された Age 列を含む変更済みのテストシナリオテンプレート



注記

Background タブで定義した GIVEN データは、同じ ***.scesim** ファイルのテストシナリオ間しか共有できず、別のテストシナリオには共有されません。

68.1. ルールベースのテストシナリオへのバックグラウンドデータの追加

以下の手順に従って、ルールベースのテストシナリオでバックグラウンドデータを追加および設定します。

前提条件

- 選択したテストシナリオ用にルールベースのテストシナリオテンプレートが作成されている。ルールベースのテストシナリオの作成に関する詳細は、「ルールベースシナリオのテストシナリオテンプレートの作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細は、*67章 テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーでルールベースのテストシナリオを開きます。
2. テストシナリオデザイナーの **Background** タブをクリックします。
3. **GIVEN** セクションでインスタンスヘッダーセルを選択し、バックグラウンドのデータオブジェクトフィールドを追加します。
4. **Test Tools** パネルからデータオブジェクトを選択します。
5. **Insert Data Object** をクリックします。
6. バックグラウンドデータオブジェクトフィールドを追加するには、プロパティーヘッダーセルを選択します。
7. **Test Tools** パネルからデータオブジェクトを選択します。
8. **Insert Data Object** をクリックします。
9. データオブジェクトにプロパティーをさらに追加するには、プロパティーヘッダーセルを右クリックして、必要に応じて **Insert column right** または **Insert column left** を選択します。
10. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。
11. 定義済みのテストシナリオを実行します。

68.2. DMN ベースのテストシナリオでのバックグラウンドデータ追加

以下の手順に従って、DMN ベースのテストシナリオでバックグラウンドデータを追加して設定します。

前提条件

- 選択したテストシナリオ用に DMN ベースのテストシナリオテンプレートが作成されている。DMN ベースのテストシナリオの作成に関する詳細は、「DMN ベースのテストシナリオのテストシナリオテンプレート作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細は、67章 *テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーで DMN ベースのテストシナリオを開きます。
2. テストシナリオデザイナーの **Background** タブをクリックします。
3. **GIVEN** セクションでインスタンスヘッダーセルを選択し、バックグラウンドのデータオブジェクトフィールドを追加します。
4. **Test Tools** パネルからデータオブジェクトを選択します。
5. **Insert Data Object** をクリックします。
6. バックグラウンドデータオブジェクトフィールドを追加するには、プロパティーヘッダーセルを選択します。

7. **Test Tools** パネルからデータオブジェクトを選択します。
8. **Insert Data Object** をクリックします。
9. データオブジェクトにプロパティーをさらに追加するには、プロパティーヘッダーセルを右クリックして、必要に応じて **Insert column right** または **Insert column left** を選択します。
10. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。
11. 定義済みのテストシナリオを実行します。

第69章 テストシナリオでのリストおよびマッピングコレクションの使用

テストシナリオデザイナーは、DMN ベースとルールベース両方のテストシナリオのリストおよびマッピングコレクションをサポートします。リストやマッピングのようなコレクションは、**GIVEN** および **EXPECT** の両列に、特定のセルの値として作成して定義できます。

注記

マップエントリーでは、エントリーキーのデータタイプは **String** である必要があります。

ルールベースのコレクションエディターの **EXPECT** 列にパラメーターを指定するにはキーワード **actualValue** を使用し、DMN ベースのテストシナリオではキーワード **?** を使用します。

手順

1. 列タイプを先に設定します (タイプがリストまたはマッピングのフィールドを使用します)。
2. 列内のセルをダブルクリックして、値を入力します。
3. コレクションエディターのポップアップでデータオブジェクトのリスト値を作成するには、以下を実行します。
   a. **Creae List** を選択します。
   b. **Add New Item** をクリックします。
   c. 必要な値を入力して、チェックアイコン  をクリックし、追加した各コレクションアイテムを保存します。
   d. **Save** をクリックします。
   e. コレクションからアイテムを編集するには、コレクションのポップアップエディターで鉛筆のアイコンをクリックします。
   f. **Save Changes** をクリックします。
   g. コレクションからアイテムを削除するには、コレクションのポップアップエディターでゴミ箱のアイコンをクリックします。
4. コレクションエディターのポップアップでデータオブジェクトのリスト値を定義するには、以下を実行します。
   a. **Define List** を選択します。
   b. MVEL 式または FEEL 式を使用して、テキストフィールドでリストの値を定義します。ルールベースのテストシナリオは MVEL 式言語を、DMN ベースのテストシナリオは FEEL 式言語を使用します。
   c. **Save** をクリックします。
5. コレクションエディターのポップアップでデータオブジェクトのマップの値を作成するには、以下を実行します。

a. **Create Map** を選択します。

b. **Add New Item** をクリックします。

c. 必要な値を入力して、チェックアイコン  をクリックし、追加した各コレクションアイテムを保存します。

d. **Save** をクリックします。

e. コレクションからアイテムを編集するには、コレクションのポップアップエディターで鉛筆のアイコンをクリックします。

f. **Save Changes** をクリックします。

g. コレクションからアイテムを削除するには、コレクションのポップアップエディターでゴミ箱のアイコンをクリックします。

6. コレクションエディターのポップアップでデータオブジェクトのマップの値を定義するには、以下を実行します。

a. **Define Map** を選択します。

b. MVEL 式または FEEL 式を使用して、テキストフィールドでマップの値を定義します。ルールベースのテストシナリオは MVEL 式言語を、DMN ベースのテストシナリオは FEEL 式言語を使用します。

c. **Save** をクリックします。

注記

DMN ベースのテストシナリオ向けにマップ値を定義するには、コレクションエディターを使用せずにファクトを追加して FEEL 式を使用します。

7. **Remove** をクリックして全コレクションを削除します。

第70章 テストシナリオの式構文

テストシナリオデザイナーでは、ルールベースおよび DMN ベースのテストシナリオの両方で、異なる式言語をサポートします。ルールベースのテストシナリオは MVFLEX 式言語 (MVEL) を、DMN ベースのテストシナリオは FEEL (Friendly Enough Expression Language) 式言語をサポートします。

70.1. ルールベースのテストシナリオの式構文

ルールベースのテストシナリオは、以下の組み込みデータ型をサポートします。

- String
- Boolean
- Integer
- Long
- Double
- Float
- Character
- Byte
- Short
- LocalDate

注記

その他のデータ型については、プリフィックスとして **#** を指定した MVEL 式を使用してください。

テストシナリオデザイナーの BigDecimal の例に従って、プリフィックス **#** を使用して java 式を設定します。

- **GIVEN** 列の値に、**# java.math.BigDecimal.valueOf(10)** と入力します。
- **EXPECT** 列の値に **# actualValue.intValue() == 10** と入力します。

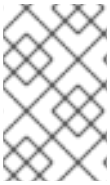
java 式の **EXPECT** 列の実際の値を参照して、条件を実行できます。

テストシナリオデザイナーでは、以下に示すルールベースのテストシナリオの定義式がサポートされます。

表70.1 式構文の説明

Operator	説明
=	値と同等であることを指定します。これは、全列でデフォルトとなっており、GIVEN 列でサポートされる唯一の演算子です。

Operator	説明
=、=!、<>	値が同等でないことを指定します。この演算子は、他の演算子と組み合わることができます。
<、>、<=、>=	比較 (未満、値よりも大きい、以下、以上) を指定します。
#	この演算子を使用して、プロパティーヘッダーセルに java 式の値を設定することで、java メソッドとして実行できます。
[value1, value2, value3]	値の一覧を指定します。1 つまたは複数の値が有効な場合は、シナリオの定義が True と評価されます。
expression1; expression2; expression3	式のリストを指定します。すべての式が有効な場合は、シナリオ定義が True と評価されます。



注記

ルールベースのテストシナリオを評価する場合、空のセルは評価からスキップされます。空の文字列を定義するには、**=**、**[]**、または **;** を使用します。null 値を定義するには、**null** を使用します。

表70.2 式の例

式	説明
-1	実際の値は、-1 と同等です。
< 0	実際の値は、0 未満です。
! > 0	実際の値は、0 以下です。
[-1, 0, 1]	実際の値は、-1、0、または 1 です。
<> [1, -1]	実際の値は、1 でも -1 でもありません。
!100; 0	実際の値は、100 ではなく、0 です。
!= < 0; <> > 1	実際の値は、0 未満でも、1 以上でもありません。
<> <= 0; >= 1	実際の値は、0 以下でなく、1 以上です。

ルールベースのテストシナリオデザイナーの右側にある **Scenario Cheatsheet** タブで、サポートされているコマンドと構文を参照できます。

70.2. DMN ベースのテストシナリオでの式の構文

以下のデータタイプは、テストシナリオデザイナーの DMN ベーステストシナリオでサポートされます。

表70.3 DMN ベースのシナリオがサポートするデータタイプ

サポートされるデータタイプ	説明
番号および文字列	文字列は、**"John Doe"**、**"Brno"**、または **""** のように、引用符で囲む必要があります。
ブール値	**true**、**false**、および **null**.
日時	例: **date("2019-05-13")** または **time("14:10:00+02:00")**
関数	**avg**、**max** などの組み込み計算機能をサポートします。
コンテキスト	例: **{x : 5, y : 3}**
範囲とリスト	例: **[1 ..10]** または **[2, 3, 4, 5]** です。

注記

DMN ベースのテストシナリオを評価する場合、空のセルは評価からスキップされます。DMN ベースのテストシナリオで空の文字列を定義するには **" "** を使用し、**null** 値を定義するには null を使用し ます。

DMN ベースのテストシナリオデザイナーの右側にある **Scenario Cheatsheet** タブで、サポートされているコマンドと構文を参照できます。

第71章 テストシナリオの実行

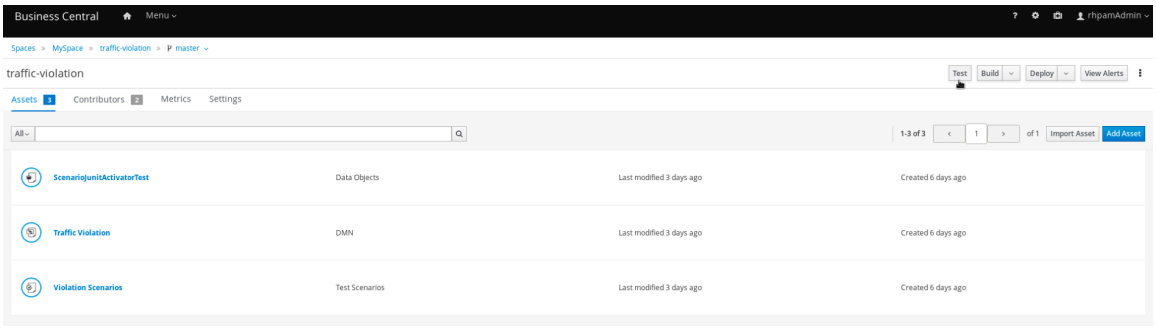
テストシナリオテンプレートを作成してテストシナリオを定義したら、テストを実行してビジネスルールとデータを検証できます。

手順

1. 定義済みのテストシナリオを実行するには、次のタスクのいずれかを実行します。

   - プロジェクトで利用可能なすべてのテストシナリオを複数のアセット内で実行するには、プロジェクトページの右上隅で **Test** をクリックします。

     図71.1 プロジェクトビューからの全テストシナリオの実行

   - **.scesim** ファイルで定義された利用可能なすべてのテストシナリオを実行するには、テストシナリオデザイナーの上部の **Run Test**  アイコンをクリックします。

   - 単一の **.scesim** ファイルで定義されている単一のテストシナリオを実行するには、実行するテストシナリオの行を右クリックし、**Run scenario** を選択します。

2. **Test Report** パネルには、テストの概要とシナリオステータスが表示されます。
   テストの実行後に、テストシナリオテーブルに入力された値が期待した値と一致しない場合には、対応するセルがハイライト表示されます。

3. テストが失敗した場合は、次のタスクを実行して、失敗内容に対してトラブルシューティングすることができます。

   - ポップアップウィンドウでエラーメッセージを確認するには、ハイライトされたセルにマウスのカーソルを合わせます。

   - デザイナーの下部にある **Alerts** パネルまたはプロジェクトビューを開いてエラーメッセージを確認するには、**View Alerts** をクリックします。

   - 必要な変更を加えて、シナリオが成功するまで再度テストを実行します。

第72章 ローカルでのテストシナリオの実行

Red Hat Process Automation Manager では、テストシナリオを Business Central で直接実行することも、コマンドラインを使用してローカルで実行することもできます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトのホームページで **Settings** タブを選択します。
3. **git URL** を選択し、**Clipboard**  をクリックして git url をコピーします。
4. コマンドターミナルを開いて、git プロジェクトのクローンを作成するディレクトリーに移動します。
5. 次のコマンドを実行します。

   ```
   git clone your_git_project_url
   ```

   your_git_project_url は、**git://localhost:9418/MySpace/ProjectTestScenarios** などの適切なデータに置き換えます。
6. プロジェクトのクローンを正しく作成したら、git プロジェクトディレクトリーに移動して、以下のコマンドを実行します。

   ```
   mvn clean test
   ```

   プロジェクトのビルド情報およびテスト結果 (テスト実行回数およびテスト実行が成功したかどうかなど) は、コマンドターミナルに表示されます。失敗がある場合には、Business Central で必要な変更を行い、変更をプルし、もう一度コマンドを実行します。

第73章 テストシナリオスプレッドシートのエクスポートおよびインポート

以下のセクションでは、テストシナリオデザイナーでテストシナリオスプレッドシートをエクスポートおよびインポートする方法を説明します。Microsoft Excel や LibreOffice Calc などのソフトウェアを使用して、テストシナリオスプレッドシートを分析および管理できます。テストシナリオデザイナーは、**.CSV** ファイル形式をサポートします。コンマ区切り値 (CSV) 形式の RFC 仕様の詳細は、Common Format and MIME Type for Comma-Separated Values (CSV) Files を参照してください。

73.1. テストシナリオスプレッドシートのエクスポート

以下の手順に従って、テストシナリオデザイナーを使用してテストシナリオスプレッドシートをエクスポートします。

手順

1. 右上のテストシナリオデザイナーツールバーで、**Export** 

ボタンをクリックします。
2. ローカルファイルディレクトリー内の宛先を選択し、**.CSV** ファイルの保存を確定します。

.CSV ファイルがローカルマシンにエクスポートされます。

73.2. テストシナリオスプレッドシートのインポート

以下の手順に従って、テストシナリオデザイナーを使用してテストシナリオスプレッドシートをインポートします。

手順

1. 右上のテストシナリオデザイナーツールバーで **Import** 

ボタンをクリックします。
2. **Select file to Import** プロンプトで **Choose File...** をクリックし、ローカルファイルディレクトリーからインポートする **.CSV** ファイルを選択します。
3. **Import** をクリックします。

.CSV ファイルがテストシナリオデザイナーにインポートされます。

警告

選択した **.CSV** ファイルのヘッダーを変更しないでください。変更すると、スプレッドシートが正常にインポートされないことがあります。

第74章 テストシナリオのカバレッジレポート

テストシナリオデザイナーは、右側の **Coverage Report** タブを使用してテストカバレッジ統計を明確かつ一貫した形で表示します。また、カバレッジレポートをダウンロードして、テストカバレッジ統計の表示や分析を行うことも可能です。ダウンロードしたテストシナリオのカバレッジレポートは、**.CSV** ファイル形式をサポートします。コンマ区切り値 (CSV) 形式の RFC 仕様の詳細は、Common Format and MIME Type for Comma-Separated Values (CSV) Files を参照してください。

ルールベースおよび DMN ベースのテストシナリオのカバレッジレポートを表示できます。

74.1. ルールベースのテストシナリオのカバレッジレポート生成

ルールベースのテストシナリオでは、**Coverage Report** タブに以下の詳細情報が含まれています。

- 利用可能なルールの数
- 実行したルールの数
- 実行したルールの割合
- 実行したルールの割合 (円グラフで表示)
- 各ルールを実行した回数
- 定義済みのテストシナリオごとに実行するルール

以下の手順に従って、ルールベースのテストシナリオのカバレッジレポートを生成します。

前提条件

- 選択したテストシナリオ用にルールベースのテストシナリオテンプレートが作成されている。ルールベースのテストシナリオの作成に関する詳細は、「ルールベースシナリオのテストシナリオテンプレートの作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細は、*67章 テストシナリオの定義* を参照してください。

注記

ルールベースのテストシナリオのカバレッジレポートを生成するには、ルールを最低でも 1 つ作成する必要があります。

手順

1. テストシナリオデザイナーでルールベースのテストシナリオを開きます。
2. 定義済みのテストシナリオを実行します。
3. テストシナリオデザイナーの右側にある **Coverage Report** をクリックして、テストカバレッジ統計を表示します。
4. (必要に応じて) テストシナリオのカバレッジレポートをダウンロードする場合は、**Download report** をクリックします。

74.2. DMN ベースのテストシナリオのカバレッジレポート生成

DMN ベースのテストシナリオには、**Coverage Report** タブに以下の詳細情報が含まれています。

- 利用可能なデシジョン数
- 実行したデシジョン数
- 実行したデシジョンの割合
- 実行したデシジョンの割合 (円グラフで表示)
- 各デシジョンを実行した回数
- 定義済みのテストシナリオごとに実行するデシジョン

以下の手順に従って、DMN ベースのテストシナリオのカバレッジレポートを生成します。

前提条件

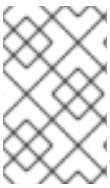
- 選択したテストシナリオ用に DMN ベースのテストシナリオテンプレートが作成されている。DMN ベースのテストシナリオの作成に関する詳細は、「DMN ベースのテストシナリオのテストシナリオテンプレート作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細は、*67章 テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーで DMN ベースのテストシナリオを開きます。
2. 定義済みのテストシナリオを実行します。
3. テストシナリオデザイナーの右側にある **Coverage Report** をクリックして、テストカバレッジ統計を表示します。
4. (必要に応じて) テストシナリオのカバレッジレポートをダウンロードする場合は、**Download report** をクリックします。

第75章 KIE SERVER REST API を使ったテストシナリオの実行

KIE Server の REST エンドポイントで直接対話することで、呼び出しコードと、意思決定ロジックの定義の分離が最大になります。外部のテストシナリオの実行には KIE Server REST API を使用できます。API は、デプロイ済みのプロジェクトにテストシナリオを実行します。



注記

この機能はデフォルトでは無効になっているため、**org.kie.scenariosimulation.server.ext.disabled** のシステムプロパティーを使用して有効化してください。

KIE Server REST API の詳細は、**KIE API を使用した Red Hat Process Automation Manager の操作**を参照してください。

前提条件

- KIE Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、**Red Hat Process Automation Manager インストールの計画** を参照してください。
- KJAR アーティファクトとしてプロジェクトをビルドし、KIE Server にデプロイしている。
- KIE コンテナーの ID がある。

手順

1. KIE Server REST API エンドポイントにアクセスするためのベース URL を決定します。これには、以下の値が必要です (例ではローカルデプロイメントのデフォルト値を使用しています)。
   - ホスト (**localhost**)
   - ポート (**8080**)
   - ルートコンテキスト (**kie-server**)
   - ベース REST パス (**services/rest/**)

   交通違反プロジェクトのローカルデプロイメントにおけるベース URL の例:

   http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT

2. ユーザー認証要件を決定します。
   ユーザーを KIE Server 設定に直接定義すると、ユーザー名およびパスワードを要求する HTTP Basic 認証が使用されます。要求を成功させるには、ユーザーに **kie-server** ルールが必要です。

   以下の例は、curl 要求に認証情報を追加する方法を示します。

   ```
   curl -u username:password <request>
   ```

   Red Hat Single Sign-On を使用して KIE Server を設定している場合は、要求にベアラートークンが必要です。

   ```
   curl -H "Authorization: bearer $TOKEN" <request>
   ```

3. 要求と応答の形式を指定します。REST API エンドポイントには XML 書式が利用でき、このエンドポイントは要求ヘッダーを使用して設定されます。

   XML

   ```
   curl -H "accept: application/xml" -H "content-type: application/xml"
   ```

4. テストシナリオを実行します。
   [POST] server/containers/{containerId}/scesim

   curl 要求例:

   ```
   curl -X POST "http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT/scesim"\ -u 'wbadmin:wbadmin;' \ -H "accept: application/xml" -H "content-type: application/xml"\ -d @Violation.scesim
   ```

   XML 要求例:

```
<ScenarioSimulationModel version="1.8">
 <simulation>
  <scesimModelDescriptor>
   <factMappings>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Index</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>#</name>
      <className>java.lang.Integer</className>
     </factIdentifier>
     <className>java.lang.Integer</className>
     <factAlias>#</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>70.0</columnWidth>
    </FactMapping>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Description</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>Scenario description</name>
      <className>java.lang.String</className>
     </factIdentifier>
     <className>java.lang.String</className>
     <factAlias>Scenario description</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>300.0</columnWidth>
    </FactMapping>
    <FactMapping>
     <expressionElements>
```

```
        <ExpressionElement>
          <step>Driver</step>
        </ExpressionElement>
        <ExpressionElement>
          <step>Points</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|1</name>
        <type>GIVEN</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Driver</name>
        <className>Driver</className>
      </factIdentifier>
      <className>number</className>
      <factAlias>Driver</factAlias>
      <expressionAlias>Points</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
    <FactMapping>
      <expressionElements>
        <ExpressionElement>
          <step>Violation</step>
        </ExpressionElement>
        <ExpressionElement>
          <step>Type</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|6</name>
        <type>GIVEN</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Violation</name>
        <className>Violation</className>
      </factIdentifier>
      <className>Type</className>
      <factAlias>Violation</factAlias>
      <expressionAlias>Type</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
    <FactMapping>
      <expressionElements>
        <ExpressionElement>
          <step>Violation</step>
        </ExpressionElement>
        <ExpressionElement>
          <step>Speed Limit</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|7</name>
        <type>GIVEN</type>
```

```
    </expressionIdentifier>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Violation</factAlias>
    <expressionAlias>Speed Limit</expressionAlias>
    <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
    <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
    <expressionElements>
      <ExpressionElement>
        <step>Violation</step>
      </ExpressionElement>
      <ExpressionElement>
        <step>Actual Speed</step>
      </ExpressionElement>
    </expressionElements>
    <expressionIdentifier>
      <name>0|8</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Violation</factAlias>
    <expressionAlias>Actual Speed</expressionAlias>
    <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
    <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
    <expressionElements>
      <ExpressionElement>
        <step>Fine</step>
      </ExpressionElement>
      <ExpressionElement>
        <step>Points</step>
      </ExpressionElement>
    </expressionElements>
    <expressionIdentifier>
      <name>0|11</name>
      <type>EXPECT</type>
    </expressionIdentifier>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Fine</factAlias>
    <expressionAlias>Points</expressionAlias>
    <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
    <columnWidth>114.0</columnWidth>
```

```
      </FactMapping>
      <FactMapping>
        <expressionElements>
          <ExpressionElement>
            <step>Fine</step>
          </ExpressionElement>
          <ExpressionElement>
            <step>Amount</step>
          </ExpressionElement>
        </expressionElements>
        <expressionIdentifier>
          <name>0|12</name>
          <type>EXPECT</type>
        </expressionIdentifier>
        <factIdentifier>
          <name>Fine</name>
          <className>Fine</className>
        </factIdentifier>
        <className>number</className>
        <factAlias>Fine</factAlias>
        <expressionAlias>Amount</expressionAlias>
        <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
        <columnWidth>114.0</columnWidth>
      </FactMapping>
      <FactMapping>
        <expressionElements>
          <ExpressionElement>
            <step>Should the driver be suspended?</step>
          </ExpressionElement>
        </expressionElements>
        <expressionIdentifier>
          <name>0|13</name>
          <type>EXPECT</type>
        </expressionIdentifier>
        <factIdentifier>
          <name>Should the driver be suspended?</name>
          <className>Should the driver be suspended?</className>
        </factIdentifier>
        <className>string</className>
        <factAlias>Should the driver be suspended?</factAlias>
        <expressionAlias>value</expressionAlias>
        <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
        <columnWidth>114.0</columnWidth>
      </FactMapping>
    </factMappings>
  </scesimModelDescriptor>
  <scesimData>
    <Scenario>
      <factMappingValues>
        <FactMappingValue>
          <factIdentifier>
            <name>Scenario description</name>
            <className>java.lang.String</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>Description</name>
```

```
      <type>OTHER</type>
    </expressionIdentifier>
    <rawValue class="string">Above speed limit: 10km/h and 30 km/h</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Driver</name>
      <className>Driver</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|1</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">10</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|6</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">&amp;quot;speed&amp;quot;</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|7</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">100</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|8</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">120</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|11</name>
      <type>EXPECT</type>
```

```
          </expressionIdentifier>
          <rawValue class="string">3</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>Fine</name>
            <className>Fine</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>0|12</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">500</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>Should the driver be suspended?</name>
            <className>Should the driver be suspended?</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>0|13</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">&amp;quot;No&amp;quot;</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>#</name>
            <className>java.lang.Integer</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>Index</name>
            <type>OTHER</type>
          </expressionIdentifier>
          <rawValue class="string">1</rawValue>
        </FactMappingValue>
      </factMappingValues>
    </Scenario>
  </scesimData>
</simulation>
<background>
  <scesimModelDescriptor>
    <factMappings>
      <FactMapping>
        <expressionElements/>
        <expressionIdentifier>
          <name>1|1</name>
          <type>GIVEN</type>
        </expressionIdentifier>
        <factIdentifier>
          <name>Empty</name>
          <className>java.lang.Void</className>
        </factIdentifier>
        <className>java.lang.Void</className>
        <factAlias>Instance 1</factAlias>
        <expressionAlias>PROPERTY 1</expressionAlias>
```

```
          <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
          <columnWidth>114.0</columnWidth>
        </FactMapping>
      </factMappings>
    </scesimModelDescriptor>
    <scesimData>
      <BackgroundData>
        <factMappingValues>
          <FactMappingValue>
            <factIdentifier>
              <name>Empty</name>
              <className>java.lang.Void</className>
            </factIdentifier>
            <expressionIdentifier>
              <name>1|1</name>
              <type>GIVEN</type>
            </expressionIdentifier>
          </FactMappingValue>
        </factMappingValues>
      </BackgroundData>
    </scesimData>
  </background>
  <settings>
    <dmnFilePath>src/main/resources/org/kie/example/traffic/traffic_violation/Traffic
Violation.dmn</dmnFilePath>
    <type>DMN</type>
    <fileName></fileName>
    <dmnNamespace>https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-
93B2-A2598F19ECFF</dmnNamespace>
    <dmnName>Traffic Violation</dmnName>
    <skipFromBuild>false</skipFromBuild>
    <stateless>false</stateless>
  </settings>
  <imports>
    <imports/>
  </imports>
</ScenarioSimulationModel>
```

XML 応答例:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Test Scenario successfully executed">
    <scenario-simulation-result>
        <run-count>5</run-count>
        <ignore-count>0</ignore-count>
        <run-time>31</run-time>
    </scenario-simulation-result>
</response>
```

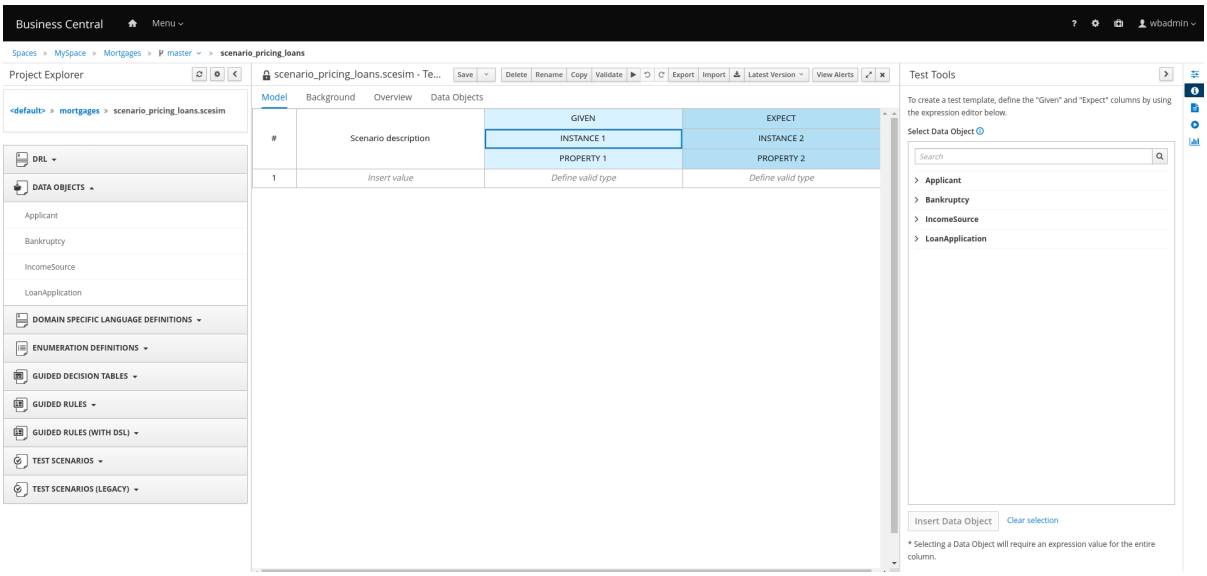
第76章 MORTGAGES サンプルプロジェクトを使用したテストシナリオの作成

本章では、テストシナリオデザイナーを使用して、Business Central に同梱されているサンプルの **Mortgages** プロジェクトからテストシナリオを作成して実行する方法を説明します。本章のテストシナリオの例は、**Mortgages** プロジェクトからの **Pricing loans** のガイド付きデシジョンテーブルに基づいています。

手順

1. Business Central にログインし、**Menu → Design → Projects** の順にクリックし、**Mortgages** をクリックします。
2. プロジェクトが **Projects** に表示されていない場合は、**MySpace** から **Try Samples → Mortgages → OK** の順にクリックします。
   アセットのウィンドウが表示されます。
3. **Add Asset → Test Scenario** の順にクリックします。
4. **Test Scenario** の名前として、**scenario_pricing_loans** を入力し、**Package** ドロップダウンリストから、デフォルトの **mortgages.mortgages** パッケージを選択します。
   選択するパッケージには、必要なルールアセットがすべて含まれている必要があります。
5. **Source type** として **RULE** を選択します。
6. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。
7. **Project Explorer** を展開して以下を確認します。
   - **Applicant**、**Bankruptcy**、**IncomeSource**、および **LoanApplication** データオブジェクトが存在する。
   - **Pricing loans** ガイド付きのデシジョンテーブルが存在する。
   - 新しいテストシナリオが **Test Scenario** に表示されていることを確認する。
8. データオブジェクトがすべて配置されていることを確認してから、テストシナリオデザイナーの **Model** タブに戻り、利用可能なデータオブジェクトのシナリオに、**GIVEN** データと **EXPECT** データを定義します。

図76.1 空のテストシナリオデザイナー



9. **GIVEN** 列の詳細を定義します。

    a. **GIVEN** 列ヘッダーにある **INSTANCE 1** という名前のセルをクリックします。

    b. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択します。

    c. **Insert Data Object** をクリックします。

10. データオブジェクトのプロパティーを作成するには、必要に応じてプロパティーヘッダーセルを右クリックして、**Insert column right** または **Insert column left** を選択します。この例では、**GIVEN** 列の下に。プロパティーセルをさらに 2 つ作成する必要があります。

11. 最初のプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. **amount** をクリックします。

    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。

12. 2 番目のプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. **deposit** をクリックします。

    c. **Insert Data Object** をクリックします。

13. 3 番目のプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. **lengthYears** をクリックします。

    c. **Insert Data Object** をクリックします。

14. **LoanApplication** ヘッダーセルを右クリックし、**Insert column right** を選択します。右側に新しい **GIVEN** 列が作成されます。

15. 新しいヘッダーセルを選択します。
    a. **Test Tools** パネルから、**IncomeSource** データオブジェクトを選択します。
    b. **Insert Data Object** をクリックして、データオブジェクトフィールドをヘッダーセルにマッピングします。
16. **IncomeSource** の下のプロパティーヘッダーセルをクリックします。
    a. **Test Tools** パネルから、**IncomeSource** データオブジェクトを選択して展開します。
    b. **type** をクリックします。
    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。
    GIVEN 列セルがすべて定義されました。
17. 次に **EXPECT** 列の詳細を定義します。
    a. **EXPECT** 列ヘッダーにある **INSTANCE 2** という名前のセルをクリックします。
    b. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択します。
    c. **Insert Data Object** をクリックします。
18. データオブジェクトのプロパティーを作成するには、必要に応じてプロパティーヘッダーセルを右クリックして、**Insert column right** または **Insert column left** を選択します。**EXPECT** 列の下に、プロパティーセルをさらに 2 つ作成します。
19. 最初のプロパティーヘッダーセルを選択します。
    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。
    b. **Approved** をクリックします。
    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。
20. 2 番目のプロパティーヘッダーセルを選択します。
    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。
    b. **insuranceCost** をクリックします。
    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。
21. 3 番目のプロパティーヘッダーセルを選択します。
    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。
    b. **approvedRate** をクリックします。
    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。
22. テストシナリオを定義するには、1 行目に以下のデータを入力します。
    - **GIVEN** 列の値として、**Scenario Description** には **Row 1 test scenario**、amount には

150000、deposit には **19000**、lengthYears には **30**、type には **Asset** を入力します。

- EXPECT 列の値として、approved には **true**、insuranceCost には **0**、approvedRate には **2** を入力します。

23. 次に 2 番目の行に、以下のデータを入力します。

    - GIVEN 列の値として、Scenario Description には **Row 2 test scenario**、amount には **100002**、deposit には **2999**、lengthYears には **20**、type には **Job** を入力します。

    - EXPECT 列の値として、approved には **true**、insuranceCost には **10**、approvedRate には **6** を入力します。

24. シナリオに対して、GIVEN、EXPECT、その他のデータをすべて定義したら、テストシナリオデザイナーで Save をクリックして、設定した内容を保存します。

25. 右上隅の Run Test をクリックして **.scesim** ファイルを実行します。
テスト結果は、Test Report パネルに表示されます。View Alerts をクリックして、Alerts セクションからメッセージを表示します。テストに失敗した場合は、ウィンドウの下部にある Alerts セクションのメッセージを参照し、シナリオの全コンポーネントをレビューして修正してから、シナリオが合格するまでシナリオの検証を再度行います。

26. テストシナリオデザイナーで Save をクリックして必要な変更をすべて加えてから、作業を保存します。

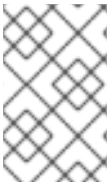
第77章 BUSINESS CENTRAL でのテストシナリオ (レガシー) デザイナー

Red Hat Process Automation Manager は現在、新規の **テストシナリオ** デザイナーと以前の **テストシナリオ (レガシー)** デザイナーの両方をサポートします。デフォルトのデザイナー、新規のテストシナリオデザイナーで、ルールと DMN モデルのテストをサポートし、テストシナリオの全体的な使用感が改善されています。必要に応じて、レガシーのテストシナリオをそのまま使用することができますが、ルールベースのテストシナリオしかサポートされません。

77.1. テストシナリオ (レガシー) の作成および実行

ビジネスルールデータをデプロイする前に、Business Central にテストシナリオを作成して、その機能をテストできます。基本的なテストシナリオには、少なくても以下のデータが必要です。

- 関連するデータオブジェクト
- **GIVEN (指定した)** ファクト
- **EXPECT (想定される)** 結果



注記

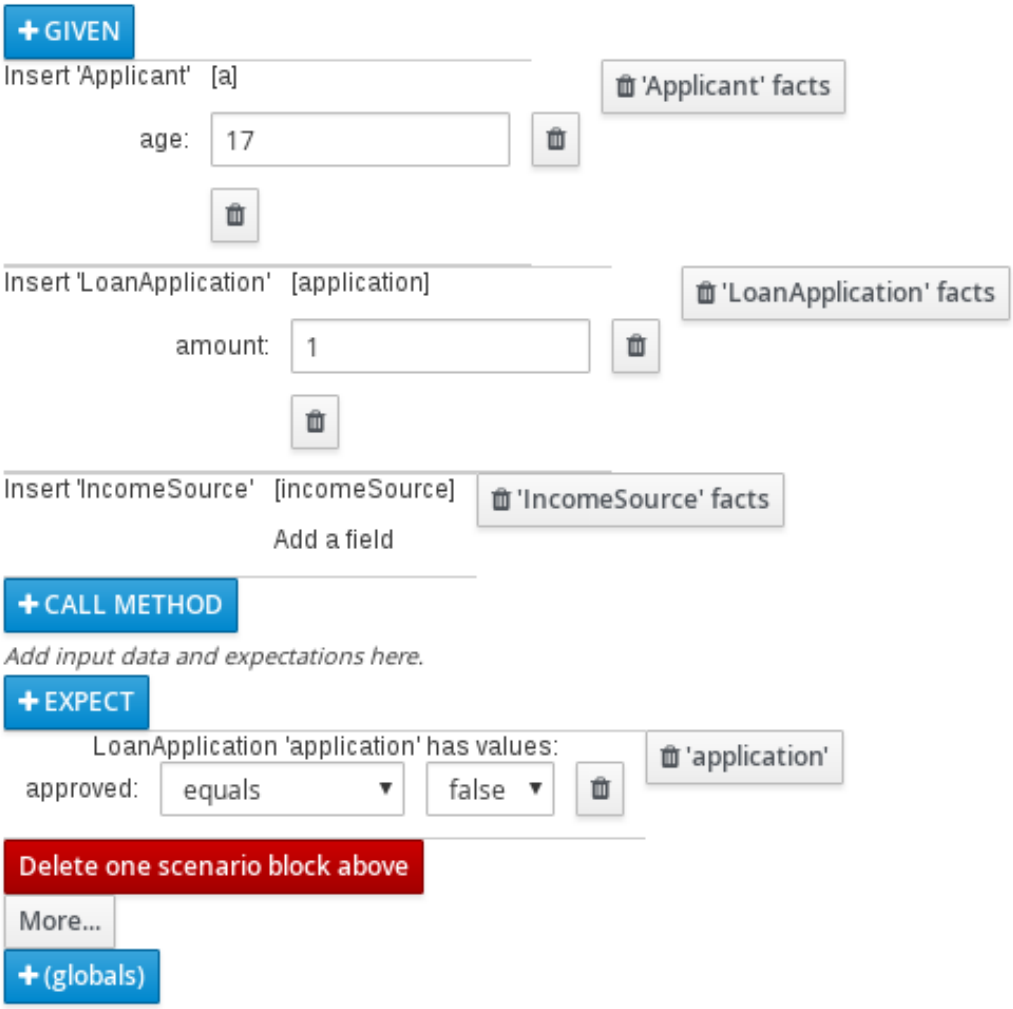
レガシーのテストシナリオデザイナーは、**LocalDate** の java 組み込みデータタイプをサポートします。**LocalDate** の java 組み込みデータタイプでは **dd-mmm-yyyy** の日付形式を使用できます。たとえば、**17-Oct-2020** の日付形式で設定できます。

テストシナリオでは、このデータを使用し、定義したファクトに基づいて、そのルールインスタンスに対して想定した結果と実際の結果の妥当性を検証できます。**CALL METHOD** と利用可能な **globals** をテストシナリオに追加することもできますが、これは必須ではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Test Scenarios (Legacy)** の順にクリックします。
3. **テストシナリオ** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なルールアセットが割り当てられている、またはこれから割り当てるパッケージと同じにする必要があります。データオブジェクトは、パッケージからアセットのデザイナーにインポートできます。
4. **OK** をクリックして、テストシナリオを作成します。
   Project Explorer の **Test Scenarios** パネルに、新しいテストシナリオが追加されました。
5. **Data Objects** タブをクリックして、テストするルールに必要なデータオブジェクトがすべてリストされていることを検証します。追加されていない場合は、**New item** をクリックして別のパッケージから必要なデータオブジェクトをインポートするか、パッケージに データオブジェクトを作成 します。
6. データオブジェクトをすべて配置したら、テストシナリオデザイナーの **Model** タブに戻り、利用可能なデータオブジェクトに基づいたシナリオに、**GIVEN** データと **EXPECT** データを定義します。

図77.1 テストシナリオデザイナー



GIVEN セクションには、テストする入力ファクトを定義します。たとえば、プロジェクトの **Underage** ルールで、ローン申請者の年齢が 21 歳未満であれば承認しない場合は、テストシナリオの GIVEN ファクトで、**Applicant** の **age** に、21 より小さい数字に設定する必要があります。

EXPECT セクションには、GIVEN に入力したファクトに基づいて想定される結果を定義します。つまり、入力ファクトを **GIVEN (指定)** すると、その他のファクトが有効であること、またはルール全体が有効であることを **EXPECT (想定)** します。たとえば、このシナリオで、申請者が 21 歳未満の場合に **想定される** 結果は、(申請者の年齢が基準を満たさないため) **LoanApplication** の **approved** が **false** になるか、**Underage** ルール全体が有効になります。

7. 必要に応じて、CALL METHOD と globals をテストシナリオに追加します。

- CALL METHOD: ルールの実行を開始する際に、別のファクトからメソッドを呼び出します。CALL METHOD をクリックし、ファクトを選択し、 をクリックして呼び出すメソッドを選択します。(可能な場合は) プロジェクトに対してインポートした Java ライブラリー、または JAR から (ArrayList のメソッドなどの) Java クラスメソッドを呼び出すことができます。

- globals: テストシナリオで妥当性を確認するプロジェクトにグローバル変数を追加します。globals をクリックして、妥当性を確認する変数を選択し、テストシナリオデザイナーでグローバル名をクリックして、グローバル変数に適用するフィールド値を定義します。グローバル変数が利用できない場合は、Business Central に新しいアセットとして作成する

必要があります。グローバル変数はデシジョンエンジンに表示されますが、ファクトに対するオブジェクトとは異なるオブジェクトの名前です。グローバルのオブジェクトを変更しても、ルールの再評価は行われません。

8. 必要に応じて、テストシナリオデザイナーの下部で **More** をクリックし、同じシナリオファイルに別のデータブロックを追加します。

9. シナリオに対して、**GIVEN**、**EXPECT**、その他のデータをすべて定義したら、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

10. 右上の **Run scenario** をクリックして、この **.scenario** ファイルを実行します。プロジェクトパッケージに **.scenario** ファイルが複数ある場合にすべて保存している場合は、**Run all scenarios** をクリックして、すべてのシナリオを実行します。**Run scenario** オプションでは、個々の **.scenario** ファイルを保存する必要はありませんが、**Run all scenarios** オプションを使用する場合は、すべての **.scenario** ファイルを保存する必要があります。
テストに失敗したら、ウィンドウ下部の **Alerts** メッセージに記載されている問題に対応し、シナリオの全コンポーネントを見直し、エラーが表示されなくなるまで妥当性確認を行います。

11. 変更がすべて終了したら、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

77.1.1. テストシナリオ (レガシー) への GIVEN ファクトの追加

GIVEN セクションには、テストする入力ファクトを定義します。たとえば、プロジェクトの **Underage** ルールで、ローン申請者の年齢が 21 歳未満であれば承認しない場合は、テストシナリオの GIVEN ファクトで、**Applicant** の **age** に、21 より小さい数字に設定する必要があります。

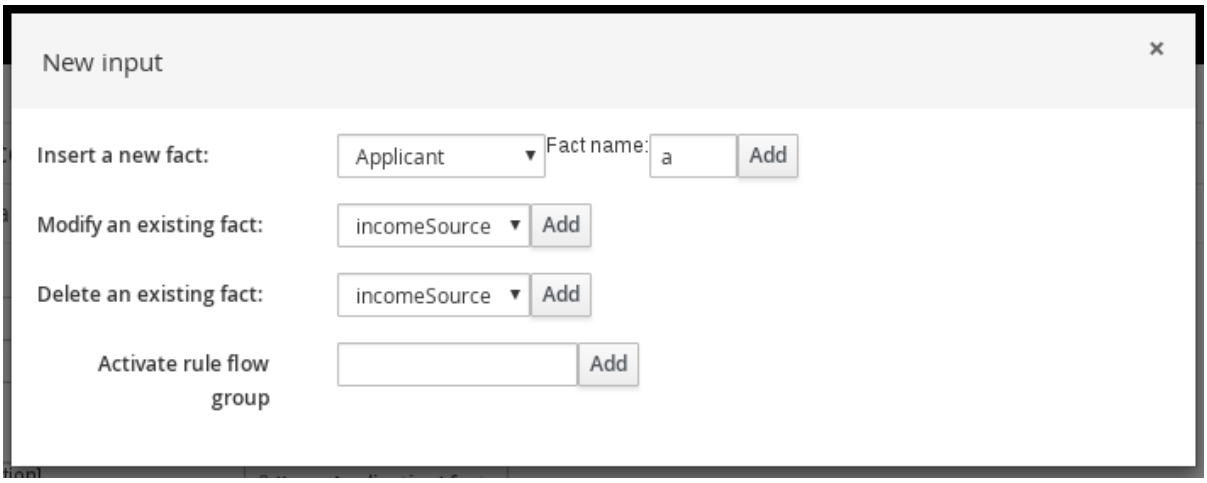
前提条件

- テストシナリオに必要なデータオブジェクトがすべて作成、またはインポートされていて、**テストシナリオ (レガシー)** デザイナーの **Data Objects** タブにリストされている。

手順

1. **テストシナリオ (レガシー)** デザイナーで、**GIVEN** をクリックして、利用可能なファクトが含まれる **New input** ウィンドウを開きます。

図77.2 テストシナリオへの GIVEN 入力の追加

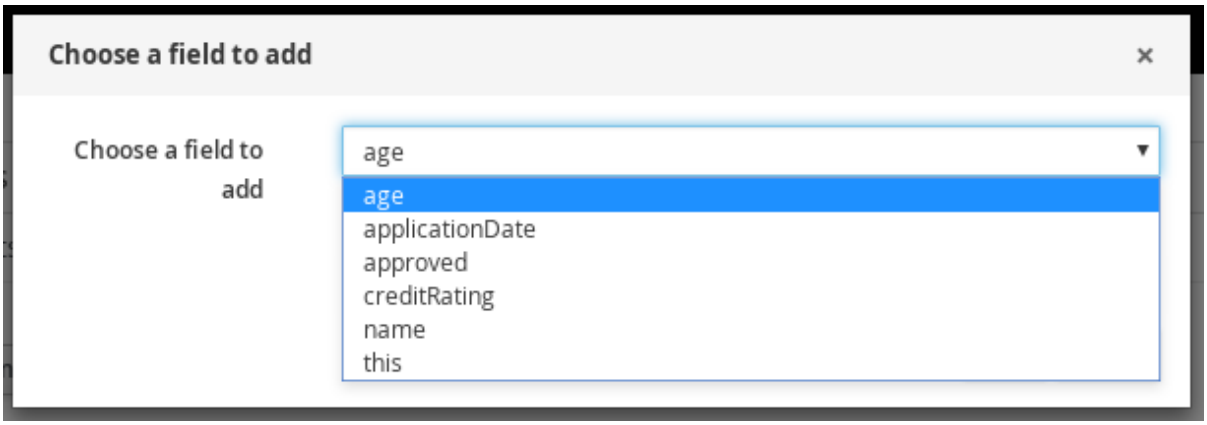


リストには以下のオプションが含まれます。表示されるオプションは、テストシナリオデザイナーの **Data Objects** タブで利用可能なデータオブジェクトによって異なります。

- **Insert a new fact:** ファクトを追加して、フィールド値を修正します。**Fact name** にファクトの変数を入力します。
- **Modify an existing fact:** 別のファクトが追加される場合に限り表示されます。 これを使用して、シナリオの実行間でデシジョンエンジンで変更される前に挿入されたファクトを指定します。
- **Delete an existing fact:** 別のファクトが追加される場合に限り表示されます。 これを使用して、シナリオの実行間でデシジョンエンジンで削除される前に挿入されたファクトを指定します。
- **Activate rule flow group:** そのグループ内にあるすべてのルールをテストできるように、有効にするルールフローグループを指定します。

2. 目的の入力オプションに対するファクトを選択し、**Add** をクリックします。たとえば、**Insert a new fact:** に **Applicant** を設定し、**Fact name** に対して **a** または **app**、もしくは別の変数を入力します。
3. テストシナリオデザイナーのファクトをクリックし、修正するフィールドを選択します。

図77.3 ファクトフィールドの修正



4. 編集アイコン ( ) をクリックし、以下のフィールド値を選択します。
   - **Literal value:** 特定のリテラル値を入力するオープンフィールドを作成します。
   - **Bound variable:** このフィールドの値を、選択した変数にバインドするファクトに設定します。フィールドタイプが、バインドした変数型に一致する必要があります。
   - **Create new fact:** 新しいファクトを作成し、そのファクトを親ファクトのフィールド値として割り当てます。テストシナリオデザイナーで子ファクトをクリックし、同じようにフィールド値を割り当てるか、別のファクトをネストできます。
5. 続いて、シナリオに別の **GIVEN** 入力データを追加し、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

77.1.2. テストシナリオ (レガシー) への EXPECT 結果の追加

EXPECT セクションには、**GIVEN** に入力したファクトに基づいて想定される結果を定義します。つまり、入力ファクトを **GIVEN (指定)** すると、その他のファクトが有効であること、またはルール全体が有効であることを **EXPECT (想定)** します。たとえば、このシナリオで、申請者が 21 歳未満の場合に **想定される** 結果は、(申請者の年齢が基準を満たさないため) **LoanApplication** の **approved** が **false** になるか、**Underage** ルール全体が有効になります。

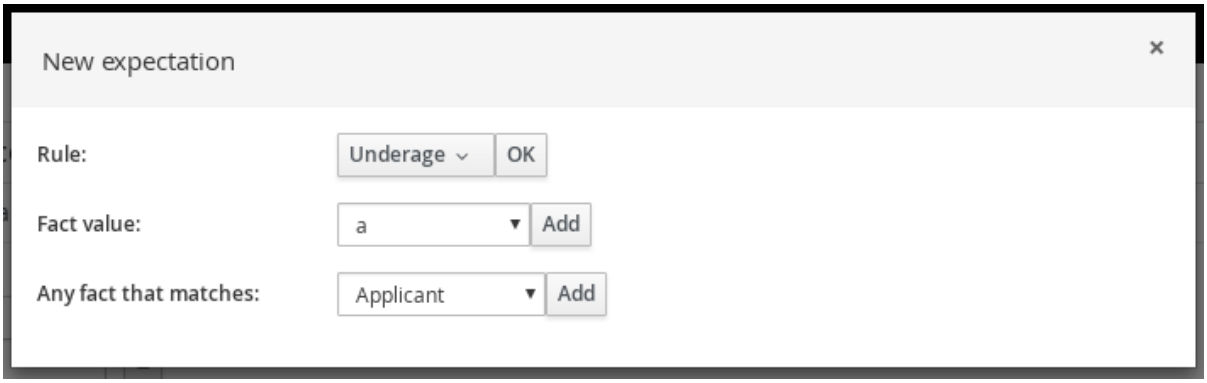
前提条件

- テストシナリオに必要なデータオブジェクトがすべて作成、またはインポートされていて、**テストシナリオ (レガシー)** デザイナーの **Data Objects** タブにリストされている。

手順

1. **テストシナリオ (レガシー)** デザイナーで、**EXPECT** をクリックして、利用可能なファクトが含まれる **New expectation** ウィンドウを開きます。

図77.4 テストシナリオへの EXPECT 結果の追加



リストには以下のオプションが含まれます。表示されるオプションは、**GIVEN** セクションのデータや、テストシナリオデザイナーの **Data Objects** タブで利用可能なデータオブジェクトによって異なります。

- **Rule:** プロジェクトに、**GIVEN** に指定した内容に対して有効になることが想定される特定のルールを指定します。ルールの名前を入力するか、ルールリストから選択します。次に、テストシナリオデザイナーで、ルールが有効になるべき回数を指定します。

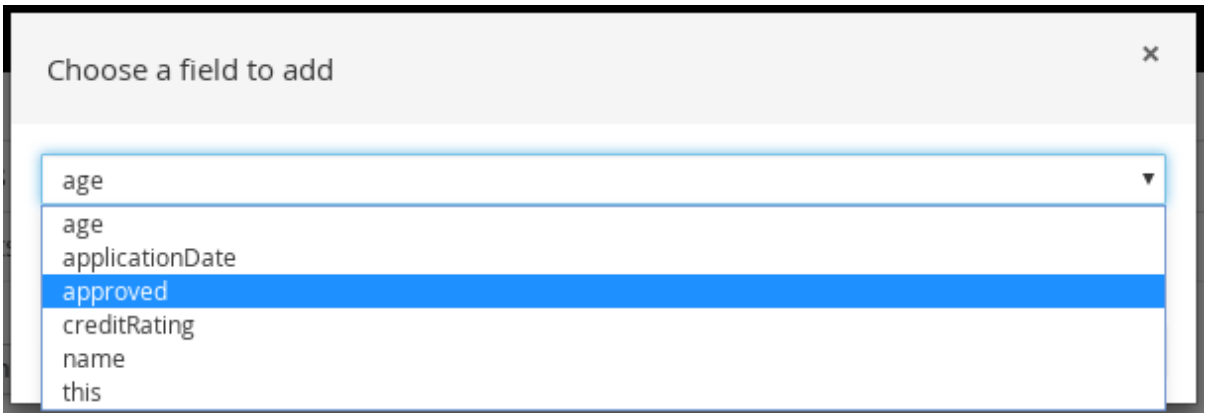
- **Fact value:** ファクトを選択し、**GIVEN** セクションに定義したファクトに対して有効になることが想定される値を定義します。ファクトは、**GIVEN** の入力に対して事前に定義した **Fact name** でリストされます。

- **Any fact that matches: GIVEN** に指定した内容に対して、指定した値を持つファクトが最低 1 つ存在するかどうかの妥当性を確認します。

2. (**Fact value: application** などの) 期待される結果のファクトを選択し、**Add** または **OK** を選択します。

3. テストシナリオデザイナーでファクトをクリックし、追加または修正するフィールドを選択します。

図77.5 ファクトフィールドの修正



4. フィールド値に、GIVEN に指定した内容に対して、有効になると想定される値 (**approved** | **equals** | **false** など) を設定します。

注記

レガシーのテストシナリオデザイナーでは、EXPECT フィールドに **["value1", "value2"]** の文字列形式を使用して文字列の一覧を検証できます。

5. 続いて、シナリオに別の EXPECT 入力データを追加し、テストシナリオデザイナーで Save をクリックして、設定内容を保存します。

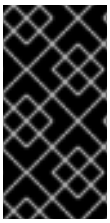
6. シナリオに GIVEN、EXPECT、その他のデータを定義して保存したら、右上の Run scenario をクリックしてこの **.scenario** ファイルを実行するか、(複数の場合は) Run all scenarios をクリックして、プロジェクトパッケージに保存したすべての **.scenario** を実行します。Run scenario オプションでは、個々の **.scenario** ファイルを保存する必要はありませんが、Run all scenarios オプションを使用する場合は、すべての **.scenario** ファイルを保存する必要があります。
テストに失敗したら、ウィンドウ下部の Alerts メッセージに記載されている問題に対応し、シナリオの全コンポーネントを見直し、エラーが表示されなくなるまで妥当性確認を行います。

7. 変更がすべて終了したら、テストシナリオデザイナーで Save をクリックして、設定した内容を保存します。

第78章 従来のテストシナリオデザイナーと新しいテストシナリオデザイナーの機能比較

Red Hat Process Automation Manager は、新しいテストシナリオデザイナーと以前のテストシナリオ (レガシー) デザイナーの両方をサポートします。

デフォルトのデザイナー、新規のテストシナリオデザイナーで、ルールと DMN モデルのテストをサポートし、テストシナリオの全体的な使用感が改善されています。必要に応じて、レガシーのテストシナリオをそのまま使用することはできますが、ルールベースのテストシナリオしかサポートされません。



重要

新しいテストシナリオデザイナーのレイアウトと機能セットは改善されており、開発が続けられています。ただし、レガシーのテストシナリオデザイナーは、Red Hat Process Automation Manager 7.3.0 で非推奨になります。このツールは、今後の Red Hat Process Automation Manager リリースで削除予定です。

以下の表では、プロジェクトに適したテストシナリオデザイナーを決定できるように、Red Hat Process Automation Manager でサポートされているレガシーおよび新しいテストシナリオデザイナーの主要機能を取り上げています。

- **+** は、機能がテストシナリオデザイナーに含まれていることを示します。
- **-** は、機能がテストシナリオデザイナーに含まれていないことを示します。

表78.1 レガシーおよび新しいテストシナリオデザイナーの主な機能

機能および特徴	新しいデザイナー	従来のデザイナー	ドキュメント
テストシナリオの作成および実行 ● ビジネスルールデータをデプロイする前に、Business Central にテストシナリオを作成して、その機能をテストできます。 ● 基本的なテストシナリオには、少なくても **GIVEN** ファクトと、**EXPECT** 結果など、関連のデータオブジェクトが必要です。 ● テストを実行して、ビジネスルールとデータを検証できます。	+	+	● ルールおよび DMN ベースのテストシナリオの作成に関する詳細は、*65章 テストシナリオテンプレート* を参照してください。 ● テストシナリオの実行に関する詳細は、*71章 テストシナリオの実行* を参照してください。 ● テストシナリオ (レガシー) の作成と実行に関する詳細は、「テストシナリオ (レガシー) の作成および実行」 を参照してください。

機能および特徴	新しいデザイナー	従来のデザイナー	ドキュメント
テストシナリオへの GIVEN ファクトの追加 ● テストの GIVEN ファクトは、挿入して検証できます。	+	+	● 新規テストシナリオデザイナーへの GIVEN ファクトの追加に関する詳細は、65章 *テストシナリオテンプレート* を参照してください。 ● テストシナリオ (レガシー) の GIVEN ファクトの追加に関する詳細は、「テストシナリオ (レガシー) への GIVEN ファクトの追加」 を参照してください。
テストシナリオへの EXPECT 結果の追加 ● EXPECT セクションには、GIVEN に入力したファクトに基づいて想定される結果を定義します。 ● これは、オブジェクトとそのフィールドを表し、これらの正確な値が指定の情報に基づいてチェックされます。	+	+	● 新規テストシナリオデザイナーへの EXPECT 結果の追加に関する詳細は、65章 *テストシナリオテンプレート* を参照してください。 ● テストシナリオ (レガシー) への EXPECT 結果の追加に関する詳細は、「テストシナリオ (レガシー) への EXPECT 結果の追加」 を参照してください。
KIE セッション ● テストシナリオのレベルオプションで、KIE セッションを設定できます。	+	+	NA
テストシナリオレベルの KIE ベース ● テストシナリオのレベルオプションで KIE ベースを設定できます。	-	+	NA
プロジェクトレベルの KIE ベース ● プロジェクトレベル設定で、KIE ベースを設定できます。	+	+	NA

機能および特徴	新しいデザイナー	従来のデザイナー	ドキュメント
シミュレーション日時 ● レガシーテストシナリオデザイナーに、シミュレーション日時を設定できます。	-	+	NA
ルールフローグループ ● アクティベートするルールフローグループを指定して、対象グループ内で全ルールをテストできます。	+	+	● 新規テストシナリオでルールフローグループを設定する方法は、「ルールベースのテストシナリオのグローバル設定の設定」 を参照してください。 ● テストシナリオ (レガシー) でのルールフローグループを設定する方法は、「テストシナリオ (レガシー) への GIVEN ファクトの追加」 を参照してください。
グローバル変数 ● グローバル変数はデシジョンエンジンに表示されますが、ファクトに対するオブジェクトとは異なるオブジェクトの名前です。 ● 新しいテストシナリオでは、グローバル変数の設定は非推奨です。 ● 異なるシナリオにデータセットを再利用する場合には、**Background** インスタンスを使用できます。	-	+	● 新規テストシナリオでの **Background** インスタンスの詳細は、*68章テストシナリオでのバックグラウンドインスタンス* を参照してください。 ● テストシナリオ (レガシー) でのグローバル変数の詳細は、「テストシナリオ (レガシー) の作成および実行」 を参照してください。

機能および特徴	新しいデザイナー	従来のデザイナー	ドキュメント
呼び出しメソッド ● ルール実行を開始するときに、別のファクトからメソッドを呼び出す場合にはこちらを使用します。 ● Java ライブラリーや、プロジェクト用にインポートした JAR から Java クラスメソッドを呼び出すことができます。	+	+	● 新規テストシナリオでメソッドを呼び出す方法は、70章 *テストシナリオの式構文* を参照してください。 ● テストシナリオ (レガシー) でメソッドを呼び出す方法は、「テストシナリオ (レガシー) の作成および実行」 を参照してください。
既存ファクトの変更 ● 次回のシナリオ実行までに、デシジョンエンジンで以前に挿入したファクトを変更できます。	-	+	テストシナリオ (レガシー) で既存のファクトを変更する方法は、「テストシナリオ (レガシー) への GIVEN ファクトの追加」 を参照してください。
バインド変数 ● 選択した変数にバインドするファクトに、フィールドの値を設定できます。 ● 新規テストシナリオデザイナーでは、テストシナリオグリッドに変数を定義して、**GIVEN** セルまたは **EXPECTED** セルで再利用できません。	-	+	テストシナリオ (レガシー) にバインド変数を設定する方法は、「テストシナリオ (レガシー) への GIVEN ファクトの追加」 を参照してください。

第79章 次のステップ

Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

パート IX. RED HAT PROCESS AUTOMATION MANAGER のデシジョンエンジン

ビジネスルールの開発者として、Red Hat Process Automation Manager のデシジョンエンジンを理解することで、より効果的なビジネスアセットおよびよりスケーラブルなデシジョン管理アーキテクチャーの設計が可能となります。デシジョンエンジンは、Red Hat Process Automation Manager のコンポーネントで、データを保存、処理、および評価してビジネスルールを実行し、お客様が定義したデシジョンを達成します。本書では、お客様が Red Hat Process Automation Manager でビジネスルールシステムおよびデシジョンサービスを作成する際に検討すべき、デシジョンエンジンに関する基本的な概念および機能を説明します。

第80章 RED HAT PROCESS AUTOMATION MANAGER のデシジョンエンジン

デシジョンエンジンは、Red Hat Process Automation Manager のルールエンジンです。デシジョンエンジンは、データを保存、処理、および評価して、お客様が定義するビジネスルールまたはデシジョンモデルを実行します。デシジョンエンジンの基本的な機能は、受信データ、または **ファクト** をルールの条件に一致させ、そのルールを実行するかどうか、そしてどのように実行するかを決定することです。

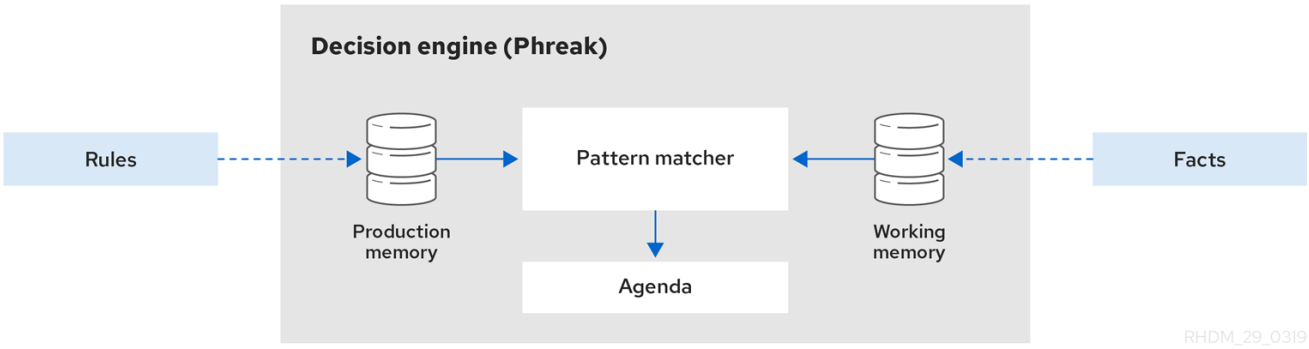
デシジョンエンジンは、以下の基本コンポーネントを使用して動作します。

- **ルール**: お客様が定義するビジネスルールまたは DMN デシジョン。すべてのルールは、ルールをトリガーする条件およびルールが指示するアクションを最小限含む必要があります。
- **ファクト**: デシジョンエンジンで入力または変更されるデータで、デシジョンエンジンをルールの条件と一致させ、適用可能なルールを実行します。
- **プロダクションメモリー**: デシジョンエンジンのルールが格納されている場所。
- **ワーキングメモリー**: デシジョンエンジンのファクトが格納されている場所。
- **アジェンダ**: 有効化されたルールが登録され、実行に備えて (必要に応じて) 並べ替えられた場所。

ビジネスユーザーまたは自動化システムが Red Hat Process Automation Manager にルール関連の情報を追加または更新した場合、その情報は 1 つ以上のファクトという形でデシジョンエンジンのワーキングメモリーに挿入されます。デシジョンエンジンは、それらのファクトをプロダクションメモリーに格納されたルールの条件と一致させ、適用可能なルールの実行を決定します。ファクトをルールに一致させるこの処理は、しばしば **パターン一致** と呼ばれます。 ルールの条件が一致すると、デシジョンエンジンはアジェンダのルールを有効にして登録します。続いてアジェンダでは、デシジョンエンジンが実行に備えて優先的なルールまたは競合するルールをソートします。

以下の図は、デシジョンエンジンのこれらの基本コンポーネントを表しています。

図80.1 基本となるデシジョンエンジンコンポーネントの概要



デシジョンエンジンのルールやファクトの動作に関する詳細や例は、*82章 デシジョンエンジンにおける推論と真理維持* を参照してください。

これらのコアなコンセプトにより、デシジョンエンジンの他のより高度なコンポーネント、プロセス、およびサブプロセスについての理解が深まるようになります。その結果、Red Hat Process Automation Manager でより効果的なビジネスアセットを設計できるようになります。

第81章 KIE セッション

Red Hat Process Automation Manager では、KIE セッションはランタイムデータを保存して実行します。KIE セッションは KIE ベースから作成されるか、またはプロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で KIE セッションを定義している場合は、KIE コンテナーから直接作成されます。

kmodule.xml ファイルの KIE セッション設定例

```
<kmodule>
  ...
  <kbase>
    ...
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
    ...
  </kbase>
  ...
</kmodule>
```

KIE ベースは、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で定義するリポジトリーで、Red Hat Process Automation Manager のすべてのルール、プロセス、およびその他のビジネスアセットが含まれていますが、ランタイムのデータは一切含まれていません。

kmodule.xml ファイルの KIE ベース設定例

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality"
declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

KIE セッションは、ステートレスでも、ステートフルでも可能です。ステートレスな KIE セッションでは、KIE セッションの以前の呼び出し (以前のセッション状態) からのデータは、次のセッションの呼び出しまでに破棄されます。ステートフルな KIE セッションでは、そのデータは保持されます。使用する KIE セッションのタイプは、プロジェクトの要件と、さまざまなアセット呼び出しからのデータをどのように維持するかにより決まります。

81.1. ステートレスな KIE セッション

ステートレスな KIE セッションは、推論を使用せずに、時間の経過とともにファクトを繰り返し変更していくセッションです。ステートレスな KIE セッションでは、KIE セッションの以前の呼び出し (以前のセッション状態) からのデータはセッションの呼び出し間で破棄されますが、ステートフルな KIE セッションではそのデータは保持されます。ステートレスな KIE セッションは、生成する結果が KIE ベースのコンテンツと、特定の時点で実行するために KIE セッションに渡されるデータによって決定されるという点で、関数と同様に動作します。KIE セッションには、以前に KIE セッションに渡されたデータのメモリーはありません。

以下のユースケースで、ステートレスな KIE セッションは一般的に使用されます。

- **検証** (住宅ローンの対象となるかを検証するなど)
- **計算** (住宅ローンのプレミアムの計算など)

- **ルーティングとフィルターリング** (受信した電子メールをフォルダーにソートしたり、受信した電子メールを送信先に送信したりすることなど)

たとえば、以下の運転免許のデータモデルおよび DRL ルールのサンプルをご覧ください。

運転免許申請のデータモデル

```
public class Applicant {
  private String name;
  private int age;
  private boolean valid;
  // Getter and setter methods
}
```

運転免許申請の DRL ルールのサンプル

```
package com.company.license

rule "Is of valid age"
when
  $a : Applicant(age < 18)
then
  $a.setValid(false);
end
```

Is of valid age ルールは、18 歳未満の申請者全員を失格にします。**Applicant** オブジェクトがデシジョンエンジンに挿入されると、デシジョンエンジンは各ルールの制約を評価し、一致するものを探します。**"objectType"** 制約は常に暗黙的で、その後に明示的なフィールド制約の任意の数が評価されます。変数 **$a** は、ルール結果で一致したオブジェクトを参照するバインディング変数です。

注記

ドル記号 (**$**) はオプションで、変数名とフィールド名を識別する上で役立ちます。

この例では、ルールのサンプルと Red Hat Process Automation Manager プロジェクトの **~/resources** ディレクトリー内の他のすべてのファイルは、以下のコードで構築されます。

KIE コンテナーの作成

```
KieServices kieServices = KieServices.Factory.get();

KieContainer kContainer = kieServices.getKieClasspathContainer();
```

このコードは、クラスパスで見つかったすべてのルールファイルをコンパイルし、このコンパイルの結果である **KieModule** オブジェクトを **KieContainer** に追加します。

最後に、**StatelessKieSession** オブジェクトが **KieContainer** からインスタンス化され、指定したデータに対して実行されます。

ステートレスな KIE セッションをインスタンス化し、データを入力

```
StatelessKieSession kSession = kContainer.newStatelessKieSession();
```

```
Applicant applicant = new Applicant("Mr John Smith", 16);

assertTrue(applicant.isValid());

ksession.execute(applicant);

assertFalse(applicant.isValid());
```

ステートレスな KIE セッションの設定では、**execute()** の呼び出しは **KieSession** オブジェクトをインスタンス化するコンビネーションメソッドとして機能し、すべてのユーザーデータを追加してユーザーコマンドを実行し、**fireAllRules()** を呼び出してから、**dispose()** を呼び出します。したがって、ステートレスな KIE セッションでは、ステートフルな KIE セッションの時のようにセッションの呼び出し後に **fireAllRules()** または **dispose()** を呼び出す必要はありません。

この場合、指定された申請者は 18 歳未満であるため、申請は拒否されます。

より複雑なユースケースについては、以下の例を参照してください。この例では、ステートレスな KIE セッションを使用し、コレクションなどの反復可能なオブジェクトのリストに対してルールを実行します。

運転免許申請の拡張データモデル

```
public class Applicant {
  private String name;
  private int age;
  // Getter and setter methods
}

public class Application {
  private Date dateApplied;
  private boolean valid;
  // Getter and setter methods
}
```

運転免許申請の拡張 DRL ルールセット

```
package com.company.license

rule "Is of valid age"
when
  Applicant(age < 18)
  $a : Application()
then
  $a.setValid(false);
end

rule "Application was made this year"
when
  $a : Application(dateApplied > "01-jan-2009")
then
  $a.setValid(false);
end
```

ステートレスな KIE セッションで実行が反復可能な拡張 Java ソース

```
StatelessKieSession ksession = kbase.newStatelessKnowledgeSession();
Applicant applicant = new Applicant("Mr John Smith", 16);
Application application = new Application();

assertTrue(application.isValid());
ksession.execute(Arrays.asList(new Object[] { application, applicant })); (1)
assertFalse(application.isValid());

ksession.execute
  (CommandFactory.newInsertIterable(new Object[] { application, applicant })); (2)

List<Command> cmds = new ArrayList<Command>(); (3)
cmds.add(CommandFactory.newInsert(new Person("Mr John Smith"), "mrSmith"));
cmds.add(CommandFactory.newInsert(new Person("Mr John Doe"), "mrDoe"));

BatchExecutionResults results = ksession.execute(CommandFactory.newBatchExecution(cmds));
assertEquals(new Person("Mr John Smith"), results.getValue("mrSmith"));
```

1. **Arrays.asList()** メソッドによって生成されたオブジェクトの反復可能なコレクションに対してルールを実行するメソッド。すべてのコレクション要素は、一致したルールが実行される前に挿入されます。**execute(Object object)** および **execute(Iterable objects)** メソッドは、**BatchExecutor** インターフェイスから派生した **execute(Command command)** メソッドを包むラッパーです。
2. **CommandFactory** インターフェイスを使用したオブジェクトの反復可能なコレクションの実行。
3. 多くのさまざまなコマンドまたは結果出力識別子と作業するための **BatchExecutor** および **CommandFactory** 設定。**CommandFactory** インターフェイスは、**StartProcess**、**Query**、**SetGlobal** など、**BatchExecutor** で使用できる他のコマンドをサポートしています。

81.1.1. ステートレスな KIE セッションのグローバル変数

StatelessKieSession オブジェクトは、セッションスコープのグローバル、デリゲートグローバル、または実行スコープのグローバルとして解決されるように設定できるグローバル変数 (グローバル) をサポートしています。

- **セッションスコープのグローバル:** セッションスコープのグローバルの場合は、**getGlobals()** メソッドを使用して、KIE セッショングローバルへのアクセスを提供する **Globals** インスタンスを返すことができます。これらのグローバルは、すべての実行呼び出しに使用されます。実行呼び出しは異なるスレッドで同時に実行する可能性があるため、可変グローバルには注意が必要です。

  セッションスコープのグローバル

  ```
  import org.kie.api.runtime.StatelessKieSession;

  StatelessKieSession ksession = kbase.newStatelessKieSession();

  // Set a global `myGlobal` that can be used in the rules.
  ksession.setGlobal("myGlobal", "I am a global");

  // Execute while resolving the `myGlobal` identifier.
  ksession.execute(collection);
  ```

- **デリゲートグローバル**: デリゲートグローバルの場合は、識別子を値にマップする内部コレクションに値を保存できるように、(**setGlobal(String, Object)** を使用して) 値をグローバルに割り当てることができます。この内部コレクションの識別子は、すべての提供されるデリゲートの中で優先されます。この内部コレクションで識別子が見つからない場合に、デリゲートグローバルが (もしあれば) 使用されます。

- **実行スコープのグローバル**: 実行スコープのグローバルの場合は、**Command** オブジェクトを使用して、実行特有のグローバル解決用に **CommandExecutor** インターフェイスに渡されるグローバルを設定できます。

CommandExecutor インターフェイスでは、グローバル、挿入されたファクト、およびクエリー結果の out identifier を使用してデータをエクスポートすることもできます。

グローバル、挿入されたファクト、およびクエリー結果の out identifier

```
import org.kie.api.runtime.ExecutionResults;

// Set up a list of commands.
List cmds = new ArrayList();
cmds.add(CommandFactory.newSetGlobal("list1", new ArrayList(), true));
cmds.add(CommandFactory.newInsert(new Person("jon", 102), "person"));
cmds.add(CommandFactory.newQuery("Get People" "getPeople"));

// Execute the list.
ExecutionResults results = ksession.execute(CommandFactory.newBatchExecution(cmds));

// Retrieve the `ArrayList`.
results.getValue("list1");
// Retrieve the inserted `Person` fact.
results.getValue("person");
// Retrieve the query as a `QueryResults` instance.
results.getValue("Get People");
```

81.2. ステートフルな KIE セッション

ステートフルな KIE セッションは、推論を使用して、時間の経過とともにファクトを繰り返し変更していくセッションです。ステートフルな KIE セッションでは、KIE セッションの以前の呼び出し (以前のセッション状態) からのデータは、セッションの呼び出し間で保持されますが、ステートレスな KIE セッションではそのデータは破棄されます。

警告

ステートフルな KIE セッションの実行後に **dispose()** メソッドを呼び出して、セッションの呼び出し間でメモリーリークが発生しないようにしてください。

以下のユースケースで、ステートフルな KIE セッションは一般的に使用されます。

- **監視** (株式市場の監視や購入プロセスの自動化など)

- **診断** (不具合検出プロセスまたは医療診断プロセスの実行など)

- **ロジスティックス** (荷物の追跡や配達のプロビジョニングなど)
- **コンプライアンスの確保** (市場取引における合法性の検証など)

たとえば、以下の火災報知機のデータモデルと DRL ルールのサンプルをご覧ください。

スプリンクラーと火災報知機のデータモデル

```
public class Room {
  private String name;
  // Getter and setter methods
}

public class Sprinkler {
  private Room room;
  private boolean on;
  // Getter and setter methods
}

public class Fire {
  private Room room;
  // Getter and setter methods
}

public class Alarm { }
```

スプリンクラーとアラームを有効にするための DRL ルールセットのサンプル

```
rule "When there is a fire turn on the sprinkler"
when
  Fire($room : room)
  $sprinkler : Sprinkler(room == $room, on == false)
then
  modify($sprinkler) { setOn(true) };
  System.out.println("Turn on the sprinkler for room "+$room.getName());
end

rule "Raise the alarm when we have one or more fires"
when
    exists Fire()
then
    insert( new Alarm() );
    System.out.println( "Raise the alarm" );
end

rule "Cancel the alarm when all the fires have gone"
when
    not Fire()
    $alarm : Alarm()
then
    delete( $alarm );
    System.out.println( "Cancel the alarm" );
end
```

```
rule "Status output when things are ok"
when
    not Alarm()
    not Sprinkler( on == true )
then
    System.out.println( "Everything is ok" );
end
```

When there is a fire turn on the sprinkler ルールの場合は、火災が発生すると、その部屋に対して **Fire** クラスのインスタンスが作成され、KIE セッションに挿入されます。このルールは、**Fire** インスタンスに一致する特定の **room** に制約を追加し、その部屋のスプリンクラーのみがチェックされるようにします。このルールが実行すると、スプリンクラーが有効になります。他のルールのサンプルは、これに基づいてアラームをいつ有効または無効にするかを決定します。

ステートレスな KIE セッションは、標準的な Java 構文に依存してフィールドを変更しますが、ステートフルな KIE セッションはルールの **modify** ステートメントに依存して、変更をデシジョンエンジンに通知します。次に、デシジョンエンジンが変更を判断し、後続のルール実行への影響を評価します。このプロセスは、**推論** および **真理維持** を使用するデシジョンエンジンの機能の一部であり、ステートフルな KIE セッションでは不可欠となります。

この例では、ルールのサンプルと Red Hat Process Automation Manager プロジェクトの ~/**resources** ディレクトリー内の他のすべてのファイルは、以下のコードで構築されます。

KIE コンテナーの作成

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kContainer = kieServices.getKieClasspathContainer();
```

このコードは、クラスパスで見つかったすべてのルールファイルをコンパイルし、このコンパイルの結果である **KieModule** オブジェクトを **KieContainer** に追加します。

最後に、**KieSession** オブジェクトが **KieContainer** からインスタンス化され、指定したデータに対して実行されます。

ステートフルな KIE セッションをインスタンス化してデータを入力

```
KieSession ksession = kContainer.newKieSession();

String[] names = new String[]{"kitchen", "bedroom", "office", "livingroom"};
Map<String,Room> name2room = new HashMap<String,Room>();
for( String name: names ){
    Room room = new Room( name );
    name2room.put( name, room );
    ksession.insert( room );
    Sprinkler sprinkler = new Sprinkler( room );
    ksession.insert( sprinkler );
}

ksession.fireAllRules();
```

コンソールの出力

```
> Everything is ok
```

データが追加されると、デシジョンエンジンはすべてのパターン一致を完了しますが、ルールは実行されていないため、設定済みの検証メッセージが表示されます。新しいデータがルール条件をトリガーすると、デシジョンエンジンはルールを実行してアラームを有効にし、後で有効になったアラームをキャンセルします。

新しいデータを入力してルールをトリガー

```
Fire kitchenFire = new Fire( name2room.get( "kitchen" ) );
Fire officeFire = new Fire( name2room.get( "office" ) );

FactHandle kitchenFireHandle = ksession.insert( kitchenFire );
FactHandle officeFireHandle = ksession.insert( officeFire );

ksession.fireAllRules();
```

コンソールの出力

```
> Raise the alarm
> Turn on the sprinkler for room kitchen
> Turn on the sprinkler for room office
```

```
ksession.delete( kitchenFireHandle );
ksession.delete( officeFireHandle );

ksession.fireAllRules();
```

コンソールの出力

```
> Cancel the alarm
> Turn off the sprinkler for room office
> Turn off the sprinkler for room kitchen
> Everything is ok
```

この場合は、返された **FactHandle** オブジェクトの参照が保持されます。ファクトハンドルは、挿入されたインスタンスへの内部エンジン参照であり、後でインスタンスを撤回または変更できるようにします。

この例が示すように、以前のステートフルな KIE セッションのデータと結果 (有効化されたアラーム) は、後続のセッションの呼び出し (アラームのキャンセル) に影響します。

81.3. KIE セッションプール

大量の KIE ランタイムデータと多くのシステムアクティビティーのあるユースケースでは、頻繁に KIE セッションが作成および破棄される可能性があります。頻繁な作成/破棄は必ずしも多大な時間を要するとは限りませんが、これが何百万回も繰り返されると、このプロセスがボトルネックとなり、膨大なクリーンアップ作業が必要となります。

このようなボリュームの高いケースには、多くの個別の KIE セッションの代わりに、KIE セッションプールを使用できます。KIE セッションプールを使用するには、KIE コンテナーから KIE セッションプールを取得し、プールでの KIE セッションの最初の数を定義して、そのプールから KIE セッションを通常どおりに作成します。

KIE セッションプールの例

```
// Obtain a KIE session pool from the KIE container
KieContainerSessionsPool pool = kContainer.newKieSessionsPool(10);

// Create KIE sessions from the KIE session pool
KieSession kSession = pool.newKieSession();
```

この例では、KIE セッションプールは 10 KIE セッションで起動しますが、必要な KIE セッション数を指定できます。この整数値は、初めにプールのみで作成された KIE セッション数です。実行中のアプリケーションで必要な場合は、プールの KIE セッション数を動的に増やすことができます。

KIE セッションプールを定義し、次に KIE セッションを通常どおりに使用し、**dispose()** を呼び出すと、KIE セッションはリセットされ、破棄されずにプールにプッシュされます。

KIE セッションプールは通常、ステートフルな KIE セッションに適用されますが、KIE セッションプールは複数の **execute()** 呼び出しで再利用するステートレスな KIE セッションにも影響を及ぼす場合があります。KIE コンテナーから直接ステートレスな KIE セッションを作成すると、KIE セッションは引き続き、**execute()** 呼び出しごとに新規の KIE セッションを内部で作成します。反対に、KIE セッションプールからステートレスな KIE セッションを作成する場合、KIE セッションはプールが提供する特定の KIE セッションのみを内部で使用します。

KIE セッションプールの使用を終了すると、メモリーリークを回避するために **shutdown()** メソッドを呼び出すことができます。または、KIE コンテナーで **dispose()** を呼び出して、KIE コンテナーから作成されたすべてのプールをシャットダウンします。

第82章 デシジョンエンジンにおける推論と真理維持

デシジョンエンジンの基本的な機能は、データをビジネスルールに一致させ、ルールを実行するかどうか、そしてどのように実行するかを決定することです。関連データが適切なルールに確実に適用されるように、デシジョンエンジンは既存の知識に基づいて **推論** を作成し、推論された情報に基づいてアクションを実行します。

たとえば、以下の DRL ルールは、バスの乗車パスに関する方針など、大人の年齢要件を決定します。

年齢要件を定義するためのルール

```
rule "Infer Adult"
when
  $p : Person(age >= 18)
then
  insert(new IsAdult($p))
end
```

このルールに基づいて、デシジョンエンジンはバス利用者が大人か子供かを推論し、指定されたアクション (**then** の結果) を実行します。18 歳以上のすべての人には、ワーキングメモリーに **IsAdult** のインスタンスが挿入されています。続いて、年齢とバスの乗車パスの推論されたこの関係は、以下のようなルールセグメントなどのルールで呼び出すことができます。

```
$p : Person()
IsAdult(person == $p)
```

多くの場合、ルールシステムの新しいデータは他のルール実行の結果であり、この新しいデータは他のルール実行に影響を与える可能性があります。デシジョンエンジンがルール実行の結果としてデータをアサートする場合、デシジョンエンジンは真理維持を使用してアサーションを正当化し、推論された情報を他のルールに適用する時に真理を強制します。真理維持は、不一致の特定と矛盾の処理にも役立ちます。たとえば、2 つのルールが実行され、矛盾したアクションが発生した場合、デシジョンエンジンは以前に計算された結論からの仮定に基づいてアクションを選択します。

デシジョンエンジンは、記述挿入または論理挿入のいずれかを使用してファクトを挿入します。

- **記述挿入: insert()** で定義されます。記述挿入の後、通常はファクトが明示的に取り消されます。(**挿入** という用語が一般的に使用される場合は **記述挿入** を指します。)
- **論理挿入: insertLogical()** で定義されます。論理挿入の後、挿入されたファクトは、ファクトを挿入したルールの条件が true でなくなると自動的に取り消されます。論理挿入をサポートする条件がない場合、ファクトは取り消されます。論理的に挿入されたファクトは、デシジョンエンジンによって **正当化される** と見なされます。

たとえば、以下の DRL ルールのサンプルでは、ファクトの記述挿入を使用して、子供用または大人用のバスの乗車パスを発行するための年齢要件を決定します。

バスの乗車パスを発行するためのルール (記述挿入)

```
rule "Issue Child Bus Pass"
when
  $p : Person(age < 18)
then
  insert(new ChildBusPass($p));
end
```

```
rule "Issue Adult Bus Pass"
when
  $p : Person(age >= 18)
then
  insert(new AdultBusPass($p));
end
```

バス利用者の年齢が上がると、子供用から大人用の乗車パスへと移行するため、デシジョンエンジンでこれらのルールを維持することは簡単ではありません。別の方法としては、ファクトの論理挿入を使用して、これらのルールをバス利用者の年齢のルールと乗車パスの種類のルールに分けることができます。ファクトを論理的に挿入することで、ファクトは **when** 節の真理に依存することになります。

以下の DRL ルールは、論理挿入を使用して子供と大人の年齢要件を決定します。

子供および大人の年齢要件 (論理挿入)

```
rule "Infer Child"
when
  $p : Person(age < 18)
then
  insertLogical(new IsChild($p))
end

rule "Infer Adult"
when
  $p : Person(age >= 18)
then
  insertLogical(new IsAdult($p))
end
```

重要

論理挿入の場合、ファクトオブジェクトは、Java 標準に従って **java.lang.Object** オブジェクトの **equals** メソッドおよび **hashCode** メソッドをオーバーライドする必要があります。2 つのオブジェクトが等しくなるのは、双方の **equals** メソッドが互いに **true** を返し、双方の **hashCode** メソッドが同じ値を返す場合です。詳細については、お使いの Java バージョンの Java API ドキュメントを参照してください。

ルールの条件が false の場合、ファクトは自動的に取り消されます。2 つのルールは相互に排他的であるため、この例ではこの動作が役立ちます。この例では、バス利用者が 18 歳未満の場合、ルールは **IsChild** ファクトを論理的に挿入します。利用者が 18 歳以上になると、**IsChild** ファクトが自動的に取り消され、**IsAdult** ファクトが挿入されます。

続いて、以下の DRL ルールが、子供用または大人用のバスの乗車パスを発行するかどうかを決定し、**ChildBusPass** ファクトおよび **AdultBusPass** ファクトを論理的に挿入します。このルールの設定が可能なのは、デシジョンエンジンの真理維持システムが、取り消しセットのカスケードに対する論理的挿入の連鎖をサポートしているためです。

バスの乗車パスを発行するためのルール (論理挿入)

```
rule "Issue Child Bus Pass"
when
  $p : Person()
    IsChild(person == $p)
```

```
then
  insertLogical(new ChildBusPass($p));
end

rule "Issue Adult Bus Pass"
when
  $p : Person()
    IsAdult(person =$p)
then
  insertLogical(new AdultBusPass($p));
end
```

バス利用者が 18 歳になると、**IsChild** ファクトとその利用者の **ChildBusPass** ファクトが取り消されます。これらの条件のセットに、18 歳になると子供用の乗車パスを返却する必要があることを示す別のルールを関連付けることができます。デシジョンエンジンが **ChildBusPass** オブジェクトを自動的に取り消すと、以下のルールが実行され、利用者にリクエストが送信されます。

バスの乗車パス利用者に新しいパスを通知するルール

```
rule "Return ChildBusPass Request"
when
  $p : Person()
    not(ChildBusPass(person == $p))
then
  requestChildBusPass($p);
end
```

以下のフローチャートは、記述挿入と論理挿入のライフサイクルを示しています。

図82.1 記述挿入

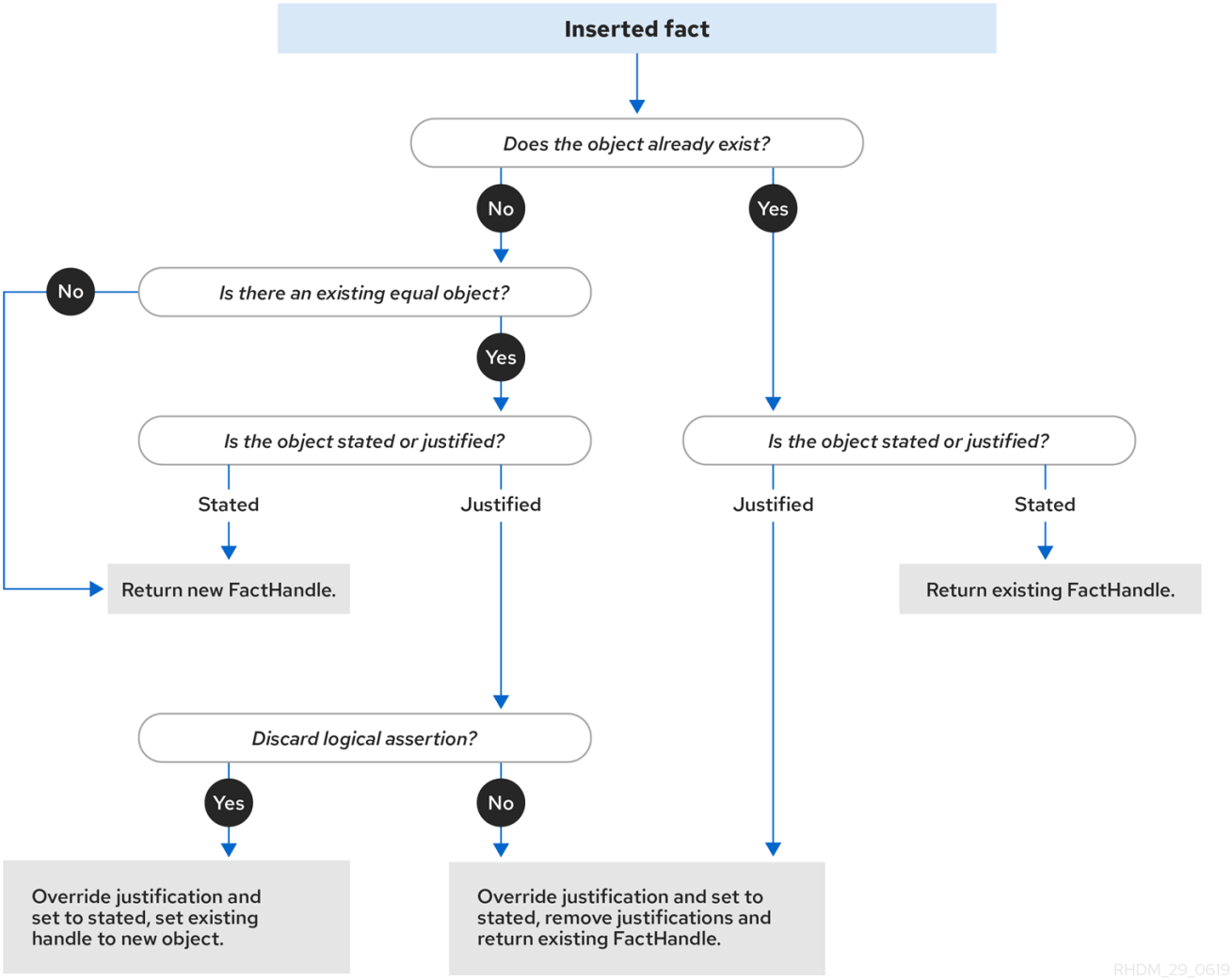
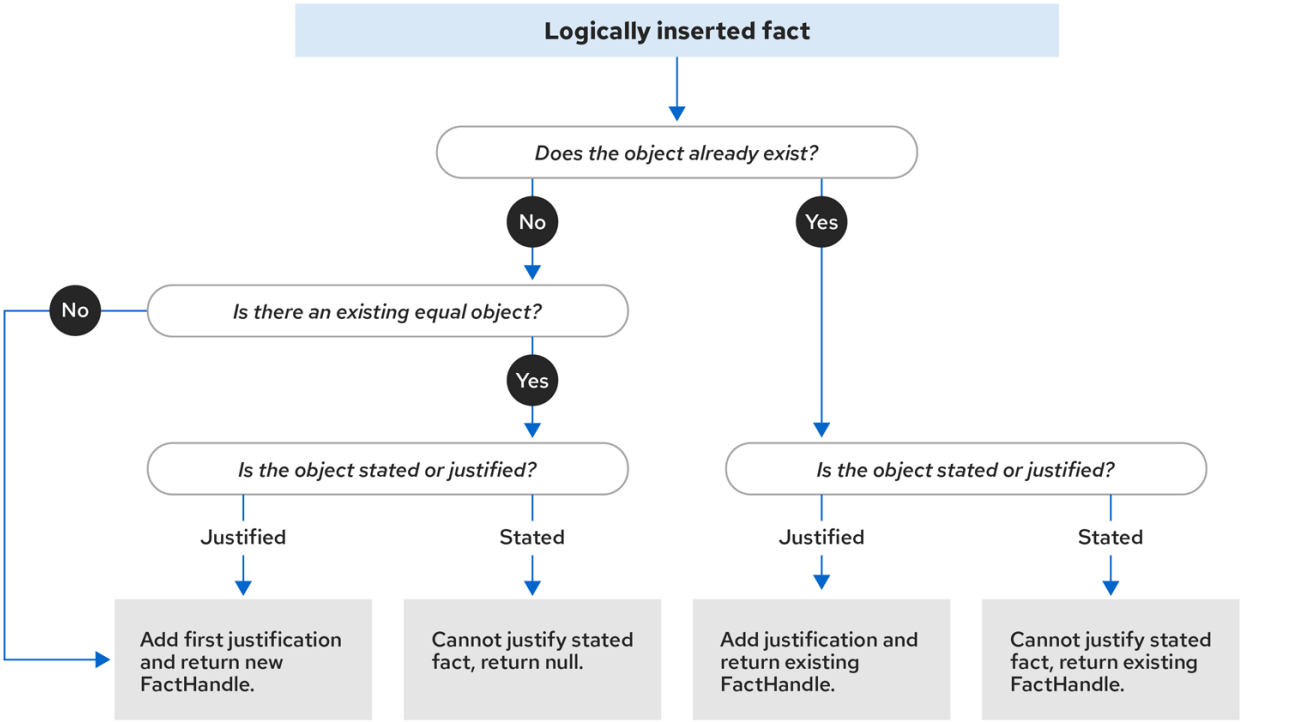

Inserted fact
Does the object already exist?
No
Yes
No
Is there an existing equal object?
Yes
Is the object stated or justified?
Is the object stated or justified?
Stated
Justified
Justified
Stated
Return new FactHandle.
Return existing FactHandle.
Discard logical assertion?
Yes
No
Override justification and set to stated, set existing handle to new object.
Override justification and set to stated, remove justifications and return existing FactHandle.
RHDM_29_0619

図82.2 論理挿入



ルールの実行中にデシジョンエンジンが論理的にオブジェクトを挿入すると、デシジョンエンジンはルールを実行してオブジェクトを **正当化** します。論理挿入ごとに、等しいオブジェクトは1つしか存在できず、後続の等しい論理挿入はそれぞれ、その論理挿入の正当化カウンターを増やします。ルールの条件が true でなくなると、正当化は削除されます。正当化がすべてなくなると、論理オブジェクトは自動的に取り消されます。

82.1. デシジョンエンジンのファクト等価モード

デシジョンエンジンは、挿入されたファクトをデシジョンエンジンが保存および比較する方法を決める以下のファクト等価モードをサポートします。

- **identity**: (デフォルト) デシジョンエンジンは **IdentityHashMap** を使用して、すべての挿入されたファクトを保存します。新しいファクトの挿入ごとに、デシジョンエンジンは、新しい **FactHandle** オブジェクトを返します。ファクトが再度挿入されると、デシジョンエンジンはオリジナルの **FactHandle** オブジェクトを返し、同じファクトに対して繰り返される挿入を無視します。このモードでは、2つのファクトが同じアイデンティティーを持つまったく同じオブジェクトである場合に限り、デシジョンエンジンにとってこの2つのファクトは同じものになります。

- **equality**: デシジョンエンジンは **HashMap** を使用して、すべての挿入されたファクトを保存します。デシジョンエンジンは、挿入されたファクトの **equals()** メソッドに従って、挿入されたファクトが既存のファクトと等しくない場合に限り、新しい **FactHandle** オブジェクトを返します。このモードでは、アイデンティティーに関係なく、2つのファクトが同じ方法で設定される場合、デシジョンエンジンにとってこの2つのファクトは同じものになります。このモードは、オブジェクトを明示的なアイデンティティーではなく、機能の等価性に基づいて評価する必要がある場合に使用します。

ファクト等価モードの説明として、以下のファクトの例をご覧ください。

ファクトの例

```
Person p1 = new Person("John", 45);
Person p2 = new Person("John", 45);
```

identity モードの場合、ファクト **p1** および **p2** は **Person** クラスの異なるインスタンスであり、別々のアイデンティティーを持つため、別のオブジェクトとして扱われます。**equality** モードの場合は、ファクト **p1** および **p2** が同じ設定を持つため、同じオブジェクトとして扱われます。このような動作の違いは、ファクトハンドルとの対話方法に影響を及ぼします。

たとえば、ファクト **p1** および **p2** をデシジョンエンジンに挿入し、その後 **p1** のファクトハンドルを取得する必要があると仮定します。**identity** モードの場合は、**p1** を指定して、正確なオブジェクトに対してファクトハンドルを返す必要があります。一方、**equality** モードの場合は、**p1**、**p2**、または **new Person("John", 45)** を指定して、ファクトハンドルを返すことができます。

ファクトを挿入して identity モードでファクトハンドルを返すコード例

```
ksession.insert(p1);

ksession.getFactHandle(p1);
```

ファクトを挿入して equality モードでファクトハンドルを返すコード例

```
ksession.insert(p1);

ksession.getFactHandle(p1);

// Alternate option:
ksession.getFactHandle(new Person("John", 45));
```

ファクト等価モードを設定するには、以下のいずれかのオプションを使用します。

- システムプロパティー **drools.equalityBehavior** を **identity** (デフォルト) または **equality** に設定
- プログラムを用いて KIE ベースを作成中に等価モードを設定

  ```
  KieServices ks = KieServices.get();
  KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
  kieBaseConf.setOption(EqualityBehaviorOption.EQUALITY);
  KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
  ```

- 特定の Red Hat Process Automation Manager プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で等価モードを設定

  ```
  <kmodule>
    ...
    <kbase name="KBase2" default="false" equalsBehavior="equality"
  packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
      ...
    </kbase>
    ...
  </kmodule>
  ```

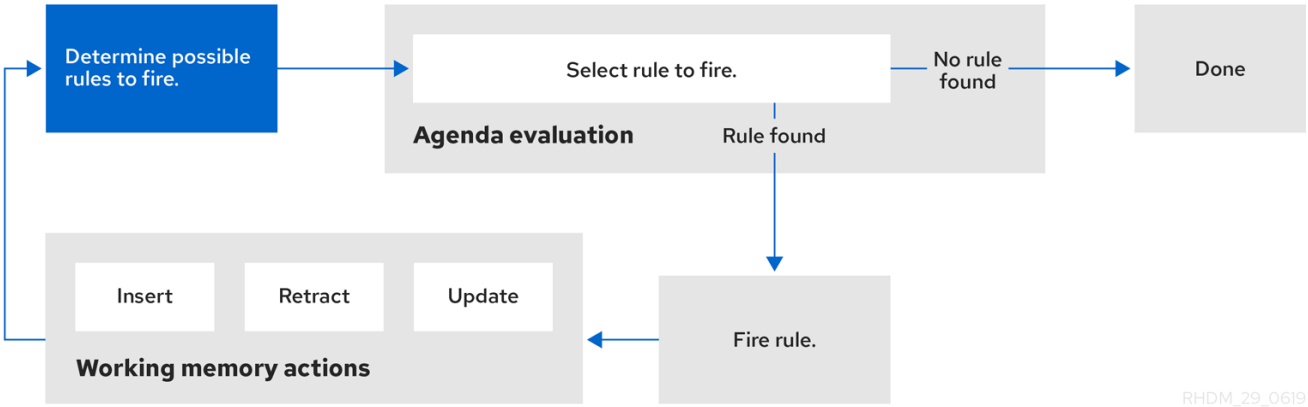
第83章 デシジョンエンジンにおける実行制御

新しいルールデータが、デシジョンエンジンのワーキングメモリーに入ると、ルールが完全に一致し、実行の対象となる場合があります。単一のワーキングメモリーアクションが、複数のルール実行の対象となる可能性があります。ルールが完全に一致すると、デシジョンエンジンはアクティベーションインスタンスを作成し、ルールと一致したファクトを参照し、アクティベーションをデシジョンエンジンのアジェンダに追加します。アジェンダは、競合解決ストラテジーを使用して、これらのルールのアクティベーションが実行される順番を制御します。

Java アプリケーションで **fireAllRules()** を最初に呼び出した後、デシジョンエンジンは 2 つのフェーズを繰り返し循環します。

- **アジェンダ評価:**このフェーズでは、デシジョンエンジンは実行可能なすべてのルールを選択します。実行可能なルールが存在しない場合は、実行サイクルが終了します。実行可能なルールが見つかると、デシジョンエンジンはアジェンダにアクティベーションを登録し、続いてワーキングメモリーアクションフェーズへと進み、ルール結果アクションを実行します。
- **ワーキングメモリーアクション:**このフェーズでは、デシジョンエンジンが、アジェンダに以前登録された有効化されたすべてのルールに対してルール結果アクション (各ルールの **then** 部分) を実行します。結果のアクションがすべて完了するか、主な Java アプリケーションプロセスが **fireAllRules()** を再度呼び出すと、デシジョンエンジンがアジェンダ評価フェーズに戻り、ルールを再評価します。

図83.1 デシジョンエンジンにおける 2 つのフェーズの実行プロセス



アジェンダに複数のルールが存在する場合は、1 つのルールを実行したことが原因で、別のルールがアジェンダから削除される場合があります。これを回避するために、デシジョンエンジンでいつ、どのようにルールを実行するかを定義できます。ルール実行の順番を定義する一般的な方法には、ルールの顕著性、アジェンダグループ、アクティベーショングループ、および DRL ルールセットのルールユニットを使用する方法があります。

83.1. ルールの顕著性

各ルールには、実行の順番を決定する整数の **salience** 属性があります。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。ルールのデフォルトの顕著性の値はゼロですが、顕著性は負の値でも正の値でもかまいません。

たとえば、以下の DRL ルールのサンプルは、以下に示す順序でデシジョンエンジンのスタックにリスト化されています。

```
rule "RuleA"
salience 95
```

```
when
    $fact : MyFact( field1 == true )
then
    System.out.println("Rule2 : " + $fact);
    update($fact);
end

rule "RuleB"
salience 100
when
   $fact : MyFact( field1 == false )
then
   System.out.println("Rule1 : " + $fact);
   $fact.setField1(true);
   update($fact);
end
```

RuleB ルールは 2 番目にリストされていますが、 **RuleA** ルールよりも顕著性の値が高いため、最初に実行されます。

83.2. ルールのアジェンダグループ

アジェンダグループは、同じ **agenda-group** ルールの属性によってバインドされている一連のルールです。アジェンダは、デシジョンエンジンのアジェンダのパーティションルールをグループ化します。常に 1 つのグループのみに **フォーカス** が設定され、そのルールのグループは、他のアジェンダグループのルールよりも優先して実行されます。アジェンダグループの **setFocus()** 呼び出しでフォーカスを決定します。**auto-focus** 属性を使用してルールを定義することもできます。これにより、次回ルールが有効になった時に、ルールが割り当てられているアジェンダグループ全体に自動的にフォーカスが設定されます。

Java アプリケーションで **setFocus()** 呼び出しが行われるたびに、デシジョンエンジンは指定されたアジェンダグループをルールスタックの一番上に追加します。デフォルトのアジェンダグループ **"MAIN"** には、指定されたアジェンダグループに属さないすべてのルールが含まれ、別のグループにフォーカスがない限り、スタックで最初に実行されます。

たとえば、以下の DRL ルールのサンプルは、指定されたアジェンダグループに属し、以下に示す順序でデシジョンエンジンのスタックにリスト化されています。

銀行取引アプリケーションにおける DRL ルールのサンプル

```
rule "Increase balance for credits"
  agenda-group "calculation"
when
  ap : AccountPeriod()
  acc : Account( $accountNo : accountNo )
  CashFlow( type == CREDIT,
            accountNo == $accountNo,
            date >= ap.start && <= ap.end,
            $amount : amount )
then
  acc.balance  += $amount;
end
```

```
rule "Print balance for AccountPeriod"
  agenda-group "report"
```

```
when
  ap : AccountPeriod()
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

この例では、**"report"** アジェンダグループのルールを常に最初に実行し、**"calculation"** アジェンダグループのルールを常に 2 番目に実行する必要があります。その後に、他のアジェンダグループの残りのルールを実行できます。したがって、**"report"** グループおよび **"calculation"** グループは、他のルールが実行する前に、この順序で実行されるフォーカスを受け取る必要があります。

アジェンダグループの実行順序にフォーカスを設定

```
Agenda agenda = ksession.getAgenda();
agenda.getAgendaGroup( "report" ).setFocus();
agenda.getAgendaGroup( "calculation" ).setFocus();
ksession.fireAllRules();
```

また、指定したアジェンダグループに属するルールで生成されたすべてのアクティベーションは、**clear()** メソッドを使用して、実行する前にキャンセルすることができます。

その他すべてのルールのアクティベーションの取り消し

```
ksession.getAgenda().getAgendaGroup( "Group A" ).clear();
```

83.3. ルールのアクティベーショングループ

アクティベーショングループは、同じ **activation-group** ルールの属性によってバインドされている一連のルールです。このグループでは、実行できるルールは 1 つだけです。実行するそのグループのルールの条件が一致すると、そのアクティベーショングループで保留となっているその他のすべてのルール実行がアジェンダから削除されます。

たとえば、以下の DRL ルールのサンプルは、指定されたアクティベーショングループに属し、以下に示す順序でデシジョンエンジンのスタックにリスト化されています。

銀行取引における DRL ルールのサンプル

```
rule "Print balance for AccountPeriod1"
  activation-group "report"
when
  ap : AccountPeriod1()
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

```
rule "Print balance for AccountPeriod2"
  activation-group "report"
when
  ap : AccountPeriod2()
```

```
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

この例では、**"report"** アクティベーショングループの最初のルールが実行すると、グループの 2 番目のルールとアジェンダにある実行可能なその他のルールがすべてアジェンダから削除されます。

83.4. デシジョンエンジンにおけるルール実行モードおよびスレッドの安全性

デシジョンエンジンは、デシジョンエンジンがルールを実行する方法とタイミングを決定する以下のルール実行モードをサポートします。

- **パッシブモード**: (デフォルト) デシジョンエンジンは、ユーザーまたはアプリケーションが **fireAllRules()** を明示的に呼び出す時にルールを評価します。デシジョンエンジンのパッシブモードは、ルールの評価および実行を直接管理する必要があるアプリケーションにとって、またはデシジョンエンジンで擬似クロック実装を使用する複合イベント処理 (CEP) のアプリケーションにとって最適です。

  パッシブモードのデシジョンエンジンを使用した CEP アプリケーションのコード例

  ```
  KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();
  config.setOption( ClockTypeOption.get("pseudo") );
  KieSession session = kbase.newKieSession( conf, null );
  SessionPseudoClock clock = session.getSessionClock();

  session.insert( tick1 );
  session.fireAllRules();

  clock.advanceTime(1, TimeUnit.SECONDS);
  session.insert( tick2 );
  session.fireAllRules();

  clock.advanceTime(1, TimeUnit.SECONDS);
  session.insert( tick3 );
  session.fireAllRules();

  session.dispose();
  ```

- **アクティブモード**: ユーザーまたはアプリケーションが **fireUntilHalt()** を呼び出す場合、デシジョンエンジンはアクティブモードで開始し、ユーザーまたはアプリケーションが明示的に **halt()** を呼び出すまで、継続的にルールを評価します。デシジョンエンジンのアクティブモードは、デシジョンエンジンにルールの評価と実行の管理を委譲するアプリケーションにとって、またはデシジョンエンジンでリアルタイムクロックの実装を使用する複合イベント処理 (CEP) アプリケーションにとって最適です。アクティブモードは、アクティブなクエリーを使用する CEP アプリケーションにも最適です。

  アクティブモードのデシジョンエンジンを使用した CEP アプリケーションのコード例

  ```
  KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();
  config.setOption( ClockTypeOption.get("realtime") );
  KieSession session = kbase.newKieSession( conf, null );
  ```

```
new Thread( new Runnable() {
  @Override
  public void run() {
     session.fireUntilHalt();
  }
} ).start();

session.insert( tick1 );

... Thread.sleep( 1000L ); ...

session.insert( tick2 );

... Thread.sleep( 1000L ); ...

session.insert( tick3 );

session.halt();
session.dispose();
```

この例では、デシジョンエンジンがルールの評価を継続している間に、現在のスレッドが永久にブロックされないように、専用の実行スレッドから **fireUntilHalt()** を呼び出します。また、専用のスレッドにより、アプリケーションコードの後の段階で **halt()** を呼び出すこともできます。

fireAllRules() および **fireUntilHalt()** の両方を呼び出すのは避けるべきですが (特に異なるスレッドからの場合)、デシジョンエンジンは、スレッドの安全性の論理および内部状態マシンを使用してこのような状況を安全に処理できます。**fireAllRules()** の呼び出し中に **fireUntilHalt()** を呼び出す場合、デシジョンエンジンは、**fireAllRules()** の操作が完了し、**fireUntilHalt()** の呼び出しに対応してアクティブモードで開始するまで、パッシブモードでの実行を続けます。しかし、デシジョンエンジンが、アクティブモードで実行中に、**fireUntilHalt()** の呼び出しに続いて **fireAllRules()** を呼び出すと、**fireAllRules()** の呼び出しは無視され、デシジョンエンジンは **halt()** が呼び出されるまで、アクティブモードでの実行を続けます。

アクティブモードでのスレッドの安全性を高めるため、デシジョンエンジンは、**submit()** メソッドをサポートします。このメソッドは、スレッドセーフでアトミックなアクションの KIE セッションで、操作をグループ化および実行するために使用できます。

アクティブモードでアトミック操作を実行する submit() メソッドを使用したアプリケーションのコード例

```
KieSession session = ...;

new Thread( new Runnable() {
  @Override
  public void run() {
     session.fireUntilHalt();
  }
} ).start();

final FactHandle fh = session.insert( fact_a );

... Thread.sleep( 1000L ); ...

session.submit( new KieSession.AtomicAction() {
```

```
  @Override
  public void execute( KieSession kieSession ) {
    fact_a.setField("value");
    kieSession.update( fh, fact_a );
    kieSession.insert( fact_1 );
    kieSession.insert( fact_2 );
    kieSession.insert( fact_3 );
  }
} );

... Thread.sleep( 1000L ); ...

session.insert( fact_z );

session.halt();
session.dispose();
```

スレッドの安全性とアトミック操作は、クライアントサイドパースペクティブからも役に立ちます。たとえば、複数のファクトを指定の時間に挿入する必要があるものの、デシジョンエンジンが挿入をアトミック操作とみなし、すべての挿入が完了してからルールの再評価を始める必要がある場合などです。

83.5. デシジョンエンジンにおけるファクトの伝播モード

デシジョンエンジンは、以下のファクト伝播モードをサポートします。このモードは、ルール実行の準備としてエンジンネットワークを介して挿入されたファクトを、デシジョンエンジンが進める方法を決定します。

- **Lazy**: (デフォルト) ファクトはルール実行時にバッチコレクションで伝播されますが、ユーザーまたはアプリケーションによってファクトは個別に挿入されるため、リアルタイムでは実行されません。その結果、ファクトが最終的にデシジョンエンジンを介して伝播される順序は、ファクトが個別に挿入された順序とは異なる可能性があります。

- **Immediate**: ファクトは、ユーザーまたはアプリケーションが挿入する順序で即座に伝播されます。

- **Eager**: ファクトは (バッチコレクション内に) 遅延して伝播されますが、ルールの実行前には伝播されます。デシジョンエンジンは、**no-loop** または **lock-on-active** の属性を持つルールに対してこの伝播動作を使用します。

デフォルトでは、デシジョンエンジンの Phreak ルールアルゴリズムは、改善されたルール評価全体に対して Lazy (遅延) ファクト伝播を使用します。ただし、場合によっては、この Lazy 伝播動作は、Immediate または Eager 伝播を必要とする特定のルール実行の想定される結果を変更する可能性があります。

たとえば、以下のルールは接頭辞に **?** を指定したクエリーを使用して、プルオンリーまたはパッシブ方式でクエリーを呼び出します。

パッシブクエリーを使用したルールの例

```
query Q (Integer i)
    String( this == i.toString() )
end

rule "Rule"
  when
```

```
    $i : Integer()
    ?Q( $i; )
  then
    System.out.println( $i );
end
```

この例では、クエリーを満たす **String** が **Integer** の前に挿入される場合にのみ、ルールを実行する必要があります。たとえば、以下のコマンド例のようになります。

ルールの実行をトリガーするコマンドの例

```
KieSession ksession = ...
ksession.insert("1");
ksession.insert(1);
ksession.fireAllRules();
```

ただし、Phreak におけるデフォルトの Lazy (遅延) 伝播動作が原因で、デシジョンエンジンはこの場合における 2 つのファクトの挿入シーケンスを検出しません。そのため、**String** および **Integer** の挿入順序に関係なく、このルールは実行されます。この例では、想定されるルール評価には Immediate 伝播が必要です。

デシジョンエンジンの伝播モードを変更して、この場合に想定されるルール評価を達成するには、ルールに **@Propagation(<type>)** タグを追加し、**<type>** を **LAZY**、**IMMEDIATE**、または **EAGER** に設定します。

同じルールの例では、想定どおりに、Immediate 伝播アノテーションによって、クエリーを満たす **String** が **Integer** の前に挿入される場合にのみ、ルールが評価されます。

パッシブクエリーと指定された伝播モードを使用したルールの例

```
query Q (Integer i)
    String( this == i.toString() )
end

rule "Rule" @Propagation(IMMEDIATE)
  when
    $i : Integer()
    ?Q( $i; )
  then
    System.out.println( $i );
end
```

83.6. アジェンダ評価フィルター

デシジョンエンジンは、アジェンダを評価している間に指定されたルールの評価を許可または拒否するために使用できるフィルターインターフェイスの **AgendaFilter** オブジェクトをサポートします。**fireAllRules()** 呼び出しの一部として、アジェンダフィルターを指定することができます。

以下のコード例では、文字列 **"Test"** で終わるルールのみの評価および実行を許可します。他のルールはすべてデシジョンエンジンのアジェンダから除外されます。

アジェンダフィルター定義の例

```
ksession.fireAllRules( new RuleNameEndsWithAgendaFilter( "Test" ) );
```

83.7. DRL ルールセットのルールユニット

ルールユニットは、データソース、グローバル変数、および DRL ルールのグループで、特定の目的に向けて互いに機能し合います。ルールユニットを使用して、ルールセットを小さなユニットに分割し、それらのユニットにさまざまなデータソースをバインドしてから、個別のユニットを実行します。ルールユニットは、実行制御用のルールアジェンダグループまたはアクティブ化グループなどの、ルールをグループ化する DRL 属性に代わるものとして、強化されています。

ルールユニットは、ルールの実行を調整することで、あるルールユニットが完全に実行されると別のルールユニットの開始をトリガーする場合などに便利です。たとえば、データ強化用の一連のルール、そのデータを処理する別の一連のルール、および処理されたデータを抽出して出力する別の一連のルールがあるとします。これらのルールセットを 3 つの異なるルールユニットに追加する場合は、これらのルールユニットを調整することで、1 つ目のユニットが完全に実行すると 2 つ目のユニットの開始をトリガーし、2 つ目のユニットが完全に実行すると 3 つ目のユニットの開始をトリガーすることができます。

ルールユニットを定義するには、以下の例に示すように **RuleUnit** インターフェイスを実装します。

ルールユニットクラスの例

```
package org.mypackage.myunit;

public static class AdultUnit implements RuleUnit {
    private int adultAge;
    private DataSource<Person> persons;

    public AdultUnit( ) { }

    public AdultUnit( DataSource<Person> persons, int age ) {
        this.persons = persons;
        this.age = age;
    }

    // A data source of `Persons` in this rule unit:
    public DataSource<Person> getPersons() {
        return persons;
    }

    // A global variable in this rule unit:
    public int getAdultAge() {
        return adultAge;
    }

    // Life-cycle methods:
    @Override
    public void onStart() {
        System.out.println("AdultUnit started.");
    }

    @Override
    public void onEnd() {
        System.out.println("AdultUnit ended.");
    }
}
```

この例では、**persons** はタイプ **Person** のファクトのソースです。ルールユニットのデータソースは、指定のルールユニットで処理されるデータのソースで、デシジョンエンジンがルールユニットの評価に使用するエントリーポイントを表します。**adultAge** グローバル変数は、このルールユニットに属するすべてのルールからアクセスできます。最後の 2 つのメソッドは、ルールユニットのライフサイクルの一部で、デシジョンエンジンによって呼び出されます。

デシジョンエンジンは、以下のようなルールユニットの任意のライフサイクルメソッドをサポートします。

表83.1 ルールユニットのライフサイクルメソッド

メソッド	呼び出されるタイミング
onStart()	ルールユニット実行開始時
onEnd()	ルールユニット実行終了時
onSuspend()	ルールユニット実行の一時停止時 (**runUntilHalt()** でのみを使用される)
onResume()	ルールユニット実行の再開時 (**runUntilHalt()** でのみ使用される)
onYield(RuleUnit other)	ルールユニットにおけるルールの結果が異なるルールユニットの実行をトリガー

ルールユニットに、ルールを 1 つ以上追加することができます。デフォルトでは、DRL ファイルのすべてのルールは、DRL ファイル名の命名規則に従うルールユニットに自動的に関連付けられます。DRL ファイルが同じパッケージにあり、**RuleUnit** インターフェイスを実装するクラスと同じ名前を持つ場合、その DRL ファイルのすべてのルールは、そのルールユニットに暗黙的に属します。たとえば、**org.mypackage.myunit** パッケージの **AdultUnit.drl** ファイルにあるすべてのルールは、自動的にルールユニット **org.mypackage.myunit.AdultUnit** の一部となります。

この命名規則をオーバーライドし、DRL ファイル内のルールが属するルールユニットを明示的に宣言するには、DRL ファイル内でキーワード **unit** を使用します。**unit** 宣言は、すぐに package 宣言に従い、DRL ファイルのルールが一部となっているパッケージ内のクラス名を含む必要があります。

DRL ファイルのルールユニット宣言の例

```
package org.mypackage.myunit
unit AdultUnit

rule Adult
  when
    $p : Person(age >= adultAge) from persons
  then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
```

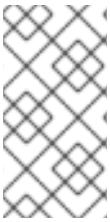
警告

同じ KIE ベースで、ルールユニットありのルールとルールユニットなしのルールを混在させないでください。KIE ベースで 2 つのルールのパラダイムを混在させると、コンパイルエラーが発生します。

以下の例のように OOPath 表記を使用して、より便利な方法で同じパターンを書き換えることもできます。

OOPath 表記を使用する DRL ファイルのルールユニット宣言の例

```
package org.mypackage.myunit
unit AdultUnit

rule Adult
  when
    $p : /persons[age >= adultAge]
  then
    System.out.println($p.getName() + " is adult and greater than " + adultAge);
end
```



注記

OOPath は、DRL ルールの条件の制約でオブジェクトのグラフを参照するために設計された XPath のオブジェクト指向構文の拡張です。OOPath は、コレクションおよびフィルター制約を処理する間に XPath からのコンパクト表記を使用して関連要素を移動します。また、OOPath は特にオブジェクトグラフの場合に役に立ちます。

この例では、ルール条件で一致するファクトはすべて、ルールユニットクラスの **DataSource** 定義で定義される **persons** のデータソースから取得されます。ルール条件およびアクションは、グローバル変数が DRL ファイルレベルで定義されるのと同じ方法で **adultAge** 変数を使用します。

KIE ベースに定義されたルールユニットを 1 つ以上実行するには、KIE ベースにバインドされている新規の **RuleUnitExecutor** クラスを作成し、関連するデータソースからルールユニットを作成して、ルールユニットエグゼキューターを実行します。

ルールユニット実行の例

```
// Create a `RuleUnitExecutor` class and bind it to the KIE base:
KieBase kbase = kieContainer.getKieBase();
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

// Create the `AdultUnit` rule unit using the `persons` data source and run the executor:
RuleUnit adultUnit = new AdultUnit(persons, 18);
executor.run( adultUnit );
```

ルールは **RuleUnitExecutor** クラスにより実行されます。**RuleUnitExecutor** クラスは KIE セッションを作成し、必要な **DataSource** オブジェクトをこれらのセッションに追加してから、**run()** メソッドへのパラメーターとして渡される **RuleUnit** をもとに、ルールを実行します。

例の実行コードは、関連する **Person** ファクトが **persons** データソースに挿入されると、以下の出力を生成します。

ルールユニット実行出力の例

```
org.mypackage.myunit.AdultUnit started.
Jane is adult and greater than 18
John is adult and greater than 18
org.mypackage.myunit.AdultUnit ended.
```

ルールユニットインスタンスを明示的に作成するのではなく、エグゼキューターにルールユニット変数を登録し、実行するルールユニットクラスをエグゼキューターに渡すと、エグゼキューターがルールユニットのインスタンスを作成します。続いて、ルールユニットを実行する前に **DataSource** 定義および他の変数を設定できます。

登録変数を含む別のルールユニット実行オプション

```
executor.bindVariable( "persons", persons );
      .bindVariable( "adultAge", 18 );
executor.run( AdultUnit.class );
```

RuleUnitExecutor.bindVariable() メソッドに渡す名前は、実行時に、同じ名前のルールユニットクラスのフィールドに変数をバインドするために使用されます。前述の例では、**RuleUnitExecutor** は、新しいルールユニットに **"persons"** の名前にバインドされているデータソースを挿入します。また、**AdultUnit** クラス内の対応する名前のフィールドに、文字列 **"adultAge"** にバインドされている値 **18** を挿入します。

このデフォルトの変数バインディング動作をオーバーライドするには、**@UnitVar** アノテーションを使用してルールユニットクラスの各フィールドに対して論理バインディング名を明示的に定義します。たとえば、以下のクラスのフィールドバインディングは、代替名で再度定義されます。

@UnitVar を使用した変数バインディング名を変更するコード例

```
package org.mypackage.myunit;

public static class AdultUnit implements RuleUnit {
    @UnitVar("minAge")
    private int adultAge = 18;

    @UnitVar("data")
    private DataSource<Person> persons;
}
```

次に、これらの代替名を使用して、変数をエグゼキューターにバインドし、ルールユニットを実行できます。

変更した変数名を使用したルールユニット実行の例

```
executor.bindVariable( "data", persons );
      .bindVariable( "minAge", 18 );
executor.run( AdultUnit.class );
```

ルールユニットは、**run()** メソッド (KIE セッションで **fireAllRules()** を呼び出す場合と同じ) を使用して **パッシブモード** で、または **runUntilHalt()** メソッド (KIE セッションで **fireUntilHalt()** を呼び出す場合

と同じ) を使用して **アクティブモード** で実行できます。デフォルトでは、デシジョンエンジンは **パッシブモード** で実行し、ユーザーまたはアプリケーションが明示的に **run()** (標準ルールでは **fireAllRules()**) を呼び出す場合にのみルールユニットを評価します。ユーザーまたはアプリケーションがルールユニットに **runUntilHalt()** (標準ルールでは **fireAllRules()**) を呼び出す場合、デシジョンエンジンは **アクティブモード** で開始し、ユーザーまたはアプリケーションが明示的に **halt()** を呼び出すまで、継続的にルールユニットを評価します。

runUntilHalt() メソッドを使用する場合は、メインスレッドをブロックしないように、別の実行スレッド上でメソッドを呼び出します。

別のスレッド上の runUntilHalt() を使用したルールユニットの実行例

```
new Thread( () -> executor.runUntilHalt( adultUnit ) ).start();
```

83.7.1. ルールユニットのデータソース

ルールユニットのデータソースは、指定のルールユニットで処理されるデータのソースで、デシジョンエンジンがルールユニットの評価に使用するエントリーポイントを表します。ルールユニットは、ゼロまたは複数のデータソースを持つことができ、ルールユニット内で宣言された各 **DataSource** の定義は、ルールユニットエグゼキューターへの異なるエントリーポイントに対応することができます。複数のルールユニットは、単一データソースを共有できます。ただし、各ルールユニットは、同じオブジェクトが挿入される異なるエントリーポイントを使用する必要があります。

以下の例で示すように、ルールユニットクラスの固定されたデータセットを使用して **DataSource** 定義を作成できます。

データソース定義の例

```
DataSource<Person> persons = DataSource.create( new Person( "John", 42 ),
                                                new Person( "Jane", 44 ),
                                                new Person( "Sally", 4 ) );
```

データソースはルールユニットのエントリーポイントを表すため、ルールユニットでファクトを挿入、更新、または削除できます。

ルールユニットでファクトを挿入、更新、削除するコード例

```
// Insert a fact:
Person john = new Person( "John", 42 );
FactHandle johnFh = persons.insert( john );

// Modify the fact and optionally specify modified properties (for property reactivity):
john.setAge( 43 );
persons.update( johnFh, john, "age" );

// Delete the fact:
persons.delete( johnFh );
```

83.7.2. ルールユニットの実行制御

一方のルールユニットの実行により、もう一方のルールユニットの開始がトリガーされるようにルールの実行を調整する必要がある場合に、ルールユニットは役に立ちます。

ルールユニットの実行制御を容易にするために、デシジョンエンジンは以下のルールユニットメソッドをサポートします。このメソッドは、DRL ルールアクションで使用して、ルールユニットの実行を調整することができます。

- **drools.run()**: 指定されたルールユニットクラスの実行をトリガーします。このメソッドでは、ルールユニットの実行を命令的に中断し、他の指定されたルールユニットを有効化します。
- **drools.guard()**: 関連付けられたルール条件が満たされるまで、指定されたルールユニットクラスが実行されないようにします (保護します)。このメソッドは、他の指定されたルールユニットの実行を宣言的にスケジュールします。デシジョンエンジンが、保護ルールの条件に対して少なくとも 1 つの一致をもたらす場合は、保護されたルールユニットが有効とみなされます。ルールユニットには、複数の保護ルールを含めることができます。

drools.run() メソッドの例として、それぞれが指定されたルールユニットに属す以下の DRL ルールを検討してください。**NotAdult** ルールは **drools.run(AdultUnit.class)** メソッドを使用して **AdultUnit** ルールユニットの実行をトリガーします。

drools.run() を使用した制御された実行を含む DRL ルールの例

```
package org.mypackage.myunit
unit AdultUnit

rule Adult
  when
    Person(age >= 18, $name : name) from persons
  then
    System.out.println($name + " is adult");
end
```

```
package org.mypackage.myunit
unit NotAdultUnit

rule NotAdult
  when
    $p : Person(age < 18, $name : name) from persons
  then
    System.out.println($name + " is NOT adult");
    modify($p) { setAge(18); }
    drools.run( AdultUnit.class );
end
```

この例では、これらのルールからビルドされた KIE ベースから作成された **RuleUnitExecutor** クラスと、これにバインドされている **persons** の **DataSource** 定義も使用します。

ルールエグゼキューターとデータソース定義の例

```
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );
DataSource<Person> persons = executor.newDataSource( "persons",
                                        new Person( "John", 42 ),
                                        new Person( "Jane", 44 ),
                                        new Person( "Sally", 4 ) );
```

この例では、**RuleUnitExecutor** クラスから **DataSource** 定義を直接作成し、これを単一ステートメントで **"persons"** 変数にバインドします。

例の実行コードは、関連する **Person** ファクトが **persons** データソースに挿入されると、以下の出力を生成します。

ルールユニット実行出力の例

```
Sally is NOT adult
John is adult
Jane is adult
Sally is adult
```

NotAdult ルールは、**"Sally"** という人物の評価時に一致を検出します。この人物は 18 歳未満です。続いてこのルールは、この人物の年齢を **18** に変更し、**drools.run(AdultUnit.class)** メソッドを使用して **AdultUnit** ルールユニットの実行をトリガーします。**AdultUnit** ルールユニットには、**DataSource** 定義の 3 人の **persons** 全員に対して実行可能となったルールが含まれています。

drools.guard() メソッドの例として、以下の **BoxOffice** クラスと **BoxOfficeUnit** ルールユニットクラスを検討してください。

BoxOffice クラスの例

```
public class BoxOffice {
    private boolean open;

    public BoxOffice( boolean open ) {
        this.open = open;
    }

    public boolean isOpen() {
        return open;
    }

    public void setOpen( boolean open ) {
        this.open = open;
    }
}
```

BoxOfficeUnit ルールユニットクラスの例

```
public class BoxOfficeUnit implements RuleUnit {
    private DataSource<BoxOffice> boxOffices;

    public DataSource<BoxOffice> getBoxOffices() {
        return boxOffices;
    }
}
```

また、この例では、以下の **TicketIssuerUnit** ルールユニットクラスを使用して、少なくとも 1 つのボックスオフィス (チケット売り場) が営業中である限り、ボックスオフィスでのイベントチケットの販売を続行します。このルールユニットは **persons** および **tickets** の **DataSource** 定義を使用します。

TicketIssuerUnit ルールユニットクラスの例

```
public class TicketIssuerUnit implements RuleUnit {
    private DataSource<Person> persons;
```

```
    private DataSource<AdultTicket> tickets;

    private List<String> results;

    public TicketIssuerUnit() { }

    public TicketIssuerUnit( DataSource<Person> persons, DataSource<AdultTicket> tickets ) {
        this.persons = persons;
        this.tickets = tickets;
    }

    public DataSource<Person> getPersons() {
        return persons;
    }

    public DataSource<AdultTicket> getTickets() {
        return tickets;
    }

    public List<String> getResults() {
        return results;
    }
}
```

BoxOfficeUnit ルールユニットには、DRL ルール **BoxOfficeIsOpen** が含まれます。これは、**drools.guard(TicketIssuerUnit.class)** メソッドを使用して、イベントチケットを配布する **TicketIssuerUnit** ルールユニットの実行を保護します。以下に DRL ルールの例を示します。

drools.guard() を使用した制御された実行を含む DRL ルールの例

```
package org.mypackage.myunit;
unit TicketIssuerUnit;

rule IssueAdultTicket when
    $p: /persons[ age >= 18 ]
then
    tickets.insert(new AdultTicket($p));
end
rule RegisterAdultTicket when
    $t: /tickets
then
    results.add( $t.getPerson().getName() );
end
```

```
package org.mypackage.myunit;
unit BoxOfficeUnit;

rule BoxOfficeIsOpen
  when
    $box: /boxOffices[ open ]
  then
    drools.guard( TicketIssuerUnit.class );
end
```

この例では、少なくとも1つのボックスオフィスが **open** である限り、保護された **TicketIssuerUnit** ルールユニットが有効なため、イベントチケットは配布されます。**open** 状態のボックスオフィスがなくなると、保護された **TicketIssuerUnit** ルールユニットは実行されなくなります。

以下のクラスの例は、より完全なボックスオフィスのシナリオを説明します。

ボックスオフィスシナリオのクラスの例

```
DataSource<Person> persons = executor.newDataSource( "persons" );
DataSource<BoxOffice> boxOffices = executor.newDataSource( "boxOffices" );
DataSource<AdultTicket> tickets = executor.newDataSource( "tickets" );

List<String> list = new ArrayList<>();
executor.bindVariable( "results", list );

// Two box offices are open:
BoxOffice office1 = new BoxOffice(true);
FactHandle officeFH1 = boxOffices.insert( office1 );
BoxOffice office2 = new BoxOffice(true);
FactHandle officeFH2 = boxOffices.insert( office2 );

persons.insert(new Person("John", 40));

// Execute `BoxOfficeIsOpen` rule, run `TicketIssuerUnit` rule unit, and execute `RegisterAdultTicket`
rule:
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "John", list.get(0) );
list.clear();

persons.insert(new Person("Matteo", 30));

// Execute `RegisterAdultTicket` rule:
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "Matteo", list.get(0) );
list.clear();

// One box office is closed, the other is open:
office1.setOpen(false);
boxOffices.update(officeFH1, office1);
persons.insert(new Person("Mark", 35));
executor.run(BoxOfficeUnit.class);

assertEquals( 1, list.size() );
assertEquals( "Mark", list.get(0) );
list.clear();

// All box offices are closed:
office2.setOpen(false);
boxOffices.update(officeFH2, office2); // Guarding rule is no longer true.
persons.insert(new Person("Edson", 35));
```

```
executor.run(BoxOfficeUnit.class); // No execution

assertEquals( 0, list.size() );
```

83.7.3. ルールユニットのアイデンティティーの競合

保護されたルールユニットを使用したルール実行のシナリオでは、1 つのルールが複数のルールユニットを保護することができます。同時に、複数のルールが 1 つのルールユニットを保護してから有効にすることもできます。このような 2 通りの保護シナリオでは、ルールユニットには、アイデンティティーの競合を避けるための明確に定義されたアイデンティティーが必要です。

デフォルトでは、ルールユニットのアイデンティティーはルールユニットクラス名で、**RuleUnitExecutor** によりシングルトンクラスとして処理されます。この識別動作は、**RuleUnit** インターフェイスの **getUnitIdentity()** のデフォルトメソッドにエンコードされています。

RuleUnit インターフェイスのデフォルトのアイデンティティーメソッド

```
default Identity getUnitIdentity() {
    return new Identity( getClass() );
}
```

場合によっては、ルールユニット間のアイデンティティーの競合を避けるために、このデフォルトの識別動作をオーバーライドする必要があります。

たとえば、以下の **RuleUnit** クラスには、あらゆる種類のオブジェクトを許可する **DataSource** 定義が含まれています。

Unit0 ルールユニットクラスの例

```
public class Unit0 implements RuleUnit {
    private DataSource<Object> input;

    public DataSource<Object> getInput() {
        return input;
    }
}
```

このルールユニットには、2 つの条件 (OOPath 表記) に基づいて別のルールユニットを保護する、以下の DRL ルールが含まれています。

ルールユニットの DRL ルール GuardAgeCheck の例

```
package org.mypackage.myunit
unit Unit0

rule GuardAgeCheck
  when
    $i: /input#Integer
    $s: /input#String
  then
    drools.guard( new AgeCheckUnit($i) );
    drools.guard( new AgeCheckUnit($s.length()) );
end
```

保護された **AgeCheckUnit** ルールユニットは、一連の **persons** の年齢を検証します。**AgeCheckUnit** には、確認用の **persons** の **DataSource** の定義、検証用の **minAge** 変数、および結果を集計する **List** が含まれます。

AgeCheckUnit ルールユニットの例

```
public class AgeCheckUnit implements RuleUnit {
    private final int minAge;
    private DataSource<Person> persons;
    private List<String> results;

    public AgeCheckUnit( int minAge ) {
        this.minAge = minAge;
    }

    public DataSource<Person> getPersons() {
        return persons;
    }

    public int getMinAge() {
        return minAge;
    }

    public List<String> getResults() {
        return results;
    }
}
```

AgeCheckUnit ルールユニットには、データソースの **persons** の検証を実行する以下の DRL ルールが含まれます。

ルールユニットの DRL ルール CheckAge の例

```
package org.mypackage.myunit
unit AgeCheckUnit

rule CheckAge
  when
    $p : /persons{ age > minAge }
  then
    results.add($p.getName() + ">" + minAge);
end
```

この例では、**RuleUnitExecutor** クラスを作成し、これらの 2 つのルールユニットが含まれる KIE ベースにクラスをバインドして、同じルールユニットの **DataSource** 定義を 2 つ作成します。

executor 定義とデータソース定義の例

```
RuleUnitExecutor executor = RuleUnitExecutor.create().bind( kbase );

DataSource<Object> input = executor.newDataSource( "input" );
DataSource<Person> persons = executor.newDataSource( "persons",
                                        new Person( "John", 42 ),
                                        new Person( "Sally", 4 ) );
```

```
List<String> results = new ArrayList<>();
executor.bindVariable( "results", results );
```

一部のオブジェクトを入力データソースに挿入し、**Unit0** ルールユニットを実行できるようになりました。

挿入されたオブジェクトを使用したルールユニット実行の例

```
ds.insert("test");
ds.insert(3);
ds.insert(4);
executor.run(Unit0.class);
```

実行結果一覧の例

```
[Sally>3, John>3]
```

この例では、**AgeCheckUnit** という名前のルールユニットはシングルトンクラスと見なされ、1 回のみ実行されます。この時、**minAge** 変数は **3** に設定されます。入力データソースに挿入された文字列 **"test"** および整数 **4** の両方は、**minAge** 変数が **4** に設定された 2 回目の実行をトリガーする可能性もあります。しかし、同じアイデンティティーを持つ別のルールユニットがすでに評価されているため、2 回目の実行はありません。

このルールユニットのアイデンティティーの競合を解決するには、**AgeCheckUnit** クラスの **getUnitIdentity()** メソッドをオーバーライドして、ルールユニットアイデンティティーに **minAge** 変数も含めます。

getUnitIdentity() メソッドをオーバーライドする変更された AgeCheckUnit ルールユニット

```
public class AgeCheckUnit implements RuleUnit {

    ...

    @Override
    public Identity getUnitIdentity() {
        return new Identity(getClass(), minAge);
    }
}
```

このオーバーライドにより、以前のルールユニットの実行例は、以下の出力を生成します。

変更したルールユニットの実行結果一覧の例

```
[John>4, Sally>3, John>3]
```

minAge が **3** と **4** に設定されたルールユニットは、2 つの異なるルールユニットとみなされるようになり、両方とも実行されます。

第84章 デシジョンエンジンにおける PHREAK ルールアルゴリズム

Red Hat Process Automation Manager のデシジョンエンジンは、ルール評価に Phreak アルゴリズムを使用します。Phreak は、Rete アルゴリズムから発展したもので、これには強化された Rete アルゴリズムの ReteOO も含まれます。ReteOO は、オブジェクト指向システム向けに Red Hat Process Automation Manager の以前のバージョンに導入されました。全体として Phreak は、Rete および ReteOO よりスケーラブルで、大規模なシステムでより迅速に対応します。

Rete は Eager (即時ルール評価) でデータ指向と考えられていますが、Phreak は Lazy (遅延ルール評価) で目標指向と考えられています。Rete アルゴリズムは、すべてのルールに対して部分的な一致を見つけ出すために insert、update、および delete の各アクションを実行中に、多数のアクションを実行します。ルールの一致において Rete アルゴリズムは活発なため、最終的にルールを実行するまでに非常に時間がかかります。これは大規模なシステムにおいて特に顕著です。Phreak の場合は、ルールの部分的な一致を意図的に遅延させ、大量のデータをより効率的に処理します。

Phreak アルゴリズムでは、これまでの Rete アルゴリズムに、以下に示す一連の拡張機能を追加しています。

- コンテキストメモリーの 3 つのレイヤー: ノード、セグメント、およびルールのメモリータイプ
- ルールベース、セグメントベース、およびノードベースのリンク
- Lazy (遅延) ルール評価
- 一時停止と再開を使用したスタックベースの評価
- 孤立したルール評価
- セット指向の伝播

84.1. PHREAK でのルール評価

デシジョンエンジンが開始すると、すべてのルールは、ルールのトリガーが可能なパターン一致データに **リンクされていない** と見なされます。この段階で、デシジョンエンジンの Phreak アルゴリズムはルールを評価しません。**insert**、**update**、および **delete** の各アクションはキューに入れられ、Phreak は、実行する可能性が最も高いルールに基づいてヒューリスティックを使用し、次に評価するルールを計算して選択します。すべての必要な入力値がルールに生成されると、ルールは関連するパターン一致データに **リンクされている** と見なされます。続いて Phreak は、このルールを表す目標を作成し、ルールの顕著性によって順序付けられた優先度キューにこの目標を置きます。目標が作成されたルールのみが評価され、その他の潜在的なルール評価は遅延されます。個別のルールは評価されますが、ノードの共有は引き続きセグメンテーション処理を通じて行われます。

タプル指向の Rete とは異なり、Phreak 伝播はコレクション指向です。評価されているルールの場合、デシジョンエンジンは最初のノードにアクセスし、キューに入れられたすべての insert、update、および delete の各アクションを処理します。結果は 1 つのセットに追加され、そのセットは子ノードに伝播されます。子ノードでは、キューに入れられたすべての insert、update、および delete の各アクションが処理され、その結果を同じセットに追加します。続いてそのセットは次の子ノードに伝播され、同じ処理が終了ノードに達するまで繰り返されます。このサイクルは、特定のルール設定にパフォーマンス上の利点を提供できるバッチ処理効果をもたらします。

ルールのリンクやリンク解除は、ネットワークセグメンテーションに基づいて、レイヤー化されたビットマスクシステムを使用して行われます。ルールネットワークが構築されると、同じ一連のルールで共有されるルールネットワークノードに対して、複数のセグメントが作成されます。ルールはセグメント

のパスで設定されます。ルールが他のどのルールともノードを一切共有しない場合、このルールは単一のセグメントになります。

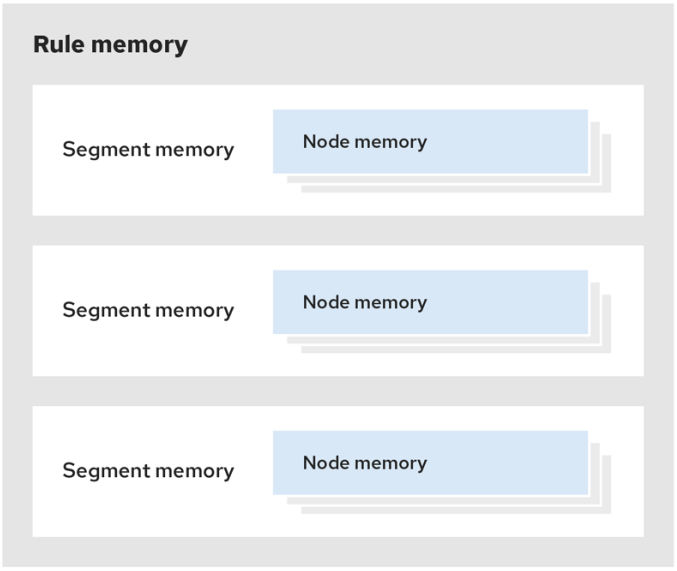
セグメントの各ノードにビットマスクの補正値が割り当てられます。別のビットマスクは、以下の要件に従って、ルールのパスの各セグメントに割り当てられます。

- ノードの入力が少なくとも 1 つ存在する場合、ノードビットは **on** 状態に設定されます。
- セグメントの各ノードのビットが **on** 状態に設定されている場合は、セグメントビットも **on** 状態に設定されます。
- いずれかのノードビットが **off** 状態に設定されている場合は、セグメントも **off** 状態に設定されます。
- ルールのパスの各セグメントが **on** 状態に設定されている場合、ルールはリンクされていると見なされ、目標が作成され、ルールを評価するためのスケジュールが組まれます。

変更されたノード、セグメント、およびルールを追跡する際に、同じビットマスクの手法が使用されます。この追跡機能により、すでにリンクされたルールの評価目標が作成後に変更された場合は、このルールの評価をスケジュールから外すことができます。その結果、部分的な一致を評価できるルールはありません。

Rete におけるメモリーの単一ユニットとは対照的に、Phreak には、ノード、セグメント、およびルールのメモリータイプから成る 3 つのレイヤーのコンテキストメモリーがあります。そのため、このルール評価の処理は Phreak で可能となります。このようなレイヤーがあることで、ルールを評価する際は、より深くコンテキストを解釈することが可能になります。

図84.1 Phreak の 3 つのレイヤーから成るメモリーシステム

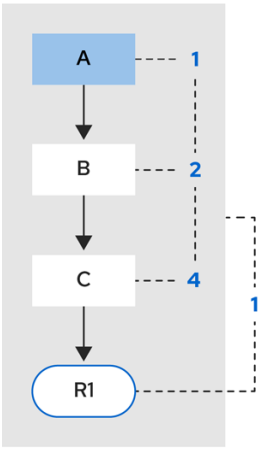


RHDM_29_0319

以下の例では、Phreak の 3 つのレイヤーから成るメモリーシステムで、ルールがどのように組織化および評価されているかを説明します。

例 1: 3 つのパターンがある単一のルール (R1): A、B、および C。ルールは、ノード用にビット 1、2、および 4 を持つ単一のセグメントを形成します。単一のセグメントのビットの補正値は 1 です。

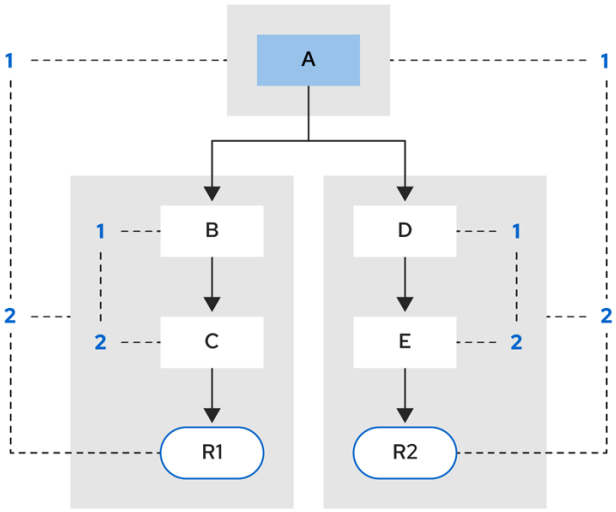
図84.2 例 1: 単一ルール



RHDM_29_0319

例 2: ルール R2 が追加され、パターン A を共有します。

図84.3 例 2: パターン共有の 2 つのルール



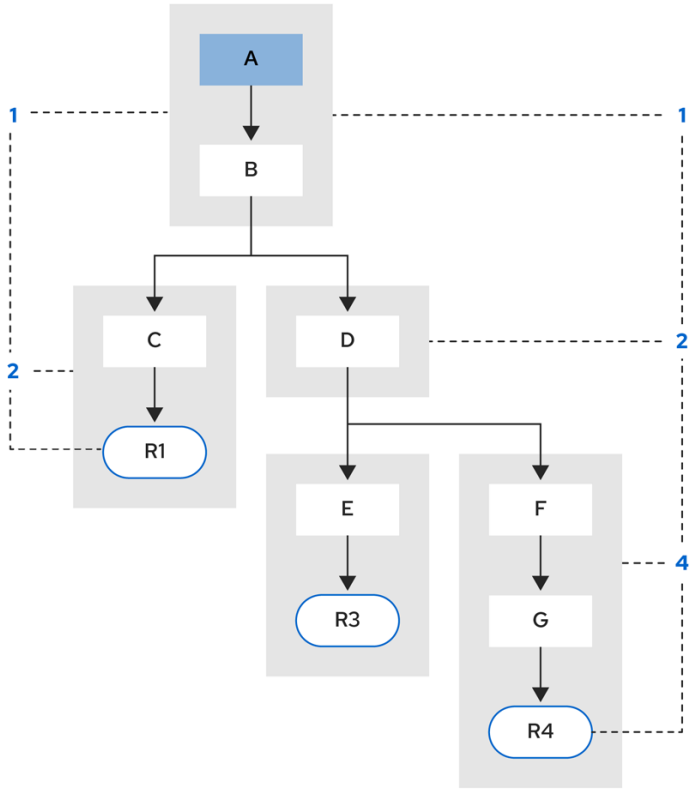
RHDM_29_0319

パターン A は独自のセグメントに置かれ、その結果、各ルールには 2 つのセグメントが作成されます。これら 2 つのセグメントは、各ルールのパスを作成します。1 つ目のセグメントは、両方のパスで共有されます。パターン A がリンクされると、セグメントはリンクされます。続いてこのセグメントは、セグメントが共有される各パスでこれを繰り返します。この時ビット 1 は **on** に設定されます。パターン B および C がその後オンになると、パス R1 の 2 つ目のセグメントがリンクされ、これによりビット 2 が R1 に対してオンになります。ビット 1 およびビット 2 が R1 に対してオンになったことで、ルールがリンクされるようになり、目標が作成され、ルールを今後評価して実行するためのスケジュールが立てられます。

ルールが評価されると、セグメントによって一致の結果が共有できるようになります。各セグメントには、そのセグメントのすべての insert、update、および delete をキューに入れるステージングメモリーがあります。R1 が評価されるとルールはパターン A を処理し、これにより一連のタプルが作成されます。アルゴリズムがセグメンテーションの分割を検出し、セット内の各 insert、update、および delete にピアタプルを作成します。そして、これらを R2 のステージングメモリーに追加します。次にこれらのタプルは、既存のステージタプルとマージされ、最終的に R2 が評価されると実行されます。

例 3: ルール R3 とルール R4 が追加され、パターン A とパターン B を共有します。

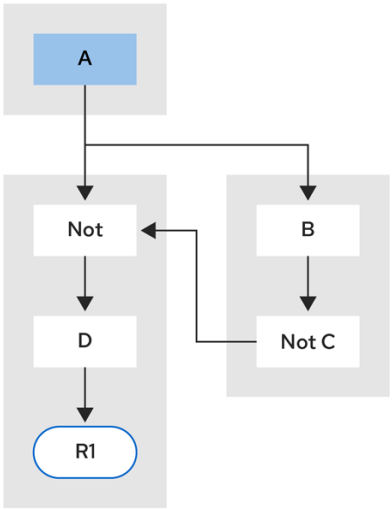
図84.4 例 3: パターン共有の 3 つのルール



ルール R3 と ルール R4 にはセグメントが 3 つあり、R1 にはセグメントが 2 つあります。R1、R3、および R4 がパターン A とパターン B を共有し、R3 と R4 がパターン D を共有しています。

例 4: パターンの共有なしでサブネットワークがある単一ルール (R1)

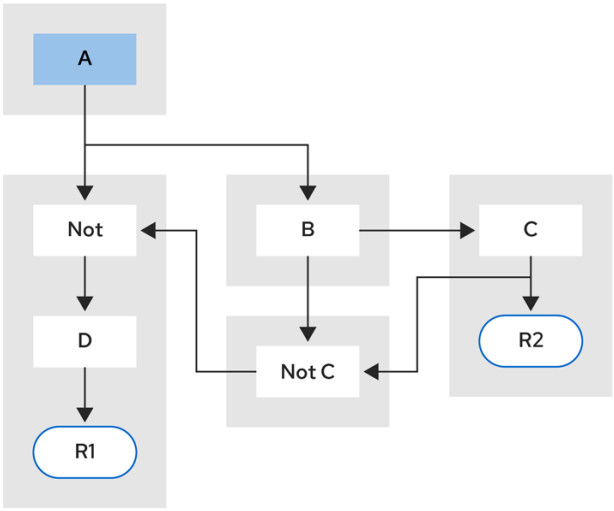
図84.5 例 4: パターンの共有なしで、サブネットワークがある単一のルール



Not、**Exists**、または **Accumulate** ノードに 1 つ以上の要素がある場合は、サブネットワークが形成されます。この例では、要素 **B not(C)** がサブネットワークを形成します。要素 **not(C)** は、サブネットワークを必要としない単一要素のため、**Not** ノード内にマージされます。サブネットワークは専用のセグメントを使用します。ルール R1 には、依然として 2 つのセグメントのパスがあり、サブネットワークは別の内部のパスを形成します。サブネットワークがリンクされると、これは外部のセグメントにもリンクされます。

例 5: ルール R2 と共有するサブネットワークのあるルール R1

図84.6 例 5: 2 つのルールのうち、1つはサブネットワークとパターンを共有



ルールのサブネットワークノードは、サブネットワークのない別のルールと共有することができます。このように共有することで、サブネットワークのセグメントが 2 つのセグメントに分割されることになります。

制約のある **Not** ノードおよび **Accumulate** ノードは、セグメントのリンクを外すことは一切できず、ビットがオンになっていると常に考えられています。

Phreak 評価のアルゴリズムは、メソッド再帰ベースではなく、スタックベースです。**StackEntry** を使用して現在評価中のノードを表す場合は、いつでもルール評価を一時的に停止したり、再開したりすることができます。

ルール評価がサブネットワークに到達すると、**StackEntry** オブジェクトが、外部パスのセグメントおよびサブネットワークセグメント用に作成されます。最初にサブネットワークのセグメントが評価され、セットがサブネットワークパスの終わりに到達すると、そのセグメントがフィードする外部ノードのステージングリストにマージされます。次に、以前の **StackEntry** オブジェクトが再開し、サブネットワークの結果を処理できるようになります。この処理には、子ノードに伝播される前にすべての作業がバッチで完了するという付加的な利点があります。これは、**Accumulate** ノードにとって特に有利です。

同じスタックシステムは、効率的な後向き連鎖に使用されます。ルール評価がクエリーノードに到達すると、評価は一時的に停止し、クエリーがスタックに追加されます。続いてクエリーは、結果セットを生成するために評価されます。結果セットは、再開した **StackEntry** オブジェクトのメモリーロケーションに保存され、回収されて子ノードに伝播されます。クエリー自体が他のクエリーを呼び出した場合はこの処理は繰り返され、その一方で、現在のクエリーは一時的に停止し、現在のクエリーノード用に新しい評価が設定されます。

84.1.1. 前向き連鎖と後向き連鎖を使用したルール評価

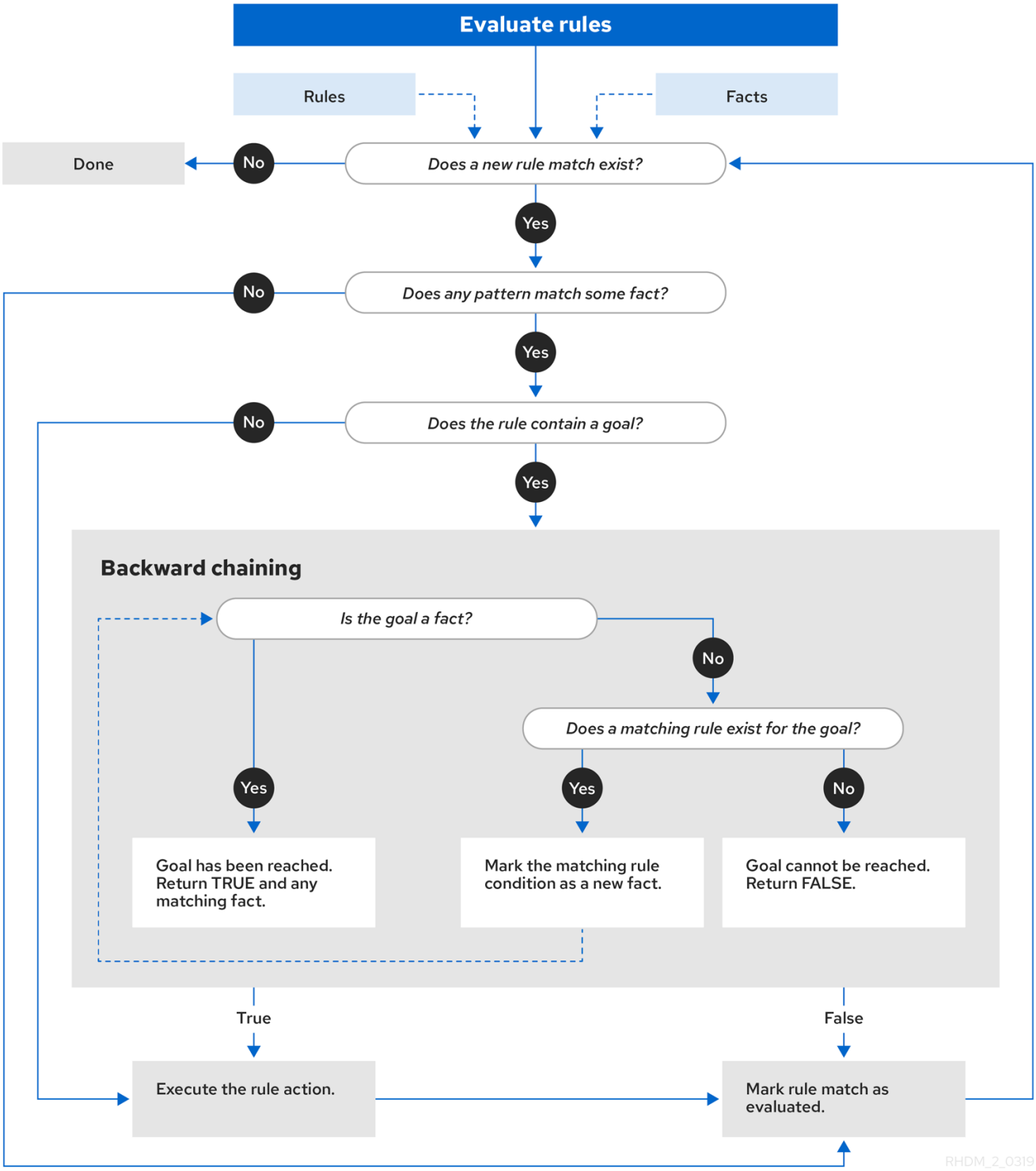
Red Hat Process Automation Manager のデシジョンエンジンは、前向き連鎖と後向き連鎖の両方を使用してルールを評価する、ハイブリッドの理由付けシステムです。前向き連鎖のルールシステムは、ディジョンエンジンのワーキングメモリーにあるファクトで開始して、そのファクトへの変更に反応するデータ駆動型のシステムです。オブジェクトがワーキングメモリーに挿入されると、その変更の結果として True となるルールの条件はすべて、アジェンダによって実行されるようにスケジュールされます。

反対に、後向き連鎖のルールシステムは、しばしば再帰を使用して、デシジョンエンジンが満たそうと

する結論から開始する目的駆動型のシステムです。システムが結論または目的に到達できない場合は、サブとなる目的、つまり、現在の目的の一部を完了する結論を検索します。システムは、最初の結論が満たされるか、すべてのサブとなる目的が満たされるまでこのプロセスを続行します。

以下の図は、デシジョンエンジンが、ロジックフローで後向き連鎖のセグメントと、前向き連鎖全体を使用してルールを評価する方法を例示します。

図84.7 前向き連鎖と後向き連鎖を使用したルール評価のロジック



84.2. ルールベースの設定

Red Hat Process Automation Manager には、**RuleBaseConfiguration.java** オブジェクトが含まれます。これを使用して、デシジョンエンジンで例外ハンドラーの設定、マルチスレッドの実行、および順次モードを設定することができます。

ルールベースの設定オプションに関しては、Red Hat カスタマーポータル から ZIP ファイル **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**~/rhpam-7.11.0-sources/src/drools-$VERSION/drools-core/src/main/java/org/drools/core/RuleBaseConfiguration.java** に移動してください。

以下のルールベースの設定オプションは、デシジョンエンジンで利用可能です。

drools.consequenceExceptionHandler

設定すると、このシステムプロパティーは、ルールの結果によって例外の出力を管理するクラスを定義します。このプロパティーを使用して、デシジョンエンジンのルール評価にカスタムの例外ハンドラーを指定できます。
デフォルト値: **org.drools.core.runtime.rule.impl.DefaultConsequenceExceptionHandler**

以下のオプションのいずれかを使用して、カスタムの例外ハンドラーを指定できます。

- システムプロパティーで例外ハンドラーを指定:

```
drools.consequenceExceptionHandler=org.drools.core.runtime.rule.impl.MyCustomConsequenceExceptionHandler
```

- プログラムを用いて KIE ベースを作成中に例外ハンドラーを指定:

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(ConsequenceExceptionHandlerOption.get(MyCustomConsequenceExceptionHandler.class));
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

drools.multithreadEvaluation

有効になると、このシステムプロパティーは、Phreak のルールネットワークを個々のパーティションに分割することで、デシジョンエンジンが並列してルールを評価できるようにします。このプロパティーを使用すると、特定のルールベースのルール評価の速度をあげることができます。
デフォルト値は **false** です。

以下のオプションのいずれかを使用してマルチスレッド評価を有効化できます。

- マルチスレッド評価のシステムプロパティーを有効化:

```
drools.multithreadEvaluation=true
```

- プログラムを用いて KIE ベースを作成中にマルチスレッド評価を有効化:

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(MultithreadEvaluationOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

警告

クエリー、顕著性、またはアジェンダグループを使用するルールは現在、並列のデシジョンエンジンではサポートされていません。これらのルールの要素が KIE ベースに存在する場合、コンパイラーは警告を発し、自動的にシングルスレッドの評価に切り替えます。しかし、ケースによっては、デシジョンエンジンはサポートされていないルールの要素を検出できず、ルールが間違って評価される可能性があります。たとえば、ルールが DRL ファイル内のルールの順序によって与えられた暗黙の顕著性に依存する場合は、デシジョンエンジンが検出できない可能性があり、その結果、サポートされていない顕著性の属性により、間違った評価となります。

drools.sequential

これを有効にすると、このシステムプロパティーは、デシジョンエンジンの順次モードを有効にします。順次モードでは、デシジョンエンジンは、ワーキングメモリーでの変更に関係なく、デシジョンエンジンアジェンダにリスト化された順番でルールを一度評価します。これは、デシジョンエンジンがルールの **insert**、**modify**、または **update** ステートメントをすべて無視し、ルールを単一シーケンスで実行することを意味します。その結果、ルールの実行は順次モードの方が速くなる可能性がありますが、重要な更新がルールに適用されない可能性があります。ステートレスな KIE セッションを使用し、アジェンダ内の後続のルールに対してルールの実行による影響を与えないようにする場合に、このプロパティーを使用できます。順次モードは、ステートレス KIE セッションのみに適用されます。
デフォルト値は **false** です。

以下のオプションのいずれかを使用して、順次モードを有効化できます。

- 順次モードのシステムプロパティーの有効化:

```
drools.sequential=true
```

- プログラムを用いて KIE ベースを作成中に順次モードを有効化:

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(SequentialOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

- 特定の Red Hat Process Automation Manager プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で順次モードを有効化

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" sequential="true" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

84.3. PHREAK における順次モード

順次モードは、デシジョンエンジンにおける高度なルールベースの設定で、Phreak がサポートしています。順次モードでは、デシジョンエンジンは、ワーキングメモリーでの変更に関係なく、デシジョンエンジンアジェンダにリスト化された順番でルールを一度評価します。順次モードでは、デシジョンエンジンがルールのステートメント **insert**、**modify**、または **update** をすべて無視し、ルールを単一シーケンスで実行します。その結果、ルールの実行は順次モードの方が速くなる可能性がありますが、重要な更新がルールに適用されない可能性があります。

ステートフルな KIE セッションは本来、以前呼び出された KIE セッションのデータを使用するため、順次モードが適用されるのはステートレスな KIE セッションのみとなります。ステートレスな KIE セッションを使用し、ルールを実行して、アジェンダ内の後続のルールを決定するには、順次モードを有効にしないでください。デシジョンエンジンでは、デフォルトで順次モードは無効となっています。

以下のオプションのいずれかを使用して、順次モードを有効にします。

- システムプロパティー **drools.sequential** を **true** に設定:
- プログラムを用いて KIE ベースを作成中に順次モードを有効化:

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(SequentialOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

- 特定の Red Hat Process Automation Manager プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で順次モードを有効化

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" sequential="true" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

以下のオプションのいずれかを使用して、順次モードが動的アジェンダを使用するように設定します。

- システムプロパティー **drools.sequential.agenda** を **dynamic** に設定:
- プログラムを用いて KIE ベースを作成中に順次アジェンダオプションを設定:

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(SequentialAgendaOption.DYNAMIC);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

順次モードを有効にすると、以下の方法でデシジョンエンジンがルールを評価します。

1. ルールは、ルールセットの顕著性および位置によって順序付けられます。
2. 考えられるルールの一致ごとの要素が作成されました。要素の位置が実行の順番を示しています。

3. right-input オブジェクトメモリーを除いて、ノードメモリーは無効化されました。

4. left-input アダプターノードの伝播は切断され、ノードを持つオブジェクトは **Command** オブジェクトで参照されます。**Command** オブジェクトは、後で実行するためにワーキングメモリーのリストに追加されます。

5. すべてのオブジェクトがアサートされると、**Command** オブジェクトのリストが確認され、実行されます。

6. リストの実行によって生じるすべての一致は、ルールのシーケンス番号に基づいて要素に追加されます。

7. 一致を含む要素は、順次実行されます。ルール実行の最大数を設定している場合は、デシジョンエンジンがアジェンダで実行するルールの数がこの最大数を超えることはありません。

順次モードでは、**LeftInputAdapterNode** ノードが **Command** オブジェクトを作成し、これをデシジョンエンジンのワーキングメモリーリストに追加します。この **Command** オブジェクトには、**LeftInputAdapterNode** ノードおよび伝播されたオブジェクトへの参照が含まれます。これらの参照は、right-input 伝播が left-input の結合を試みる必要がないように、挿入時にあらゆる left-input 伝播を停止します。また、参照により、left-input メモリーが不要となります。

すべてのノードのメモリーはオフになっています。これには、left-input のタプルメモリーも含まれますが、right-input オブジェクトメモリーは除外されます。すべてのアサーションが終了し、すべてのオブジェクトの right-input メモリーが生成されると、デシジョンエンジンは **LeftInputAdatperNode** オブジェクトと **Command** オブジェクトのリストを繰り返します。このオブジェクトはネットワークを伝播し、right-input オブジェクトに結合しようと試みますが、left input では保持されません。

タプルをスケジュールするための優先度キューがあるアジェンダは、各ルールの要素に置き換えます。**RuleTerminalNode** ノードのシーケンス番号は、一致を配置する要素を示します。**Command** オブジェクトがすべて終了すると、要素が確認され、既存の一致が実行されます。要素内で最初と最後に生成されたセルを保持してパフォーマンスを向上します。

ネットワークが構築されると、各 **RuleTerminalNode** ノードは、顕著性の番号と、ネットワークに追加された順序をベースとするシーケンス番号を受け取ります。

right-input ノードメモリーは、通常、オブジェクトをすばやく削除するためのハッシュマップです。オブジェクトの削除はサポートされていないため、オブジェクトの値がインデックス化されていない場合は、Phreak がオブジェクトリストを使用します。大量のオブジェクトに対しては、インデックス化されたハッシュマップがパフォーマンスを向上させます。オブジェクトのインスタンスが少ししかないと、Phreak はインデックスの代わりにオブジェクトリストを使用します。

第85章 複合イベント処理 (CEP)

Red Hat Process Automation Manager では、イベントとは、ある時点でのアプリケーションドメインの状態の大幅な変化の記録です。ドメインのモデル化方法に応じて、状態の変化は単一のイベント、複数のアトミックイベント、または相関イベントの階層によって表される場合があります。複合イベント処理 (CEP) の観点から見ると、イベントは特定の時点で発生するファクトまたはオブジェクトのタイプであり、ビジネスルールはそのファクトまたはオブジェクトからのデータにどのように反応するかを定義したものです。たとえば、株式ブローカーアプリケーションでは、株価の変動、売り手から買い手への所有権の変更、またはアカウント所有者の残高の変更はすべて、所定の時間にアプリケーションドメインの状態で変更が発生したため、イベントと見なされます。

Red Hat Process Automation Manager のデシジョンエンジンは、複合イベント処理 (CEP) を使用して、イベントのコレクション内の複数のイベントを検出および処理し、イベント間に存在する関係を明らかにするほか、イベントとイベント同士の関係から新しいデータを推論します。

CEP のユースケースは、複数の要件と目標をビジネスルールのユースケースと共有しています。

ビジネスの観点から見ると、ビジネスルールの定義は多くの場合は、イベントによってトリガーされるシナリオの発生に基づいて定義されます。以下の例では、イベントがビジネスルールの基礎を形成しています。

- アルゴリズム取引アプリケーションでは、株価が始値を X パーセント上回った場合は、ルールがアクションを実行します。価格の上昇は、株式取引アプリケーションのイベントによって示されます。
- 監視アプリケーションでは、サーバールームの温度が Y 分で X 度上昇すると、ルールがアクションを実行します。センサーの測定値はイベントによって示されます。

技術的な観点から見ると、ビジネスルールの評価と CEP には、以下の重要な類似点があります。

- ビジネスルールの評価と CEP の両方で、エンタープライズインフラストラクチャーとアプリケーションとのシームレスな統合が必要です。これは、ライフサイクル管理、監査、およびセキュリティーにおいて特に重要です。
- ビジネスルールの評価と CEP の両方には、パターン一致などの機能要件と、応答時間の制限やクエリールールの説明などの非機能要件があります。

CEP シナリオには、以下の重要な特徴があります。

- シナリオは通常、大量のイベントを処理しますが、関連するイベントはごく一部です。
- 通常、イベントは不変であり、状態の変化の記録を表します。
- ルールとクエリーはイベントに対して実行され、検出されたイベントパターンに対応する必要があります。
- 通常、関連するイベントには強い一時的な関係があります。
- 個々のイベントは優先されません。CEP システムは、関連するイベントのパターンとイベント間の関係に優先順位を付けます。
- 通常、イベントは設定および集約を行う必要があります。

これらの一般的な CEP シナリオの特徴を前提として、Red Hat Process Automation Manager の CEP システムは、イベント処理を最適化するために以下の機能をサポートしています。

- 適切なセマンティクスによるイベント処理

- イベントの検出、相関、集約、および設定
- イベントストリーム処理
- イベント間の一時的な関係をモデル化する一時的な制約
- 重要なイベントのスライディングウィンドウ
- セッションスコープの統合クロック
- CEP ユースケースに必要なイベントのボリューム
- リアクティブルール
- デシジョンエンジンへのイベント入力アダプター (パイプライン)

85.1. 複合イベント処理 (CEP) におけるイベント

Red Hat Process Automation Manager では、イベントとは、ある時点でのアプリケーションドメインの状態の大幅な変化の記録です。ドメインのモデル化方法に応じて、状態の変化は単一のイベント、複数のアトミックイベント、または相関イベントの階層によって表される場合があります。複合イベント処理 (CEP) の観点から見ると、イベントは特定の時点で発生するファクトまたはオブジェクトのタイプであり、ビジネスルールはそのファクトまたはオブジェクトからのデータにどのように反応するかを定義したものです。たとえば、株式ブローカーアプリケーションでは、株価の変動、売り手から買い手への所有権の変更、またはアカウント所有者の残高の変更はすべて、所定の時間にアプリケーションドメインの状態で変更が発生したため、イベントと見なされます。

イベントには、以下の重要な特徴があります。

- **不変性:** イベントは、過去のある時点で発生した変更の記録であり、変更することはできません。

注記

デシジョンエンジンは、イベントを表す Java オブジェクトに不変性を強制しません。この動作により、イベントデータの強化が可能になります。アプリケーションは、未入力のイベント属性を入力できる必要があります。そしてこれらの属性は、推論データでイベントを強化するためにデシジョンエンジンによって使用されます。ただし、すでに入力されているイベント属性は変更しないでください。

- **強力な一時的な制約:** 通常、イベントに関係するルールは、相互に関連する異なる時点で発生する複数のイベントの相関を必要とします。
- **管理されたライフサイクル:** イベントは不変であり、一時的な制約があるため、通常は特定の期間にのみ関連します。これは、デシジョンエンジンがイベントのライフサイクルを自動的に管理できることを意味します。
- **スライディングウィンドウが使用可能:** イベントで時間または長さのスライディングウィンドウを定義できます。スライディングタイムウィンドウは、イベントを処理できる特定の期間です。スライディングレングスウィンドウは、処理可能な指定されたイベントの数です。

85.2. ファクトのイベントとしての宣言

Java クラスまたは DRL ルールファイルでファクトをイベントとして宣言すると、デシジョンエンジン

が複雑なイベント処理中にファクトをイベントとして処理できます。ファクトは、interval-based イベントまたは point-in-time イベントとして宣言できます。interval-based のイベントには持続期間があり、その持続期間が経過するまでデシジョンエンジンのワーキングメモリーで持続します。point-in-time イベントに持続期間はなく、基本的には期間がゼロの interval-based イベントになります。

手順

Java クラスまたは DRL ルールファイルの関連するファクトタイプについては、**@role(event)** メタデータタグとパラメーターを入力します。**@role** メタデータタグは、以下の 2 つの値を受け入れます。

- **fact**: (デフォルト) タイプを通常のファクトとして宣言
- **event**: タイプをイベントとして宣言

たとえば、以下のスニペットは、株式ブローカーアプリケーションの **StockPoint** ファクトタイプをイベントとして処理する必要があることを宣言しています。

ファクトタイプをイベントとして宣言

```
import some.package.StockPoint

declare StockPoint
  @role( event )
end
```

StockPoint が、既存のクラスではなく DRL ルールファイルで宣言されたファクトタイプである場合は、アプリケーションコードでイベントをインラインで宣言できます。

ファクトタイプをインラインで宣言し、イベントロールに割り当て

```
declare StockPoint
  @role( event )

  datetime : java.util.Date
  symbol : String
  price : double
end
```

85.3. イベントのメタデータタグ

デシジョンエンジンは、デシジョンエンジンのワーキングメモリーに挿入されるイベントに以下のメタデータタグを使用します。必要に応じて、Java クラスまたは DRL ルールファイルでデフォルトのメタデータタグ値を変更できます。

注記

VoiceCall クラスを参照する本セクションの例では、サンプルアプリケーションドメインモデルに以下のクラスの詳細が含まれていることを前提としています。

Telecom ドメインモデルの例における VoiceCall ファクトクラス

```
public class VoiceCall {
  private String  originNumber;
  private String  destinationNumber;
  private Date    callDateTime;
  private long    callDuration;  // in milliseconds

  // Constructors, getters, and setters
}
```

@role

このタグは、指定のファクトタイプが複雑なイベントの処理時にデシジョンエンジンにて通常のファクトまたはイベントとして処理されるかどうかを決定します。
デフォルトパラメーター: **fact**

サポート対象のパラメーター: **fact**、**event**

```
@role( fact | event )
```

例: イベントタイプとして VoiceCall の宣言

```
declare VoiceCall
  @role( event )
end
```

@timestamp

このタグは、デシジョンエンジンのすべてのイベントに自動的に割り当てられます。デフォルトでは、この時間はセッションクロックにより提供され、デシジョンエンジンのワーキングメモリーへの挿入時にイベントに割り当てられます。セッションクロックが追加するデフォルトのタイムスタンプの代わりに、カスタムのタイムスタンプ属性を指定できます。
デフォルトパラメーター: デシジョンエンジンのセッションクロックが追加する時間

サポート対象のパラメーター: セッションクロックタイムまたはカスタムのタイムスタンプ属性

```
@timestamp( <attributeName> )
```

例: VoiceCall のタイムスタンプ属性の宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
end
```

@duration

このタグは、デシジョンエンジンのイベントの持続期間を決定します。イベントは、interval-based

イベントまたは point-in-time イベントのいずれかになります。interval-based のイベントには持続期間があり、その持続期間が経過するまでデシジョンエンジンのワーキングメモリーで持続します。point-in-time イベントに持続期間はなく、基本的には期間がゼロの interval-based イベントになります。デフォルトでは、デシジョンエンジンのすべてのイベントの持続期間は 0 です。デフォルトの代わりに、カスタムの持続期間属性を指定できます。
デフォルトパラメーター: null (ゼロ)

サポート対象のパラメーター: カスタムの持続期間属性

```
@duration( <attributeName> )
```

例: VoiceCall の持続期間属性の宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
end
```

@expires

このタグは、デシジョンエンジンのワーキングメモリーでイベントの有効期限が切れるまでの時間を決定します。デフォルトでは、イベントは現在のルールのいずれにも一致せず、それらのいずれもアクティベートできなくなった時点で失効します。イベント失効までの期間を定義できます。また、このタグの定義は、KIE ベースの一時的な制約やスライディングウィンドウから算出した暗黙的な有効期限のオフセットもオーバーライドします。デシジョンエンジンがストリームモードで実行中の場合にのみ、このタグを使用できます。
デフォルトパラメーター: Null (イベントがルールに一致せず、ルールをアクティブにできなくなるとイベントの有効期限が切れる)

サポート対象のパラメーター: **[#d][#h][#m][#s][[ms]]** 形式のカスタムの **timeOffset** 属性

```
@expires( <timeOffset> )
```

例: VoiceCall イベントに対する有効期限のオフセットの宣言

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
  @expires( 1h35m )
end
```

85.4. デシジョンエンジンのイベント処理モード

デシジョンエンジンは、クラウドモードまたはストリームモードで実行します。クラウドモードでは、デシジョンエンジンは、ファクトを、時間に依存せず、特定の順序で、時間的な制約のないファクトとして処理します。ストリームモードでは、デシジョンエンジンは、ファクトをリアルタイムまたはほぼリアルタイムで、強力な時間的制約のあるイベントとして処理します。ストリームモードは同期を使用して、Red Hat Process Automation Manager でのイベント処理を可能にします。

クラウドモード

クラウドモードは、デシジョンエンジンのデフォルトの動作モードです。クラウドモードでは、デシジョンエンジンはイベントを順不同のクラウドとして扱います。イベントには依然としてタイムスタンプがありますが、クラウドモードでは現在の時刻が無視されるため、クラウドモードで実行されているデシジョンエンジンは、タイムスタンプから関連性を引き出すことができません。このモードでは、ルール制約を使用して一致するタプルを検索し、ルールを有効にして実行します。
クラウドモードでは、ファクトに対して追加要件を課すことは一切ありません。ただし、このモードのデシジョンエンジンには時間の概念がないため、スライディングウィンドウや自動ライフサイクル管理などの時間機能は使用できません。クラウドモードでは、イベントが必要なくなると、明示的にイベントを取り消す必要があります。

クラウドモードでは、以下の要件が課されることはありません。

- デシジョンエンジンには時間の概念がないため、クロックの同期はありません
- デシジョンエンジンは、イベントを順不同のクラウドとして処理するため、イベントの順序付けはありませんが、デシジョンエンジンは順不同のクラウドに対してルールを一致させます。

関連する設定ファイルでシステムプロパティーを設定するか、または Java クライアント API を使用して、クラウドモードを指定できます。

システムプロパティーを使用してクラウドモードを設定

```
drools.eventProcessingMode=cloud
```

Java クライアント API を使用してクラウドモードを設定

```
import org.kie.api.conf.EventProcessingOption;
import org.kie.api.KieBaseConfiguration;
import org.kie.api.KieServices.Factory;

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption(EventProcessingOption.CLOUD);
```

特定の Red Hat Process Automation Manager プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) の **eventProcessingMode="<mode>"** KIE ベース属性を使用して、クラウドモードを指定することもできます。

プロジェクト kmodule.xml ファイルを使用してクラウドモードを設定

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="cloud"
packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

ストリームモード

イベントがデシジョンエンジンに挿入されると、デシジョンエンジンは、ストリームモードを使用することで、イベントを時系列およびリアルタイムに処理できます。ストリームモードでは、(異なるストリームのイベントを時系列で処理できるように) デシジョンエンジンはイベントのストリーム

を同期し、時間または長さのスライディングウィンドウを実装し、自動ライフサイクル管理を可能にします。
以下の要件がストリームモードに適用されます。

- 各ストリームのイベントは、時系列に並べる必要があります。
- イベントストリームを同期するには、セッションクロックが必要です。

注記

お使いのアプリケーションが、ストリーム間でイベントの順序付けを強制する必要はありませんが、同期されていないイベントストリームを使用すると、予期しない結果が生じる可能性があります。

関連する設定ファイルでシステムプロパティーを設定するか、または Java クライアント API を使用して、ストリームモードを指定できます。

システムプロパティーを使用してストリームモードを設定

```
drools.eventProcessingMode=stream
```

Java クライアント API を使用してストリームモードを設定

```
import org.kie.api.conf.EventProcessingOption;
import org.kie.api.KieBaseConfiguration;
import org.kie.api.KieServices.Factory;

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption(EventProcessingOption.STREAM);
```

特定の Red Hat Process Automation Manager プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) の **eventProcessingMode="<mode>"** KIE ベース属性を使用して、ストリームモードを指定することもできます。

プロジェクト kmodule.xml ファイルを使用してストリームモードを設定

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

85.4.1. デシジョンエンジンのストリームモードにおける負のパターン

負のパターンは、条件が一致しない場合のパターンです。たとえば、以下の DRL ルールは、火災が検知されてもスプリンクラーが有効になっていない場合に、火災報知機を有効にします。

負のパターンでの火災報知機ルール

```
rule "Sound the alarm"
when
  $f : FireDetected()
  not(SprinklerActivated())
then
  // Sound the alarm.
end
```

クラウドモードでは、デシジョンエンジンはすべてのファクト (通常のファクトとイベント) が前もって知らされているものと想定し、負のパターンをすぐに評価します。ストリームモードでは、意思決定エンジンは、ルールをアクティブ化する前に設定された時間待機するファクトの時間的制約をサポートできます。

ストリームモードでは、同じ例のルールは、通常どおりに火災報知機を有効にしますが、10 秒間の遅延が適用されます。

負のパターンと遅延時間のある火災報知機ルール (ストリームモードのみ)

```
rule "Sound the alarm"
when
  $f : FireDetected()
  not(SprinklerActivated(this after[0s,10s] $f))
then
  // Sound the alarm.
end
```

以下の修正された火災報知器ルールは、10 秒ごとに 1 つの **Heartbeat** イベントが発生することを想定しています。想定されるイベントが発生しないと、ルールが実行します。このルールは、最初のパターンと負のパターンの両方で同じタイプのオブジェクトを使用します。負のパターンには、実行前に 0 秒から 10 秒待機する一時的な制約があり、**$h** にバインドされている **Heartbeat** イベントを除外して、ルールを実行できるようにします。時間的制約 **[0s, ...]** が本質的にそのイベントの再照合を除外しないため、ルールを実行するには、バインドされたイベント **$h** を明示的に除外する必要があります。

負のパターンでバインドされたイベントを除外する火災報知器ルール (ストリームモードのみ)

```
rule "Sound the alarm"
when
  $h: Heartbeat() from entry-point "MonitoringStream"
  not(Heartbeat(this != $h, this after[0s,10s] $h) from entry-point "MonitoringStream")
then
  // Sound the alarm.
end
```

85.5. ファクトタイプに対するプロパティー変更の設定およびリスナー

デフォルトでは、デシジョンエンジンは、ルールがトリガーされるたびに、ファクトタイプに対するすべてのファクトパターンを再評価しません。代わりに、指定のパターン内に制約またはバインドされている変更されたプロパティーのみに対応します。たとえば、ルールが、ルールアクションの一環として **modify()** を呼び出すものの、アクションが KIE ベースで新しいデータを生成しない場合は、データが変更されないため、デシジョンエンジンはすべてのファクトパターンを自動的に再評価しません。このプロパティーのリアクティビティー動作は、KIE ベースでの不要な再帰を阻止し、より効率的なルール評価をもたらします。また、この動作は無限再帰を回避するために **no-loop** ルール属性を必ずしも使用する必要がないことを意味します。

以下の **KnowledgeBuilderConfiguration** オプションを使用して、このプロパティーリアクティビティー動作を変更または無効にできます。次に、Java クラスまたは DRL ファイルでプロパティー変更設定を使用し、必要に応じてプロパティーリアクティビティーを調整します。

- **ALWAYS**: (デフォルト) すべてのタイプはプロパティーリアクティブです。ただし、**@classReactive** プロパティー変更設定を使用して、特定タイプのプロパティーリアクティビティーを無効にできます。
- **ALLOWED**: すべてのタイプはプロパティーリアクティブではありません。ただし、**@propertyReactive** プロパティー変更設定を使用して、特定タイプのプロパティーリアクティビティーを有効にできます。
- **DISABLED**: すべてのタイプはプロパティーリアクティブではありません。すべてのプロパティー変更リスナーは無視されます。

KnowledgeBuilderConfiguration におけるプロパティーリアクティビティー設定の例

```
KnowledgeBuilderConfiguration config =
KnowledgeBuilderFactory.newKnowledgeBuilderConfiguration();
config.setOption(PropertySpecificOption.ALLOWED);
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder(config);
```

または、Red Hat Process Automation Manager ディストリビューションにおける **standalone.xml** ファイルの **drools.propertySpecific** システムプロパティーを更新できます。

システムプロパティーにおけるプロパティーリアクティビティー設定の例

```
<system-properties>
  ...
  <property name="drools.propertySpecific" value="ALLOWED"/>
  ...
</system-properties>
```

デシジョンエンジンは、ファクトクラスまたは宣言された DRL ファクトタイプに対して、以下のプロパティー変更の設定およびリスナーをサポートします。

@classReactive

デシジョンエンジンでプロパティーリアクティビティーが **ALWAYS** に設定されている場合 (すべてのタイプはプロパティーリアクティブ)、このタグは特定の Java クラスまたは宣言された DRL ファクトタイプに対してデフォルトのプロパティーリアクティビティー動作を無効にします。このタグは、特定パターン内に制約またはバインドされる変更されたプロパティーのみに対応するのではなく、ルールがトリガーされるたびに指定されたファクトタイプのすべてのファクトパターンをデシジョンエンジンが再評価する必要がある場合に使用できます。

例: DRL タイプの宣言におけるデフォルトのプロパティーリアクティビティーの無効化

```
declare Person
  @classReactive
    firstName : String
    lastName : String
end
```

例: Java クラスにおけるデフォルトのプロパティーリアクティビティーの無効化

```
@classReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@propertyReactive

プロパティーリアクティビティーがデシジョンエンジンで **ALLOWED** に設定されている場合 (指定されていない場合、すべてのタイプはプロパティーリアクティブではない)、このタグは特定の Java クラスまたは宣言された DRL ファクトタイプに対してプロパティーリアクティビティーを有効にします。ルールがトリガーされるたびにファクトのすべてのファクトパターンを再評価するのではなく、指定されたファクトタイプの特定のパターン内で制約またはバインドされた変更済みプロパティーにのみデシジョンエンジンを反応させる場合は、このタグを使用できます。

例: DRL タイプの宣言におけるプロパティーリアクティビティーの有効化 (リアクティビティーがグローバルに無効にされる場合)

```
declare Person
  @propertyReactive
    firstName : String
    lastName : String
end
```

例: Java クラスでのプロパティーのリアクティビティーの有効化 (リアクティビティーがグローバルに無効にされる場合)

```
@propertyReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@watch

このタグは、DRL ルールのファクトパターンで、インラインで指定する追加のプロパティーに対するプロパティーリアクティビティーを有効にします。このタグがサポートされるのは、デシジョンエンジンでプロパティーリアクティビティーが **ALWAYS** に設定されている場合か、プロパティーリアクティビティーが **ALLOWED** に設定され、関連するファクトタイプが **@propertyReactive** タグを使用する場合に限られます。DRL ルールでこのタグを使用して、ファクトプロパティーリアクティビティー論理の指定されたプロパティーを追加または除外できます。
デフォルトパラメーター: なし

サポートされているパラメーター: プロパティー名、***** (all)、**!** (not)、**!*** (no properties)

```
<factPattern> @watch ( <property> )
```

例: ファクトパターンにおけるプロパティーリアクティビティーの有効化または無効化

```
// Listens for changes in both `firstName` (inferred) and `lastName`:
Person(firstName == $expectedFirstName) @watch( lastName )

// Listens for changes in all properties of the `Person` fact:
Person(firstName == $expectedFirstName) @watch( * )
```

```
// Listens for changes in `lastName` and explicitly excludes changes in `firstName`:
Person(firstName == $expectedFirstName) @watch( lastName, !firstName )

// Listens for changes in all properties of the `Person` fact except `age`:
Person(firstName == $expectedFirstName) @watch( *, !age )

// Excludes changes in all properties of the `Person` fact (equivalent to using `@classReactivity`
tag):
Person(firstName == $expectedFirstName) @watch( !* )
```

デシジョンエンジンは、**@classReactive** タグ (プロパティーリアクティビティーを無効にする) を使用するファクトタイプのプロパティーに対して **@watch** タグを使用する場合、またはデシジョンエンジンでプロパティーリアクティビティーが **ALLOWED** に設定され、関連するファクトタイプが **@propertyReactive** タグを使用しない場合は、コンパイルエラーを生成します。また、**@watch(firstName, ! firstName)** などのリスナーアノテーションでプロパティーを複製する場合でも、コンパイルエラーが生じます。

@propertyChangeSupport

JavaBeans Specification で定義されたプロパティー変更のサポートを実装するファクトの場合は、このタグによりデシジョンエンジンがファクトプロパティーの変更を監視できるようになります。

例: JavaBeans オブジェクトでのプロパティー変更のサポートの宣言

```
declare Person
    @propertyChangeSupport
end
```

85.6. イベントの時間オペレーター

ストリームモードでは、デシジョンエンジンは、デシジョンエンジンのワーキングメモリーに挿入されるイベントに対して以下の時間オペレーターをサポートします。これらのオペレーターを使用して、Java クラスまたは DRL ルールファイルで宣言するイベントの時間的な理由付け動作を定義できます。デシジョンエンジンがクラウドモードで実行されている場合、時間オペレーターはサポートされません。

- **after**
- **before**
- **coincides**
- **during**
- **includes**
- **finishes**
- **finished by**
- **meets**
- **met by**

- **overlaps**
- **overlapped by**
- **starts**
- **started by**

after

このオペレーターは、相関イベントの後に現在のイベントが発生するかどうかを指定します。また、このオペレーターは時間を定義でき、この時間の後に、現在のイベントは相関イベントを追跡することができます。または、現在のイベントが相関イベントを追跡できる区切られた時間範囲を定義することもできます。
たとえば、以下のパターンは、**$eventA** が **$eventB** の終了後 3 分 30 秒から 4 分の間に開始する場合に一致します。**$eventA** が **$eventB** の終了後 3 分 30 秒よりも前に開始する場合、または **$eventB** の終了後 4 分よりも後に開始する場合は、このパターンが一致しません。

```
$eventA : EventA(this after[3m30s, 4m] $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
3m30s <= $eventA.startTimestamp - $eventB.endTimeStamp <= 4m
```

after オペレーターは、パラメーター値を 2 つまでサポートします。

- 2 つの値が定義されると、間隔は 1 番目の値 (例では 3 分 30 秒) で開始し、2 番目の値 (例では 4 分) で終了します。
- 1 つの値のみ定義すると、間隔は提示した値で開始し、終了時間なしで無期限に実行されます。
- 値が定義されない場合、間隔は 1 ミリ秒から開始し、終了時間なしで無期限に実行されます。

after オペレーターは、負の時間範囲もサポートしています。

```
$eventA : EventA(this after[-3m30s, -2m] $eventB)
```

1 番目の値が 2 番目の値より大きい場合、デシジョンエンジンは順番を自動的に入れ替えます。たとえば、デシジョンエンジンは以下の 2 つのパターンを同じものと解釈します。

```
$eventA : EventA(this after[-3m30s, -2m] $eventB)
$eventA : EventA(this after[-2m, -3m30s] $eventB)
```

before

このオペレーターは、相関イベントの前に現在のイベントが発生するかどうかを指定します。このオペレーターは、現在のイベントが相関イベントに先行できる時間、または現在のイベントが相関イベントに先行できる区切り時間範囲を定義することもできます。
たとえば、以下のパターンは、**$eventA** が **$eventB** の開始前 3 分 30 秒から 4 分の間に終了する場合に一致します。**$eventA** が **$eventB** の開始前 3 分 30 秒よりも前に終了する場合、または **$eventB** の開始前 4 分よりも後に終了する場合は、パターンは一致しません。

```
$eventA : EventA(this before[3m30s, 4m] $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
3m30s <= $eventB.startTimestamp - $eventA.endTimeStamp <= 4m
```

before オペレーターは、パラメーター値を 2 つまでサポートします。

- 2 つの値が定義されると、間隔は 1 番目の値 (例では 3 分 30 秒) で開始し、2 番目の値 (例では 4 分) で終了します。
- 1 つの値のみ定義すると、間隔は提示した値で開始し、終了時間なしで無期限に実行されます。
- 値が定義されない場合、間隔は 1 ミリ秒から開始し、終了時間なしで無期限に実行されます。

before オペレーターは、負の時間範囲もサポートしています。

```
$eventA : EventA(this before[-3m30s, -2m] $eventB)
```

1 番目の値が 2 番目の値より大きい場合、デシジョンエンジンは順番を自動的に入れ替えます。たとえば、デシジョンエンジンは以下の 2 つのパターンを同じものと解釈します。

```
$eventA : EventA(this before[-3m30s, -2m] $eventB)
$eventA : EventA(this before[-2m, -3m30s] $eventB)
```

coincides

このオペレーターは、2 つのイベントが同じ開始時刻と終了時刻で同時に発生するかどうかを指定します。
たとえば、**$eventA** と **$eventB** の開始タイムスタンプと終了タイムスタンプの両方が同一の場合、以下のパターンは一致します。

```
$eventA : EventA(this coincides $eventB)
```

coincides オペレーターは、イベントの開始時間と終了時間の間隔が同じではない場合は、最大 2 つのパラメーター値をサポートします。

- パラメーターが 1 つだけ指定されている場合は、このパラメーターを使用して、両方のイベントの開始時間と終了時間のしきい値が設定されます。
- パラメーターが 2 つ指定されている場合、1 番目のパラメーターは開始時間のしきい値として使用され、2 番目のパラメーターは終了時間のしきい値として使用されます。

以下のパターンでは、開始時間と終了時間のしきい値を使用しています。

```
$eventA : EventA(this coincides[15s, 10s] $eventB)
```

以下の条件が一致する場合は、パターンが一致します。

```
abs($eventA.startTimestamp - $eventB.startTimestamp) <= 15s
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 10s
```

警告

デシジョンエンジンは、**coincides** オペレーターの負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

during

このオペレーターは、相関イベントが開始および終了する時間枠内で現在のイベントが発生するかどうかを指定します。現在のイベントは、相関イベントの開始後に開始し、相関イベントの終了前に終了する必要があります。(**coincides** オペレーターを使用すると、開始時間と終了時間は同じか、ほぼ同じになります)。
たとえば、以下のパターンは、**$eventA** が **$eventB** の開始後に開始し、**$eventB** の終了前に終了する場合に一致します。

```
$eventA : EventA(this during $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventB.startTimestamp < $eventA.startTimestamp <= $eventA.endTimestamp <
$eventB.endTimestamp
```

during オペレーターは、1、2、または 4 つの任意のパラメーターをサポートします。

- 1 つの値が定義されている場合、この値は 2 つのイベントのそれぞれの開始時間の間隔が最大であるほか、2 つのイベントのそれぞれの終了時間の間隔が最大であることを示しています。
- 2 つの値が定義されている場合、これらの値はしきい値で、これらのしきい値の間では、現在のイベントの開始時間と終了時間が、相関イベントの開始時間と終了時間に関連して発生する必要があります。
  たとえば、値が **5s** と **10s** である場合、現在のイベントは相関イベントの開始後 5 秒から 10 秒の間に開始し、相関イベント終了の 5 秒から 10 秒前に終了する必要があります。
- 4 つの値が定義されている場合、1 番目と 2 番目の値は、各イベントの開始時間の最小間隔と最大間隔を表しています。また、3 番目と 4 番目の値は、2 つのイベントの終了時間の最小間隔と最大間隔を表しています。

includes

このオペレーターは、相関イベントが、現在のイベントが発生する時間枠内で発生するかどうかを指定します。相関イベントは、現在のイベントの開始後に開始し、現在のイベントの終了前に終了する必要があります。(このオペレーターの動作は、**during** オペレーターと逆の動作になります)。
たとえば、以下のパターンは、**$eventB** が **$eventA** の開始後に開始し、**$eventA** の終了前に終了する場合に一致します。

```
$eventA : EventA(this includes $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventA.startTimestamp < $eventB.startTimestamp <= $eventB.endTimestamp <
$eventA.endTimestamp
```

includes オペレーターは、1 つ、2 つ、または 4 つのオプションのパラメーターをサポートします。

- 1 つの値が定義されている場合、この値は 2 つのイベントのそれぞれの開始時間の間隔が最大であるほか、2 つのイベントのそれぞれの終了時間の間隔が最大であることを示しています。
- 2 つの値が定義されている場合、これらの値はしきい値で、これらのしきい値の間では、相関イベントの開始時間と終了時間が、現在のイベントの開始時間と終了時間に関連して発生する必要があります。
  たとえば、値が **5s** と **10s** である場合、相関イベントは現在のイベントの開始後 5 秒から 10 秒の間に開始し、現在のイベント終了の 5 秒から 10 秒前に終了する必要があります。
- 4 つの値が定義されている場合、1 番目と 2 番目の値は、各イベントの開始時間の最小間隔と最大間隔を表しています。また、3 番目と 4 番目の値は、2 つのイベントの終了時間の最小間隔と最大間隔を表しています。

finishes

このオペレーターは、現在のイベントが相関イベントの後に開始して、両方のイベントが同時に終了するかどうかを指定します。
たとえば、以下のパターンは、**$eventA** が **$eventB** の開始後に開始し、**$eventB** と同時に終了する場合に一致します。

```
$eventA : EventA(this finishes $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventB.startTimestamp < $eventA.startTimestamp
&&
$eventA.endTimestamp == $eventB.endTimestamp
```

finishes オペレーターは、2 つのイベントのそれぞれの終了時間の間隔に最大許容時間を設定する 1 つのオプションパラメーターをサポートします。

```
$eventA : EventA(this finishes[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
$eventB.startTimestamp < $eventA.startTimestamp
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 5s
```

警告

デシジョンエンジンは、**finishes** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

finished by

このオペレーターは、相関イベントが現在のイベントの後に開始して、両方のイベントが同時に終了するかどうかを指定します。(このオペレーターの動作は、**finishes** オペレーターと逆の動作になります)。
たとえば、以下のパターンは、**$eventB** が **$eventA** の開始後に開始し、**$eventA** と同時に終了する場合に一致します。

```
$eventA : EventA(this finishedby $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventA.startTimestamp < $eventB.startTimestamp
&&
$eventA.endTimestamp == $eventB.endTimestamp
```

finished by オペレーターは、2 つのイベントのそれぞれの終了時間の間隔に最大許容時間を設定する 1 つの任意のパラメーターをサポートします。

```
$eventA : EventA(this finishedby[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
$eventA.startTimestamp < $eventB.startTimestamp
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 5s
```

警告

デシジョンエンジンは、**finished by** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

meets

このオペレーターは、現在のイベントが相関イベントの開始と同時に終了するかどうかを指定します。
たとえば、以下のパターンは、**$eventA** が **$eventB** の開始と同時に終了する場合に一致します。

```
$eventA : EventA(this meets $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
abs($eventB.startTimestamp - $eventA.endTimestamp) == 0
```

meets オペレーターは、現在のイベントの終了時間と相関イベントの開始時間との間隔に最大許容時間を設定する 1 つの任意のパラメーターをサポートします。

```
$eventA : EventA(this meets[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
abs($eventB.startTimestamp - $eventA.endTimestamp) <= 5s
```

警告

デシジョンエンジンは、**meets** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

met by

このオペレーターは、相関イベントが現在のイベントの開始と同時に終了するかどうかを指定します。(このオペレーターの動作は、**meets** オペレーターと逆の動作になります)。たとえば、以下のパターンは、**$eventB** が **$eventA** の開始と同時に終了する場合に一致します。

```
$eventA : EventA(this metby $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
abs($eventA.startTimestamp - $eventB.endTimestamp) == 0
```

met by オペレーターは、相関イベントの終了時間と現在のイベントの開始時間との間に最大距離を設定する 1 つの任意のパラメーターをサポートします。

```
$eventA : EventA(this metby[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
abs($eventA.startTimestamp - $eventB.endTimestamp) <= 5s
```

警告

デシジョンエンジンは、**met by** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

overlaps

このオペレーターは、現在のイベントが相関イベントの開始前に開始し、相関イベントが発生する時間枠内で終了するかどうかを指定します。現在のイベントは、相関イベントの開始時間と終了時間の間に終了する必要があります。
たとえば、以下のパターンは、**$eventA** が **$eventB** の開始前に開始し、**$eventB** の終了前に **$eventB** が発生する間に終了する場合に一致します。

```
$eventA : EventA(this overlaps $eventB)
```

overlaps オペレーターは、パラメーター値を 2 つまでサポートします。

- 1 つのパラメーターが定義されている場合、値は相関イベントの開始時間と現在のイベントの終了時間との間の最大間隔になります。
- 2 つのパラメーターが定義されている場合、値は相関イベントの開始時間と現在のイベントの終了時間との間の最短間隔 (1 番目の値) と最大間隔 (2 番目の値) になります。

overlapped by

このオペレーターは、相関イベントが、現在のイベントの開始前に開始し、現在のイベントが発生する時間枠内で終了するかどうかを指定します。相関イベントは、現在のイベントの開始時間と終了時間の間に終了する必要があります。(このオペレーターの動作は、**overlaps** オペレーターと逆の動作になります)。
たとえば、以下のパターンは、**$eventB** が **$eventA** の開始前に開始し、**$eventA** の終了前に **$eventA** が発生する前に終了する場合に一致します。

```
$eventA : EventA(this overlappedby $eventB)
```

overlapped by オペレーターは、パラメーター値を 2 つまでサポートします。

- 1 つのパラメーターが定義されている場合、値は現在のイベントの開始時間と相関イベントの終了時間との間の最大間隔になります。
- 2 つのパラメーターが定義されている場合、値は現在のイベントの開始時間と相関イベントの終了時間との間の最短間隔 (1 番目の値) と最大間隔 (2 番目の値) になります。

starts

このオペレーターは、2 つのイベントが同時に開始するが、現在のイベントが相関イベントの終了前に終了するかどうかを指定します。
たとえば、以下のパターンは、**$eventA** と **$eventB** が同時に開始し、**$eventA** が **$eventB** の終了前に終了する場合に一致します。

```
$eventA : EventA(this starts $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventA.startTimestamp == $eventB.startTimestamp
&&
$eventA.endTimestamp < $eventB.endTimestamp
```

starts オペレーターは、2 つのイベントのそれぞれの開始時間の間の最大間隔を設定する 1 つの任意のパラメーターをサポートします。

```
$eventA : EventA(this starts[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
abs($eventA.startTimestamp - $eventB.startTimestamp) <= 5s
&&
$eventA.endTimestamp < $eventB.endTimestamp
```

警告

デシジョンエンジンは、**starts** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

started by

このオペレーターは、2 つのイベントが同時に開始し、現在のイベントの終了前に相関イベントが終了するかどうかを指定します。(このオペレーターの動作は、**starts** オペレーターと逆の動作になります)。
たとえば、以下のパターンは、**$eventA** と **$eventB** が同時に開始し、**$eventB** が **$eventA** の終了前に終了する場合に一致します。

```
$eventA : EventA(this startedby $eventB)
```

以下の方法で、このオペレーターを表すこともできます。

```
$eventA.startTimestamp == $eventB.startTimestamp
&&
$eventA.endTimestamp > $eventB.endTimestamp
```

started by オペレーターは、2 つのイベントのそれぞれの開始時間の間の最大間隔を設定する 1 つの任意のパラメーターをサポートします。

```
$eventA : EventA( this starts[5s] $eventB)
```

これらの条件が一致する場合は、パターンが一致します。

```
abs( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5s
&&
$eventA.endTimestamp > $eventB.endTimestamp
```

警告

デシジョンエンジンは、**started by** オペレーターに対して負の間隔をサポートしていません。負の間隔を使用すると、デシジョンエンジンはエラーを生成します。

85.7. デシジョンエンジンにおけるセッションクロックの実装

複雑なイベントの処理中、デシジョンエンジンのイベントには時間的な制約があるかもしれないため、現在の時刻を提供するセッションクロックが必要になります。たとえば、ルールが過去 60 分間の指定の株式の平均価格を決定する必要がある場合、デシジョンエンジンは株価イベントのタイムスタンプとセッションクロックの現在の時刻と比較できなければなりません。

デシジョンエンジンは、リアルタイムクロックと擬似クロックをサポートしています。シナリオに応じて、1 つまたは両方のクロックタイプを使用できます。

- **ルールのテスト:** テストには管理された環境が必要です。テストに時間的制約を持つルールが含まれる場合は、入力ルール、ファクト、および時間のフローを制御できる必要があります。
- **通常の実行:** デシジョンエンジンは、リアルタイムでイベントに反応するため、リアルタイムクロックが必要です。
- **特別な環境:** 特定の環境には、特定の時間制御要件がある場合があります。たとえば、クラスター環境ではクロックの同期が必要な場合があり、Java Enterprise Edition (JEE) 環境ではアプリケーションサーバーが提供するクロックが必要な場合があります。
- **ルールの再生またはシミュレーション:** シナリオを再生またはシミュレートするには、アプリケーションが時間のフローを制御できるようにする必要があります。

デシジョンエンジンで、リアルタイムクロックを使用するか、または擬似クロックを使用するかを決定する際には、環境要件を考慮してください。

リアルタイムクロック

リアルタイムクロックは、デシジョンエンジンのデフォルトのクロック実装であり、システムクロックを使用してタイムスタンプの現在の時刻を決定します。リアルタイムクロックを使用するようにデシジョンエンジンを設定するには、KIE セッション設定パラメーターを **realtime** に設定します。

KIE セッションでのリアルタイムクロックの設定

```
import org.kie.api.KieServices.Factory;
import org.kie.api.runtime.conf.ClockTypeOption;
import org.kie.api.runtime.KieSessionConfiguration;
```

```
KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption(ClockTypeOption.get("realtime"));
```

擬似クロック

デシジョンエンジンでの擬似クロックの実装は、時間的なルールのテストに役立ち、アプリケーションによって制御できます。疑似クロックを使用するようにデシジョンエンジンを設定するには、KIE セッション設定パラメーターを **pseudo** に設定します。

KIE セッションでの擬似クロックの設定

```
import org.kie.api.runtime.conf.ClockTypeOption;
import org.kie.api.runtime.KieSessionConfiguration;
import org.kie.api.KieServices.Factory;

KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();

config.setOption(ClockTypeOption.get("pseudo"));
```

追加の設定とファクトハンドラーを使用して、擬似クロックを制御することもできます。

KIE セッションで擬似クロックの動作を制御

```
import java.util.concurrent.TimeUnit;

import org.kie.api.runtime.KieSessionConfiguration;
import org.kie.api.KieServices.Factory;
import org.kie.api.runtime.KieSession;
import org.drools.core.time.SessionPseudoClock;
import org.kie.api.runtime.rule.FactHandle;
import org.kie.api.runtime.conf.ClockTypeOption;

KieSessionConfiguration conf = KieServices.Factory.get().newKieSessionConfiguration();

conf.setOption( ClockTypeOption.get("pseudo"));
KieSession session = kbase.newKieSession(conf, null);

SessionPseudoClock clock = session.getSessionClock();

// While inserting facts, advance the clock as necessary.
FactHandle handle1 = session.insert(tick1);
clock.advanceTime(10, TimeUnit.SECONDS);

FactHandle handle2 = session.insert(tick2);
clock.advanceTime(30, TimeUnit.SECONDS);

FactHandle handle3 = session.insert(tick3);
```

85.8. イベントストリームとエントリーポイント

デシジョンエンジンは、大量のイベントをイベントストリームの形式で処理できます。DRL ルール宣言では、ストリームは **エントリーポイント** としても知られています。DRL ルールまたは Java アプリ

ケーションでエントリーポイントを宣言すると、デシジョンエンジンはコンパイル時に適切な内部構造を特定して作成し、そのエントリーポイントのみからのデータを使用してそのルールを評価します。

1 つのエントリーポイント、またはストリームからのファクトは、デシジョンエンジンのワーキングメモリーにすでにあるファクトに加えて、他のエントリーポイントからのファクトに参加できます。ファクトは、常にデシジョンエンジンに入ったエントリーポイントに関連付けられたままとなっています。同じタイプのファクトは、複数のエントリーポイントからデシジョンエンジンに入ることができますが、エントリーポイント A からデシジョンエンジンに入るファクトは、エントリーポイント B からのパターンと一致することはありません。

イベントストリームには、以下の特徴があります。

- ストリーム内のイベントは、タイムスタンプ別に並べられます。タイムスタンプは、ストリームごとにさまざまなセマンティクスを持つ場合もありますが、常に内部で並べ替えが行われます。
- 通常、イベントストリームには大量のイベントがあります。
- 通常、ストリームに含まれるアトミックなイベントは、それだけでは実用的ではなく、ストリームで集合的な場合にのみ実用的です。
- イベントストリームは、同種 (単一タイプのイベントを含む) の場合も、異種 (異るタイプのイベントを含む) の場合もあります。

85.8.1. ルールデータのエントリーポイントの宣言

イベントのエントリーポイント (イベントストリーム) を宣言して、デシジョンエンジンがそのエントリーポイントのみからのデータを使用してルールを評価することが可能です。エントリーポイントは、DRL ルールで参照することで暗黙的に宣言することも、Java アプリケーションで明示的に宣言することもできます。

手順

以下のいずれかの方法を使用して、エントリーポイントを宣言します。

- DRL ルールファイルで、挿入されたファクトの **from entry-point "<name>"** を指定します。

  "ATM Stream" エントリーポイントで取り消しルールを認可

  ```
  rule "Authorize withdrawal"
  when
    WithdrawRequest($ai : accountId, $am : amount) from entry-point "ATM Stream"
    CheckingAccount(accountId == $ai, balance > $am)
  then
    // Authorize withdrawal.
  end
  ```

  "Branch Stream" エントリーポイントで手数料ルールを適用

  ```
  rule "Apply fee on withdraws on branches"
  when
    WithdrawRequest($ai : accountId, processed == true) from entry-point "Branch Stream"
    CheckingAccount(accountId == $ai)
  then
    // Apply a $2 fee on the account.
  end
  ```

–

銀行取引アプリケーションにおける両方の DRL ルールのサンプルは、イベント **WithdrawalRequest** にファクト **CheckingAccount** を挿入しますが、エントリーポイントは異なります。実行時にデシジョンエンジンは、**"ATM Stream"** エントリーポイントのみからのデータを使用して **Authorize withdrawal** ルールを評価し、**"Branch Stream"** エントリーポイントのみからのデータを使用して **Apply fee** ルールを評価します。**"ATM Stream"** に挿入されたイベントは、**"Apply fee"** ルールのパターンと一致することはありません。また、**"Branch Stream"** に挿入されたイベントは、**"Authorize withdrawal rule"** のパターンと一致することはありません。

- Java アプリケーションコードで、**getEntryPoint()** メソッドを使用して **EntryPoint** オブジェクトを指定および取得し、以下に従ってファクトをそのエントリーポイントに挿入します。

EntryPoint オブジェクトと挿入されたファクトを持つ Java アプリケーションコード

```
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.EntryPoint;

// Create your KIE base and KIE session as usual.
KieSession session = ...

// Create a reference to the entry point.
EntryPoint atmStream = session.getEntryPoint("ATM Stream");

// Start inserting your facts into the entry point.
atmStream.insert(aWithdrawRequest);
```

from entry-point "ATM Stream" を指定する DRL ルールはすべて、このエントリーポイントのデータのみに基づいて評価されます。

85.9. 時間または長さのスライディングウィンドウ

ストリームモードでは、デシジョンエンジンは指定された時間または長さのスライディングウィンドウからのイベントを処理できます。スライディングタイムウィンドウは、イベントを処理できる特定の期間です。スライディングレングスウィンドウは、処理可能な指定されたイベントの数です。DRL ルールまたは Java アプリケーションでスライディングウィンドウを宣言すると、デシジョンエンジンは、コンパイル時に適切な内部構造を特定して作成し、そのスライディングウィンドウのみからのデータを使用してそのルールを評価します。

たとえば、以下の DRL ルールスニペットは、最後の 2 分間のストックポイントのみを処理する (スライディングタイムウィンドウ) か、最後の 10 のストックポイントのみを処理する (スライディングレングスウィンドウ) かをデシジョンエンジンに指示します。

過去 2 分間のストックポイントを処理 (スライディングタイムウィンドウ)

```
StockPoint() over window:time(2m)
```

最後の 10 のストックポイントを処理 (スライディングレングスウィンドウ)

```
StockPoint() over window:length(10)
```

85.9.1. ルールデータのスライディングタイムウィンドウを宣言

イベントの時間 (時間のフロー) または長さ (発生回数) のスライディングウィンドウを宣言して、デシジョンエンジンがそのウィンドウのみのデータを使用してルールを評価することが可能です。

手順

DRL ルールファイルで、挿入されたファクトに **over window:<time_or_length>(<value>)** を指定します。

たとえば、以下の 2 つの DRL ルールは、平均温度に基づいて火災報知器を有効にします。ただし、1 番目のルールはスライディングタイムウィンドウを使用して最後の 10 分間の平均を計算し、2 番目のルールはスライディングレングスウィンドウを使用して最後の 100 の温度測定値の平均を計算します。

スライディングタイムウィンドウにおける平均温度

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:time(10m),
    average($temp))
then
  // Sound the alarm.
end
```

スライディングレングスウィンドウにおける平均温度

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:length(100),
    average($temp))
then
  // Sound the alarm.
end
```

デシジョンエンジンは、**SensorReading** イベントのうち、10 分を経過したもの、または最後の 100 の読み取り値以外のものをすべて破棄し、時間または読み取り値がリアルタイムで先へとスライドしていくなか、平均を再計算し続けます。

デシジョンエンジンは、KIE セッションから古いイベントを自動的に削除しません。これは、スライディングウィンドウの宣言がない他のルールが、古いイベントに依存する可能性があるからです。デシジョンエンジンは、明示的なルール宣言または KIE ベースの推論データに基づくデシジョンエンジン内の暗黙的な理由付けのいずれかによって、イベントの有効期限が切れるまで、KIE セッションにイベントを保存します。

85.10. イベントのメモリー管理

ストリームモードでは、デシジョンエンジンは自動メモリー管理を使用して、KIE セッションに保存されているイベントを維持します。デシジョンエンジンは、時間制約が原因でルールと一致しなくなったイベントを KIE セッションから取り除き、取り除かれたイベントが使っていたリソースを解放します。

デシジョンエンジンは、明示的な有効期限または推論された有効期限のいずれかを使用して、古いイベントを取り消します。

- **明示的な有効期限:** デシジョンエンジンは、**@expires** タグを宣言するルールで明示的に有効期限が切れるよう設定されているイベントを削除します。

  有効期限が明示的な DRL ルールスニペット

  ```
  declare StockPoint
    @expires( 30m )
  end
  ```

  この例のルールは、**StockPoint** イベントが 30 分後に有効期限となるよう設定し、他のルールがイベントを使用しない場合は KIE セッションから削除するよう設定します。

- **推論された有効期限:** デシジョンエンジンは、ルールの時間制約を解析することで、特定のイベントに関する有効期限の補正値を暗黙的に算出できます。

  時間制約のある DRL ルール

  ```
  rule "Correlate orders"
  when
    $bo : BuyOrder($id : id)
    $ae : AckOrder(id == $id, this after[0,10s] $bo)
  then
    // Perform an action.
  end
  ```

  この例のルールでは、デシジョンエンジンは、**BuyOrder** イベントが発生するたびに、デシジョンエンジンがイベントを最大 10 秒間保存し、一致する **AckOrder** イベントを待つ必要があることを自動的に計算します。10 秒後に、デシジョンエンジンは有効期限を推論し、KIE セッションからイベントを削除します。**AckOrder** イベントは既存の **BuyOrder** イベントとのみ一致するため、一致が発生しない場合はデシジョンエンジンが有効期限を推論し、イベントをすぐに削除します。

  デシジョンエンジンは、KIE ベース全体を分析して、すべてのイベントタイプの補正値を見つけ、他のルールが削除を保留しているイベントを使用しないようにします。暗黙的な有効期限が明示的な有効期限の値と衝突するたびに、デシジョンエンジンはこの 2 つのうちの大きい方の時間枠を使用して、イベントをより長く保存します。

第86章 デシジョンエンジンクエリーおよびライブクエリー

デシジョンエンジンでクエリーを使用して、ルールで使用されるファクトパターンに基づいてファクトセットを取得できます。また、パターンはオプションのパラメーターを使用することもできます。

デシジョンエンジンでクエリーを使用するには、DRL ファイルにクエリー定義を追加し、アプリケーションコードで一致する結果を取得します。クエリーは結果コレクション上で反復しますが、クエリーにバインドされている識別子を使用して、バインディング変数名を引数として使用する **get()** メソッドを呼び出すことで、対応するファクトまたはファクトフィールドにアクセスできます。バインディングがファクトオブジェクトを参照する場合は、変数名をパラメーターとして使用する **getFactHandle()** を呼び出すことで、ファクトハンドルを取得できます。

DRL ファイルにおけるクエリー定義の例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

クエリー結果を取得および反復するアプリケーションのコード例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age of 21" );

System.out.println( "These people are under the age of 21:" );

for ( QueryResultsRow row : results ) {
    Person person = ( Person ) row.get( "person" );
    System.out.println( person.getName() + "\n" );
}
```

経時的な変化をモニターリングする場合は、クエリーを呼び出して返された値のセットを反復して結果を処理することが困難な場合があります。進行中のクエリーでこの困難な作業を軽減するために、Red Hat Process Automation Manager は **ライブクエリー** を提供しています。ライブクエリーは、反復可能な結果セットを返す代わりに、変更イベントにアタッチされたリスナーを使用します。ライブクエリーは、ビューを作成し、このビューのコンテンツ向けに変更イベントを公開することで、オープンの状態を保ちます。

ライブクエリーをアクティブ化するには、パラメーターを使用してクエリーを開始し、結果ビューの変更を監視します。**dispose()** メソッドを使用してクエリーを終了し、このリアクティブシナリオを中断できます。

DRL ファイルにおけるクエリー定義の例

```
query colors(String $color1, String $color2)
    TShirt(mainColor = $color1, secondColor = $color2, $price: manufactureCost)
end
```

イベントリスナーとライブクエリーを使用したアプリケーションのコード例

```
final List updated = new ArrayList();
final List removed = new ArrayList();
final List added = new ArrayList();

ViewChangedEventListener listener = new ViewChangedEventListener() {
```

```
 public void rowUpdated(Row row) {
  updated.add( row.get( "$price" ) );
 }

 public void rowRemoved(Row row) {
  removed.add( row.get( "$price" ) );
 }

 public void rowAdded(Row row) {
  added.add( row.get( "$price" ) );
 }
};

// Open the live query:
LiveQuery query = ksession.openLiveQuery( "colors",
                                          new Object[] { "red", "blue" },
                                          listener );
...
...

// Terminate the live query:
query.dispose()
```

第87章 デシジョンエンジンのイベントリスナーおよびデバッグロギング

Red Hat Process Automation Manager では、ファクトの挿入やルールの実行など、デシジョンエンジンのイベントリスナーを追加または削除できます。デシジョンエンジンのイベントリスナーを使用すると、デシジョンエンジンのアクティビティーの通知を受けることができ、ロギングと監査をアプリケーションのコアから分けることができます。

デシジョンエンジンは、アジェンダおよびワーキングメモリーに対して、以下のデフォルトのイベントリスナーをサポートします。

- **AgendaEventListener**
- **WorkingMemoryEventListener**

各イベントリスナーについて、デシジョンエンジンは、監視するように指定できる以下の特定のイベントもサポートします。

- **MatchCreatedEvent**
- **MatchCancelledEvent**
- **BeforeMatchFiredEvent**
- **AfterMatchFiredEvent**
- **AgendaGroupPushedEvent**
- **AgendaGroupPoppedEvent**
- **ObjectInsertEvent**
- **ObjectDeletedEvent**
- **ObjectUpdatedEvent**
- **ProcessCompletedEvent**
- **ProcessNodeLeftEvent**
- **ProcessNodeTriggeredEvent**
- **ProcessStartEvent**

たとえば、以下のコードは、KIE セッションにアタッチされた **DefaultAgendaEventListener** リスナーを使用して、**AfterMatchFiredEvent** イベントが監視されるように指定します。コードは、ルールの実行後にパターンの一致を出力します。

アジェンダの AfterMatchFiredEvent イベントを監視および出力するコード例

```
ksession.addEventListener( new DefaultAgendaEventListener() {
  public void afterMatchFired(AfterMatchFiredEvent event) {
     super.afterMatchFired( event );
     System.out.println( event );
  }
});
```

デシジョンエンジンは、デバッグロギングに対して以下のアジェンダおよびワーキングメモリーイベントリスナーもサポートします。

- **DebugAgendaEventListener**
- **DebugRuleRuntimeEventListener**

これらのイベントリスナーは、同様のサポートされているイベントリスナーメソッドを実装し、デフォルトでデバッグ出力ステートメントを含んでいます。特定のサポートされるイベントを追加して監視およびドキュメント化するか、すべてのアジェンダまたはワーキングメモリーアクティビティーを監視することができます。

たとえば、以下のコードは **DebugRuleRuntimeEventListener** イベントリスナーを使用して、すべてのワーキングメモリーイベントを監視および出力します。

すべてのワーキングメモリーイベントを監視および出力するコード例

```
ksession.addEventListener( new DebugRuleRuntimeEventListener() );
```

87.1. デシジョンエンジンでのロギングユーティリティーの設定

デシジョンエンジンは、システムロギングに Java ロギング API SLF4J を使用します。以下のロギングユーティリティーのいずれかをデシジョンエンジンで使用して、トラブルシューティングまたはデータ収集などのデシジョンエンジンのアクティビティーを調査できます。

- Logback
- Apache Commons Logging
- Apache Log4j
- **java.util.logging** パッケージ

手順

使用するロギングユーティリティーについては、Maven プロジェクトに関連する依存関係を追加するか、Red Hat Process Automation Manager ディストリビューションの **org.drools** パッケージに関連する XML 設定ファイルを保存します。

Logback の Maven 依存関係の例

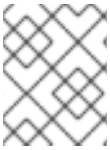
```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

org.drools パッケージにおける logback.xml 設定ファイルの例

```
<configuration>
  <logger name="org.drools" level="debug"/>
  ...
<configuration>
```

org.drools パッケージにおける log4j.xml 設定ファイルの例

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <category name="org.drools">
    <priority value="debug" />
  </category>
  ...
</log4j:configuration>
```



注記

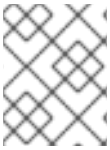
超軽量環境向けに開発している場合は **slf4j-nop** または **slf4j-simple** ロガーを使用します。

第88章 RED HAT PROCESS AUTOMATION MANAGER の IDE 向けのデシジョン例

Red Hat Process Automation Manager は、統合開発環境 (IDE: integrated development environment) にインポートできるように Java クラスとして配信されるデシジョン例を提供します。これらの例は、デシジョンエンジン機能をさらに理解するために使用する目的か、Red Hat Process Automation Manager プロジェクトに定義するデシジョンの参考として使用してください。

以下のデシジョンセットの例は、Red Hat Process Automation Manager で利用可能な例の一部です。

- **Hello World の例**: 基本的なルール実行や、デバッグ出力の使用方法を例示します。
- **状態の例**: ルールの顕著性やアジェンダグループを使用した前向き連鎖や競合解決を例示します。
- **フィボナッチの例**: ルールの顕著性を使用した再帰や競合解決を例示します。
- **銀行の例**: パターン一致、基本的なソート、計算を例示します。
- **ペットショップの例**: ルールアジェンダグループ、グローバル変数、コールバック、GUI 統合を例示します。
- **数独の例**: 複雑なパターン一致、問題解決、コールバック、GUI 統合を例示します。
- **House of Doom の例**: 後向き連鎖と再帰を例示します。



注記

Red Hat ビルドの OptaPlanner で提供される最適化の例は、**Red Hat ビルドの OptaPlanner のスタートガイド** を参照してください。

88.1. IDE での RED HAT PROCESS AUTOMATION MANAGER のデシジョン例のインポートおよび実行

Red Hat Process Automation Manager のデシジョン例を統合開発環境 (IDE) にインポートして実行し、ルールとコードがどのように機能するかをチェックできます。これらの例は、デシジョンエンジン機能をさらに理解するために使用する目的か、Red Hat Process Automation Manager プロジェクトに定義するデシジョンの参考として使用してください。

前提条件

- Java 8 以降がインストールされている。
- Maven 3.5.x 以降がインストールされている。
- Red Hat CodeReady Studio などの IDE がインストールされている。

手順

1. Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.11.0 Source Distribution** をダウンロードし、**/rhpam-7.11.0-sources** などの一時的なディレクトリーに展開します。

2. IDE を開き、**File → Import → Maven → Existing Maven Projects**を選択するか、同等のオプションを選択して、Maven プロジェクトをインポートします。

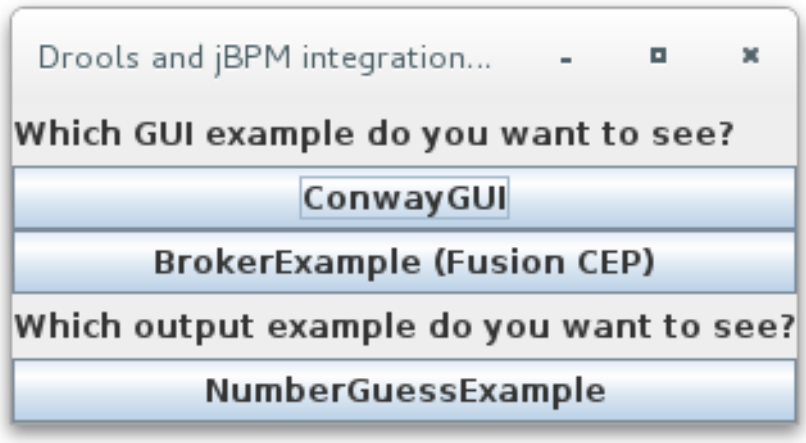
3. **Browse** をクリックして、**~/rhpam-7.11.0-sources/src/drools-$VERSION/drools-examples** (または、Conway の Game of Life の例の場合は、**~/rhpam-7.11.0-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples**) に移動して、プロジェクトをインポートします。

4. 実行するパッケージ例に移動して、**main** メソッドが含まれる Java クラスを検索します。

5. Java クラスを右クリックし、**Run As → Java Application** を選択して例を実行します。
基本的なユーザーインターフェイスですべての例を実行するには、Main クラス **org.drools.examples** の **DroolsExamplesApp.java** クラス (または Conway の Game of Life の場合は **DroolsJbpmIntegrationExamplesApp.java** クラス) を実行します。

図88.1 drools-examples (DroolsExamplesApp.java) 内のすべての例のインターフェイス

図88.2 droolsjbpm-integration-examples (DroolsJbpmIntegrationExamplesApp.java) のすべての例のインターフェイス



88.2. HELLO WORLD の例のデシジョン (基本ルールおよびデバッグ)

Hello World のデシジョンセットの例では、オブジェクトをデシジョンエンジンのワーキングメモリーに挿入する方法、ルールを使用してオブジェクトを一致させる方法、エンジンの内部アクティビティーを追跡するロギングの設定方法を説明します。

以下は、Hello World の例の概要です。

- **名前**: **helloworld**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.helloworld.HelloWorldExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.helloworld.HelloWorld.drl**
- **目的**: 基本的なルール実行とデバッグ出力の使用方法を例示します。

Hello World の例では、KIE セッションが生成されて、ルールの実行が可能になります。すべてのルールは、実行するのに KIE セッションが必要です。

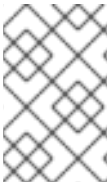
ルール実行の KIE セッション

```
KieServices ks = KieServices.Factory.get(); (1)
KieContainer kc = ks.getKieClasspathContainer(); (2)
KieSession ksession = kc.newKieSession("HelloWorldKS"); (3)
```

(1) **KieServices** ファクトリーを取得します。これは、アプリケーションがデシジョンエンジンとの対話に使用する主なインターフェイスです。

(2) プロジェクトクラスパスから **KieContainer** を作成します。これは、**KieModule** で **KieContainer** を設定してインスタンス化し、そこから **/META-INF/kmodule.xml** ファイルを検出します。

(3) **/META-INF/kmodule.xml** ファイルに定義された KIE セッション設定 **"HelloWorldKS"** をもとに **KieSession** を作成します。



注記

Red Hat Process Automation Manager プロジェクトのパッケージ化に関する詳細は、**Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ** を参照してください。

Red Hat Process Automation Manager には、内部エンジンアクティビティーを公開するイベントモデルがあります。デフォルトのデバッグリスナー **DebugAgendaEventListener** と **DebugRuleRuntimeEventListener** により、デバッグイベント情報が **System.err** の出力に表示されます。**KieRuntimeLogger** では、実行監査と、グラフィックビューワーで確認可能な結果が提供されます。

リスナーと監査ロガーのデバッグ

```
// Set up listeners.
ksession.addEventListener( new DebugAgendaEventListener() );
ksession.addEventListener( new DebugRuleRuntimeEventListener() );

// Set up a file-based audit logger.
KieRuntimeLogger logger = KieServices.get().getLoggers().newFileLogger( ksession,
"./target/helloworld" );

// Set up a ThreadedFileLogger so that the audit view reflects events while debugging.
KieRuntimeLogger logger = ks.getLoggers().newThreadedFileLogger( ksession, "./target/helloworld",
1000 );
```

ロガーは、**Agenda** と **RuleRuntime** リスナーにビルドされる特別な実装です。デシジョンエンジンが実行を終えると、**logger.close()** が呼び出されます。

この例では、**"Hello World"** というメッセージを含む **Message** オブジェクトを作成し、ステータス **HELLO** を **KieSession** に挿入して、**fireAllRules()** でルールを実行します。

データの挿入および実行

```
// Insert facts into the KIE session.
final Message message = new Message();
message.setMessage( "Hello World" );
message.setStatus( Message.HELLO );
ksession.insert( message );

// Fire the rules.
ksession.fireAllRules();
```

ルール実行は、データモデルを使用して、**KieSession** への出入力としてデータを渡します。この例のデータモデルには **message** (**String**) と **status** (**HELLO** または **GOODBYE**) の 2 つのフィールドが含まれます。

データモデルクラス

```
public static class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String         message;
```

```
    private int           status;
    ...
}
```

この 2 つのルールは、**src/main/resources/org/drools/examples/helloworld/HelloWorld.drl** ファイルに配置されます。

"Hello World" ルールの **when** 条件では、ステータスが **Message.HELLO** の KIE セッションに、**Message** オブジェクトが挿入されるたびに、このルールをアクティベートすると記述しています。さらに、変数のバインドが 2 つ作成されます (**message** 変数を **message** 属性に、**m** 変数を一致する **Message** オブジェクト自体にバインド)。

ルールの **then** アクションは、バインドされた変数 **message** のコンテンツを **System.out** に出力するよう指定し、続いて **m** にバインドされている **Message** オブジェクトの **message** と **status** 属性値を変更します。このルールは **modify** ステートメントを使用して、1 つのステートメントに割り当てブロックを適用し、ブロックの最後にデシジョンエンジンにこの変更について通知します。

"Hello World" のルール

```
rule "Hello World"
  when
    m : Message( status == Message.HELLO, message : message )
  then
    System.out.println( message );
    modify ( m ) { message = "Goodbye cruel world",
                   status = Message.GOODBYE };
end
```

"Good Bye" ルールは、ステータスが **Message.GOODBYE** の **Message** オブジェクトと一致する点を除き、**"Hello World"** ルールによく似ています。

"Good Bye" ルール

```
rule "Good Bye"
  when
    Message( status == Message.GOODBYE, message : message )
  then
    System.out.println( message );
end
```

この例を実行するには、**org.drools.examples.helloworld.HelloWorldExample** クラスを IDE で Java アプリケーションとして実行します。このルールは **System.out** に、デバッグリスナーは **System.err** に書き込み、監査ロガーは **target/helloworld.log** のログファイルを作成します。

IDE コンソールの System.out 出力

```
Hello World
Goodbye cruel world
```

IDE コンソールでの System.err の出力

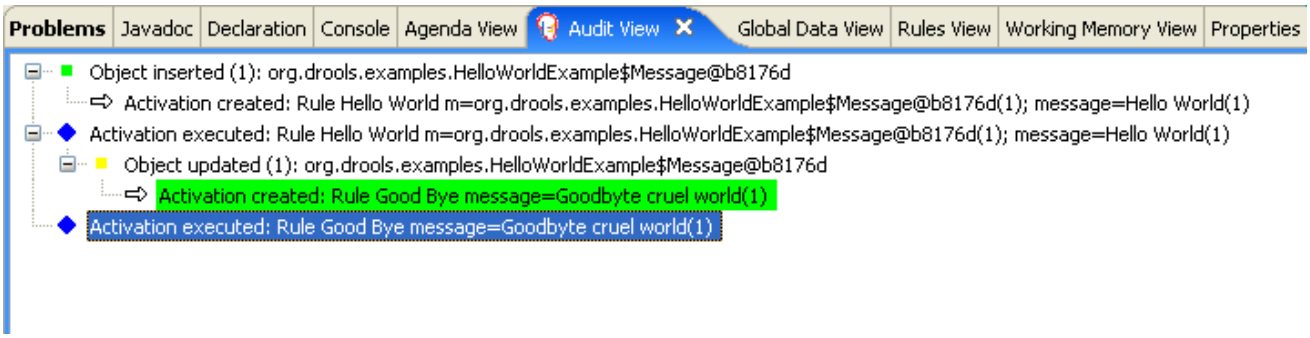
```
==>[ActivationCreated(0): rule=Hello World;
                  tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectInserted: handle=
```

```
[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
            object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[BeforeActivationFired: rule=Hello World;
             tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
==>[ActivationCreated(4): rule=Good Bye;
             tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectUpdated: handle=
[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
          old_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96;
          new_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[AfterActivationFired(0): rule=Hello World]
[BeforeActivationFired: rule=Good Bye;
             tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[AfterActivationFired(4): rule=Good Bye]
```

この例の実行フローをさらに理解するには、**target/helloworld.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**Audit view** で、オブジェクトが挿入され、**"Hello World"** ルールのアクティベーションが作成されます。次に、このアクティベーションが実行され、**Message** オブジェクトを更新して、**"Good Bye"** ルールのアクティベーションをトリガーします。最後に、**"Good Bye"** ルールが実行します。**Audit View** でイベントが選択されると、この例の **"Activation created"** イベントである元のイベントが緑色にハイライトされます。

図88.3 Hello World の例の監査ビュー



88.3. 状態の例のデシジョン (前向き連鎖および競合解決)

状態の例のディジョンセットでは、デシジョンエンジンが前向き連鎖と、ワーキングメモリー内のファクトへの変更をどのように使用してルールの実行競合を順番に解決していくのかを例示します。この例では、ルールで定義可能な顕著性の値またはアジェンダグループを使用して競合を解決することにフォーカスします。

以下は、状態の例の概要です。

- **名前**: **state**
- Main **クラス**: (**src/main/java** 内の) **org.drools.examples.state.StateExampleUsingSalience**、**org.drools.examples.state.StateExampleUsingAgendaGroup**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション

- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.state.*.drl**
- **目的**: ルールの顕著性やアジェンダグループを使用した前向き連鎖や競合解決を例示します。

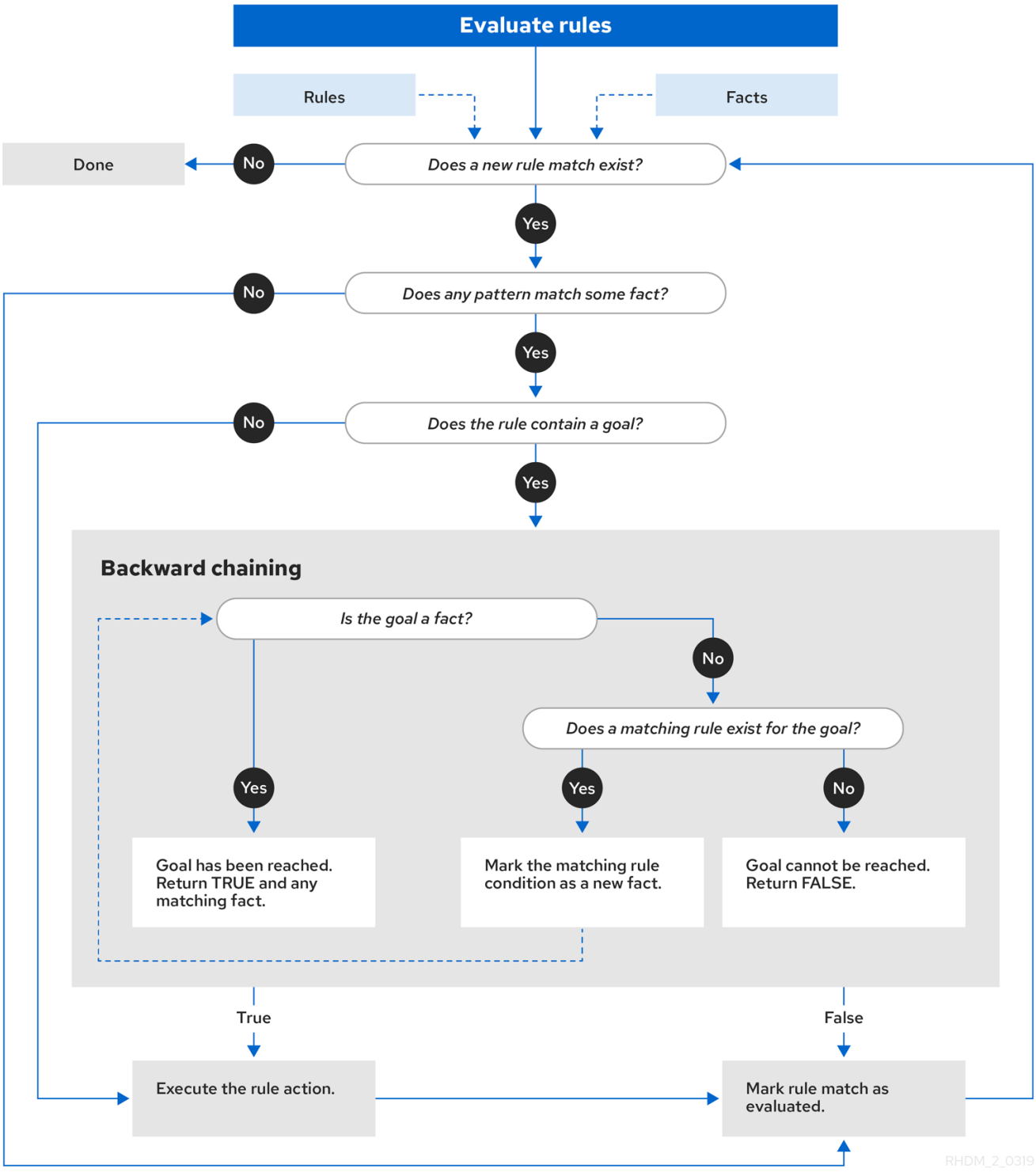
前向き連鎖のルールシステムは、ディジョンエンジンのワーキングメモリーにあるファクトで開始して、そのファクトへの変更に反応するデータ駆動型のシステムです。オブジェクトがワーキングメモリーに挿入されると、その変更の結果として True となるルールの条件はすべて、アジェンダによって実行されるようにスケジュールされます。

反対に、後向き連鎖のルールシステムは、しばしば再帰を使用して、デシジョンエンジンが満たそうとする結論から開始する目的駆動型のシステムです。システムが結論または目的に到達できない場合は、サブとなる目的、つまり、現在の目的の一部を完了する結論を検索します。システムは、最初の結論が満たされるか、すべてのサブとなる目的が満たされるまでこのプロセスを続行します。

Red Hat Process Automation Manager のデシジョンエンジンは、前向き連鎖と後向き連鎖の両方を使用してルールを評価します。

以下の図は、デシジョンエンジンが、ロジックフローで後向き連鎖のセグメントと、前向き連鎖全体を使用してルールを評価する方法を例示します。

図88.4 前向き連鎖と後向き連鎖を使用したルール評価のロジック



状態の例では、**State** クラスごとに、名前や現在の状態のフィールドが含まれます (**org.drools.examples.state.State** のクラス参照)。以下の状態は、各プロジェクトで考えられる 2 つの状態です。

- **NOTRUN**
- **FINISHED**

State クラス

```
public class State {
```

```
    public static final int NOTRUN   = 0;
    public static final int FINISHED = 1;

    private final PropertyChangeSupport changes =
        new PropertyChangeSupport( this );

    private String name;
    private int    state;

    ... setters and getters go here...
}
```

状態の例には、同じ例が 2 つのバージョンとして提供されており、それぞれルール実行の競合を解決します。

- ルールの顕著性を使用して競合を解決する **StateExampleUsingSalience** バージョン
- ルールアジェンダグループを使用して競合を解決する **StateExampleUsingAgendaGroups** バージョン

状態の例のバージョンはいずれも、**A**、**B**、**C**、および **D** の 4 つの **State** オブジェクトを使用します。最初に、それぞれの状態は、**NOTRUN** に設定されます。NOTRUN は、例が使用するコンストラクターのデフォルト値です。

顕著性を使用した状態の例
状態の例の **StateExampleUsingSalience** バージョンでは、ルールで顕著性の値を使用し、ルール実行の競合を解決します。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。

この例では、各 **State** インスタンスを KIE セッションに挿入して、**fireAllRules()** を呼び出します。

顕著性の状態例の実行

```
final State a = new State( "A" );
final State b = new State( "B" );
final State c = new State( "C" );
final State d = new State( "D" );

ksession.insert( a );
ksession.insert( b );
ksession.insert( c );
ksession.insert( d );

ksession.fireAllRules();

// Dispose KIE session if stateful (not required if stateless).
ksession.dispose();
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.state.StateExampleUsingSalience** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの顕著性の状態例の出力

```
A finished
B finished
C finished
D finished
```

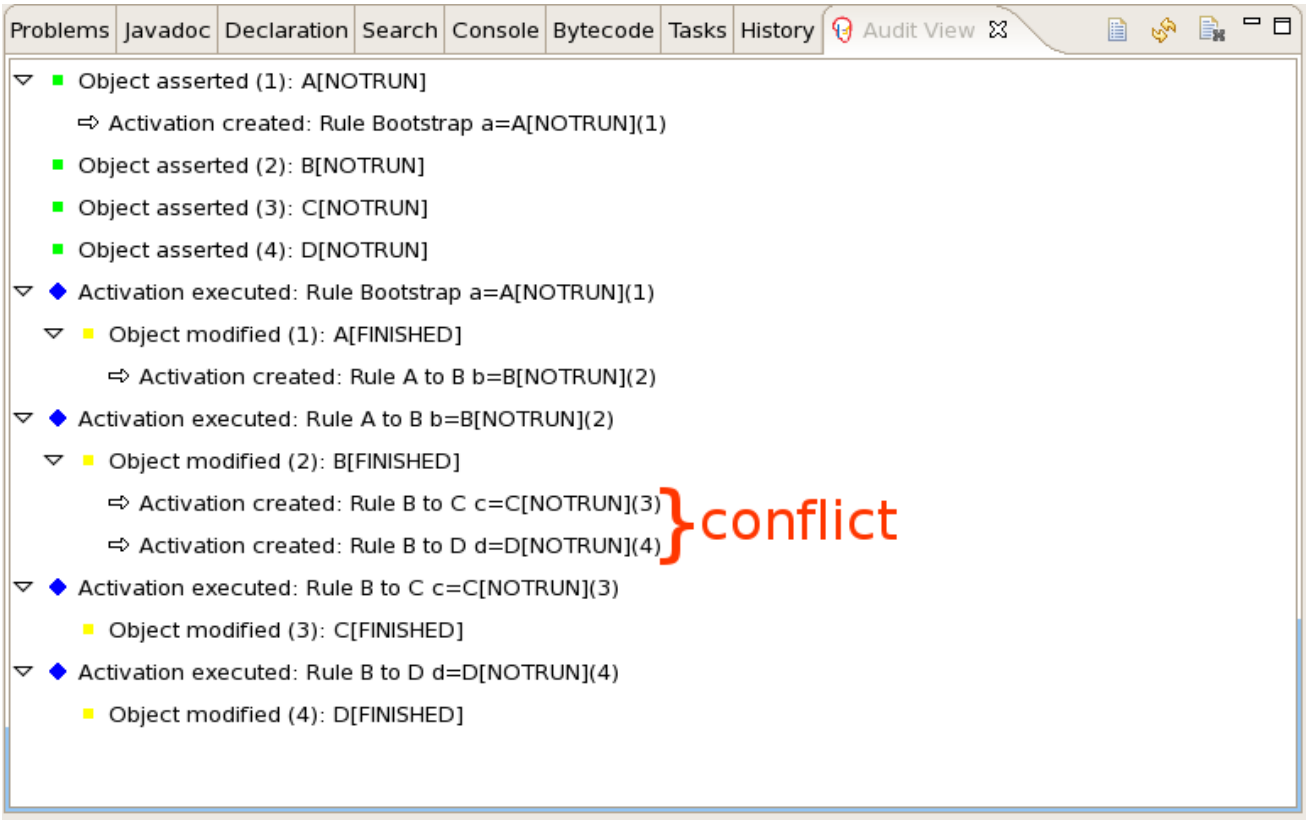
4 つのルールが存在します。

まず、**"Bootstrap"** ルールが実行され、**A** の状態が **FINISHED** に設定されます。次に、**B** の状態が **FINISHED** に変更されます。オブジェクト **C** と **D** はいずれも **B** に依存するため競合が発生しますが、顕著性の値で解決されます。

この例の実行フローをさらに理解するには、**target/state.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**Audit View** は、状態が **NOTRUN** のオブジェクト **A** のアサーションが **"Bootstrap"** ルールをアクティベートしますが、他のオブジェクトのアサーションはすぐに有効になりません。

図88.5 顕著性の状態例の監査ビュー



顕著性の状態例の "Bootstrap" ルール

```
rule "Bootstrap"
  when
    a : State(name == "A", state == State.NOTRUN )
  then
    System.out.println(a.getName() + " finished" );
    a.setState( State.FINISHED );
end
```

"Bootstrap" ルールを実行すると、**A** の状態が **FINISHED** に変わり、ルール **"A to B"** をアクティベートします。

顕著性の状態例の "A to B" ルール

```
rule "A to B"
  when
    State(name == "A", state == State.FINISHED )
    b : State(name == "B", state == State.NOTRUN )
  then
    System.out.println(b.getName() + " finished" );
    b.setState( State.FINISHED );
end
```

"A to B" ルールを実行すると、**B** の状態を **FINISHED** に変更し、**"B to C"** と **"B to D"** の両方のルールをアクティベートして、これらのアクティベーションをデシジョンエンジンアジェンダに配置します。

顕著性の状態例の "B to C" および "B to D" ルール

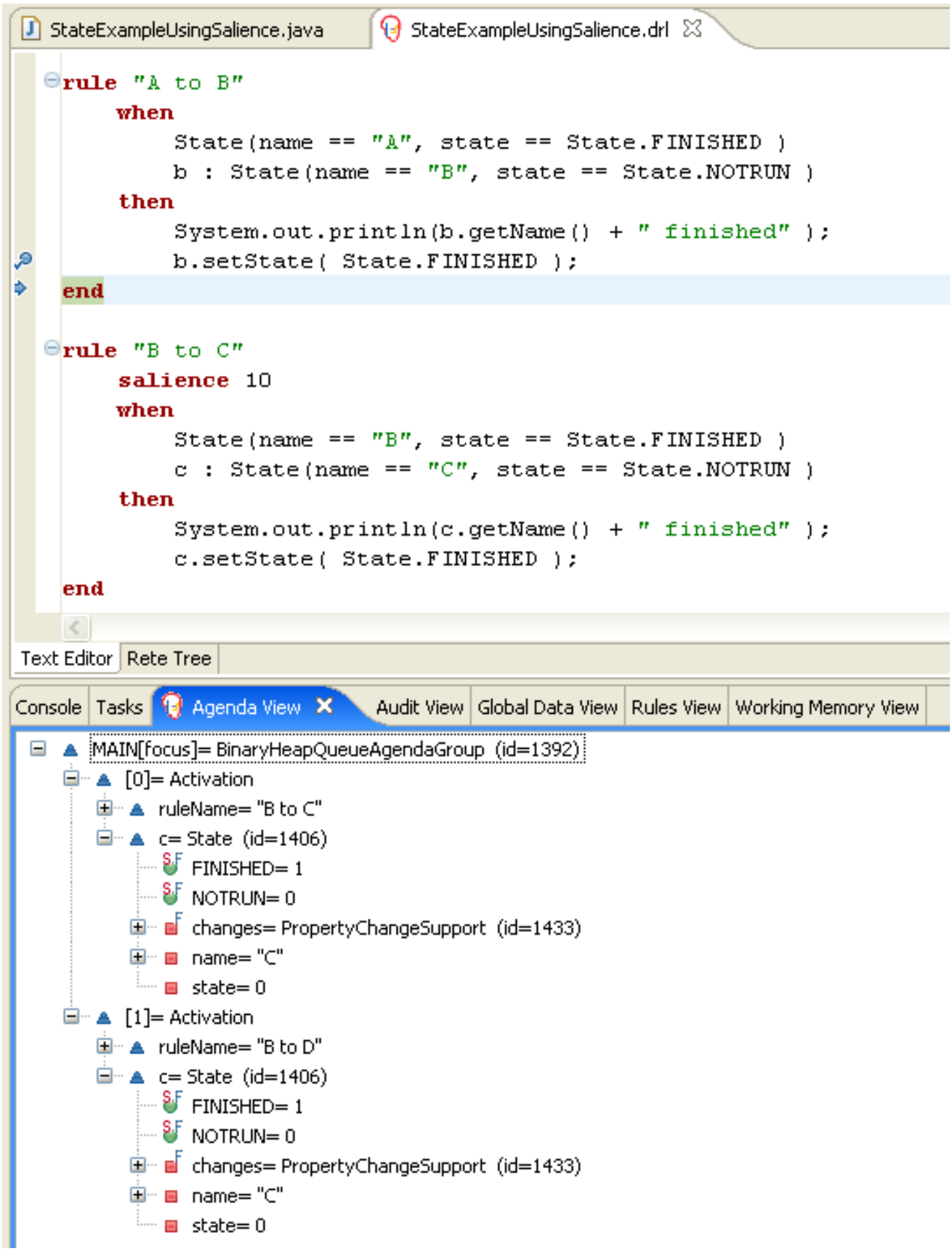
```
rule "B to C"
    salience 10
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
end

rule "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

この時点から、両方のルールが実行される可能性があるため、これらのルールは競合しています。競合解決ストラテジーを使用すると、デシジョンエンジンアジェンダがどのルールを実行するかを決定できます。**"B to C"** は、顕著性の値が高いため (デフォルトの顕著性の値 **0** に対して **10**) 先に実行し、オブジェクト **C** の状態が **FINISHED** に変更されます。

IDE の **Audit View** では、ルール **"A to B"** の **State** オブジェクトが変更され、2 つのアクティベーションが競合する結果になることが分かります。

IDE で **Agenda View** を使用して、デシジョンエンジンアジェンダの状態を調査できます。この例では **Agenda View** で、ルール **"A to B"** のブレークポイントと、2 つの競合するルールを持つアジェンダの状態が分かります。最後にルール **"B to D"** が実行され、オブジェクト **D** の状態が **FINISHED** に変更されます。

図88.6 顕著性の状態例のアジェンダビュー



アジェンダグループを使用した状態の例

状態の例の **StateExampleUsingAgendaGroups** バージョンでは、ルールでアジェンダグループを使用し、ルール実行における競合を解決します。アジェンダグループを使用すると、デシジョンエンジンアジェンダが分割され、ルールのグループの実行に対してこれまで以上に制御ができるようになります。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

最初は、ワーキングメモリーは、アジェンダグループ **MAIN** にフォーカスを当てます。アジェンダグループのルールは、グループがこのフォーカスを受けた場合のみ実行されます。**setFocus()** メソッドか、**auto-focus** ルール属性を使用してフォーカスを設定できます。**auto-focus** 属性を使用すると、ルールが一致してアクティベートされた場合のみ、ルールにアジェンダグループのフォーカスが自動的に当てられます。

この例では、**auto-focus** 属性を使用すると **"B to D"** の前に **"B to C"** ルールを実行できます。

アジェンダグループの状態例のルール "B to C"

```
rule "B to C"
    agenda-group "B to C"
    auto-focus true
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "B to D" ).setFocus();
end
```

ルール **"B to C"** は、アジェンダグループ **"B to D"** の **setFocus()** を呼び出し、アクティブなルールを実行できるようにします。その後にルール **"B to D"** が実行できるようになります。

アジェンダグループの状態例のルール "B to D"

```
rule "B to D"
    agenda-group "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.state.StateExampleUsingAgendaGroups** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます (状態の例の顕著性バージョンと同じ)。

IDE コンソールでのアジェンダグループの状態例の出力

```
A finished
B finished
C finished
D finished
```

状態の例の含まれる動的なファクト

状態の例に含まれる主なコンセプトとしては、他にも **PropertyChangeListener** オブジェクトを実装するオブジェクトに基づいて **動的ファクト** を使用するというものがあります。デシジョンエンジンがファクトプロパティーへの変更を確認し、対応するためには、アプリケーションがデシジョンエンジン

に対して、変更があったことを通知する必要があります。**modify** ステートメントを使用して、このコミュニケーションをルールで明示的に設定するか、JavaBeans 仕様で定義されているようにファクトが **PropertyChangeSupport** インターフェイスを実装するように指定することで暗黙的に設定できます。

この例は、ルールで **modify** ステートメントを明示的に指定しなくても良いように **PropertyChangeSupport** インターフェイスを使用する方法が示されています。このインターフェイスを使用するには、**org.drools.example.State** クラスと同じ方法で、ファクトに **PropertyChangeSupport** が実装されていることを確認し、DRL ルールファイルで以下のコードを使用して、これらのファクトでプロパティー変更がないかをリッスンするようにデシジョンエンジンを設定してください。

動的ファクトの宣言

```
declare type State
  @propertyChangeSupport
end
```

PropertyChangeListener オブジェクトを使用する場合に、各セッターは通知用に追加のコードを実装する必要があります。たとえば、**state** の以下のセッターは **org.drools.examples** のクラスに含まれます。

PropertyChangeSupport のセッター例

```
public void setState(final int newState) {
    int oldState = this.state;
    this.state = newState;
    this.changes.firePropertyChange( "state",
                                     oldState,
                                     newState );
}
```

88.4. フィボナッチの例のデシジョン (再帰および競合解決)

フィボナッチの例のデシジョンセットでは、デシジョンエンジンが再帰をどのように使用してルールの実行競合を順番に解決していくのかを例示します。この例では、ルールで定義可能な顕著性の値を使用して競合を解決することにフォーカスします。

以下は、フィボナッチの例の概要です。

- **名前**: **フィボナッチ**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.fibonacci.FibonacciExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.fibonacci.Fibonacci.drl**
- **目的**: ルールの顕著性を使用した再帰や競合解決を例示します。

フィボナッチ数は、0 または 1 で開始する数列です。0、1、1、2、3、5、8、13、21、34、55、89、144、233、377、610、987、1597、2584、4181、6765、10946 などのように、2 つの先行する数を足すことにより、次にくるフィボナッチ数が求められます。

フィボナッチの例では、**Fibonacci** のファクトクラスを 1 つ使用し、このクラスに以下の 2 つのフィールドが含まれています。

- **sequence**
- **value**

sequence フィールドは、フィボナッチ数列のオブジェクトの位置を示します。**value** フィールドは、その数列の位置のフィボナッチオブジェクトの値を示します。**-1** は、計算する必要がある値という意味です。

フィボナッチクラス

```
public static class Fibonacci {
    private int  sequence;
    private long value;

    public Fibonacci( final int sequence ) {
        this.sequence = sequence;
        this.value = -1;
    }

    ... setters and getters go here...
}
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.fibonacci.FibonacciExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでのフィボナッチの例の出力

```
recurse for 50
recurse for 49
recurse for 48
recurse for 47
...
recurse for 5
recurse for 4
recurse for 3
recurse for 2
1 == 1
2 == 1
3 == 2
4 == 3
5 == 5
6 == 8
...
47 == 2971215073
48 == 4807526976
49 == 7778742049
50 == 12586269025
```

Java でこの動作を実現するには、sequence フィールドに **50** を指定して、**Fibonacci** オブジェクトを挿入します。この例では、次に再帰ルールを使用して、他の 49 個の **Fibonacci** オブジェクトを挿入します。

PropertyChangeSupport インターフェイスを実装して動的ファクトを使用する代わりに、この例では MVEL 方言の **modify** キーワードを使用して、ブロックセッターアクションを有効にしてデシジョンエンジンに変更を通知しています。

フィボナッチの例の実行

```
ksession.insert( new Fibonacci( 50 ) );
ksession.fireAllRules();
```

この例では、以下の 3 つのルールを使用します。

- **"Recurse"**
- **"Bootstrap"**
- **"Calculate"**

"Recurse" ルールは、値が **-1** の、アサートされた各 **Fibonacci** オブジェクトを照合して、現在の値よりも数列が 1 つ小さい **Fibonacci** オブジェクトを新たに作成し、アサートします。数列フィールドが **1** に相当するオブジェクトが存在しない場合に、フィボナッチオブジェクトが追加されると毎回、このルールは再度照合され、実行されます。メモリーにフィボナッチオブジェクト 50 個がすべて存在する場合は、**not** 条件要素を使用して、ルールの合致を停止します。また、**"Bootstrap"** ルールを実行する前に **Fibonacci** オブジェクト 50 個をすべてアサートする必要があるため、このルールには **salience** の値も含まれます。

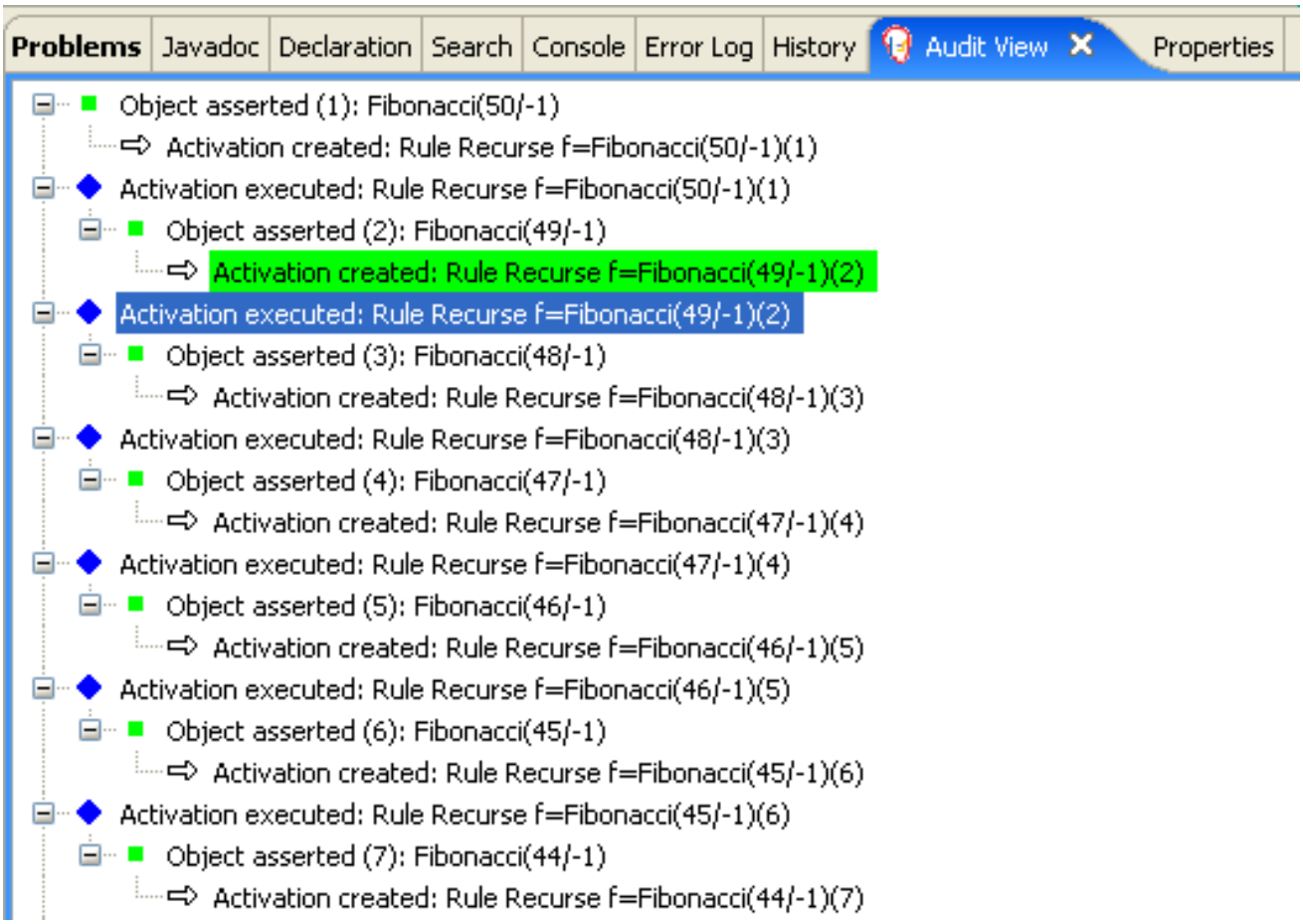
ルール "Recurse"

```
rule "Recurse"
    salience 10
  when
    f : Fibonacci ( value == -1 )
    not ( Fibonacci ( sequence == 1 ) )
  then
    insert( new Fibonacci( f.sequence - 1 ) );
    System.out.println( "recurse for " + f.sequence );
end
```

この例の実行フローをさらに理解するには、**target/fibonacci.log** の監査ログファイルを IDE デバッグビュー (または **Audit View** が利用できる場合は Audit View (例: IDE の **Window → Show View**)) に読み込みます。

この例では、**監査ビュー** に、**sequence** フィールドが **50** に指定された、**Fibonacci** の元のアサーションが表示されます。これは Java コードで実行されています。これ以降、**監査ビュー** で、ルールの再帰が継続して行われ、アサートされた **Fibonacci** オブジェクトにより、**"Recurse"** ルールがアクティベートされて、再度実行されます。

図88.7 監査ビューでのルール "Recurse"



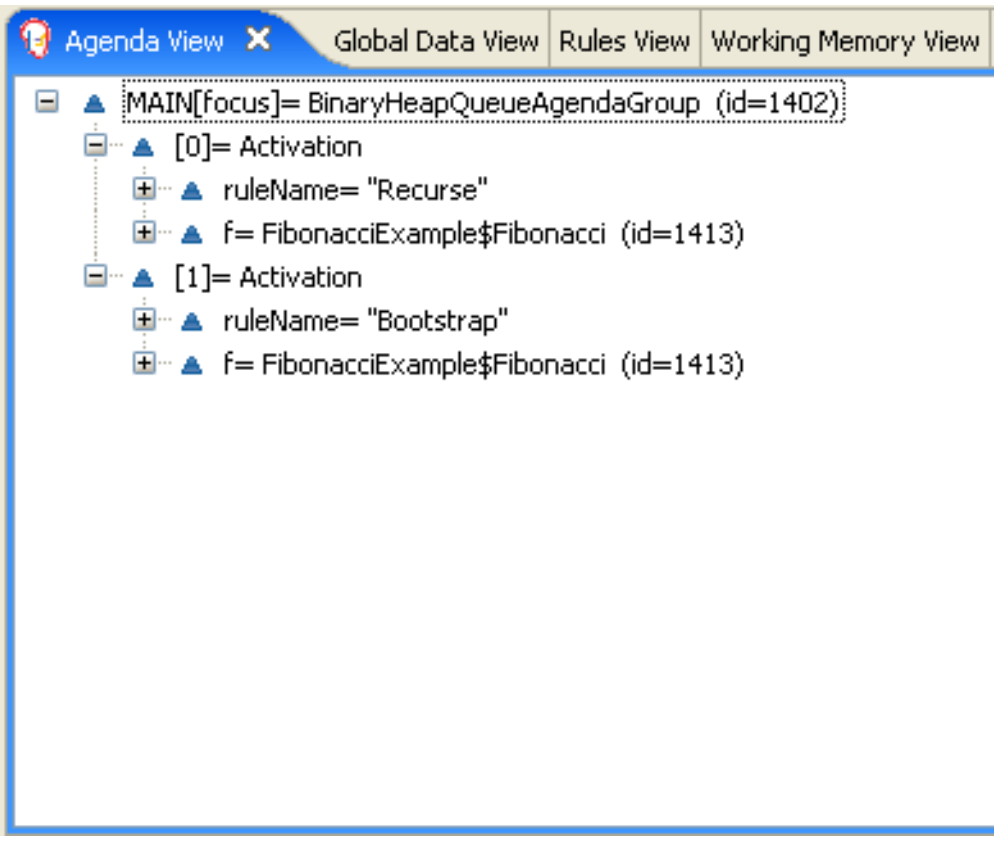
sequence フィールドが **2** の **Fibonacci** オブジェクトがアサートされると、**"Bootstrap"** ルールが一致し、**"Recurse"** ルールとともにアクティベートされます。フィールド **sequence** には複数の制約があり、**1** または **2** と同等かをテストしている点に注目してください。

ルール "Bootstrap"

```
rule "Bootstrap"
  when
    f : Fibonacci( sequence == 1 || == 2, value == -1 ) // multi-restriction
  then
    modify ( f ){ value = 1 };
    System.out.println( f.sequence + " == " + f.value );
end
```

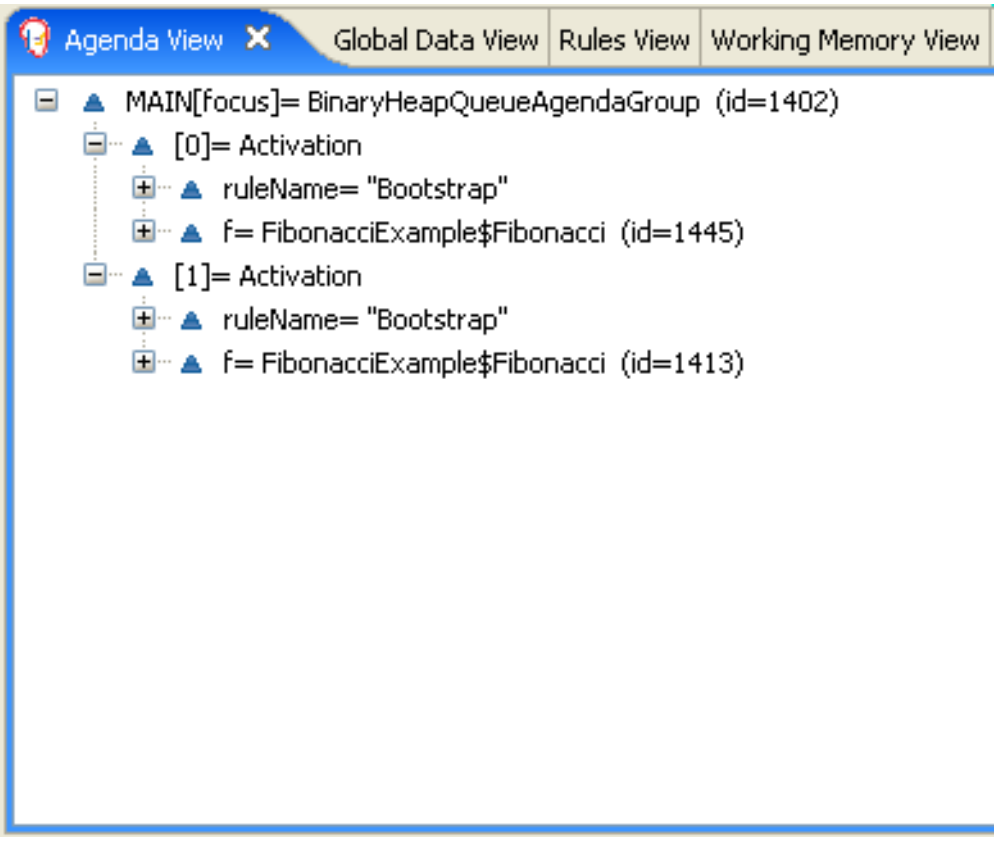
IDE で **Agenda View** を使用して、デシジョンエンジンアジェンダの状態を調査できます。**"Recurse"** の顕著性の値のほうが高いため、**"Bootstrap"** ルールは実行していません。

図88.8 アジェンダビュー 1 でのルール "Recurse" および "Bootstrap"



sequence が **1** の **Fibonacci** オブジェクトがアサートされると、**"Bootstrap"** ルールが再度一致し、このルールに含まれる 2 つのルールがアクティベートされます。**sequence** が **1** の **Fibonacci** オブジェクトが存在すると、すぐに **not** 条件要素でルールが一致しなくなるため、**"Recurse"** ルールの照合やアクティベーションはされません。

図88.9 アジェンダビュー 2 でのルール "Recurse" および "Bootstrap"



"Bootstrap" ルールは、**sequence** が **1** と **2** のオブジェクトの値を **1** に設定します。値が **-1** でない **Fibonacci** オブジェクトが 2 つあるため、**"Calculate"** ルールの照合が可能になります。

この例のある時点で、ワーキングメモリーに 50 近くの **Fibonacci** オブジェクトが存在します。3 つ選択してそれぞれを乗算し、順番に各値を計算する必要があります。フィールドの制約なしに、ルールで 3 つの Fibonacci パターンを使用してクラス積候補を絞り込む場合に、考えられる組み合わせとして 50x49x48 通りあり、約 12 万 5000 のルールを実行できるにもかかわらず、その大半が誤っていることになります。

"Calculate" ルールは、フィールドの制約を使用して正しい順番にフィボナッチパターンを 3 つ評価します。この手法は **cross-product matching** と呼ばれます。

最初のパターンでは、値が **!= -1** の **Fibonacci** オブジェクトを検索して、このパターンとフィールド両方をバインドします。2 番目の **Fibonacci** オブジェクトが実行する内容は同じですが、別のフィールド制約を追加して、シーケンスが **f1** にバインドされている **Fibonacci** オブジェクトより 1 つ大きくなるようにします。このルールが初めて実行されると、シーケンスが **1** と **2** にだけ、値 **1** が割り当てられていることが分かります。また、この 2 つの制約で、**f1** がシーケンス **1** を参照し、**f2** がシーケンス **2** を参照するようにします。

最後のパターンでは、値が **-1** と等しく、シーケンスが **f2** よりも大きい **Fibonacci** オブジェクトを検索します。

フィボナッチの例のこの時点で、3 つの **Fibonacci** オブジェクトが利用可能なクロス積から正しく選択され、**f3** にバインドされている 3 番目の **Fibonacci** オブジェクトの値を計算できます。

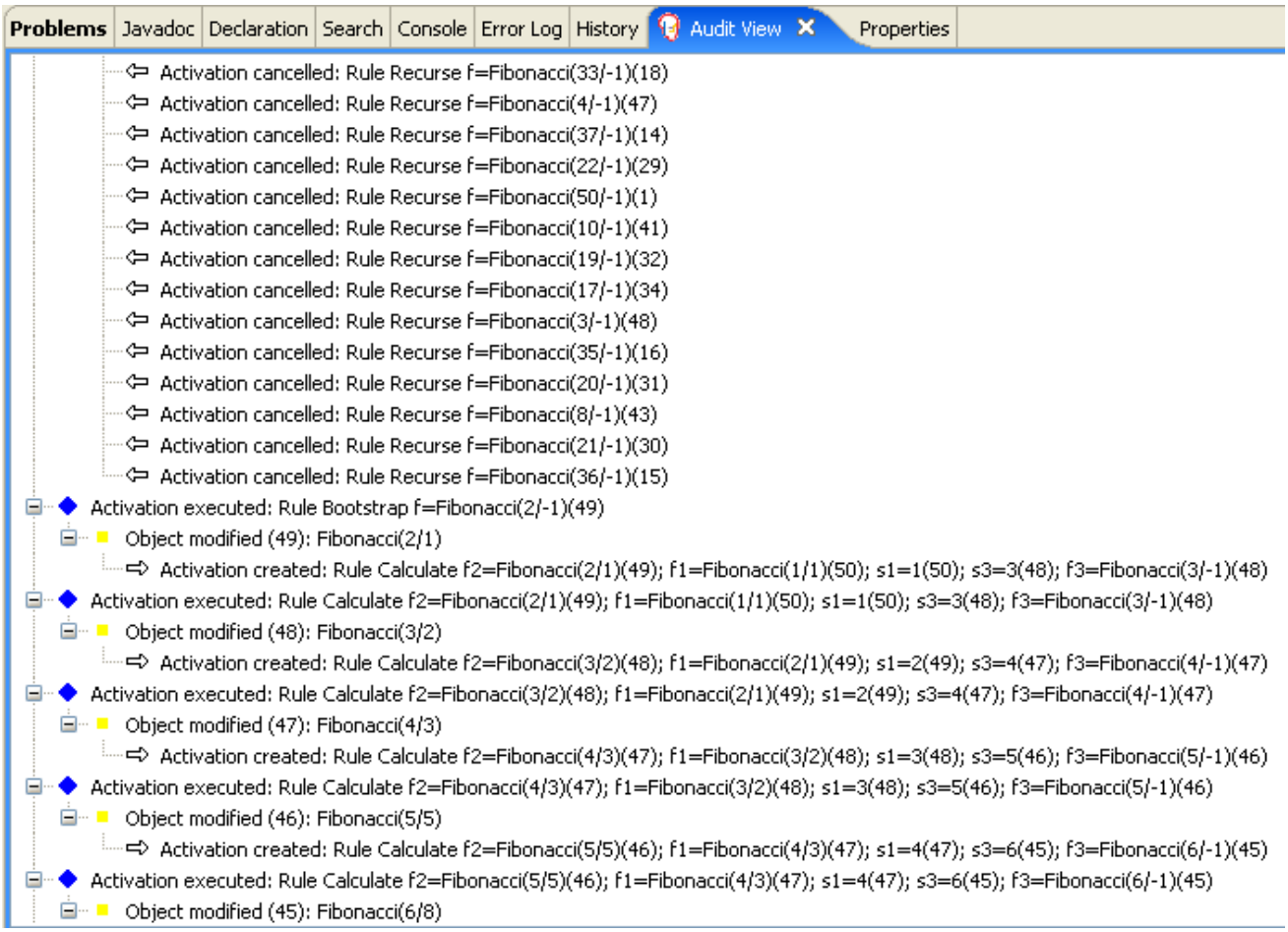
ルール "Calculate"

```
rule "Calculate"
  when
    // Bind f1 and s1.
    f1 : Fibonacci( s1 : sequence, value != -1 )
    // Bind f2 and v2, refer to bound variable s1.
    f2 : Fibonacci( sequence == (s1 + 1), v2 : value != -1 )
    // Bind f3 and s3, alternative reference of f2.sequence.
    f3 : Fibonacci( s3 : sequence == (f2.sequence + 1 ), value == -1 )
  then
    // Note the various referencing techniques.
    modify ( f3 ) { value = f1.value + v2 };
    System.out.println( s3 + " == " + f3.value );
end
```

modify ステートメントにより、**f3** にバインドされた **Fibonacci** オブジェクトの値が更新されます。つまり、値が **-1** 以外の **Fibonacci** オブジェクトが新たに存在するということで、**"Calculate"** ルールにより、再度合致があるか検索して次のフィボナッチ番号を算出することができます。

IDE のデバッグビューまたは **監査ビュー** では、最後の **"Bootstrap"** ルールが実行されることで **Fibonacci** オブジェクトが変更され、**"Calculate"** ルールに合致し、次に、別の **Fibonacci** オブジェクトが変更され、この **"Calculate"** ルールに再度合致できていることが分かります。このプロセスは、すべての **Fibonacci** オブジェクトに値が設定されるまで継続されます。

図88.10 監査ビューのルール



88.5. 価格設定のデシジョン例 (デシジョンテーブル)

価格設定のデシジョンセットの例では、スプレッドシートのデシジョンテーブルを使用して、DRL ファイルに直接ではなく、表形式で保険料金の価格を計算する方法を説明します。

以下は価格設定の例の概要です。

- **名前**: **decisiontable**
- Main **クラス**: (**src/main/java** の場合)
  org.drools.examples.decisiontable.PricingRuleDTExample
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: **org.drools.examples.decisiontable.ExamplePolicyPricing.xls**
  (**src/main/resources** 内)
- **目的**: スプレッドシートのデシジョンテーブルを使用してルールを定義する方法を示します。

スプレッドシートのデシジョンテーブルは、表形式でビジネスルールを定義する XLS 形式または XLSX 形式のスプレッドシートです。スプレッドシートのデシジョンテーブルは、スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにアップロードしたりできます。スプレッドシートの各行がルールになり、各列が条件、アクション、または別

のルール属性になります。最初に Red Hat Process Automation Manager でデシジョンテーブルを作成してアップロードします。次に、その他のすべてのルールアセットと同じように、定義したルールを Drools Rule Language (DRL) ルールにコンパイルします。

この価格設定の例では、特定タイプの保険を申請するドライバーに対して基本価格と割引を計算するビジネスルールセットを提供します。ドライバーの年齢と履歴、およびポリシータイプはすべて、基本保険料の計算に役立ち、追加のルールは、ドライバーが適格となる可能性のある潜在的な割引を計算します。

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.decisiontable.PricingRuleDTExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

```
Cheapest possible
BASE PRICE IS: 120
DISCOUNT IS: 20
```

この例を実行するコードは、標準の実行パターンに準拠しています。ルールが読み込まれ、ファクトが挿入されて、ステートレス KIE セッションが作成されます。この例における違いは、DRL ファイルや他のソースではなく、**ExamplePolicyPricing.xls** ファイルでルールが定義されるという点です。このスプレッドシートファイルは、テンプレートと DRL ルールを使用してデシジョンエンジンに読み込まれます。

スプレッドシートのデシジョンテーブルの設定

ExamplePolicyPricing.xls スプレッドシートには、以下の 2 つのデシジョンテーブルが含まれています。

- **Base pricing rules**
- **Promotional discount rules**

この例のスプレッドシートで分かるように、デシジョンテーブルの作成にはスプレッドシートの最初のタブしか使用できませんが、単一のタブ内に複数のテーブルが作成できます。デシジョンテーブルは必ずしもトップダウンの論理に従うものではなく、ルールとなるデータを補足する手段です。ルールの評価は、ルールエンジンのすべての通常のメカニックが適用されるため、必ずしも特定の順序で行われるわけではありません。このために、スプレッドシートの同一タブ内に複数のデシジョンテーブルが作成可能となります。

デシジョンテーブルは、対応するルールテンプレートファイルである **BasePricing.drt** と **PromotionalPricing.drt** で実行されます。これらのテンプレートファイルはテンプレートパラメーターによってデシジョンテーブルを参照し、デシジョンテーブルの条件およびアクションの各種ヘッダーを直接参照します。

BasePricing.drt ルールテンプレートファイル

```
template header
age[]
profile
priorClaims
policyType
base
reason

package org.drools.examples.decisiontable;
```

```
template "Pricing bracket"
age
policyType
base

rule "Pricing bracket_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}
        , priorClaims == "@{priorClaims}"
        , locationRiskProfile == "@{profile}"
    )
    policy: Policy(type == "@{policyType}")
  then
    policy.setBasePrice(@{base});
    System.out.println("@{reason}");
end
end template
```

PromotionalPricing.drt ルールテンプレートファイル

```
template header
age[]
priorClaims
policyType
discount

package org.drools.examples.decisiontable;

template "discounts"
age
priorClaims
policyType
discount

rule "Discounts_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}, priorClaims == "@{priorClaims}")
    policy: Policy(type == "@{policyType}")
  then
    policy.applyDiscount(@{discount});
end
end template
```

ルールは、KIE セッション **DTableWithTemplateKB** の **kmodule.xml** 参照によって実行されます。これは **ExamplePolicyPricing.xls** スプレッドシートを特定して参照するもので、ルールの実行の成功には必要なものです。この実行方法により、ルールをスタンドアロンユニットとして実行したり (ここでの例) パッケージ化されたナレッジ JAR (KJAR) ファイルにルールを含めたりすることができるため、スプレッドシートはルール実行とともにパッケージ化されます。

kmodule.xml ファイルの以下のセクションは、ルールが正常に実行され、スプレッドシートが機能するために必要になります。

```
<kbase name="DecisionTableKB" packages="org.drools.examples.decisiontable">
    <ksession name="DecisionTableKS" type="stateless"/>
</kbase>
```

```
<kbase name="DTableWithTemplateKB" packages="org.drools.examples.decisiontable-template">
  <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
          template="org/drools/examples/decisiontable-template/BasePricing.drt"
          row="3" col="3"/>
  <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
          template="org/drools/examples/decisiontable-template/PromotionalPricing.drt"
          row="18" col="3"/>
  <ksession name="DTableWithTemplateKS"/>
</kbase>
```

ルールテンプレートファイルを使用したデシジョンテーブルの実行方法とは別に、**DecisionTableConfiguration** オブジェクトを使用して、**DecisionTableInputType.xls** のような入力スプレッドシートを入力タイプとして指定することもできます。

```
DecisionTableConfiguration dtableconfiguration =
  KnowledgeBuilderFactory.newDecisionTableConfiguration();
  dtableconfiguration.setInputType( DecisionTableInputType.XLS );

  KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

  Resource xlsRes = ResourceFactory.newClassPathResource( "ExamplePolicyPricing.xls",
                                          getClass() );
  kbuilder.add( xlsRes,
          ResourceType.DTABLE,
          dtableconfiguration );
```

価格設定の例では以下の 2 つのファクトタイプを使用します。

- **Driver**
- **Policy**

この例では、これらのファクトのデフォルト値をそれぞれの Java クラス **Driver.java** と **Policy.java** に設定します。この **Driver** は 30 歳で、これまでに保険の請求をしたことがなく、現在のリスクプロファイルが **LOW** となっています。申請している **Policy** は **COMPREHENSIVE** です。

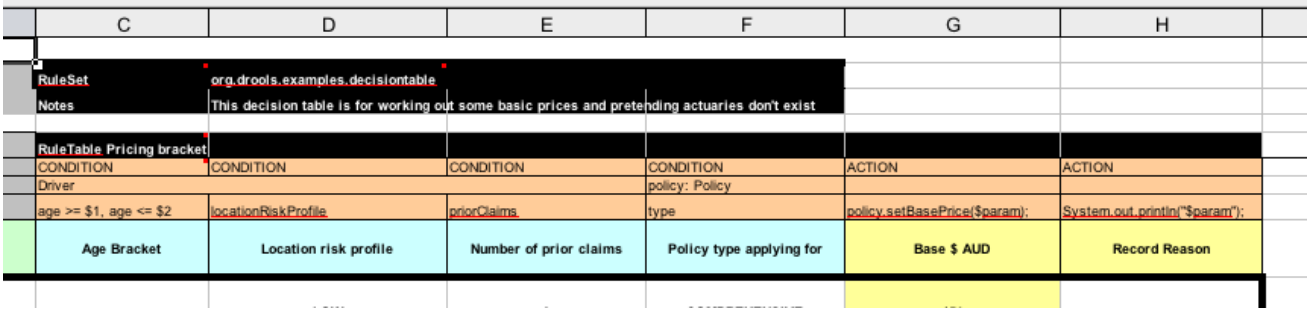
デシジョンテーブルでは、各行は異なるルールと見なされ、各列は条件またはアクションとみなされます。実行時にアジェンダがクリアされなければ、デシジョンテーブルの各行が評価されます。

デシジョンテーブルのスプレッドシート (XLS または XLSX) には、ルールデータを定義する以下の 2 つの主要なエリアが必要です。

- **RuleSet** エリア
- **RuleTable** エリア

RuleSet 領域では、ルールセット名、ユニバーサルルール属性など、(このスプレッドシートだけでなく) すべてのルールをパッケージ全体に、グローバルに適用する要素を定義します。**RuleTable** 領域では、実際のルール (行) と、指定したルールセットのルールテーブルを設定する条件、アクション、その他のルール属性 (列) を定義します。デシジョンテーブルのスプレッドシートには複数の **RuleTable** エリアを追加できますが、**RuleSet** エリアは 1 つのみとなります。

図88.11 デシジョンテーブルの設定



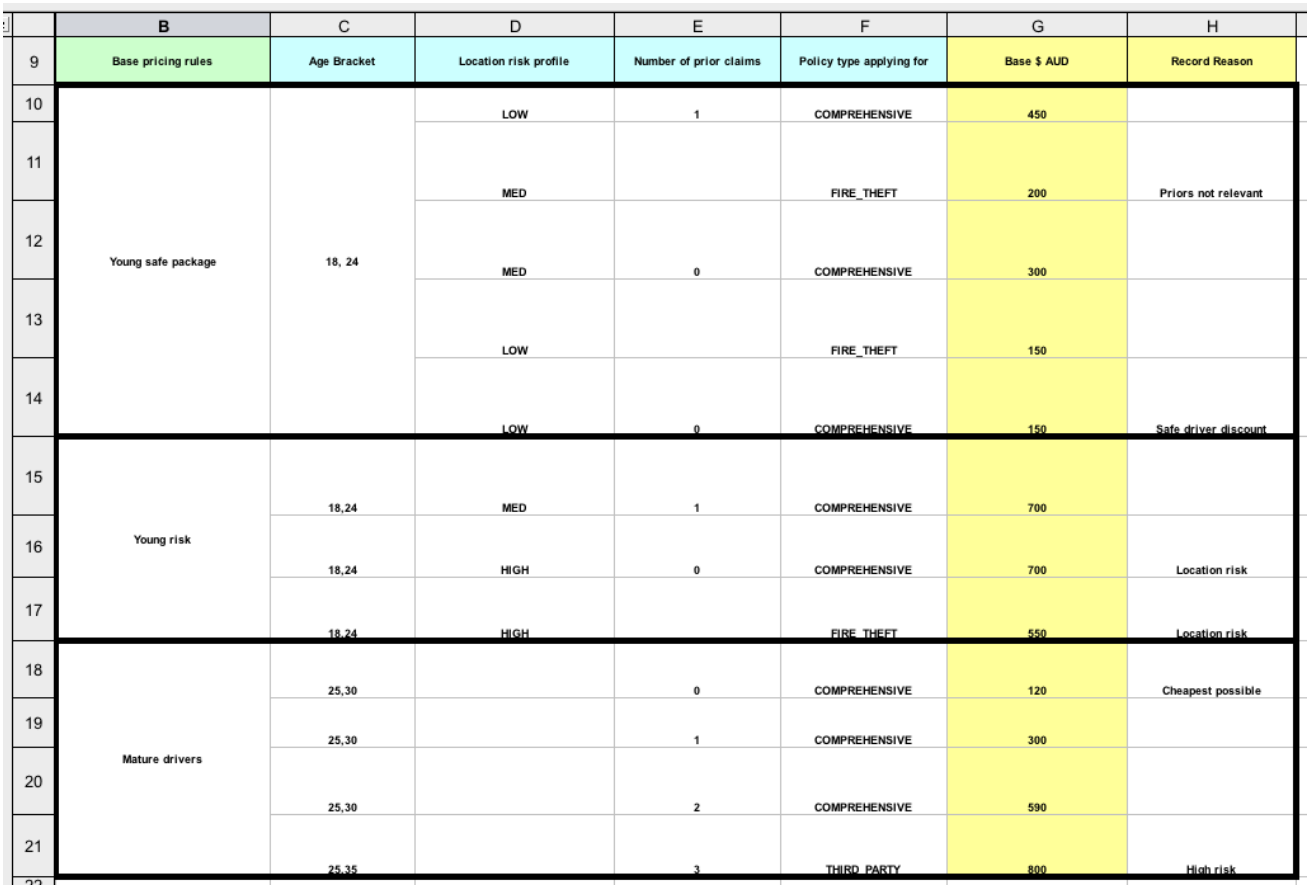
	C	D	E	F	G	H
	RuleSet	org.drools.examples.decisiontable				
	Notes	This decision table is for working out some basic prices and pretending actuaries don't exist				
	RuleTable Pricing bracket					
	CONDITION	CONDITION	CONDITION	CONDITION	ACTION	ACTION
	Driver			policy: Policy		
	age >= $1, age <= $2	locationRiskProfile	priorClaims	type	policy.setBasePrice($param);	System.out.println("$param");
	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason

RuleTable エリアでは、ルール属性を適用するオブジェクトも定義します。この例では、**Driver** と **Policy**、さらにオブジェクトの制限です。たとえば、**Driver** オブジェクトの制限では、**Age Bracket** 列が **age >= $1, age <= $2** と定義されています。ここでのコンマ区切りの範囲は、**18,24** などのテーブルの列値で定義されます。

Base pricing rules

価格設定例での **Base pricing rules** デシジョンテーブルでは、ドライバーの年齢、リスクプロファイル、請求数、ポリシータイプを評価し、これらの条件をベースにしたポリシーの基本価格を算定します。

図88.12 基本価格の計算



	B	C	D	E	F	G	H
9	Base pricing rules	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason
10	Young safe package	18, 24	LOW	1	COMPREHENSIVE	450	
11			MED		FIRE_THEFT	200	Priors not relevant
12			MED	0	COMPREHENSIVE	300	
13			LOW		FIRE_THEFT	150	
14			LOW	0	COMPREHENSIVE	150	Safe driver discount
15	Young risk	18,24	MED	1	COMPREHENSIVE	700	
16		18,24	HIGH	0	COMPREHENSIVE	700	Location risk
17		18,24	HIGH		FIRE_THEFT	550	Location risk
18	Mature drivers	25,30		0	COMPREHENSIVE	120	Cheapest possible
19		25,30		1	COMPREHENSIVE	300	
20		25,30		2	COMPREHENSIVE	590	
21		25,35		3	THIRD_PARTY	800	High risk

Driver 属性は、以下のテーブル列で定義されます。

- **Age Bracket**: この年齢層には、ドライバー年齢の条件範囲を定義する条件 **age >=$1, age <=$2** の定義があります。この条件列では **$1 and $2** を使用しており、スプレッドシートではコンマ区切りになります。ここでの値入力は **18,24** や **18, 24** の形式となり、両方ともビジネスルールの実行で機能する形式です。

- **Location risk profile**: リスクプロファイルは、この例のプログラムでは常に **LOW** として渡す文字列です。ただし、**MED** または **HIGH** に変更することが可能です。

- **Number of prior claims**: これまでの請求数は整数で定義し、アクションをトリガーするには条件列のものと同一である必要があります。この値は範囲ではなく、完全一致のみになります。

デシジョンテーブルの **Policy** は、ルールの条件とアクションの両方で使用され、属性は以下のテーブル列で定義されます。

- **Policy type applying for**: ポリシータイプは文字列として渡される条件で、適用する以下のいずれかのポリシータイプ (**COMPREHENSIVE**、**FIRE_THEFT**、または **THIRD_PARTY**) を定義します。

- **Base $ AUD**: **basePrice** は **ACTION** として定義され、これはこの値に対応するスプレッドシートのセルをベースに制限 **policy.setBasePrice($param);** で価格を設定します。このデシジョンテーブルの対応する DRL ルールを実行する際には、ファクトに合致する true 条件でルールの **then** 部分がこのアクションステートメントを実行し、基本価格を対応する値に設定します。

- **Record Reason**: ルールが正常に実行されると、アクションは出力メッセージを **System.out** コンソールに生成し、どのルールが適用されたかが反映されます。これは後でアプリケーションにキャプチャーされ、出力されます。

この例では、左側の最初の列でルールをカテゴリー分けしています。この列は注釈目的で、ルール実行には影響がありません。

Promotional discount rules

価格設定例での **Promotional discount rules** デシジョンテーブルでは、ドライバーの年齢、請求数、ポリシータイプを評価し、ポリシー価格の割引を算定します。

図88.13 割引計算

29	Promotional discount rules	Age Bracket	Number of prior claims	Policy type applying for	Discount %
30	Rewards for safe drivers	18,24	0	COMPREHENSIVE	1
31		18,24	0	FIRE_THEFT	2
32		25,30	1	COMPREHENSIVE	5
33		25,30	2	COMPREHENSIVE	1
34		25,30	0	COMPREHENSIVE	20

このデシジョンテーブルには、ドライバーに適用可能な割引条件が含まれています。基本価格の算定と同様に、このテーブルではドライバーの **Age**、**Number of prior claims**、および **Policy type applying for** を評価して、適用する **Discount %** 率を判定します。たとえば、ドライバーは 30 歳であり、請求履歴がなく、**COMPREHENSIVE** ポリシーを申請している場合は、**20** パーセントの割引率が導き出されます。

88.6. ペットショップの例のデシジョン (アジェンダグループ、グローバル変数、コールバック、および GUI 統合)

ペットショップの例のデシジョンセットでは、ルールでのアジェンダグループとグローバル変数の使用方法と、Red Hat Process Automation Manager ルールとグラフィカルユーザーインターフェイス (GUI) (この場合は、Swing ベースのデスクトップアプリケーション) の統合方法が分かります。また、この例では、コールバックを使用して実行中のデシジョンエンジンと通信し、ランタイム時に加えられたワーキングメモリー内の変更をもとに GUI を更新する方法を例示しています。

以下は、ペットショップの例の概要です。

- **名前**: **petstore**
- Main **クラス**: (**src/main/java** 内の) **org.drools.examples.petstore.PetStoreExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.petstore.PetStore.drl**
- **目的**: ルールアジェンダグループ、グローバル変数、コールバック、および GUI 統合を例示します。

ペットショップの例では、**PetStoreExample.java** クラス例を使用して (Swing イベントを処理する複数のクラスに加え)、以下のクラスを主に定義しています。

- **Petstore** には **main()** メソッドが含まれます。
- **PetStoreUI** は Swing ベースの GUI を作成して表示します。このクラスには複数の小さいクラスが含まれており、マウスボタンのクリックなど、さまざまな GUI イベントに主に対応します。
- **TableModel** には表データが含まれています。このクラスは基本的に、Swing クラス **AbstractTableModel** を拡張する JavaBean です。
- **CheckoutCallback** により、GUI がルールと対話できるようになります。
- **Ordershow** は購入するアイテムを保持します。
- **Purchase** には、注文の詳細と、購入する製品が保存されます。
- **Product** は、販売可能な商品と価格の詳細を含む JavaBean です。

この例の Java コードはほぼ、プレーンな JavaBean か Swing ベースとなっています。Swing コンポーネントの詳細は、Creating a GUI with JFC/Swing の Java チュートリアルを参照してください。

ペットショップの例でのルール実行動作

他の例のデシジョンセットではファクトがすぐにアサートされて実行されるのに対し、ペットショップの例では、ユーザーの対話をもとに他のファクトが収集されるまでルールが実行します。このルールでは、コンストラクターで作成される **PetStoreUI** オブジェクトを使用してルールを実行し、**Vector** オブジェクトの **stock** を受けれ入れて商品を収集します。次に、この例では、以前の読み込まれたルールベースを含む **CheckoutCallback** クラスのインスタンスを使用します。

ペットショップの KIE コンテナーおよびファクト実行の設定

```
// KieServices is the factory for all KIE services.
KieServices ks = KieServices.Factory.get();

// Create a KIE container on the class path.
KieContainer kc = ks.getKieClasspathContainer();

// Create the stock.
Vector<Product> stock = new Vector<Product>();
stock.add( new Product( "Gold Fish", 5 ) );
stock.add( new Product( "Fish Tank", 25 ) );
```

```
stock.add( new Product( "Fish Food", 2 ) );

// A callback is responsible for populating the working memory and for firing all rules.
PetStoreUI ui = new PetStoreUI( stock,
                                new CheckoutCallback( kc ) );
ui.createAndShowGUI();
```

ルールを実行する Java コードは **CheckoutCallBack.checkout()** メソッドに含まれます。このメソッドは、ユーザーが UI で **チェックアウト** をクリックするとトリガーされます。

CheckoutCallBack.checkout() からのルール実行

```
public String checkout(JFrame frame, List<Product> items) {
    Order order = new Order();

    // Iterate through list and add to cart.
    for ( Product p: items ) {
        order.addItem( new Purchase( order, p ) );
    }

    // Add the JFrame to the ApplicationData to allow for user interaction.

    // From the KIE container, a KIE session is created based on
    // its definition and configuration in the META-INF/kmodule.xml file.
    KieSession ksession = kcontainer.newKieSession("PetStoreKS");

    ksession.setGlobal( "frame", frame );
    ksession.setGlobal( "textArea", this.output );

    ksession.insert( new Product( "Gold Fish", 5 ) );
    ksession.insert( new Product( "Fish Tank", 25 ) );
    ksession.insert( new Product( "Fish Food", 2 ) );

    ksession.insert( new Product( "Fish Food Sample", 0 ) );

    ksession.insert( order );

    // Execute rules.
    ksession.fireAllRules();

    // Return the state of the cart
    return order.toString();
}
```

このコード例では、2 つの要素を **CheckoutCallBack.checkout()** メソッドに渡します。1 つ目の要素は、GUI の一番下にある出力テキストのフレームを囲む Swing コンポーネント **JFrame** のハンドルです。2 つ目の要素は注文アイテムのリストで、GUI の右上のセクションにある **Table** エリアからの情報を保存する **TableModel** から取得します。

for ループは GUI からの注文アイテム一覧を JavaBean **Order** に変換します。これは、**PetStoreExample.java** ファイルにも含まれています。

今回の例では、データはすべて Swing コンポーネントに含まれており、ユーザーが UI の **チェックアウト** をクリックしない限り実行しないため、ルールはステートレスの KIE セッションで実行します。ユーザーが **チェックアウト** をクリックするたびに、リストの内容を Swing **TableModel** から KIE セッショ

ンのワーキングメモリーに移動し、**ksession.fireAllRules()** メソッドで実行します。

このコード内には、**KieSession** への呼び出しが 9 個あります。1 つ目は、**KieContainer** から新しい **KieSession** を作成します (この例では、**main()** メソッドの **CheckoutCallBack** クラスから **KieContainer** に渡されます)。次の 2 つの呼び出しは、ルールでグローバル変数として保持されるオブジェクト を 2 つ渡します (メッセージの書き込みに使用する Swing テキストエリアと Swing フレーム)。より多くの商品の情報を **KieSession** と注文リストに入力します。最後の呼び出しは、標準の **fireAllRules()** です。

ペットショップのルールファイルのインポート、グローバル変数、Java 関数

PetStore.drl ファイルには、さまざまな Java クラスをルールで利用できるように、標準のパッケージとインポートステートメントが含まれています。このルールファイルには、**frame**、**textArea** などのように、ルール内で使用する **グローバル変数** が含まれています。グローバル変数では、Swing コンポーネント **JFrame** と、**setGlobal()** メソッドを呼び出した Java コードにより以前に渡された **JTextArea** コンポーネントへの参照を保持します。ルールが実行するとすぐに失効するルールの標準変数とは異なり、グローバル変数は KIE セッションの有効期間中この値を保持します。これは、このグローバル変数の内容が、後続のすべてのルールで評価できることを意味します。

PetStore.drl パッケージ、インポート、およびグローバル変数

```
package org.drools.examples;

import org.kie.api.runtime.KieRuntime;
import org.drools.examples.petstore.PetStoreExample.Order;
import org.drools.examples.petstore.PetStoreExample.Purchase;
import org.drools.examples.petstore.PetStoreExample.Product;
import java.util.ArrayList;
import javax.swing.JOptionPane;

import javax.swing.JFrame;

global JFrame frame
global javax.swing.JTextArea textArea
```

PetStore.drl ファイルには、このファイル内のルールが使用する関数が 2 つも含まれています。

PetStore.drl Java 関数

```
function void doCheckout(JFrame frame, KieRuntime krt) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to checkout?",
                                             "",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

        if (n == 0) {
            krt.getAgenda().getAgendaGroup( "checkout" ).setFocus();
        }
}
```

```
function boolean requireTank(JFrame frame, KieRuntime krt, Order order, Product fishTank, int total)
{
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to buy a tank for your " + total + " fish?",
                                             "Purchase Suggestion",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

        System.out.print( "SUGGESTION: Would you like to buy a tank for your "
                          + total + " fish? - " );

        if (n == 0) {
            Purchase purchase = new Purchase( order, fishTank );
            krt.insert( purchase );
            order.addItem( purchase );
            System.out.println( "Yes" );
        } else {
           System.out.println( "No" );
        }
        return true;
}
```

この 2 つの関数は以下のアクションを実行します。

- **doCheckout()** は、チェックアウトするかどうかユーザーに尋ねるダイアログボックスを表示します。チェックアウトする場合は、フォーカスが **checkout** アジェンダグループに設定され、そのグループのルールを (今後) 実行できるようにします。
- **requireTank()** は、水槽を購入するかどうかを確認するダイアログを表示します。購入する場合は、新しい水槽の **Product** がワーキングメモリーの注文リストに追加されます。

注記

この例では、効率化を図るため、すべてのルールと関数が同じルールファイルで実行しています。実稼働環境では、通常、ルールと関数を別のファイルに分けるか、静的な Java メソッドを構築して、**import function my.package.name.hello** などのインポート関数を使用し、ファイルをインポートします。

アジェンダグループを使用したペットショップルール

ペットショップの例のルールはほぼ、アジェンダグループを使用してルールの実行を制御しています。アジェンダグループを使用すると、デシジョンエンジンアジェンダを分割し、ルールのグループの実行を、詳細にわたり制御できるようになります。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

最初は、ワーキングメモリーは、アジェンダグループ **MAIN** にフォーカスを当てます。アジェンダグループのルールは、グループがこのフォーカスを受けた場合のみ実行されます。**setFocus()** メソッドか、**auto-focus** ルール属性を使用してフォーカスを設定できます。**auto-focus** 属性を使用すると、

ルールが一致してアクティベートされた場合のみ、ルールにアジェンダグループのフォーカスが自動的に当てられます。

ペットショップの例では、ルールに以下のアジェンダグループを使用します。

- **"init"**
- **"evaluate"**
- **"show items"**
- **"checkout"**

たとえば、同じルール **"Explode Cart"** は **"init"** のアジェンダグループを使用して、ショッピングカードのアイテムを起動して KIE セッションのワーキングメモリーに挿入するオプションが提供されるようにします。

ルール "Explode Cart"

```
// Insert each item in the shopping cart into the working memory.
rule "Explode Cart"
    agenda-group "init"
    auto-focus true
    salience 10
  when
    $order : Order( grossTotal == -1 )
    $item : Purchase() from $order.items
  then
    insert( $item );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "show items" ).setFocus();
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "evaluate" ).setFocus();
end
```

このルールは、**grossTotal** がまだ計算されていない全注文に対して照合が行われます。購入アイテムごとに、順番に実行がループされます。

ルールは、アジェンダグループに関連する以下の機能を使用します。

- **agenda-group "init"** はアジェンダグループの名前を定義します。この例では、グループにはルールが 1 つしかありません。ただし、Java コードもルール結果もこのグループにフォーカスされていないため、**auto-focus** の属性により、ルールが実行されるかが決まります。
- このルールはアジェンダグループで唯一のルールですが、**auto-focus true** を使用して、**fireAllRules()** が Java コードから呼び出されると、必ず実行されるようにします。
- **kcontext….setFocus()** は、**"show items"** および **"evaluate"** アジェンダグループにフォーカスを設定し、ルールを実行できるようにします。実際には、すべての項目をその順序でループしてメモリーに挿入し、各挿入の後に他のルールを実行します。

"show items" アジェンダグループには **"Show Items"** というルールだけが含まれます。KIE セッションのワーキングメモリーに現在含まれる注文で購入があるたびに、このルールを使用して、ルールファイルに定義した **textArea** 変数をもとに、GUI の下の部分にあるテキストエリアに詳細がロギングされます。

ルール "Show Items"

```
rule "Show Items"
    agenda-group "show items"
  when
    $order : Order()
    $p : Purchase( order == $order )
  then
   textArea.append( $p.product + "\n");
end
```

また、**"evaluate"** アジェンダグループにより、**"Explode Cart"** ルールからフォーカスを取得します。このアジェンダグループには、2 つのルール (**"Free Fish Food Sample"** と **"Suggest Tank"**) が含まれます。この順番で実行されます。

ルール "Free Fish Food Sample"

```
// Free fish food sample when users buy a goldfish if they did not already buy
// fish food and do not already have a fish food sample.
rule "Free Fish Food Sample"
    agenda-group "evaluate" (1)
  when
    $order : Order()
    not ( $p : Product( name == "Fish Food") && Purchase( product == $p ) ) (2)
    not ( $p : Product( name == "Fish Food Sample") && Purchase( product == $p ) ) (3)
    exists ( $p : Product( name == "Gold Fish") && Purchase( product == $p ) ) (4)
    $fishFoodSample : Product( name == "Fish Food Sample" );
  then
    System.out.println( "Adding free Fish Food Sample to cart" );
    purchase = new Purchase($order, $fishFoodSample);
    insert( purchase );
    $order.addItem( purchase );
end
```

ルール **"Free Fish Food Sample"** は、以下の条件がすべて該当する場合のみ実行されます。

1. アジェンダグループ **"evaluate"** がルール実行で評価されている
2. ユーザーが魚の餌をまだ持っていない
3. ユーザーが無料の魚の餌サンプルをまだ持っていない
4. ユーザーが金魚を注文している

この注文ファクトが上記の要件すべてを満たす場合は、新しい商品 (Fish Food Sample) が作成され、ワーキングメモリーの注文に追加されます。

ルール "Suggest Tank"

```
// Suggest a fish tank if users buy more than five goldfish and
// do not already have a tank.
rule "Suggest Tank"
    agenda-group "evaluate"
  when
    $order : Order()
    not ( $p : Product( name == "Fish Tank") && Purchase( product == $p ) ) (1)
```

```
    ArrayList( $total : size > 5 ) from collect( Purchase( product.name == "Gold Fish" ) ) (2)
    $fishTank : Product( name == "Fish Tank" )
  then
    requireTank(frame, kcontext.getKieRuntime(), $order, $fishTank, $total);
end
```

ルール **"Suggest Tank"** は以下の条件がすべて該当する場合のみ実行されます。

1. ユーザーが水槽を注文していない
2. ユーザーが 6 匹以上注文した

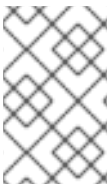
このルールが実行すると、ルールファイルに定義されている **requireTank()** 関数が呼び出されます。この関数により、水槽を購入するかどうかを尋ねるダイアログが表示されます。購入する場合は、新しい水槽の **Product** がワーキングメモリーの注文リストに追加されます。ルールが **requireTank()** 関数を呼び出した場合は、このルールを使用して、関数に Swing GUI のハンドルが含まれるように、**frame** のグローバル変数を渡します。

ペットショップの例の **"do checkout"** ルールにはアジェンダグループや **when** 条件がないため、ルールは常に実行され、デフォルトの **MAIN** のアジェンダグループの一部とみなされます。

ルール "do checkout"

```
rule "do checkout"
  when
  then
    doCheckout(frame, kcontext.getKieRuntime());
end
```

このルールが実行されると、ルールファイルで定義されている **doCheckout()** 関数を呼び出します。この関数により、チェックアウトするかどうかをユーザーに尋ねるダイアログボックスが表示されます。チェックアウトする場合は、フォーカスが **checkout** アジェンダグループに設定され、そのグループのルールを (今後) 実行できるようにします。このルールで **doCheckout()** 関数を呼び出し、この変数に Swing GUI のハンドルが含まれるように **frame** グローバル変数を渡します。



注記

この例では、結果が想定どおりに実行されない場合のトラブルシューティングの方法を例示します。ルールの **when** ステートメントから条件を削除して、**then** ステートメントのアクションをテストし、アクションが正しく実行されることを検証します。

"checkout" アジェンダグループには、注文のチェックアウト処理、および割引の適用の 3 つのルール (**"Gross Total"**、**"Apply 5% Discount"**、および **"Apply 10% Discount"**) が含まれています。

ルール "Gross Total"、"Apply 5% Discount"、および "Apply 10% Discount"

```
rule "Gross Total"
    agenda-group "checkout"
  when
    $order : Order( grossTotal == -1)
    Number( total : doubleValue ) from accumulate( Purchase( $price : product.price ),
                                                   sum( $price ) )
  then
    modify( $order ) { grossTotal = total }
```

```
    textArea.append( "\ngross total=" + total + "\n" );
end

rule "Apply 5% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 10 && < 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.95;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end

rule "Apply 10% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.90;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end
```

ユーザーがまだ総計を算出していない場合には、**Gross Total** で、商品の価格を累積して合計を出し、この合計を KIE セッションに渡して、**textArea** のグローバル変数を使用し、Swing **JTextArea** で合計を表示します。

総計が **10** から **20** (通貨単位) の場合は、**"Apply 5% Discount"** ルールで割引合計を計算し、KIE セッションに追加して、テキストエリアに表示します。

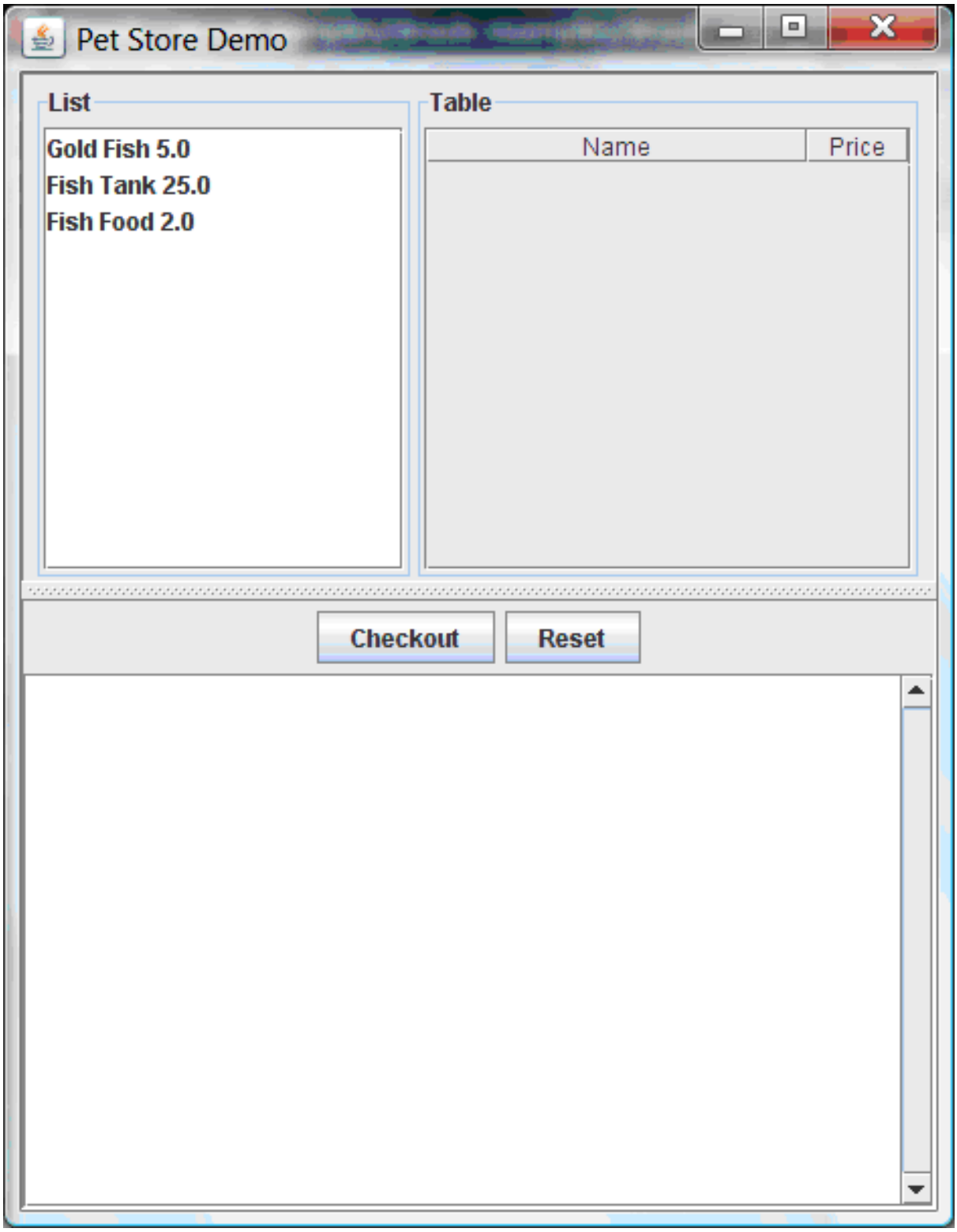
総計が **20** 未満の場合は、**"Apply 10% Discount"** ルールで割引合計を計算し、KIE セッションに追加して、テキストエリアに表示します。

ペットショップ例の実行

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.petstore.PetStoreExample** クラスを Java アプリケーションとして実行し、ペットショップの例を実行します。

ペットショップの例を実行すると、**Pet Store Demo** GUI ウィンドウが表示されます。このウィンドウでは、購入可能な商品 (左上)、選択済み商品の空白のリスト (右上)、**チェックアウト** および **リセット** ボタン (真ん中)、空白のシステムメッセージエリア (下) が表示されます。

図88.14 起動後のペットショップ例の GUI

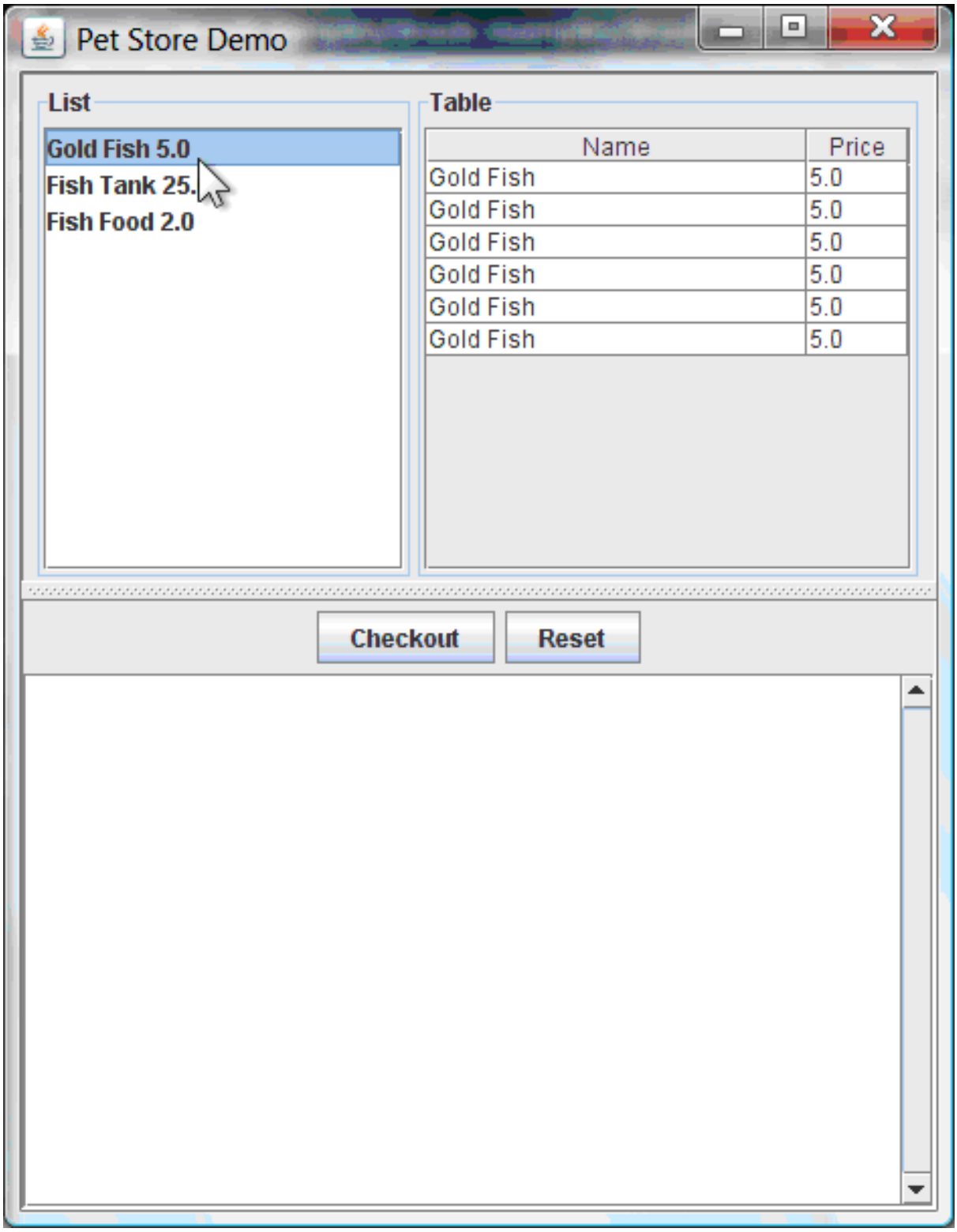


この例では、以下のイベントが発生して、この実行動作を確立します。

1. **main()** メソッドがルールベースの実行と読み込みを終えていますが、ルールは実行していない。今のところ、これが、実行されたルールに関連する唯一のコードになります。
2. 新しい **PetStoreUI** オブジェクトが作成され、後で使用できるようにルールベースにハンドルを渡している。
3. さまざまな Swing コンポーネントが関数を実行し、最初の UI 画面が表示され、ユーザーの入力を待っている。

リストからさまざまな商品をクリックして、UI 設定をチェックできます。

図88.15 ペットショップ例の GUI のチェック



ルールコードはまだ実行されていません。UI は Swing コードを使用してユーザーによるマウスクリックを検出し、選択済みの商品を **TableModel** オブジェクトに追加して、UI の右上隅に表示します。この例では、Model-View-Controller 設計パターンを紹介しています。

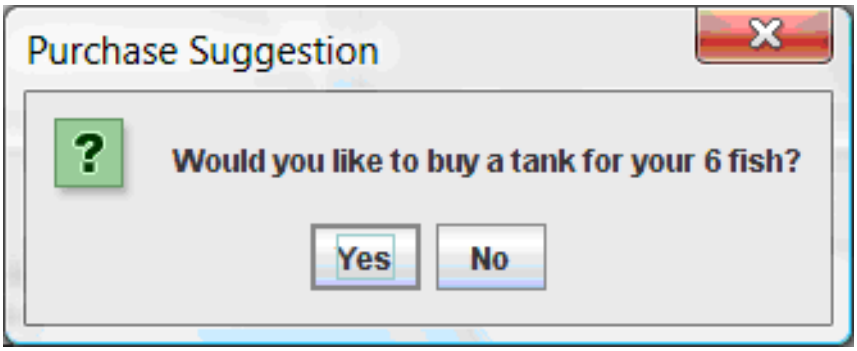
チェックアウト をクリックすると、ルールが以下の方法で実行されます。

1. Swing クラスは **Checkout** がクリックされるまで待機して、(最終的に) **CheckOutCallBack.checkout()** メソッドを呼び出します。これにより、**TableModel** オブジェ

クト (UI の右上隅) から KIE セッションのワーキングメモリーにデータを挿入します。その後に、メソッドによりルールが実行されます。

2. **"Explode Cart"** ルールは、**auto-focus** 属性を **true** に設定して最初に実行します。このルールは、カートの商品をすべて順にループしていき、商品がワーキングメモリーに含まれていることを確認し、アジェンダグループ **"show Items"** と **"evaluate"** に実行するオプションを提供します。このグループのルールは、カートのコンテンツをテキストエリア (UI の下) に追加して、魚の餌を無料で受け取る資格があるかどうかを評価し、また水槽購入の有無を尋ねるかどうかを決定します。

   図88.16 水槽の資格



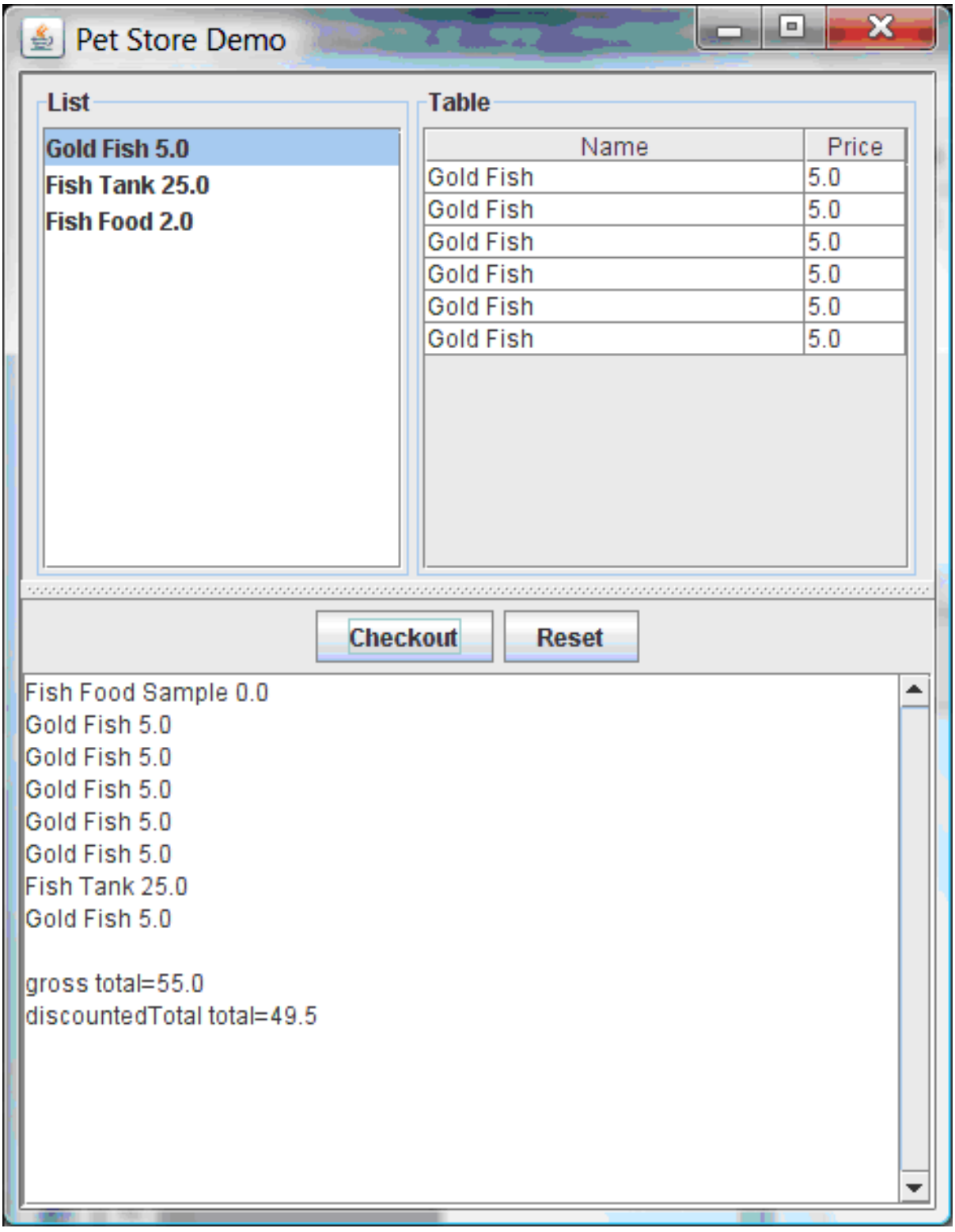
3. 現在、他のアジェンダグループにフォーカスが当たっておらず、**"do checkout"** ルールは、デフォルトの **MAIN** アジェンダグループに含まれているため、次に実行されます。このルールは常に **doCheckout()** 関数を呼び出し、この関数によりチェックアウトをするかどうかが尋ねられます。

4. **doCheckout()** 関数は、フォーカスを **"checkout"** アジェンダグループに設定し、そのグループ内のルールに、実行するオプションを提供します。

5. **"checkout"** アジェンダグループ内のルールは、カート内の内容を表示し、適切な割引を適用します。

6. Swing は、別の商品の選択 (およびもう一度ルールを実行) または GUI の終了のいずれかのユーザー入力を待ちます。

図88.17 全ルールが実行された後のペットショップ例の GUI



IDE コンソールでイベントのこのフローを例示するには、他の **System.out** 呼び出しを追加します。

IDE コンソールの System.out 出力

```
Adding free Fish Food Sample to cart
SUGGESTION: Would you like to buy a tank for your 6 fish? - Yes
```

88.7. 誠実な政治家の例のデシジョン (真理維持および顕著性)

誠実な政治家のデシジョンセットの例では、論理挿入を使用した真理維持の概念およびルールでの顕著性の使用方法を説明します。

以下は、誠実な政治家の例の概要です。

- **名前**: **honestpolitician**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.honestpolitician.HonestPoliticianExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.honestpolitician.HonestPolitician.drl**
- **目的**: ファクトの論理挿入をもとにした真理維持の概念およびルールでの顕著性の使用方法を紹介します。

誠実な政治家例の前提として基本的に、ステートメントが True の場合にのみ、オブジェクトが存在できます。**insertLogical()** メソッドを使用して、ルールの結果により、オブジェクトを論理的に挿入します。つまり、論理的に挿入されたルールが True の状態であれば、オブジェクトは KIE セッションのワーキングメモリー内に留まります。ルールが True でなくなると、オブジェクトは自動的に取り消されます。

この例では、ルールを実行することで、企業による政治家の買収が原因で、政治家グループが誠実から不誠実に変わります。各政治家が評価されるにつれ、最初は honesty 属性を **true** に設定して開始しますが、ルールが実行すると政治家は誠実ではなくなります。状態が誠実から不誠実に切り替わると、ワーキングメモリーから削除されます。ルールの顕著性により、顕著性が定義されているルールをどのように優先付けするかを、デシジョンエンジンに通知します。通知しないと、デフォルトの顕著性である **0** が使用されます。顕著性の値が高いルールは、アクティベーションキューの順番で、優先度が高くなります。

Politician クラスおよび Hope クラス

この例の **Politician** クラス例は、誠実な政治家として設定されています。**Politician** クラスは、文字列アイテム **name** とブール値アイテム **honest** で設定されています。

Politician クラス

```
public class Politician {
    private String name;
    private boolean honest;
    ...
}
```

Hope クラスは、**Hope** オブジェクトが存在するかどうかを判断します。このクラスには意味を持つメンバーは存在しませんが、社会に希望がある限り、ワーキングメモリーに存在します。

Hope クラス

```
public class Hope {

    public Hope() {
```

```
    }
  }
```

政治家の誠実性に関するルール定義
誠実な政治家の例では、ワーキングメモリーに最低でも 1 名誠実な政治家が存在する場合は、**"We have an honest Politician"** ルールで論理的に新しい **Hope** オブジェクトを挿入します。すべての政治家が不誠実になると、**Hope** オブジェクトは自動的に取り除かれます。このルールでは、**salience** 属性の値が **10** となっており、他のルールより先に実行されます。理由は、この時点では **"Hope is Dead"** ルールが True となっているためです。

ルール "We have an honest politician"

```
rule "We have an honest Politician"
    salience 10
  when
    exists( Politician( honest == true ) )
  then
    insertLogical( new Hope() );
end
```

Hope オブジェクトが存在すると、すぐに **"Hope Lives"** ルールが一致して実行されます。**"Corrupt the Honest"** ルールよりも優先されるように、このルールにも **salience** 値を **10** に指定しています。

ルール "Hope Lives"

```
rule "Hope Lives"
    salience 10
  when
    exists( Hope() )
  then
    System.out.println("Hurrah!!! Democracy Lives");
end
```

最初は、誠実な政治家が 4 人いるため、このルールには 4 つのアクティベーションが存在し、すべてが競合しています。各ルールが順番に実行し、政治家が誠実でなくなるように、企業により各政治家を買収させていきます。政治家 4 人が全員買収されたら、プロパティーが **honest == true** の政治家はいなくなります。**"We have an honest Politician"** のルールは True でなくなり、論理的に挿入されるオブジェクト (最後に実行された **new Hope()** による) は自動的に取り除かれます。

ルール "Corrupt the Honest"

```
rule "Corrupt the Honest"
  when
    politician : Politician( honest == true )
    exists( Hope() )
  then
    System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );
    modify ( politician ) { honest = false };
end
```

真理維持システムにより **Hope** オブジェクトが自動的に取り除かれると、**Hope** に適用された条件付き要素 **not** は True でなくなり、 **"Hope is Dead"** ルールが一致して実行されます。

ルール "Hope is Dead"

```
rule "Hope is Dead"
  when
    not( Hope() )
  then
    System.out.println( "We are all Doomed!!! Democracy is Dead" );
end
```

実行と監査証跡

HonestPoliticianExample.java クラスでは、honest の状態が **true** に設定されている政治家 4 人が挿入され、定義したビジネスルールに対して評価されます。

HonestPoliticianExample.java クラスの実行

```
public static void execute( KieContainer kc ) {
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        ksession.fireAllRules();

        ksession.dispose();
    }
```

この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.honestpolitician.HonestPoliticianExample** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの実行出力

```
Hurrah!!! Democracy Lives
I'm an evil corporation and I have corrupted President of Umpa Lumpa
I'm an evil corporation and I have corrupted Prime Minster of Cheeseland
I'm an evil corporation and I have corrupted Tsar of Pringapopaloo
I'm an evil corporation and I have corrupted Omnipotence Om
We are all Doomed!!! Democracy is Dead
```

この出力では、democracy lives に誠実な政治家が最低でも 1 人いることが分かります。ただし、各政治家は企業に買収されているため、全政治家が不誠実になり、民主性がなくなります。

この例の実行フローをさらに理解するために、**HonestPoliticianExample.java** クラスを変更し、**DebugRuleRuntimeEventListener** リスナーと監査ロガーを追加して実行の詳細を表示することができます。

監査ロガーを含む HonestPoliticianExample.java クラス

```
package org.drools.examples.honestpolitician;

import org.kie.api.KieServices;
import org.kie.api.event.rule.DebugAgendaEventListener; (1)
import org.kie.api.event.rule.DebugRuleRuntimeEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class HonestPoliticianExample {

    /**
     * @param args
     */
    public static void main(final String[] args) {
     KieServices ks = KieServices.Factory.get(); (2)
     //ks = KieServices.Factory.get();
        KieContainer kc = KieServices.Factory.get().getKieClasspathContainer();
        System.out.println(kc.verify().getMessages().toString());
        //execute( kc );
        execute( ks, kc); (3)
    }

    public static void execute( KieServices ks, KieContainer kc ) { (4)
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        // The application can also setup listeners (5)
        ksession.addEventListener( new DebugAgendaEventListener() );
        ksession.addEventListener( new DebugRuleRuntimeEventListener() );

        // Set up a file-based audit logger.
        ks.getLoggers().newFileLogger( ksession, "./target/honestpolitician" ); (6)

        ksession.fireAllRules();

        ksession.dispose();
    }

}
```

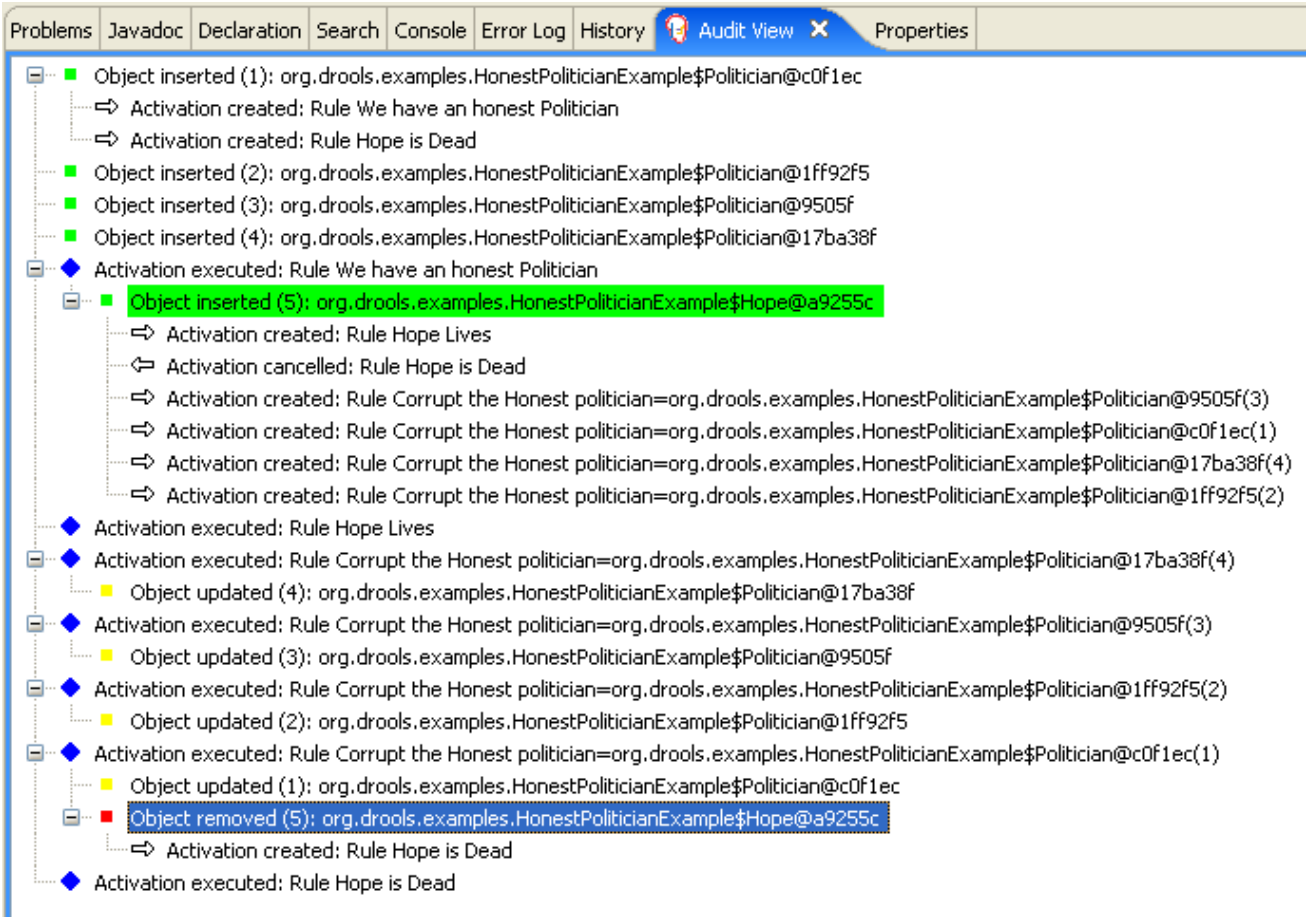
(1) **DebugAgendaEventListener** と **DebugRuleRuntimeEventListener** を処理するインポートにパッケージを追加します。

2. この監査ログは **KieContainer** レベルでは利用できないため、**KieServices Factory** 要素および **ks** 要素を作成してログを生成します。
3. **execute** メソッドを変更して **KieServices** と **KieContainer** の両方を使用します。
4. **execute** メソッドを変更して **KieContainer** に加えて **KieServices** で渡します。
5. リスナーを作成します。
6. ルールの実行後にデバッグビュー、**監査ビュー**、または IDE に渡すことが可能なログを構築します。

ログ機能を変更して誠実な政治家のサンプルを実行すると、**target/honestpolitician.log** から IDE デバッグビュー、または利用可能な場合には **監査ビュー** (IDE の一部では **Window → Show View**) に、監査ログファイルを読み込むことができます。

この例では、**監査ビュー** では、クラスやルールのサンプルで定義されているように、実行、挿入、取り消しのフローが示されています。

図88.18 誠実な政治家の例での監査ビュー



最初の政治家が挿入されると、2 つのアクティベーションが発生します。**"We have an honest Politician"** のルールは、**exists** の条件付き要素を使用するため、最初に挿入された政治家に対してのみ一度だけアクティベートされます。この条件付き要素は、政治家が最低でも 1 人挿入されると一致します。**Hope** オブジェクトがまだ挿入されていないため、ルール **"Hope is Dead"** もこの時点でアクティベートになります。**"We have an honest Politician"** ルールは、**"Hope is Dead"** ルールより、**salience** の値が高いため先に実行され、**Hope** オブジェクト (緑にハイライト) を挿入します。**Hope** オブジェクトを挿入すると、ルール **"Hope Lives"** が有効になり、ルール **"Hope is Dead"**

が無効になります。この挿入により、挿入された誠実な各政治家に対して **"Corrupt the Honest"** ルールがアクティベートになります。**"Hope Lives"** のルールが実行して、**"Hurrah!!!Democracy Lives"** が出力されます。

次に、政治家ごとに **"Corrupt the Honest"** ルールを実行して **"I'm an evil corporation and I have corrupted X"** と出力します。**X** は政治家の名前で、その政治家の誠実値が **false** に変更になります。最後の誠実な政治家が買収されると、真理維持システム (青でハイライト) により **Hope** が自動的に取り消されます。緑でハイライトされたエリアは、現在選択されている青のハイライトエリアの出元です。**Hope** ファクトが取り消されると、**"Hope is dead"** ルールが実行して **"We are all Doomed!!!Democracy is Dead"** が出力されます。

88.8. 数独例のデシジョン (複雑なパターン一致、コールバック、および GUI 統合)

人気の数字パズルに基づく数独の例の決定セットは、Red Hat Process Automation Manager のルールを使用して、さまざまな制約に基づいて大きな潜在的なソリューションを空間でソリューションを見つける方法を示しています。この例では、Red Hat Process Automation Manager ルールをグラフィカルユーザーインターフェイス (GUI) (この場合は Swing ベースのデスクトップアプリケーション) に統合する方法と、コールバックを使用して実行中の意思決定エンジンと対話し、ランタイム時に加えられた作業メモリー内の変更をもとに GUI を更新する方法を例示しています。

以下は数独の例の概要です。

- **名前**: **sudoku**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.sudoku.SudokuExample**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.sudoku.*.drl**
- **目的**: 複雑なパターン一致、問題解決、コールバック、および GUI 統合を例示します。

数独は、ロジックベースの数字配置パズルです。目的は、各列、各行、および各 3x3 ゾーンに 1 から 9 の数字が一度だけ含まれるように 9x9 のグリッドを埋めることです。パズルセッターでは、グリッド内の一部だけ記入されており、上記の制約ですべての空白を埋めるのがパズルの回答者のタスクです。

問題解決の一般的なストラテジーとして、新しい番号の挿入時に、特定の 3x3 ゾーン、行、および列で同じ番号がないことを確認します。この数独例のデシジョンセットでは、Red Hat Process Automation Manager ルールを使用して、さまざまな難易度の数独パズルを解き、無効なエントリーが含まれ、不備のあるパズルの解決を試みます。

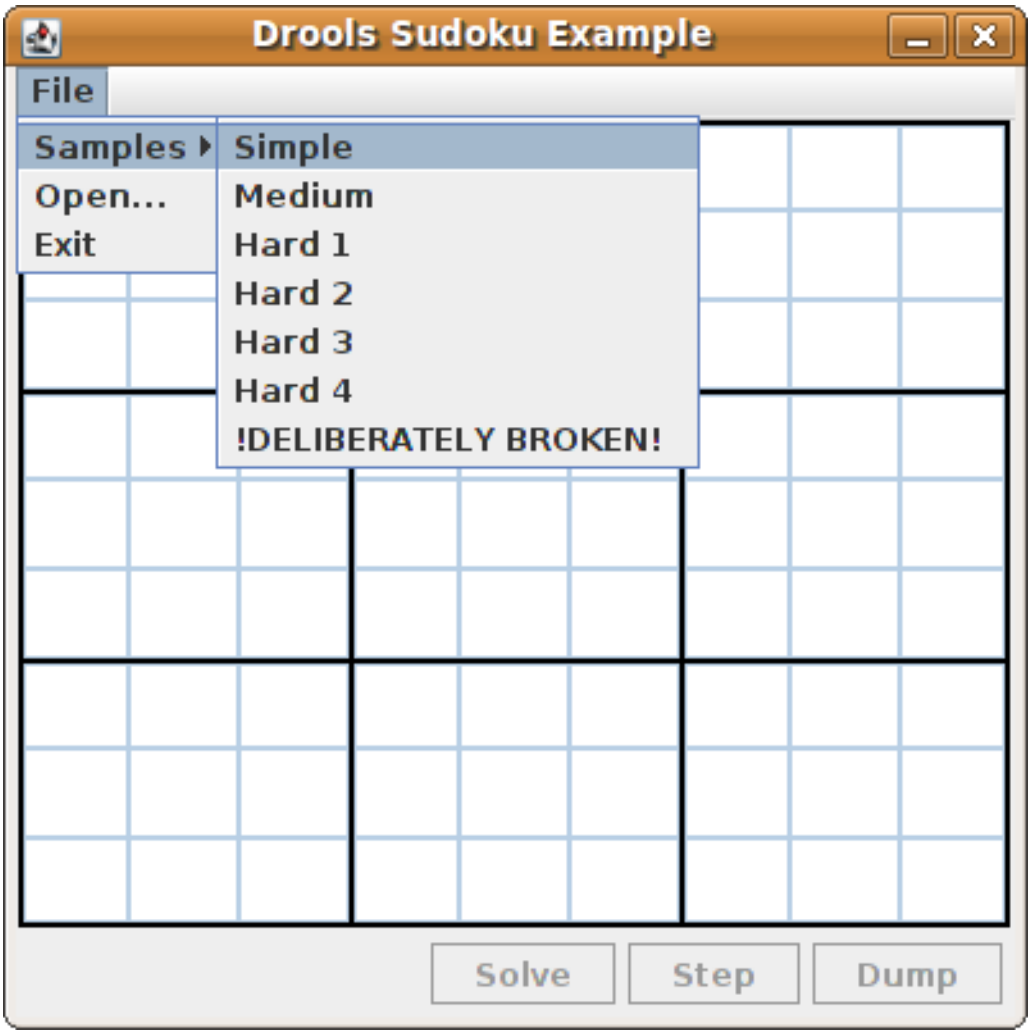
数独例の実行および対話

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.sudoku.SudokuExample** クラスを Java アプリケーションとして実行し、数独の例を実行します。

数独の例を実行すると、GUI ウィンドウ **Drools Sudoku Example** が表示されます。このウィンドウには空のグリッドが含まれていますが、プログラムには内部に保存されたさまざまなグリッドが含まれ、読み込んで解決できます。

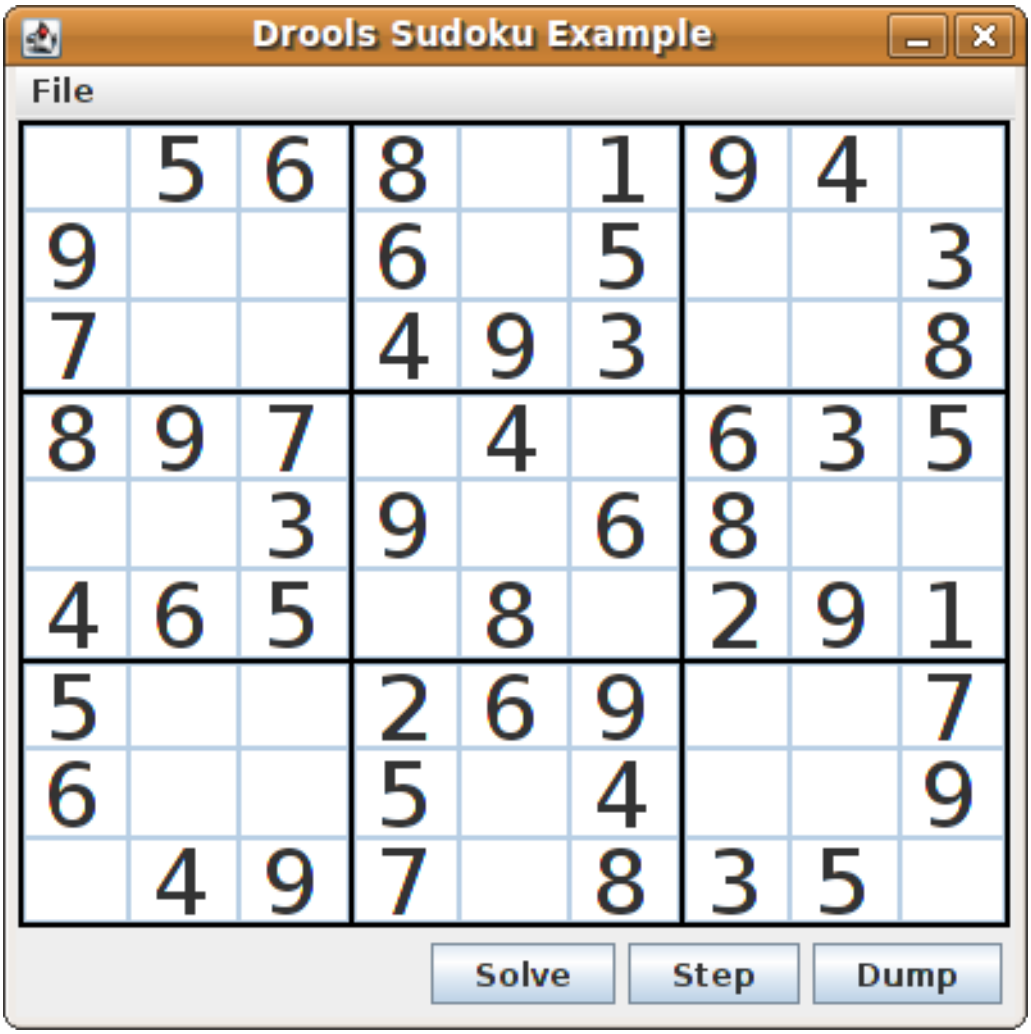
File → Samples → Simple をクリックして、例の 1 つを読み込みます。グリッドが読み込まれるまで、すべてのボタンが無効になっている点に注目してください。

図88.19 起動後の数独例の GUI



Simple サンプルを読み込むと、パズルの最初の状態に合わせて、グリッドが埋められます。

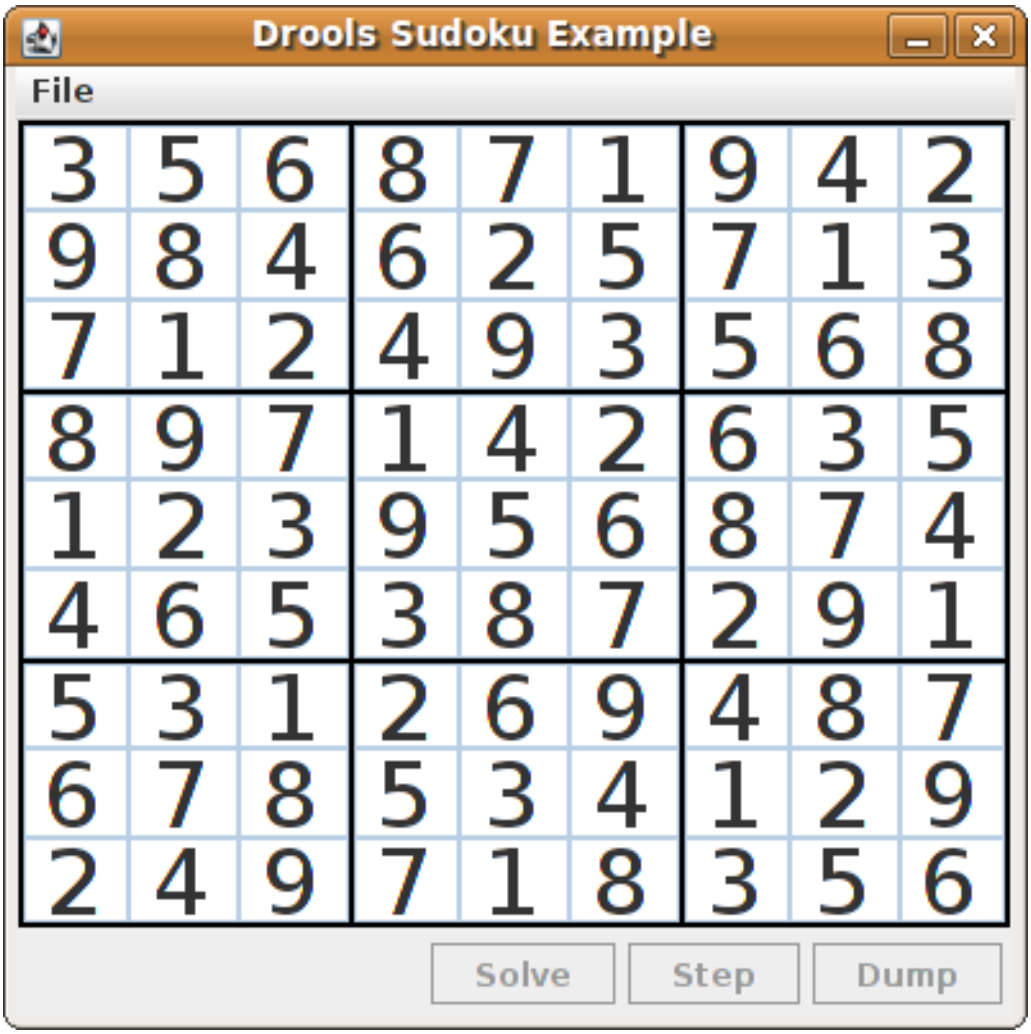
図88.20 Simple サンプルを読み込んだ後の数独例の GUI



以下のオプションから選択します。

- Solve をクリックして、数独の例に定義されているルールを実行し、残りの値を埋めていき、このボタンを再度無効にします。

図88.21 Simple サンプルの解決



- **Step** をクリックして、ルールセットに含まれる次の数字を表示します。IDE のコンソールウィンドウでは、解決手順を実行するルールに関する情報が詳細に表示されます。

IDE コンソールでの手順実行の出力

```
single 8 at [0,1]
column elimination due to [1,2]: remove 9 from [4,2]
hidden single 9 at [1,2]
row elimination due to [2,8]: remove 7 from [2,4]
remove 6 from [3,8] due to naked pair at [3,2] and [3,7]
hidden pair in row at [4,6] and [4,4]
```

- **Dump** をクリックしてグリッドの状態を表示します。セルには、解決済みの値か、残りの候補値が表示されます。

IDE コンソールでのダンプ実行の出力

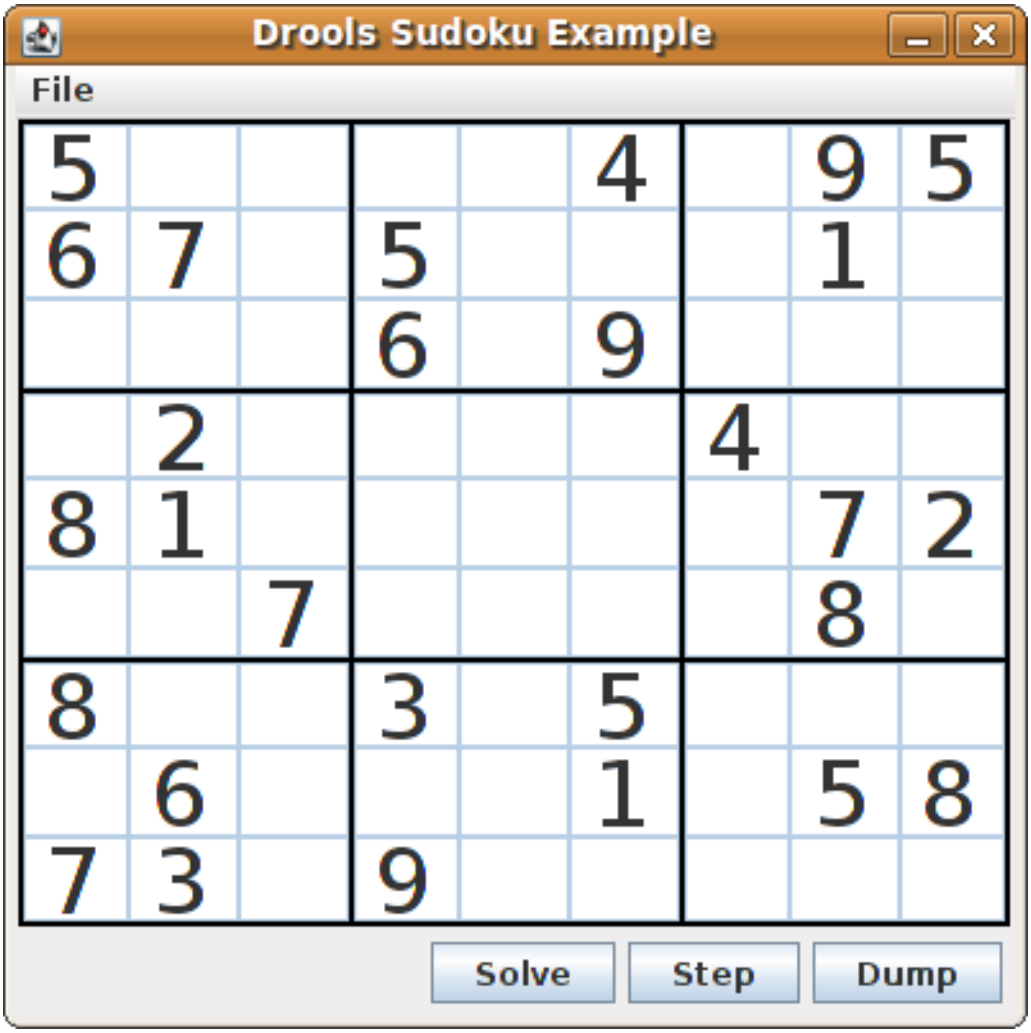
```
      Col: 0     Col: 1     Col: 2     Col: 3     Col: 4     Col: 5     Col: 6     Col: 7     Col: 8
Row 0:  123456789  --- 5 ---  --- 6 ---  --- 8 ---  123456789  --- 1 ---  --- 9 ---  --- 4 ---
123456789
Row 1:  --- 9 ---  123456789  123456789  --- 6 ---  123456789  --- 5 ---  123456789
123456789  --- 3 ---
Row 2:  --- 7 ---  123456789  123456789  --- 4 ---  --- 9 ---  --- 3 ---  123456789  123456789
--- 8 ---
```

```
Row 3:  --- 8 ---  --- 9 ---  --- 7 ---  123456789  --- 4 ---  123456789  --- 6 ---  --- 3 ---  --- 5 ---
Row 4:  123456789  123456789  --- 3 ---  --- 9 ---  123456789  --- 6 ---  --- 8 ---  123456789
123456789
Row 5:  --- 4 ---  --- 6 ---  --- 5 ---  123456789  --- 8 ---  123456789  --- 2 ---  --- 9 ---  --- 1 ---
Row 6:  --- 5 ---  123456789  123456789  --- 2 ---  --- 6 ---  --- 9 ---  123456789  123456789
--- 7 ---
Row 7:  --- 6 ---  123456789  123456789  --- 5 ---  123456789  --- 4 ---  123456789
123456789  --- 9 ---
Row 8:  123456789  --- 4 ---  --- 9 ---  --- 7 ---  123456789  --- 8 ---  --- 3 ---  --- 5 ---
123456789
```

数独の例には、不備のあるサンプルファイルが意図的に含められています。このファイルは、例で定義したルールを使用して解決できます。

File → Samples → !DELIBERATELY BROKEN! をクリックして、不備のあるサンプルを読み込みます。グリッドは、最初の行に **5** の値を 2 回表示できないにもかかわらず表示されるなど、問題が含まれた状態で表示されます。

図88.22 不備のある数独例の最初の状態



Solve をクリックしてこの無効なグリッドに解決ルールを適用します。数独の例に含まれる関連の解決ルールにより、サンプルの問題が検出され、できる限りパズルを解決します。このプロセスでは、すべてを完了させず、空白のセルをいくつか残します。

解決ルールのアクティビティーが IDE コンソールウィンドウに表示されます。

不備のあるサンプルでの問題検出

```
cell [0,8]: 5 has a duplicate in row 0
cell [0,0]: 5 has a duplicate in row 0
cell [6,0]: 8 has a duplicate in col 0
cell [4,0]: 8 has a duplicate in col 0
Validation complete.
```

図88.23 不備のあるサンプルの解決試行

Hard のラベルの付いた数独サンプルファイルはより複雑で、解決ルールを使用しても解決できない可能性があります。解決をしようとして失敗した場合は、IDE コンソールウィンドウに表示されます。

解決不可の Hard サンプル

```
Validation complete.
...
Sorry - can't solve this grid.
```

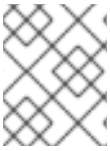
不備のあるサンプルを解決するためのルールでは、セルの候補となりえる値をもとにした標準の解決手法を実装します。たとえば、セットに値が 1 つ含まれる場合は、これが値になります。セルが 9 個あるグループの 1 つに値が 1 度挿入された場合に、ルールを使用して、特定のセルに対する値を持ち、タイプが **Setting** のファクトを挿入します。このファクトにより、そのセルが含まれるグループにある他のすべてのセルからこの値が削除され、この値が取り消されます。

この例の他のルールで、セルに入力可能な値を減らしていきます。**"naked pair"**、**"hidden pair in row"**、**"hidden pair in column"**、および **"hidden pair in square"** のルールでは、候補の絞り込みはできますが、回答を得ることはできません。**"X-wings in rows"**、"`X-wings in columns"`、**"intersection**

removal row"、および **"intersection removal column"** のルールは、より高度な絞り込みを実行します。

数独例のクラス
org.drools.examples.sudoku.swing パッケージには、以下のように、数独パズルのフレームワークを実装する主なクラスセットが含まれます。

- **SudokuGridModel** は、9x9 グリッドの **Cell** オブジェクトとして数独パズルを格納するために実装可能なインターフェイスを定義しています。
- **SudokuGridView** クラスは Swing コンポーネントで、**SudokuGridModel** クラス実装の視覚化が可能です。
- **SudokuGridEvent** クラスおよび **SudokuGridListener** クラスは、モデルとビューの間のステータスの変化をやり取りするために使用します。セルの値が解決または変更すると、イベントが実行します。
- **SudokuGridSamples** クラスは、デモ目的に一部入力されている数独パズルを複数提供します。



注記

このパッケージには、Red Hat Process Automation Manager ライブラリーの依存関係は含まれません。

org.drools.examples.sudoku パッケージには、以下のように、基本的な **Cell** オブジェクトと各種アグリゲーションを実装する主なクラスセットが含まれます。

- **CellRow**、**CellCol**、および **CellSqr** のサブタイプを含む **CellFile** クラス。これはすべて、**CellGroup** クラスのサブタイプになります。
- **Cell** と **CellGroup** は **SetOfNine** のサブクラスで、**Set<Integer>** 型の **free** プロパティーを提供します。**Cell** クラスは、個別の候補セットを表します。**CellGroup** は、セルの全候補セットの統合 (割り当ての必要がある数値セット) です。
  数独の例には、81 個の **Cell** と 27 個の **CellGroup** オブジェクト、**Cell** プロパティーの **cellRow**、**cellCol**、および **cellSqr** が提供するリンク、**CellGroup** プロパティー **cells** (**Cell** オブジェクトリスト) が提供するリストが含まれます。これらのコンポーネントを使用して、セルに値を割り当てたり、候補セットから値を取り除いたりできるように、特定の状態を検出するルールを記述できます。
- **Setting** クラスを使用して、値の割り当てに伴うオペレーションをトリガーします。**Setting** ファクトは、整合性の取れない中間の状態に対して反応しないように、新しい状況を検出する全ルールに配置して使用します。
- **Stepping** クラスは、優先順位が低いルールに使用して、**"Step"** が予期なく中断された場合に緊急停止を行います。この動作は、プログラムでパズルを解決できないということです。
- Main クラス **org.drools.examples.sudoku.SudokuExample** は、全コンポーネントを統合する Java アプリケーションを実装します。

数独の検証ルール (validate.drl)
数独の例の **validate.drl** ファイルには、セルグループで数が重複している状況を検出する検証ルールが含まれます。このグループは、**"validate"** アジェンダグループに統合され、ユーザーがパズルを読み込むと、明示的にルールをアクティベートできます。

"duplicate in cell ..." の 3 つのルールの **when** 条件はすべて以下の方法で機能します。

- このルールの最初の条件で、割り当てられた値でセルを特定します。
- このルールの 2 番目の条件では、3 つのセルグループのどれかを所属先にプルします。
- 最終条件は、ルールに従い、最初のセル、同じ行、列、または四角に入る値と同じセル (上記のセル以外) を検索します。

ルール "duplicate in cell ..."

```
rule "duplicate in cell row"
  when
    $c: Cell( $v: value != null )
    $cr: CellRow( cells contains $c )
    exists Cell( this != $c, value == $v, cellRow == $cr )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in row " + $cr.getNumber() );
end

rule "duplicate in cell col"
  when
    $c: Cell( $v: value != null )
    $cc: CellCol( cells contains $c )
    exists Cell( this != $c, value == $v, cellCol == $cc )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in col " + $cc.getNumber() );
end

rule "duplicate in cell sqr"
  when
    $c: Cell( $v: value != null )
    $cs: CellSqr( cells contains $c )
    exists Cell( this != $c, value == $v, cellSqr == $cs )
  then
    System.out.println( "cell " + $c.toString() + " has duplicate in its square of nine." );
end
```

ルール **"terminate group"** は最後に実行されます。このルールは、メッセージを出力して、シーケンスを停止します。

ルール "terminate group"

```
rule "terminate group"
    salience -100
  when
  then
    System.out.println( "Validation complete." );
    drools.halt();
end
```

数独の解決ルール (sudoku.drl)
数独の例の **sudoku.drl** ファイルには、3 種類のルールタイプが含まれます。1 つ目のグループは、セルへの数値の割り当てを処理して、2 つ目は実行可能な割り当てを検出して、3 つ目は候補セットからの値を削除します。

"set a value"、**"eliminate a value from Cell"**、および **"retract setting"** のルールは、**Setting** オブジェ

クトの有無により左右されます。最初のルールは、セルへの割り当てと、3 つのセルグループの **free** セットから値を削除する操作を処理します。また、ゼロの場合は、このグループでカウンターが 1 つ減り、**fireUntilHalt()** を呼び出した Java アプリケーションに制御を戻します。

"eliminate a value from Cell" ルールの目的は、新たに割り当てられたセルに関連する全セルの候補リストを絞り込むことです。最後に、すべての除外が完了したら、**"retract setting"** ルールにより、トリガーされている **Setting** ファクトを取り消します。

ルール "set a value"、"eliminate a value from a Cell"、および "retract setting"

```
// A Setting object is inserted to define the value of a Cell.
// Rule for updating the cell and all cell groups that contain it
rule "set a value"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // A matching Cell, with no value set
    $c: Cell( rowNo == $rn, colNo == $cn, value == null,
              $cr: cellRow, $cc: cellCol, $cs: cellSqr )

    // Count down
    $ctr: Counter( $count: count )
  then
    // Modify the Cell by setting its value.
    modify( $c ){ setValue( $v ) }
    // System.out.println( "set cell " + $c.toString() );
    modify( $cr ){ blockValue( $v ) }
    modify( $cc ){ blockValue( $v ) }
    modify( $cs ){ blockValue( $v ) }
    modify( $ctr ){ setCount( $count - 1 ) }
end

// Rule for removing a value from all cells that are siblings
// in one of the three cell groups
rule "eliminate a value from Cell"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    Cell( rowNo == $rn, colNo == $cn, value == $v, $exCells: exCells )

    // For all Cells that are associated with the updated cell
    $c: Cell( free contains $v ) from $exCells
  then
    // System.out.println( "clear " + $v + " from cell " + $c.posAsString()  );
    // Modify a related Cell by blocking the assigned value.
    modify( $c ){ blockValue( $v ) }
end

// Rule for eliminating the Setting fact
rule "retract setting"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )
```

```
    // The matching Cell, with the value already set
    $c: Cell( rowNo == $rn, colNo == $cn, value == $v )

    // This is the negation of the last pattern in the previous rule.
    // Now the Setting fact can be safely retracted.
    not( $x: Cell( free contains $v )
         and
         Cell( this == $c, exCells contains $x ) )
  then
    // System.out.println( "done setting cell " + $c.toString() );
    // Discard the Setter fact.
    delete( $s );
    // Sudoku.sudoku.consistencyCheck();
end
```

解決ルールを 2 つ使用して、セルに数字を割り当てることができる状況を検出します。**"single"** のルールは、**Cell** に、数字が 1 つだけの候補セットが含まれる場合に実行します。**"hidden single"** ルールは、候補が 1 つだけのセルが存在しない場合に実行しますが、セルに候補が含まれる場合は、セルが所属する 3 つのグループの 1 つに含まれるその他のすべてのセルに、この候補が存在しないということです。いずれのルールも **Setting** ファクトを作成して、挿入します。

ルール "single" および "hidden single"

```
// Detect a set of candidate values with cardinality 1 for some Cell.
// This is the value to be set.
rule "single"
  when
    // Currently no setting underway
    not Setting()

    // One element in the "free" set
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount == 1 )
  then
    Integer i = $c.getFreeValue();
    if (explain) System.out.println( "single " + i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, i ) );
end

// Detect a set of candidate values with a value that is the only one
// in one of its groups. This is the value to be set.
rule "hidden single"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Some integer
    $i: Integer()

    // The "free" set contains this number
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount > 1, free contains $i )

    // A cell group contains this cell $c.
    $cg: CellGroup( cells contains $c )
```

```
    // No other cell from that group contains $i.
    not ( Cell( this != $c, free contains $i ) from $cg.getCells() )
  then
    if (explain) System.out.println( "hidden single " + $i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, $i ) );
end
```

最大グループからのルール (個別または 2 ~ 3 のグループ単位) は、数独パズルを手作業で解決するのに使用する、さまざまな解決手法を実装します。

"naked pair" ルールは、グループの 2 つのセルで、全く同じ候補セットでサイズ **2** のものを検出します。これらの 2 つの値は、対象グループにあるその他のすべての候補セットから削除できます。

ルール "naked pair"

```
// A "naked pair" is two cells in some cell group with their sets of
// permissible values being equal with cardinality 2. These two values
// can be removed from all other candidate lists in the group.
rule "naked pair"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // One cell with two candidates
    $c1: Cell( freeCount == 2, $f1: free, $r1: cellRow, $rn1: rowNo, $cn1: colNo, $b1: cellSqr )

    // The containing cell group
    $cg: CellGroup( freeCount > 2, cells contains $c1 )

    // Another cell with two candidates, not the one we already have
    $c2: Cell( this != $c1, free == $f1 /*** , rowNo >= $rn1, colNo >= $cn1 ***/ ) from $cg.cells

    // Get one of the "naked pair".
    Integer( $v: intValue ) from $c1.getFree()

    // Get some other cell with a candidate equal to one from the pair.
    $c3: Cell( this != $c1 && != $c2, freeCount > 1, free contains $v ) from $cg.cells
  then
    if (explain) System.out.println( "remove " + $v + " from " + $c3.posAsString() + " due to naked pair
at " + $c1.posAsString() + " and " + $c2.posAsString() );
    // Remove the value.
    modify( $c3 ){ blockValue( $v ) }
end
```

3 つのルールの **"hidden pair in ..."** 関数は、ルール **"naked pair"** と同じように機能します。ルールはグループの 2 つのセルで 2 つの数字を検出します。どの値もこのグループの他のセルには入りません。つまり、他の候補はすべて、隠れたペアを持つ 2 つのセルから削除します。

ルール "hidden pair in ..."

```
// If two cells within the same cell group contain candidate sets with more than
// two values, with two values being in both of them but in none of the other
// cells, then we have a "hidden pair". We can remove all other candidates from
```

```
// these two cells.
rule "hidden pair in row"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Establish a pair of Integer facts.
    $i1: Integer()
    $i2: Integer( this > $i1 )

    // Look for a Cell with these two among its candidates. (The upper bound on
    // the number of candidates avoids a lot of useless work during startup.)
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellRow: cellRow )

    // Get another one from the same row, with the same pair among its candidates.
    $c2: Cell( this != $c1, cellRow == $cellRow, freeCount > 2, free contains $i1 && contains $i2 )

    // Ascertain that no other cell in the group has one of these two values.
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellRow.getCells() )
  then
    if( explain) System.out.println( "hidden pair in row at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    // Set the candidate lists of these two Cells to the "hidden pair".
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellCol: cellCol )
    $c2: Cell( this != $c1, cellCol == $cellCol, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellCol.getCells() )
  then
    if (explain) System.out.println( "hidden pair in column at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in square"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
            $cellSqr: cellSqr )
```

```
    $c2: Cell( this != $c1, cellSqr == $cellSqr, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellSqr.getCells() )
  then
    if (explain) System.out.println( "hidden pair in square " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end
```

2 つのルールは行と列で **"X-wings"** を処理します。2 つの異なる行 (または列) で、ある値を入力できるセルが 2 つしかなく、これらの候補が同じ列 (または行) に入る場合に、この列 (または行) のこの値に対する他の候補は除外できます。これらのルールの 1 つに含まれるパターンシーケンスに従うと、**same**、**only** などの用語で都合よく表現されている条件は、適切な制約が付けられたパターンになるか、**not** のプリフィックスが付きます。

ルール "X-wings in ..."

```
rule "X-wings in rows"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
                $ra: cellRow, $rano: rowNo,         $c1: cellCol,        $c1no: colNo )
    $cb1: Cell( freeCount > 1, free contains $i,
                $rb: cellRow, $rbno: rowNo > $rano,      cellCol == $c1 )
    not( Cell( this != $ca1 && != $cb1, free contains $i ) from $c1.getCells() )

    $ca2: Cell( freeCount > 1, free contains $i,
                cellRow == $ra, $c2: cellCol,       $c2no: colNo > $c1no )
    $cb2: Cell( freeCount > 1, free contains $i,
                cellRow == $rb,      cellCol == $c2 )
    not( Cell( this != $ca2 && != $cb2, free contains $i ) from $c2.getCells() )

    $cx: Cell( rowNo == $rano || == $rbno, colNo != $c1no && != $c2no,
               freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in rows " +
          $ca1.posAsString() + " - " + $cb1.posAsString() +
          $ca2.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "X-wings in columns"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
                $c1: cellCol, $c1no: colNo,         $ra: cellRow,        $rano: rowNo )
    $ca2: Cell( freeCount > 1, free contains $i,
                $c2: cellCol, $c2no: colNo > $c1no,      cellRow == $ra )
```

```
    not( Cell( this != $ca1 && != $ca2, free contains $i ) from $ra.getCells() )

    $cb1: Cell( freeCount > 1, free contains $i,
                cellCol == $c1, $rb: cellRow,  $rbno: rowNo > $rano )
    $cb2: Cell( freeCount > 1, free contains $i,
                cellCol == $c2,       cellRow == $rb )
    not( Cell( this != $cb1 && != $cb2, free contains $i ) from $rb.getCells() )

    $cx: Cell( colNo == $c1no || == $c2no, rowNo != $rano && != $rbno,
               freeCount > 1, free contains $i )
  then
    if (explain) {
        System.out.println( "X-wing with " + $i + " in columns " +
            $ca1.posAsString() + " - " + $ca2.posAsString() +
            $cb1.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString()  );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

この 2 つのルール **intersection removal …**は、1 つの行または 1 つの列のいずれかで、1 つの四角の中で一部の数字が制限されたことに基づきます。これは、この番号が行または列の 2 つまたは 3 つのセルのいずれかにある必要があり、グループの他のすべてのセルの候補セットから削除できることを意味します。このパターンは、発生制限を確立して、同じセルファイルの中、かつ四角の外のセルそれぞれに対して実行されます。

ルール "intersection removal ..."

```
rule "intersection removal column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cc: cellCol )
    // Does not occur in another cell of the same square and a different column
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellCol != $cc )

    // A cell exists in the same column and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellCol == $cc, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "column elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "intersection removal row"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
```

```
    $c: Cell( free contains $i, $cs: cellSqr, $cr: cellRow )
    // Does not occur in another cell of the same square and a different row.
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellRow != $cr )

    // A cell exists in the same row and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellRow == $cr, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "row elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

これらのルールは、すべてではありませんが、多くの数独パズルでは十分です。非常に難度の高いグリッドを解決するには、ルールセットにさらに複雑なルールが必要です。(最終的には、パズルは試行錯誤でしか解決できません)。

88.9. CONWAY の GAME OF LIFE 例のデシジョン (ルールフローグループおよび GUI 統合)

John Conway による有名なセルオートマトン (CA: Cellular automation) をベースにした Conway の Game of Life 例のデシジョンセットは、ルールでルールフローグループを使用してルール実行を制御する方法を例示します。またこの例は、Red Hat Process Automation Manager ルールをグラフィカルユーザーインターフェイス (GUI) と統合する方法も例示しています。今回は、Conway の Game of Life を Swing ベースで実装しています。

以下は、Conway の Game of Life の例の概要です。

- **名前**: **conway**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.conway.ConwayRuleFlowGroupRun**、**org.drools.examples.conway.ConwayAgendaGroupRun**
- **モジュール**: **droolsjbpm-integration-examples**
- **タイプ**: Java アプリケーション
- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.conway.*.drl**
- **目的**: ルールフローグループと GUI 統合を例示します。

注記

Conway の Game of Life の例は、Red Hat Process Automation Manager に含まれる他のデシジョンセットの例の多くとは異なり、Red Hat カスタマーポータル から取得する **Red Hat Process Automation Manager 7.11.0 Source Distribution**の ~/**rhpam-7.11.0-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples** におかれています。

Conway の Game of Life では、初期設定または定義済みのプロパティーで高度なパターンを作成して、初期状態からどのように進化していくかを観察することで、ユーザーはゲームと対話します。ゲームの目的は、世代ごとに人口の成長を表示します。各世代は、すべてのセル (細胞) が同時に進化していき、

前の世代をもとにして生み出されます。

以下の基本的なルールで、次の世代がどのようになるかを制御していきます。

- 生きているセルの近傍に、生きているセルが 2 個未満の場合は、孤独で死んでしまう。
- 生きているセルの近傍に、生きているセルが 4 個以上ある場合は、過密で死んでしまう。
- 死亡したセルの近傍に、生きているセルがちょうど 3 つある場合には、このセルは生き返る。

この基準のいずれも満たさないセルは、そのまま次の世代に残ります。

Conway の Game of Life の例は、**ruleflow-group** 属性が含まれる Red Hat Process Automation Manager ルールで、ゲームに実装されているパターンを定義します。この例には、アジェンダグループを使用して同じ動作を行うデシジョンセットのバージョンも含まれています。アジェンダグループは、デシジョンエンジンアジェンダのパーティションを作成して、ルールのグループを実行制御できるようにします。デフォルトでは、ルールはすべてアジェンダグループ **MAIN** に含まれています。**agenda-group** 属性を使用して、ルールに異なるアジェンダグループを指定できます。

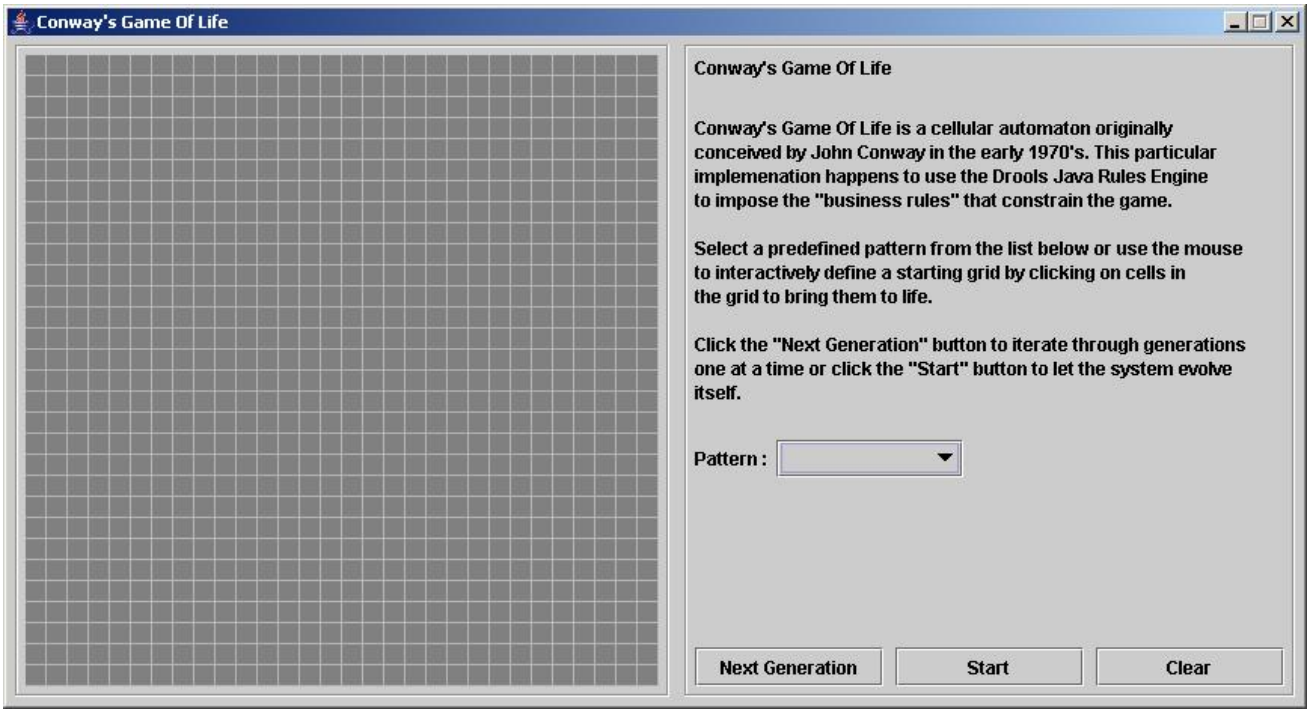
この概要では、Conway の例でアジェンダグループを使用したバージョンは触れません。アジェンダグループの詳細情報は、特にアジェンダグループについて対応している Red Hat Process Automation Manager 例のデシジョンセットを参照してください。

Conway 例の実行および対話

他の Red Hat Process Automation Manager のデシジョン例と同じように、お使いの IDE で **org.drools.examples.conway.ConwayRuleFlowGroupRun** クラスを Java アプリケーションとして実行し、Conway の例を実行します。

Conway の例を実行すると、**Conway's Game of Life** GUI ウィンドウが表示されます。このウィンドウには、空のグリッドまたはアリーナが含まれており、ここで生命のシミュレーションが行われます。システムにまだ生きているセルが含まれていないため、グリッドは最初は空白です。

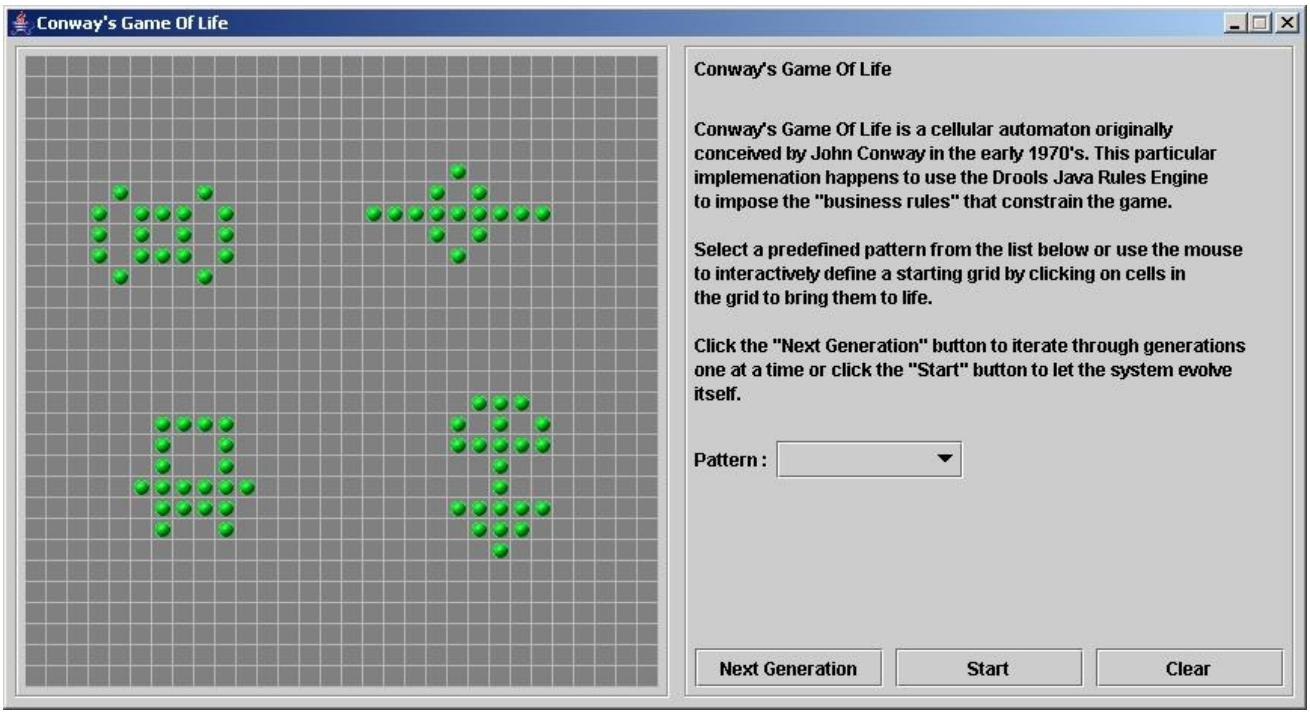
図88.24 起動後の Conway 例の GUI



パターン のドロップダウンメニューから事前定義済みのパターンを選択して、**次の世代** をクリックし、各人口の世代をクリックしていきます。セルは生きているか、死んでいるかのどちらかで、生きて

いるセルには緑のボールが含まれます。最初のパターンから人口が進化するにつれ、ゲームのルールをもとに、セルが近傍のセルに合わせて、生存するか、死亡していきます。

図88.25 Conway の例の世代進化



近傍には、上下左右のセルだけでなく対角線上につながっているセルも含まれるため、各セルには合計 8 つの近傍があります。例外は、角のセルと 4 辺上にあるセルで、それぞれ順に近傍が 3 つだけと、5 つだけになります。

セルをクリックすることで手動で介入して、セルを作成することも、死亡させることもできます。

最初のパターンから自動的に進化を実行するには、**スタート** をクリックします。

ルールグループを使用する Conway 例のルール

ConwayRuleFlowGroupRun の例のルールは、ルールフローグループを使用して、ルール実行を制御します。ルールフローグループは、**ruleflow-group** ルール属性に関連付けられたルールのグループです。これらのルールは、このグループがアクティベートされたときにしか実行されません。グループ自体は、ルールフローの図の詳細がグループを表すノードに到達してからでないと、アクティブになりません。

Conway の例では、ルールに以下のルールフローグループを使用します。

- **"register neighbor"**
- **"evaluate"**
- **"calculate"**
- **"reset calculate"**
- **"birth"**
- **"kill"**
- **"kill all"**

Cell オブジェクトはすべて KIE セッションに挿入され、**"register neighbor"** ルールフローグループの

"register ..." ルールがルールフロー処理により実行できるようになります。4 つのルールが含まれるこのグループは、セル同士の Neighbor の関係と、北東、北、北西、西の近傍との **Neighbor** の関係を作り出します。

この関係は双方向で、他の 4 方向を処理します。各辺上のセルは、特別な対応は必要ありません。これらのセルは、近傍のセルがなければペアは作成されません。

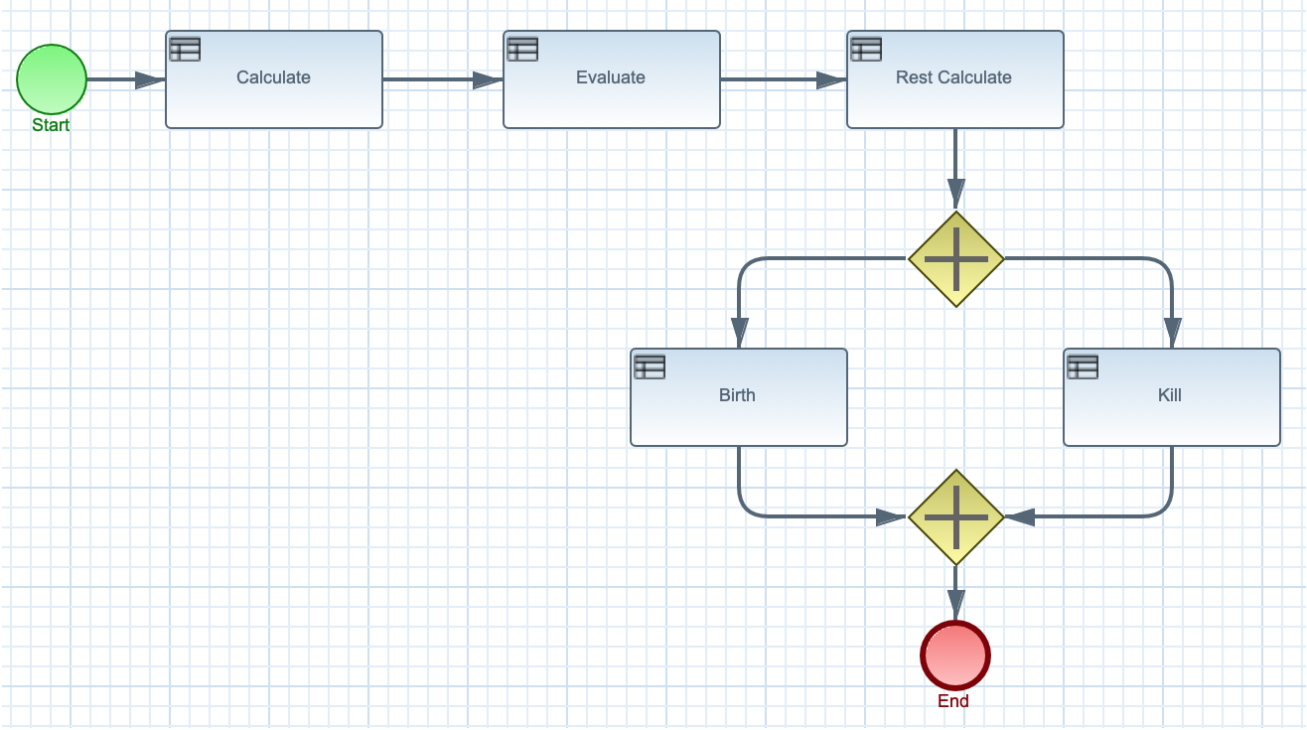
これらのルールに対して、すべてのアクティベーションが実行されるまで、全セルは、近傍の全セルと関係があります。

ルール "register ..."

```
rule "register north east"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northEast : Cell( row  == ($row - 1), col == ( $col + 1 ) )
  then
    insert( new Neighbor( $cell, $northEast ) );
    insert( new Neighbor( $northEast, $cell ) );
end

rule "register north"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $north : Cell( row  == ($row - 1), col == $col )
  then
    insert( new Neighbor( $cell, $north ) );
    insert( new Neighbor( $north, $cell ) );
end

rule "register north west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northWest : Cell( row  == ($row - 1), col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $northWest ) );
    insert( new Neighbor( $northWest, $cell ) );
end

rule "register west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $west : Cell( row  == $row, col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $west ) );
    insert( new Neighbor( $west, $cell ) );
end
```

全セルが挿入されたら、Java コードはグリッドにパターンを適用し、特定のセルを **Live** に設定します。次に、ユーザーが **スタート** または **次の世代** をクリックすると、**Generation** のルールフローが実行されます。このルールフローは、世代のサイクルごとにセルの変更をすべて管理します。

図88.26 世代のルールフロー



ルールフロープロセスは、実行可能なグループに **"evaluate"** ルールフローグループおよびアクティブなルールを追加します。このグループの **"Kill the ..."** と **"Give Birth"** ルールを使用して、細胞の誕生または死亡セルにゲームのルールを適用します。この例では、**phase** 属性を使用して、特定のルールグループで **Cell** オブジェクトの推論をトリガーします。通常は、フェーズはルールフロープロセス定義に含まれるルールフローグループに紐づけされています。

この例では、変更の適用前に評価を完全に完了しておく必要があるため、この時点では **Cell** オブジェクトの状態は変更されません。細胞の **phase** を **Phase.KILL** または **Phase.BIRTH** に適用し、後ほど **Cell** オブジェクトに適用されたアクションを制御するのに使用します。

ルール "Kill the ..." および "Give Birth"

```
rule "Kill The Lonely"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has fewer than 2 live neighbors.
    theCell: Cell( liveNeighbors < 2, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Kill The Overcrowded"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has more than 3 live neighbors.
    theCell: Cell( liveNeighbors > 3, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
```

```
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Give Birth"
    ruleflow-group "evaluate"
    no-loop
  when
    // A dead cell has 3 live neighbors.
    theCell: Cell( liveNeighbors == 3, cellState == CellState.DEAD,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        theCell.setPhase( Phase.BIRTH );
    }
end
```

グリッド内の全 **Cell** オブジェクトが評価されると、この例では **"reset calculate"** ルールを使用して **"calculate"** ルールフローグループのアクティベーションを消去します。次に、ルールフローグループがアクティベートされると、**"kill"** と **"birth"** のルールを有効にするルールフローに分岐を挿入します。これらのルールにより状態の変更が適用されます。

ルール "reset calculate"、"kill"、および "birth"

```
rule "reset calculate"
    ruleflow-group "reset calculate"
  when
  then
    WorkingMemory wm = drools.getWorkingMemory();
    wm.clearRuleFlowGroup( "calculate" );
end

rule "kill"
    ruleflow-group "kill"
    no-loop
  when
    theCell: Cell( phase == Phase.KILL )
  then
    modify( theCell ){
        setCellState( CellState.DEAD ),
        setPhase( Phase.DONE );
    }
end

rule "birth"
    ruleflow-group "birth"
    no-loop
  when
    theCell: Cell( phase == Phase.BIRTH )
  then
    modify( theCell ){
        setCellState( CellState.LIVE ),
        setPhase( Phase.DONE );
    }
end
```

この段階では、複数の **Cell** オブジェクトの状態が **LIVE** または **DEAD** のいずれかに変更されています。この例では、細胞が生存または死亡すると、**"Calculate ..."** ルールの **Neighbor** 関係を使用して、周辺のすべての細胞に繰り返し実行することで、**liveNeighbor** の数が増減します。数が変更した細胞は、**EVALUATE** フェーズに設定され、ルールフロー処理の評価段階の推論に含められるようにします。

生存数が判断され、全細胞に設定されると、ルールフロープロセスが終了します。ユーザーが最初に **Start** をクリックした場合は、その時点でデシジョンエンジンによりルールフローが再起動します。ユーザーが最初に **Next Generation** をクリックした場合は、ユーザーが別の世代を要求することができます。

ルール "Calculate ..."

```
rule "Calculate Live"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.LIVE )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() + 1 ),
        setPhase( Phase.EVALUATE );
    }
end

rule "Calculate Dead"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.DEAD )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() - 1 ),
        setPhase( Phase.EVALUATE );
    }
end
```

88.10. HOUSE OF DOOM 例のデシジョン (後向き連鎖および再帰)

House of Doom のデシジョンセットの例では、デシジョンエンジンが後向き連鎖と再帰を使用して、階層システムで定義した目的やサブゴールに到達する方法を説明します。

以下は House of Doom の例の概要です。

- **名前**: **backwardchaining**
- **Main クラス**: (**src/main/java** 内の) **org.drools.examples.backwardchaining.HouseOfDoomMain**
- **モジュール**: **drools-examples**
- **タイプ**: Java アプリケーション

- **ルールファイル**: (**src/main/resources** 内の) **org.drools.examples.backwardchaining.BC-Example.drl**

- **目的**: 後向き連鎖と再帰を例示します。

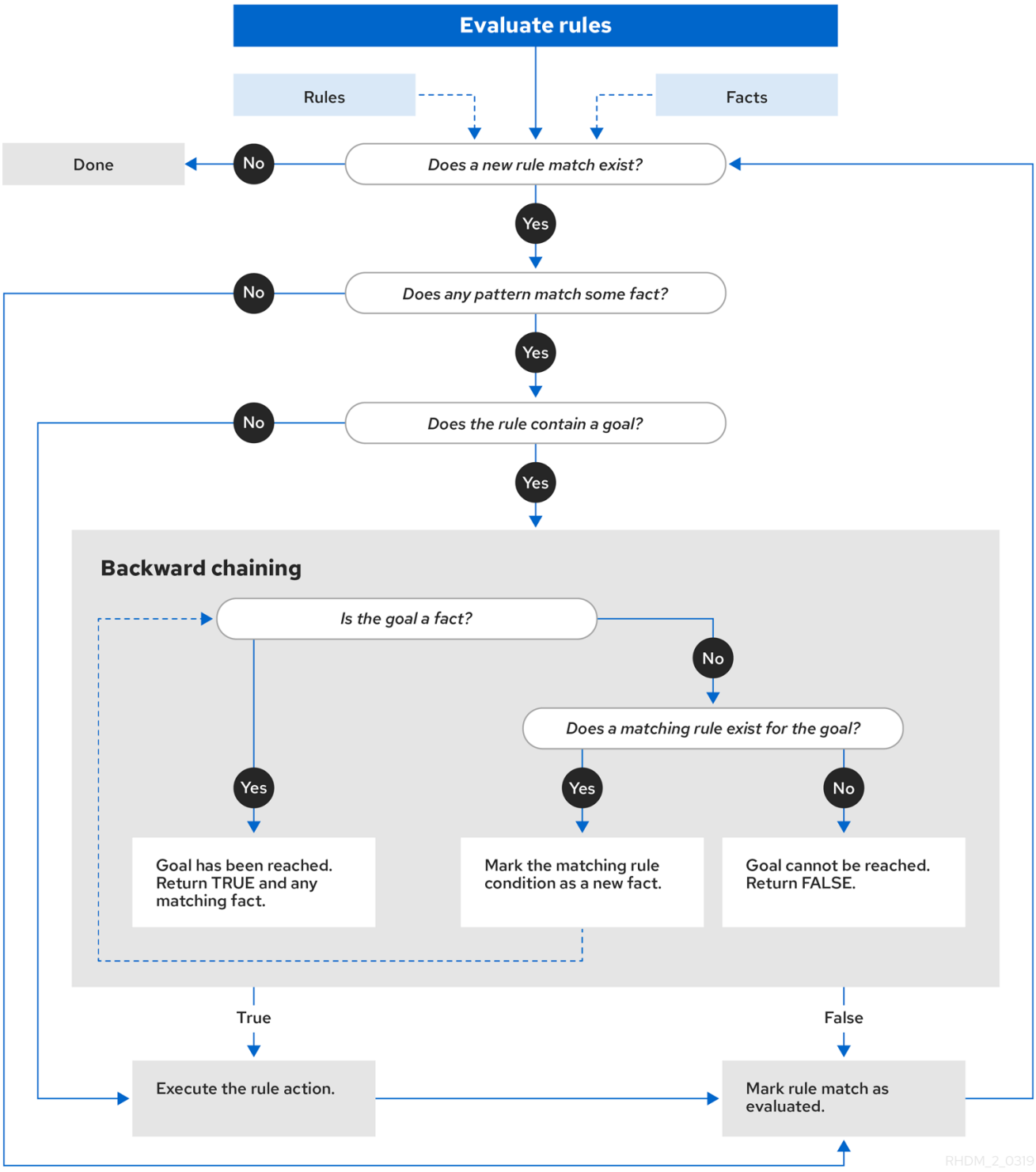
後向き連鎖のルールシステムは、通常再帰を使用して、デシジョンエンジンが満たそうとする結論から開始する目的駆動型のシステムです。システムが結論または目的に到達できない場合は、サブとなる目的、つまり、現在の目的の一部を完了する結論を検索します。システムは、最初の結論が満たされるか、すべてのサブとなる目的が満たされるまでこのプロセスを続行します。

反対に、前向き連鎖のルールシステムは、デシジョンエンジンのワーキングメモリーにあるファクトで開始して、そのファクトへの変更に反応するデータ駆動型のシステムです。オブジェクトがワーキングメモリーに挿入されると、その変更の結果として True となるルールの条件はすべて、アジェンダによって実行されるようにスケジュールされます。

Red Hat Process Automation Manager のデシジョンエンジンは、前向き連鎖と後向き連鎖の両方を使用してルールを評価します。

以下の図は、デシジョンエンジンが、ロジックフローで後向き連鎖のセグメントと、前向き連鎖全体を使用してルールを評価する方法を例示します。

図88.27 前向き連鎖と後向き連鎖を使用したルール評価のロジック



House of Doom の例は、さまざまなクエリータイプが含まれるルールを使用し、部屋の場所と家の中のアイテムを探し出します。**Location.java** のサンプルクラスには、この例で使用する **item** と **location** 要素が含まれます。**HouseOfDoomMain.java** のサンプルクラスで、家の該当の場所にアイテムまたは部屋を挿入して、ルールを実行します。

HouseOfDoomMain.java クラスでのアイテムと場所

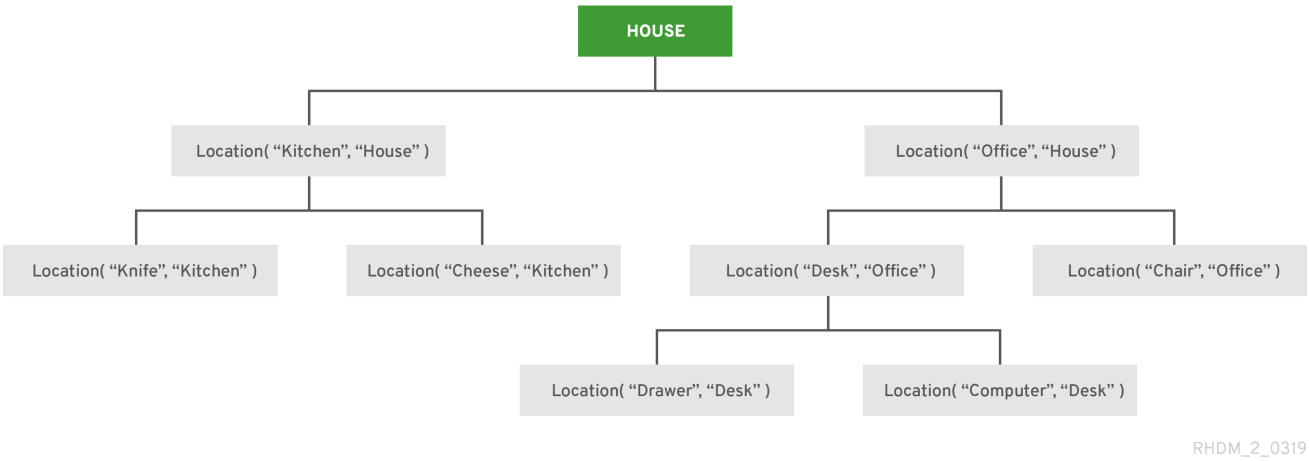
```
ksession.insert( new Location("Office", "House") );
ksession.insert( new Location("Kitchen", "House") );
ksession.insert( new Location("Knife", "Kitchen") );
ksession.insert( new Location("Cheese", "Kitchen") );
```

```
ksession.insert( new Location("Desk", "Office") );
ksession.insert( new Location("Chair", "Office") );
ksession.insert( new Location("Computer", "Desk") );
ksession.insert( new Location("Drawer", "Desk") );
```

ルールの例では、家の構造の中で全アイテムおよび部屋の場所を判断するのに、後向き連鎖と再帰を使用します。

以下の図は、House of Doom の構造と、その構造内のアイテムと部屋を示しています。

図88.28 House of Doom の構造



この例を実行するには、IDE で Java アプリケーションとして **org.drools.examples.backwardchaining.HouseOfDoomMain** クラスを実行します。

実行後に、以下の出力が IDE コンソールウィンドウに表示されます。

IDE コンソールでの実行出力

```
go1
Office is in the House
---
go2
Drawer is in the House
---
go3
---
Key is in the Office
---
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
---
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
```

```
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

この例のルールはすべて実行し、家の中の全アイテムの場所を検出して、出力でそれぞれの場所を出力します。

再帰クエリーおよび関連のルール
再帰クエリーは、要素間の関係におけるデータ構造階層を使用して繰り返し検索を行います。

House of Doom の例では、**BC-Example.drl** ファイルに、この例のルールの大半が使用する **isContainedIn** クエリーが含まれており、家のデータ構造を再帰的に評価して、デシジョンエンジンに挿入するデータがないかを確認します。

BC-Example.drl の再帰クエリー

```
query isContainedIn( String x, String y )
  Location( x, y; )
  or
  ( Location( z, y; ) and isContainedIn( x, z; ) )
end
```

"go" のルールは、システムに挿入する文字列をすべて出力し、アイテムをどのように導入し、**"go1"** ルールが **isContainedIn** クエリーを呼び出すかを判断します。

ルール "go" および "go1"

```
rule "go" salience 10
  when
    $s : String()
  then
    System.out.println( $s );
end

rule "go1"
  when
    String( this == "go1" )
    isContainedIn("Office", "House"; )
  then
    System.out.println( "Office is in the House" );
end
```

この例は、**"go1"** 文字列をデシジョンエンジンに挿入して、**"go1"** ルールを有効化し、**House** の場所にある **Office** アイテムを検出します。

文字列の挿入とルールの実行

```
ksession.insert( "go1" );
ksession.fireAllRules();
```

IDE コンソールでの ルール "go1" の出力

```
go1
Office is in the House
```

推移閉包ルール
推移閉包は、階層構造の複数レベルであり、上層にある親要素に含まれる要素間の関係です。

"go2" ルールは、**Drawer** と **House** の推移閉包の関係を特定します。**Drawer** は、**House** の中の、**Office** の中の、**Desk** の中にあります。

```
rule "go2"
  when
    String( this == "go2" )
    isContainedIn("Drawer", "House"; )
  then
    System.out.println( "Drawer is in the House" );
end
```

この例は、**"go2"** 文字列をデシジョンエンジンに挿入して、**"go2"** ルールを有効化し、最終的に **House** の場所に含まれる **Drawer** アイテムを検出します。

文字列の挿入とルールの実行

```
ksession.insert( "go2" );
ksession.fireAllRules();
```

IDE コンソールのルール "go2" の出力

```
go2
Drawer is in the House
```

デシジョンエンジンは、以下のロジックをもとにこの結果を判断します。

1. クエリーは再帰的に、家の中の複数レベルを検索して、**Drawer** と **House** の間の推移閉包を検出します。
2. **Drawer** は **House** に直接含まれないため、**Location(x, y;)** を使用する代わりに、このクエリーは **(z, y;)** の値を使用します。
3. **z** の引数は現在バインドされておらず、値が指定されていないため、引数に含まれるものはすべて返されます。
4. **y** の引数は現在、**House** にバインドされているため、**z** は **Office** と **Kitchen** を返します。

5. クエリーは、**Office** からの情報を収集して、**Drawer** が **Office** に含まれているかを再帰的にチェックします。これらのパラメーターに対して、クエリーの行 **isContainedIn(x, z;)** が呼び出されます。

6. **Office** に直接含まれる **Drawer** が存在しないため、一致するものはありません。

7. **z** のバインドがない場合、このクエリーでは **Office** 内のデータが返され、**z == Desk** と判断されます。

```
isContainedIn(x==drawer, z==desk)
```

8. **isContainedIn** クエリーは再帰的に 3 回検索し、3 回目に、このクエリーにより **Desk** の中に **Drawer** があることが検出されます。

```
Location(x==drawer, y==desk)
```

9. 最初の場所で上記が一致した後に、このクエリーにより再帰的に構造を上方向に検索し、**Drawer** が **Desk** の中に、**Desk** が **Office** の中に、**Office** が **House** の中にあることを判断します。このように、**Drawer** は **House** の中にあるため、このルールは満たされます。

リアクティブクエリールール
リアクティブクエリーでは、データ構造の階層を検索して、要素間に関係があるかを確認し、構造内の要素が変更されると動的に更新されます。

"go3" ルールは、リアクティブクエリーとして機能し、推移閉包により、新しいアイテム **Key** が **Office** に含まれるかどうかを検出します (**Office** の中の **Drawer** の中の **Key** など)。

ルール "go3"

```
rule "go3"
  when
    String( this == "go3" )
    isContainedIn("Key", "Office"; )
  then
    System.out.println( "Key is in the Office" );
end
```

この例は、**"go3"** 文字列をデシジョンエンジンに挿入して、**"go3"** ルールを有効にします。最初は、**Key** が家の構造に存在するため、このルールは満たされず、出力が生成されません。

文字列の挿入とルールの実行

```
ksession.insert( "go3" );
ksession.fireAllRules();
```

IDE コンソールのルール "go3" の出力 (条件を満たさない)

```
go3
```

この例では、**Office** の中にある **Drawer** の場所に、新しいアイテム **Key** を挿入します。この変更で、**"go3"** ルールの推移閉包が満たされ、それに合わせて出力が生成されます。

新規アイテムの場所の挿入とルールの実行

```
ksession.insert( new Location("Key", "Drawer") );
ksession.fireAllRules();
```

IDE コンソールのルール "go3" の出力 (条件を満たす)

```
Key is in the Office
```

またこの変更で、クエリーにより、後に続く再帰検索に含まれるよう、この構造に別のレベルが追加されます。

ルールにバインドなしの引数が含まれたクエリー
バインドなしの引数が 1 つ以上あるクエリーでは、クエリーの定義済み (バインドされている) 引数に含まれる未定義 (バインドされていない) のアイテムをすべて返します。クエリー内の引数でバインドされているものがない場合、クエリーはクエリーの範囲内のアイテムをすべて返します。

"go4" ルールは、バインドされている引数を使用して、**Office** 内の特定のアイテムを検索するのではなく、バインドされていない引数 **thing** を使用して、バインドされている引数 **Office** 内の全アイテムを検索します。

ルール "go4"

```
rule "go4"
  when
    String( this == "go4" )
    isContainedIn(thing, "Office"; )
  then
    System.out.println( thing + "is in the Office" );
end
```

この例では **"go4"** 文字列をデシジョンエンジンに挿入して、**"go4"** ルールをアクティベートし、**Office** の全アイテムを返します。

文字列の挿入とルールの実行

```
ksession.insert( "go4" );
ksession.fireAllRules();
```

IDE コンソールのルール "go4" の出力

```
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
```

"go5" ルールは、バインドされていない引数 **thing** と **location** を使用して、**House** の全データ構造の中に含まれる全アイテムとその場所を検索します。

ルール "go5"

```
rule "go5"
  when
```

```
    String( this == "go5" )
    isContainedIn(thing, location; )
  then
    System.out.println(thing + " is in " + location );
end
```

この例は **"go5"** 文字列をデシジョンエンジンに挿入して、**"go5"** ルールをアクティベートし、**House** データ構造に含まれる全アイテムとその場所を返します。

文字列の挿入とルールの実行

```
ksession.insert( "go5" );
ksession.fireAllRules();
```

IDE コンソールのルール "go5" の出力

```
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

第89章 デシジョンエンジン使用時のパフォーマンスチューニングに関する考慮点

以下の主要な概念または推奨のプラクティスを使用すると、デシジョンエンジンのパフォーマンス最適化に役立ちます。本セクションではこの概念についてまとめており、随時、他のドキュメントを相互参照して詳細を説明します。本セクションは、Red Hat Process Automation Manager の新しいリリースで、必要に応じて拡張または変更します。

デシジョンエンジンの重要な更新を必要としない、ステートレス KIE セッションには順次モードを使用する

順次モードは、デシジョンエンジンにおける高度なルールベースの設定です。順次モードでは、デシジョンエンジンは、ワーキングメモリーでの変更に関係なく、デシジョンエンジンアジェンダにリスト化された順番でルールを一度評価します。その結果、ルールの実行は順次モードの方が速くなる可能性がありますが、重要な更新がルールに適用されない可能性があります。順次モードは、ステートレス KIE セッションのみに適用されます。
順次モードを有効にするには、システムプロパティー **drools.sequential** を **true** に設定します。

順次モードや、このモードの有効化に関する他のオプションの情報は、「Phreak における順次モード」 を参照してください。

イベントリスナーを使用する場合は簡単な操作を使用する

イベントリスナーの数や、実行する操作の種類を制限します。デバッグロギングや設定プロパティーなどの簡単な操作にイベントリスナーを使用します。リスナーでネットワーク呼び出しなどの複雑な操作を行うと、ルールの実行が阻害される可能性があります。KIE セッションでの作業が完了した後には、セッションを消去できるように、アタッチされているイベントリスナーを以下の例のように削除します。

使用後に削除されるイベントリスナーの例

```
Listener listener = ...;
StatelessKnowledgeSession ksession = createSession();
try {
    ksession.insert(fact);
    ksession.fireAllRules();
    ...
} finally {
    if (session != null) {
        ksession.detachListener(listener);
        ksession.dispose();
    }
}
```

同梱のイベントリスナーとデシジョンエンジンでのデバッグロギングに関する情報は、87章 *デシジョンエンジンのイベントリスナーおよびデバッグロギング* を参照してください。

実行可能なモデルビルドの LambdaIntrospector のキャッシュサイズ設定

実行可能なモデルビルドで使用される **LambdaIntrospector.methodFingerprintsMap** キャッシュのサイズを設定できます。キャッシュのデフォルトサイズは **32** です。キャッシュサイズに小さい値を設定すると、メモリーの使用量が減少します。たとえば、システムプロパティー **drools.lambda.introspector.cache.size** を **0** に設定すると、メモリーの使用量を最小限に抑えることができます。キャッシュサイズが小さくなると、ビルドパフォーマンスが低下する点に注意してください。

実行可能モデルでの lambda 外部化の使用

lambda 外部化を有効にして、実行時のメモリーの消費量を最適化します。この設定は、実行可能モデルで生成され使用される lambda を書き換えます。そのため、同じ lambda を全パターンおよび同じ制約で複数回再利用できます。Rete または Phreak がインスタンス化されると、実行可能なモデルをガベージコレクションにすることができます。
実行可能モデルの lambda 外部化を有効にするには、以下のプロパティーを追加します。

```
-Ddrools.externaliseCanonicalModelLambda=true
```

アルファノード範囲のインデックスしきい値の設定

アルファノード範囲のインデックスは、ルール制約を評価するのに使用されます。**drools.alphaNodeRangeIndexThreshold** システムプロパティーを使用して、アルファノードの範囲インデックスのしきい値を設定できます。しきい値のデフォルト値は **9** です。これは、先行ノードが不等式制約を持つ 9 個以上のアルファノードを含む場合にアルファノード範囲インデックスが有効になることを示しています。たとえば、**Person(age > 10)**、**Person(age > 20)** … **Person(age > 90)** のようなルールがあると、同様の 9 つのアルファノードを持つことができます。しきい値のデフォルト値は、関連する利点とオーバーヘッドに基づきます。ただし、しきい値に小さい値を設定すると、ルールに応じてパフォーマンスを向上させることができます。たとえば、**drools.alphaNodeRangeIndexThreshold** の値を **6** に設定すると、先行ノード用に 6 を超えるアルファノード範囲のインデックスを有効にできます。ルールのパフォーマンステスト結果に基づいて、しきい値に適切な値を設定できます。

結合ノード範囲のインデックスの有効化

結合ノードの範囲インデックス機能は、複数のファクトを結合する場合にのみパフォーマンスが向上します (例: 256*16 の組み合わせ)。アプリケーションが多数のファクトを挿入すると、結合ノード範囲のインデックスを有効にして、パフォーマンスの向上を評価できます。デフォルトでは、結合ノード範囲インデックスは無効になります。

kmodule.xml ファイルの例

```
<kbase name="KBase1" betaRangeIndex="enabled">
```

BetaRangeIndexOption のシステムプロパティー

```
drools.betaNodeRangeIndexEnabled=true
```

第90章 関連情報

- Red Hat Process Automation Manager 用のデシジョン管理アーキテクチャーの設計
- デシジョンサービスのスタートガイド
- DRL ルールを使用したデシジョンサービスの作成
- Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

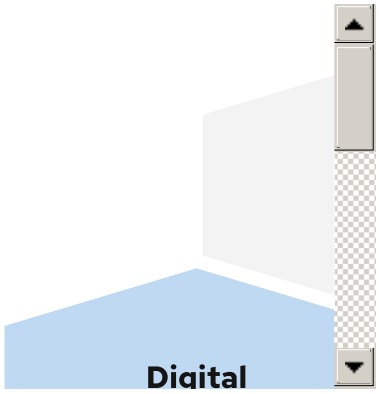
パート X. RED HAT PROCESS AUTOMATION MANAGER での機械学習の統合

ビジネス分析者またはビジネスルールの開発者は、DMN (Decision Model and Notation) モデルと PMML ファイルを使用して、機械学習を Red Hat Process Automation Manager に統合できます。

第91章 PRAGMATIC AI

人工知能 (AI) について考えるとき、機械学習とビッグデータが思い浮かぶかもしれません。ただし、機械学習は全体像の一部にすぎません。人工知能には、以下の技術が含まれます。

- ロボット工学: 人間が実行する物理的なタスクを実行できる機械を製造する技術、科学、工学の統合
- 機械学習: 明示的にプログラムされていなくても、データにさらされたときに学習または改善するアルゴリズムのコレクションの機能
- 自然言語処理: 人間の音声を処理する機械学習のサブセット
- 数理最適化: 問題の最適解を見つけるための条件と制約の使用
- デジタル意思決定: 定義された基準、条件、および一連の機械的タスクおよび人的タスクを使用して意思決定を行う



サイエンスフィクションの世界では、人よりも優れたパフォーマンスを発揮し、人と区別できず、人間の介入や制御なしに学習して進化する、いわゆる人工知能 (AGI) が多用されていますが、AGI は数十年先にあります。一方、Pragmatic AI は、それほど恐ろしくなく、今日の私たちにとってはるかに便利です。Pragmatic AI は、AI テクノロジーのコレクションであり、組み合わせることで、顧客の行動の予測、自動化された顧客サービスの提供、顧客の購入決定の支援などの問題に対するソリューションを提供します。

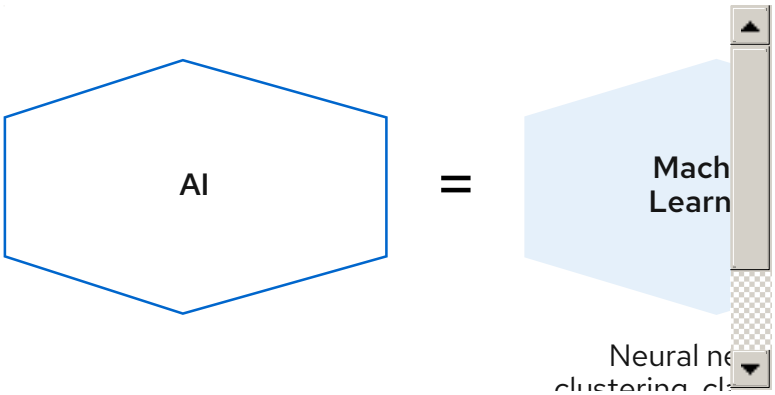
業界をリードするアナリストは、以前の組織は AI が今日提供できるものの現実ではなく、AI の可能性に投資したため、AI テクノロジーに苦労していたと報告しています。AI プロジェクトは生産的ではなく、その結果、AI への投資が鈍化し、AI プロジェクトの予算が削減されました。この AI に対する幻滅は、しばしば AI の冬と呼ばれています。AI は、AI の冬とそれに続く AI の春のサイクルを数回経験しており、今では明らかに AI の春にいます。組織は、AI が提供できる実際の状況を把握しています。実利的であることは、実用的かつ現実的であることを意味します。AI への実利的なアプローチでは、現在利用可能な AI テクノロジーを検討し、有用な場合はそれらを組み合わせ、必要に応じて人間の介入を追加して、現実の問題に対する解を作成します。

Pragmatic AI の解の例

Pragmatic AI の適用の 1 つは、カスタマーサポートです。顧客が問題 (ログインエラーなど) を報告するサポートチケットを定義します。機械学習アルゴリズムがこのチケットに適用され、キーワードまたは自然言語処理 (NLP) に基づいて、チケットのコンテンツを既存の解と照合します。キーワードは多くの解に表示される可能性があり、関連するものもあれば、関連性がないものもあります。デジタル署名を使用して、お客様に提示する解を判断できます。ただし、アルゴリズムにより提案された解のいずれも、お客様に提案された解に適していない場合があります。これは、すべての解の信頼スコアが低いか、複数のソリューションの信頼スコアが高いことが原因である可能性があります。適切な解が見つか

らない場合は、デジタル決定にヒューマンサポートチームが関与する可能性があります。可用性と専門知識に基づいて最適なサポート担当者を見つけるために、数理最適化では、従業員名簿の制約を考慮して、サポートチケットに最適な担当者を選択します。

この例が示すように、機械学習を組み合わせてデータ分析とデジタル決定から情報を抽出し、人間の知識と経験をモデル化できます。次に、数理最適化を適用して、人間による支援をスケジュールできます。これは、クレジットカードの異議申し立てやクレジットカードの不正検出など、他の状況に適用できるパターンです。



これらの技術は、4 つの業界標準を使用します。

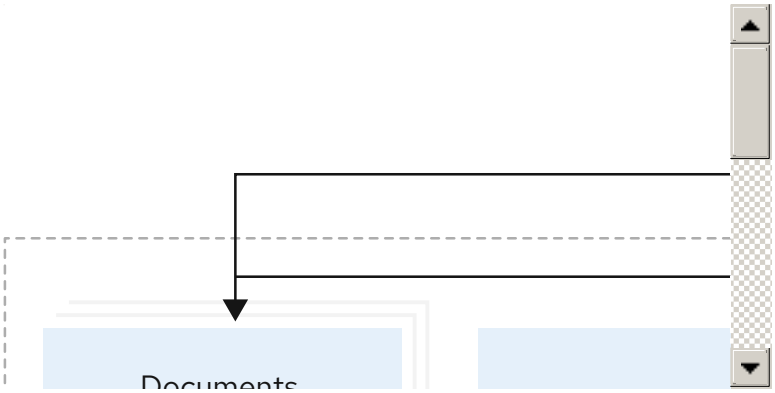
- Case Management Model and Notation (CMMN)
  CMMN は、状況に応じて予測できない順番で実行される可能性のあるさまざまなアクティビティーを含む作業メソッドをモデル化します。CMMN モデルはイベントを中心とします。CMMN は、構造化されていない作業タスクと人間が行うタスクをサポートすることにより、BPMN2 でモデル化できるものの制限を克服します。BPMN と CMMN を組み合わせることで、より強力なモデルを作成できます。

- Business Process Model and Notation (BPMN2)
  BPMN2 仕様は、ビジネスプロセスを描画表現するための標準定義、要素の実行セマンティクス定義、および XML 形式でのプロセス定義を行うときに使用する Object Management Group (OMG) 仕様です。BPMN2 はコンピューターおよびヒューマンタスクをモデル化できます。

- DMN (Decision Model and Notation)
  DMN (Decision Model and Notation) は、業務的意思決定を説明してモデル化するために、OMG が確立している規格です。DMN は XML スキーマを定義して、DMN モデルを DMN 準拠のプラットフォーム間や組織間で共有し、ビジネスアナリストやビジネスルール開発者が DMN デシジョンサービスの設計と実装で協力できるようにするものです。DMN 規格は、ビジネスプロセスを開発してモデル化する BPMN (Business Process Model and Notation) 規格と類似しており、一緒に使用できます。

- Predictive Model Markup Language (PMML)
  PMML は、予測モデル、つまり統計的手法を使用して大量のデータのパターンを検出または学習する数学モデルを表すのに使用される言語です。予測モデルは、学習したパターンを使用して、新しいデータのパターンの存在を予測します。PMML を使用すると、アプリケーション間で予測可能なモデルを共有できます。このデータは、DMN モデルで使用できる PMML ファイルとしてエクスポートされます。機械学習フレームワークがモデルのトレーニングを継続すると、更新されたデータを既存の PMML ファイルに保存できます。つまり、モデルを PMML ファイルとして保存できるアプリケーションにより作成された予測モデルを使用できます。したがって、DMN および PMML が適切に統合されます。

すべてを 1 つにまとめる



この図は、予測可能なデシジョン自動化が機能する仕組みを示しています。

1. ローン申請からのデータなど、ビジネスデータがシステムに入力されます。
2. 予測モデルと統合されたデシジョンモデルは、ローンを承認するかどうか、または追加のタスクが必要であるかどうかを判断します。
3. 拒否レターやローンの申し出などのビジネスアクションの結果が顧客に送信されます。

次のセクションでは、予測的意思決定の自動化が Red Hat Process Automation Manager でどのように機能するかを示します。

第92章 クレジットカード詐欺紛争のユースケース

金融業界は、いくつかの分野で意思決定に Pragmatic AI を使用しています。その 1 つは、クレジットカードの請求に関する紛争です。顧客がクレジットカードの請求書に誤った、または認識していない請求を特定した場合、顧客はその請求に異議を申し立てることができます。クレジットカード詐欺の検出に人間の介入が必要な場合もありますが、報告されたクレジットカード詐欺の大部分は、Pragmatic AI で完全にまたは部分的に解決できます。

このユースケースは、架空の Fortress Bank、銀行の顧客である Joe、およびビジネスプロセス管理 (BPM) 開発者の Michelle に関するものです。最初に、銀行が Red Hat Process Automation Manager デジタル意思決定を通じて AI を最初にどのように使用したかを見てから、Michelle が機械学習から作成された予測モデルマークアップ言語 (PMML) モデルで意思決定モデルをどのように強化したかを見ていきます。

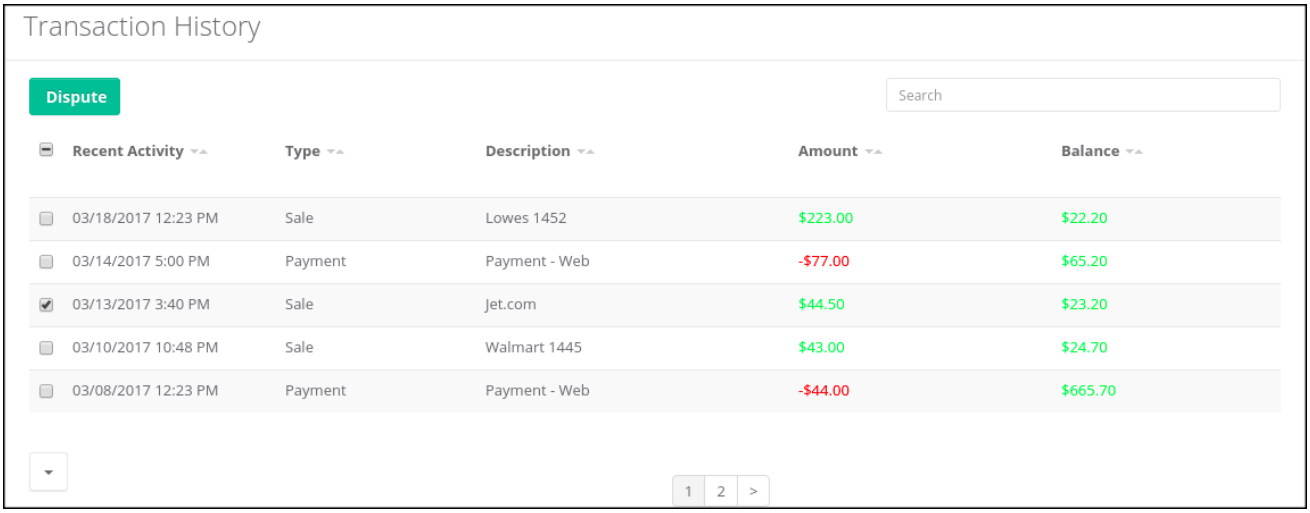
Tensorflow™ および R™ などのマシンラーニングモデルは予測可能なモデルを生成します。この予測モデルは、PMML などのオープン標準で保存できます。これにより、Red Hat Process Automation Manager または PMML 標準をサポートする他の製品でモデルを使用できます。

Red Hat Process Automation Manager および Red Hat OpenShift Container Platform

Fortress Bank は、Red Hat OpenShift Container Platform で Red Hat Process Automation Manager を使用して、Fortress Bank デシジョンサービスを開発し、実行します。Red Hat OpenShift Container Platform は、コンテナー化されたアプリケーションを開発し、実行するためのクラウドプラットフォームです。これは、アプリケーションとそれらをサポートするデータセンターを、ほんの数台のマシンとアプリケーションから、数百万のクライアントにサービスを提供する数千のマシンに拡張できるように設計されています。Red Hat Process Automation Manager は、クラウドネイティブのビジネス自動化アプリケーションとマイクロサービスを作成するための Red Hat ミドルウェアプラットフォームです。これにより、企業のビジネスユーザーと IT ユーザーが、ビジネスプロセスおよび決定を文書化、シミュレート、管理、自動化、およびモニターできます。Business Central は Red Hat Process Automation Manager ダッシュボードです。

Red Hat Process AutomationManager でのクレジットカード紛争のデジタル決定

Joe は Fortress Bank の顧客です。毎月、請求書を支払う前に彼は Fortress Bank の Web サイトにログインして、請求書のすべての請求を確認します。今月、Joe が認識しているトランザクションを確認しましたが、金額が正しくありません。ベンダーは彼に 4.50 ドルではなく 44.50 ドルを請求しました。Joe は誤ったアイテムを含む行を選択し、**Dispute** をクリックします。



このアクションは、紛争に関する一連の質問を開始します。

1. これらのトランザクションを破棄する理由

2. カードはずっと手許にあったか

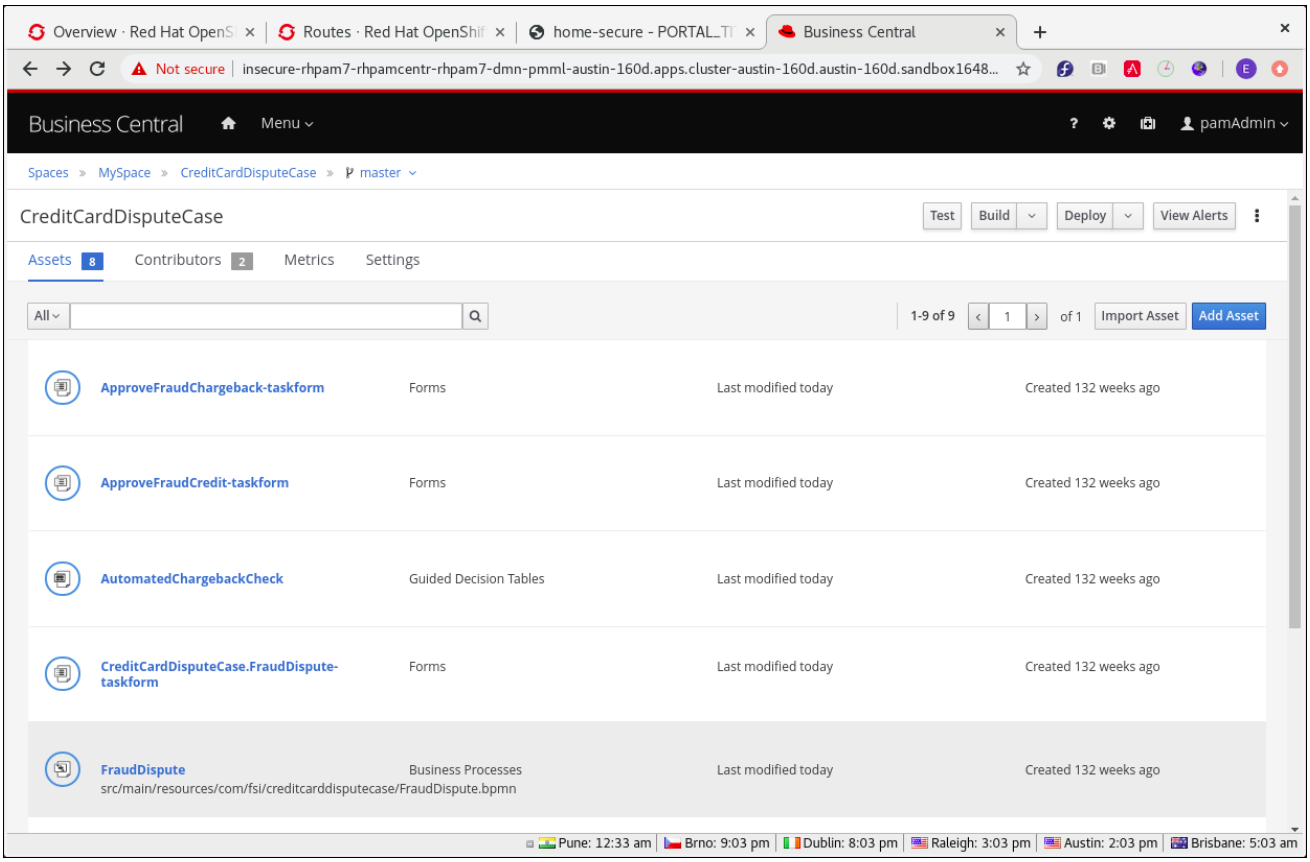
3. この異議申し立てについて他に何か伝えたいことはあるか

Joe が質問に回答すると、この Web サイトではインシデント番号 **000004** が提供されます。

Fortress Bank は、人的調査なしに係争金額を払い戻すか、請求を手動で調査するかを決定する必要があります。人的調査にはより多くのリソースが必要であるため、異議が申し立てられた額の自動処理により、銀行の人的リソースのコストが削減されます。ただし、銀行が異議が申し立てられた金額をすべて自動的に受け入れる場合は、不正請求に対して支払われる金額が原因で、最終的に銀行が負担するコストが高くなります。誰かまたは何かが調査するかどうかを決定しなければなりません。

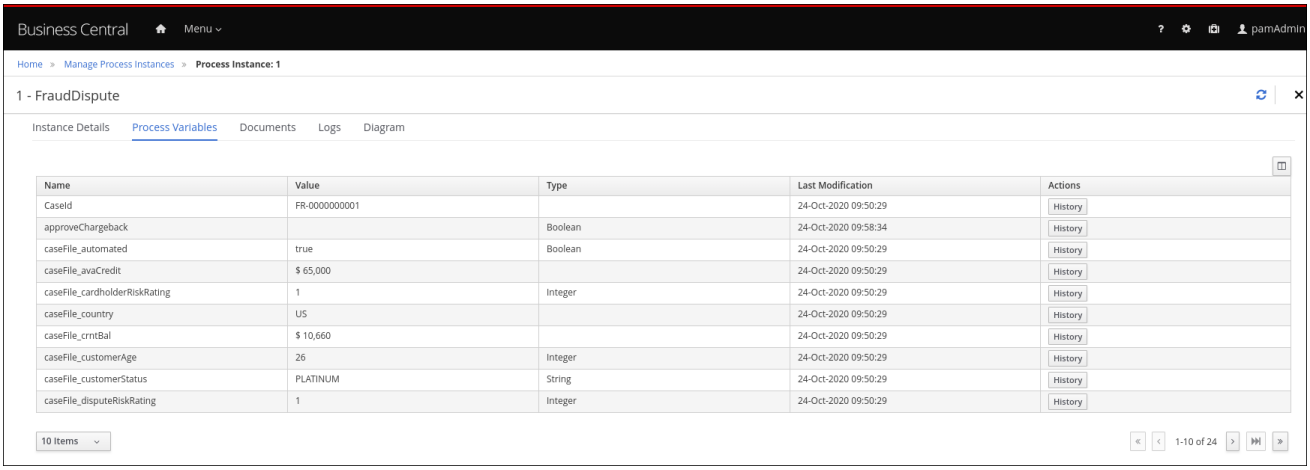
Credit Card Dispute プロジェクト

この決定を容易にするために、Fortress Bank は Business Central を使用して、その紛争プロセスをモデル化する **fraudDispute** ビジネスプロセスが含まれる **CreditCardDisputeCase** プロジェクトを作成します。



プロセス変数

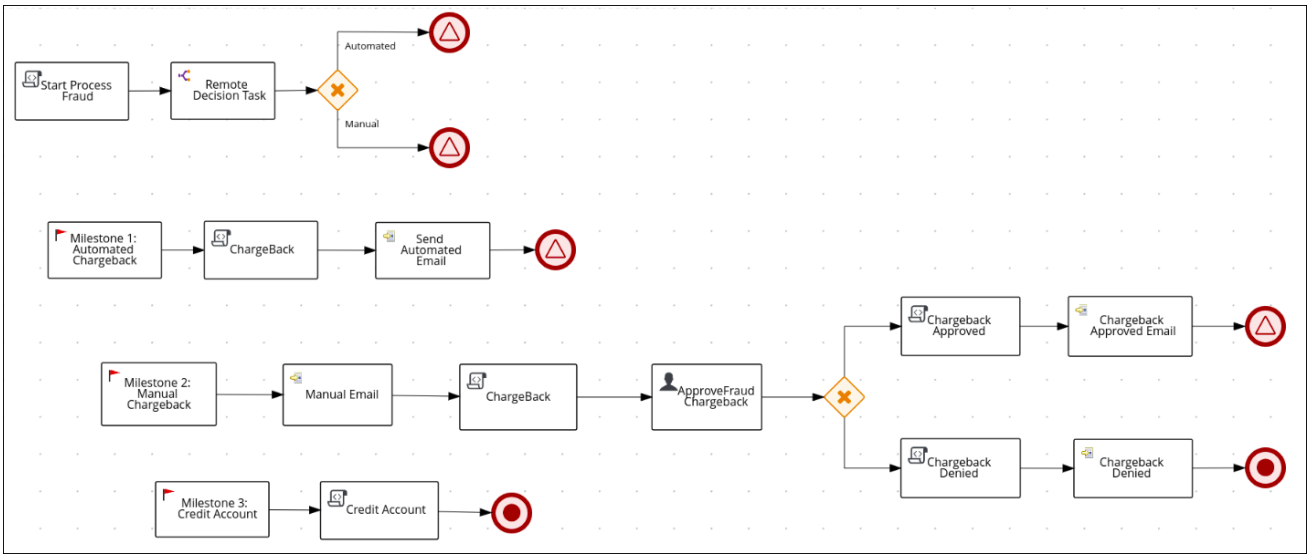
Joe が論争を報告したとき、**fraudDispute** プロセスのインスタンスがケース ID **FR-00000004** で作成されました。**Process Variables** タブには、**CaseID**、**caseFile_cardholderRiskRating** (クレジットカード所有者のリスク評価)、および **caseFile_disputeRiskRating** (この異議申し立てのリスク評価) など、Joe のアカウントに固有のいくつかの変数と値が含まれています。



この表には、値が **true** の **casefile_automated** 変数も含まれています。これは、異議申し立てが自動的に処理される基準を満たしていることを示しています。

プロセスダイアグラム

Diagram タブには BPMN ダイアグラムが含まれており、自動的に処理されるか手動で処理するかを決定する際に銀行が使用するデシジョンパスが表示されます。

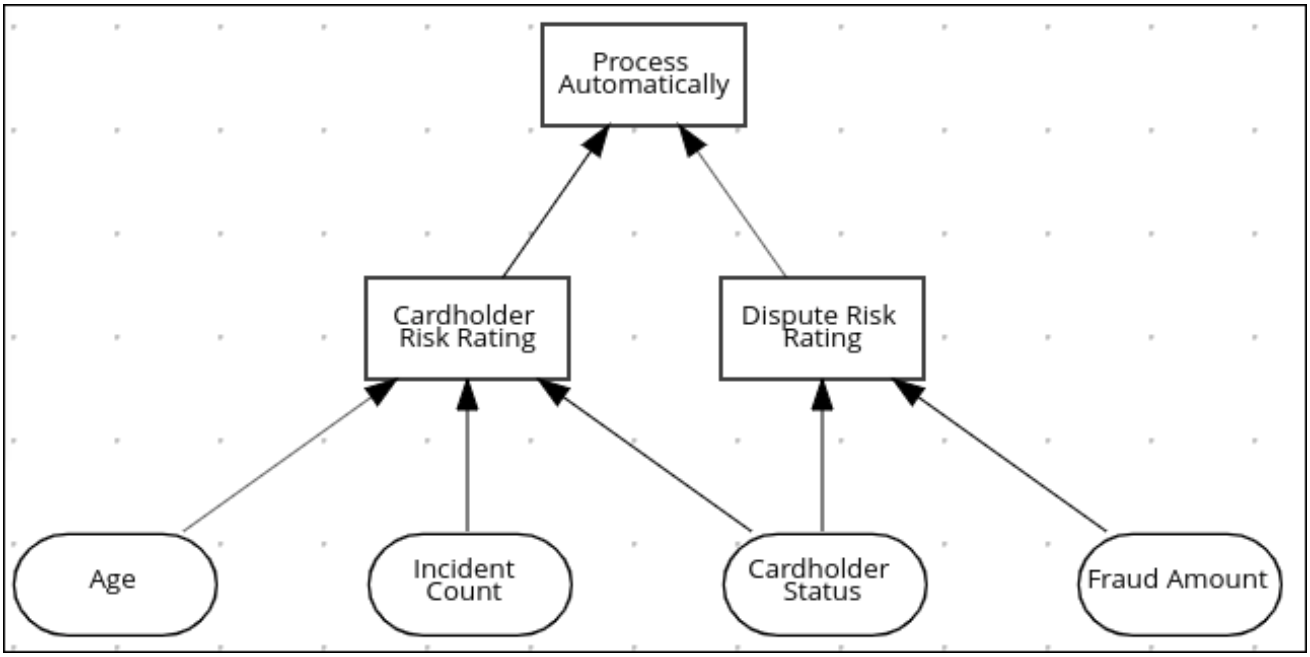


Decision Task タスクには、**caseFile_cardholderRiskRating** 変数および **caseFile_disputeRiskRating** 変数の値に基づいて、この異議申し立てが自動的に Joe のアカウントに請求される (**Milestone 1**) か、さらなる調査が必要 (**Milestone 2**) かを決定するルールが含まれています。Joe の異議申し立てが自動承認の基準に一致する場合は、**Milestone 1** が実行され、異議申し立てられた金額が彼の口座に請求されます。このサブプロセスは非常に無駄がなく効率的です。Joe の異議申し立てが手動評価を必要とする場合は、**Milestone 2** のサブプロセスが開始します。これには、人間のアクターの関与が必要であり、処理により多くのリソースが必要になります。

この場合、**Decision Task** タスクは、Joe の異議申し立てを自動的に処理することを決定したため、**Milestone 1: Automatic Chargeback** に従いました。

DMN モデル

以下の簡単な DMN モデルは、**fraudDispute Decision Task** タスクの一部であるデシジョンプロセスを示しています。



入力変数は、**Age** (カード所有者の年齢)、**Incident Count** (このカード所有者の以前の異議申し立ての数)、**Cardholder Status** (スタンダート、シルバー、ゴールド、プラチナ)、および **Fraud Amount** です。

Red Hat Process Automation Manager は、デシジョンテーブルで入力変数を使用して、デジタルデシジョンをサポートします。デシジョンテーブルは、人間のビジネスアナリストが作成します。アナリストは、利害関係者がレビューおよび承認するドラフトビジネス要件分析ドキュメントまたはスプレッドシートを作成します。プロジェクトデザイナーは、Business Central DMN エディターを使用して、分析ドキュメントのデータを DMN モデルに転送します。Fortress Bank の **Credit Card Dispute** プロセスには、**Cardholder Risk Rating** テーブルと **Dispute Risk Rating** テーブルの 2 つのデシジョンテーブルがあります。**Cardholder Risk Rating** テーブルには、**Incident Count**、**Cardholder Status**、および **Age** の 3 つの入力変数が含まれます。**Dispute Risk Rating** テーブルには **Cardholder Status** 入力変数が含まれます。この表は、カード所有者のステータスと異議申し立ての金額に基づいて、異議申し立てのリスクを計算します。

Cardholder Risk Rating *(Decision Table)*

C+	Incident Count *(number)*	Cardholder Status *(string)*	Age *(number)*	Cardholder Risk Rating *(number)*
1	> 3	"PLATINUM"	-	1
2	> 2	"GOLD"	-	1
3	> 2	"SILVER"	-	2
4	> 2	"STANDARD"	-	3
5	-	"SILVER"	< 25	1
6	-	"STANDARD"	< 25	2
7	-	"STANDARD"	>= 25	1
8	-	-	-	0

Dispute Risk Rating *(Decision Table)*

U	Cardholder Status *(string)*	Fraud Amount *(number)*	Dispute Risk Rating *(number)*	Description
1	"STANDARD"	< 25	1	
2	"SILVER"	< 50	1	
3	"GOLD"	< 75	1	
4	"PLATINUM"	< 100	1	
5	"STANDARD"	[25..150)	3	
6	"SILVER"	[50..150)	2	
7	"GOLD"	[75..150)	2	
8	"PLATINUM"	[100..150)	2	
9	"STANDARD"	[150..200)	4	
10	"SILVER"	[150..200)	3	
11	"GOLD"	[150..200)	2	
12	-	>= 200	5	

- **カード所有者のリスク評価**
  Joe は、25 歳以上の Silver カードホルダーです。彼は以前に異議申し立てを 2 回以上行ったため、リスク評価は 2 です。Joe がこれまで異議申し立てを行わなかった場合、彼のリスク評価は 0 になります。

- **異議申し立てリスク評価**
  Joe はシルバーカード所有者であり、異議申し立て額は 40 ドルであるため、**Dispute Risk Rating** テーブルでの Joe の格付けは 1 になります。異議申し立てのあった金額が 140 ドルであれば、彼のリスク評価は 2 になります。

DMN モデルで **Process Automatically** の最終決定の一部として実装された以下の式は、異議申し立てが行われた額を自動的に返還する (**Milestone 1**) か、より詳細な調査が必要である (**Milestone 2**) かどうかを決定するために、この 2 つのデシジョンテーブルから得たスコアを使用します。

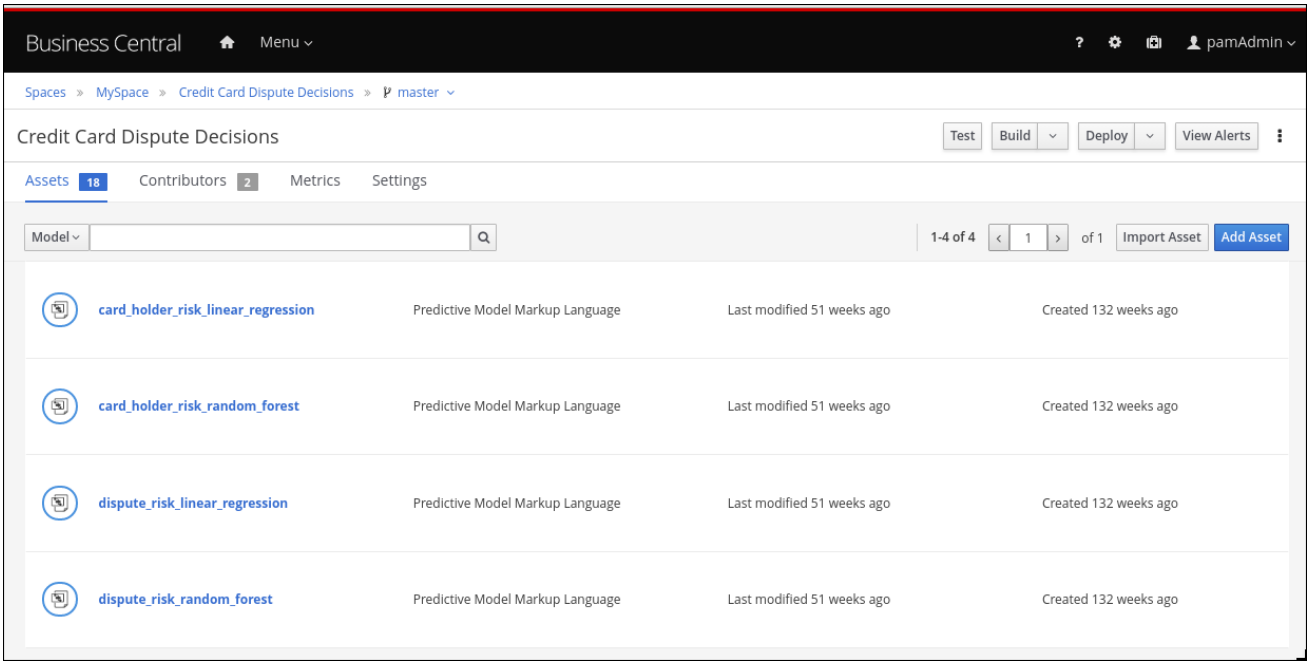
> (Cardholder Risk Rating + Dispute Risk Rating) < 5

Joe の全体的なリスクスコアが 5 未満の場合、異議申し立てが行われた金額は自動的に返金されます (**Milestone 1**)。全体のスコアが 5 以上の場合、彼の異議申し立ては手動で処理されます (**Milestone 2**)。

Red Hat Process Automation Manager への機械学習の追加

Fortress Bank には、過去の取引や異議申し立ての履歴など、顧客に関する履歴データがあるため、銀行はそのデータを機械学習で使用して、DMN モデルの意思決定タスクで使用できる予測モデルを作成できます。これにより、ビジネスアナリストが作成したデシジョンテーブルと比較すると、リスクの評価がより正確になります。

Fortress Bank には、リスク予測をより正確に評価するモデルを含む 2 セットの PMML ファイルがあります。1 つのセットは線形回帰アルゴリズムに基づいており、もう 1 つのセットはランダムフォレストアルゴリズムに基づいています。



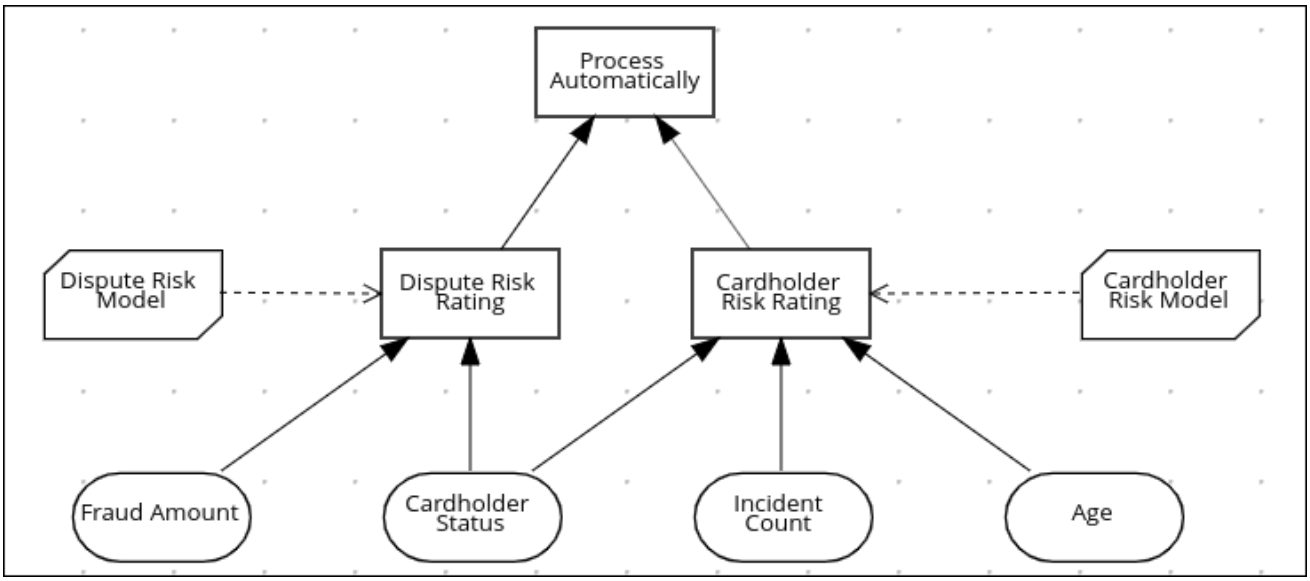
線形回帰は、統計と機械学習の両方で最も広く使用されているアルゴリズムの 1 つです。これは、数値の入力値と出力値のセットを組み合わせた線形方程式を使用します。ランダムフォレストは、多くのデシジョンツリーを入力として使用し、予測モデルを作成します。

PMML ファイルの追加

Michelle は、PMML ファイル **dispute_risk_linear_regression** を自分のプロジェクトにインポートします。彼女は、**Cardholder Risk Model** ビジネスモデルナレッジノードを DMN モデルに追加し、PMML ファイル **dispute_risk_linear_regression** をノードに関連付けます。Red Hat Process Automation Manager は PMML ファイルを分析し、入力パラメーターをノードに追加します。Michelle は、**Cardholder Risk Model** ノードを **Dispute Risk Rating** に関連付けます。

次に、PMML モデル **credit_card_holder_risk_linear_regression** をこのプロジェクトに追加し、**Dispute Risk Model** モードの DMN ファイルを作成し、PMML ファイル **credit_card_holder_risk_linear_regression** を作成してこのノードに関連付けます。Red Hat Process Automation Manager は PMML ファイルを分析し、入力パラメーターをノードに追加します。

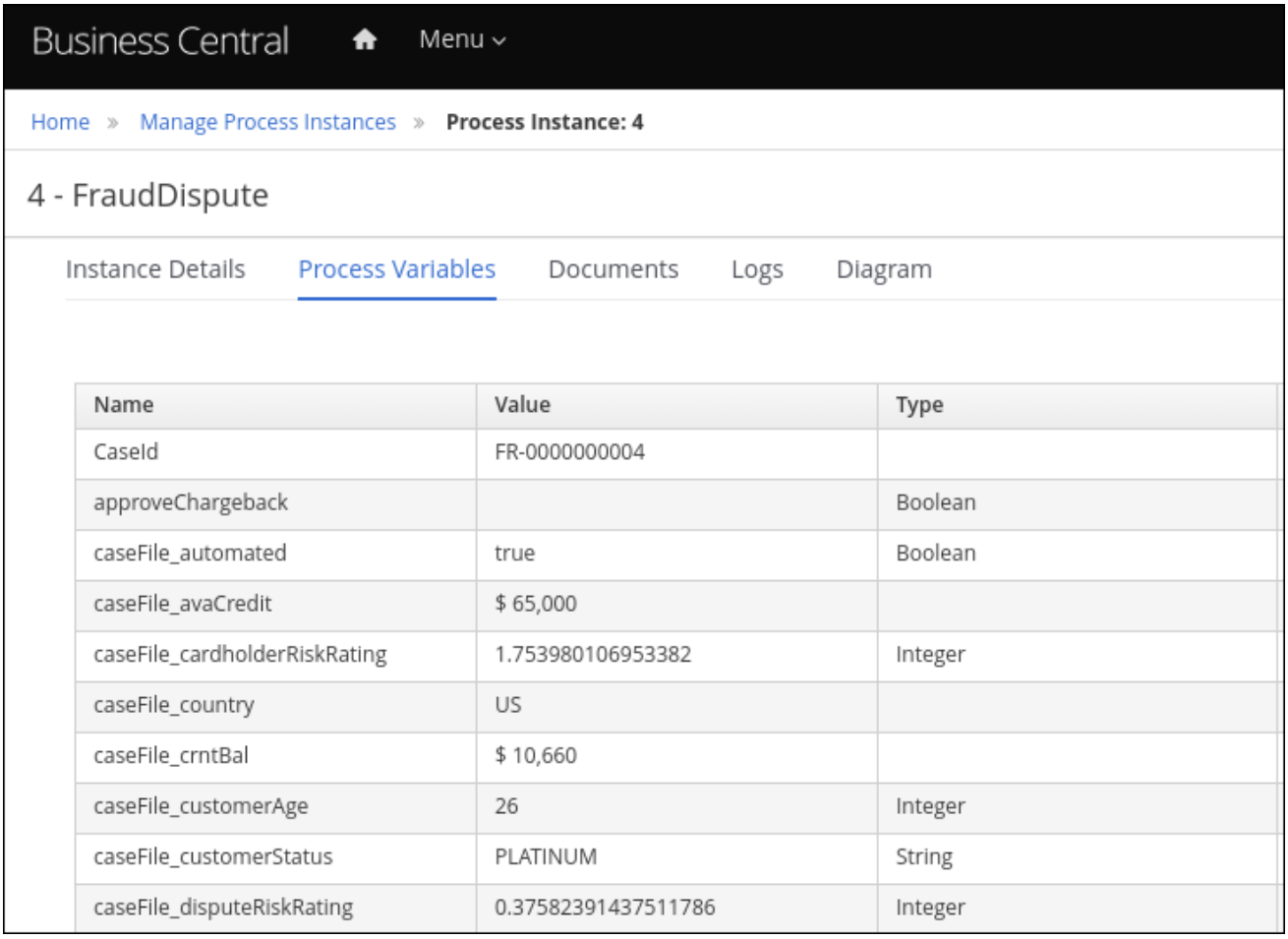
次のイメージは、Michelle の完成した DMN モデルです。これは、分析デシジョンテーブルを PMML ファイルの予測モデルに置き換えます。



Michelle は、BPMN モデル **fraudDispute** に戻り、追加した PMML ファイルでモデルを更新します。その後、プロジェクトを再デプロイします。

スコアの精度の向上

Michelle が PMML モデルを使用して Fortress Bank プロジェクトを再デプロイしたこの新しいシナリオでは、Joe が Fortress Bank アカウントにログインし、同じトランザクションを正しくないと報告するとどうなるかを確認できます。Business Central で、Michelle は **Process Instances** ウィンドウに移動するように促され、新しい紛争インスタンスを確認します。Michelle は、**Process Variables** タブで、**cardHolderRiskRating** の値および **disputeRiskRating** の値を確認します。モデルは、PMML ファイルを使用するようになったため、変更しています。これにより、履歴データに基づく機械学習モデルを利用することで、リスクをより正確に予測できます。同時に、銀行のポリシーは引き続き DMN 意思決定モデルによって実施されます。リスク予測子は指定されたしきい値を下回っており、この異議申し立てを自動的に処理できます。



Name	Value	Type
CaseId	FR-0000000004	
approveChargeback		Boolean
caseFile_automated	true	Boolean
caseFile_avaCredit	$ 65,000	
caseFile_cardholderRiskRating	1.753980106953382	Integer
caseFile_country	US	
caseFile_crntBal	$ 10,660	
caseFile_customerAge	26	Integer
caseFile_customerStatus	PLATINUM	String
caseFile_disputeRiskRating	0.37582391437511786	Integer

モニターリング

最後に、Fortress Bank は、Prometheus を使用してクレジットカードの異議申し立てに関するメトリックスを収集し、Grafana を使用してそのメトリックスをリアルタイムで視覚化します。モニターの上のセクションには、ビジネスメトリックの主要業績評価指標 (KPI) が表示され、下のセクションには、運用メトリックの KPI が表示されます。

92.1. PMML モデルと DMN モデルを使用して、クレジットカード取引の異議申し立てを解決

この例は、Red Hat Process Automation Manager を使用して、PMML モデルを使用してクレジットカード取引の異議申し立てを解決する DMN モデルを作成する方法を示しています。顧客がクレジットカード取引に異議を唱えると、システムは取引を自動的に処理するかどうかを決定します。

前提条件

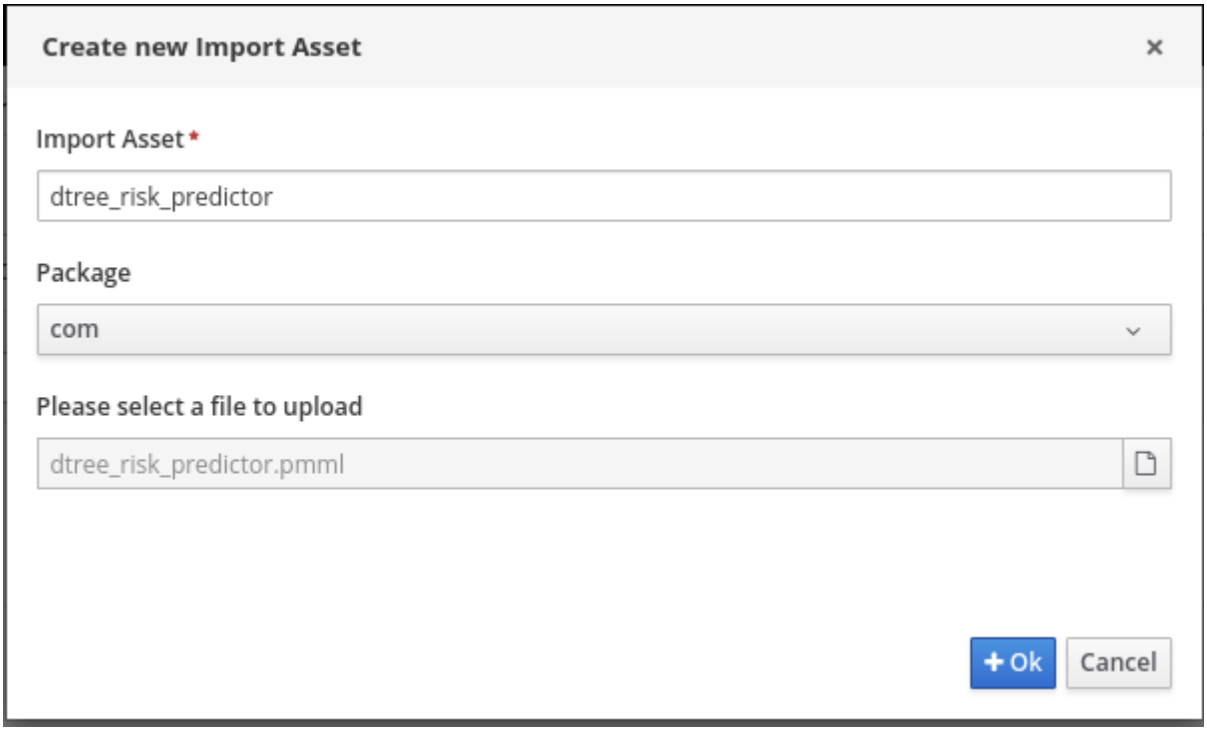
- Red Hat Process Automation Manager が使用可能であり、次の JAR ファイルが Red Hat Process Automation Manager の **~/kie-server.war/WEB-INF/lib** ディレクトリーおよび **~/business-central.war/WEB-INF/lib** ディレクトリーに追加されている。
  - **kie-dmn-jpmml-7.52.0.Final-redhat-00007.jar**
    このファイルは、Red Hat カスタマーポータルの Software Downloads ページから利用できる Red Hat Decision Manager 7.11 Maven リポジトリーディストリビューションで利用できます (ログインが必要です)。このファイルのグループ ID、アーティファクト ID、およびバージョン (GAV) 識別子は、**org.kie:kie-dmn-jpmml:7.52.0.Final-redhat-00007** です。詳細は、**DMN モデルを使用したデシジョンサービスの作成** の Business Central の DMN ファイルへの PMML モデルの追加セクションを参照してください。
  - JPMML Evaluator 1.5.1 JAR ファイル
  - JPMML Evaluator Extensions 1.5.1 JAR ファイル
    これらのファイルは、KIE Server と Business Central で JPMML 評価を有効にするのに必要です。

重要

Red Hat は、Red Hat Process Automation Manager で PMML を実行するため、Java Evaluator API for PMML (JPMML) との統合をサポートしています。しかし、Red Hat は JPMML ライブラリーを直接サポートしません。Red Hat Process Automation Manager ディストリビューションに JPMML ライブラリーを含む場合は、JPMML の Openscoring.io ライセンス条件を確認してください。

手順

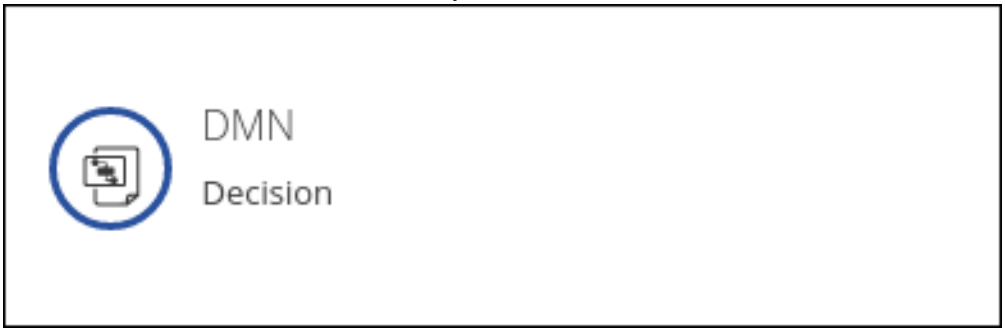
1. 「クレジットカード取引紛争の例 PMML ファイル」 の XML 例の内容で **dtree_risk_predictor.pmml** ファイルを作成します。
2. Business Central で **Credit Card Dispute** プロジェクトを作成します。
   a. **Menu → Design → Projects** に移動します。
   b. **Add Project** をクリックします。
   c. **Name** ボックスに **Credit Card Dispute** と入力し、**Add** をクリックします。
3. **Credit Card Dispute** プロジェクトの **Assets** ウィンドウで、**dtree_risk_predictor.pmml** ファイルを **com** パッケージにインポートします。



a. **Import Asset** をクリックします。

b. **Create new Import Asset** ダイアログで、**Name** ボックスに **dtree_risk_predictor** と入力し、**Package** メニューから **com** を選択し、**dtree_risk_predictor.pmml** ファイルを選択して **OK** をクリックします。
dtree_risk_predictor.pmml ファイルの内容が **Overview** ウィンドウに表示されます。

4. **com** パッケージで DMN モデル **Dispute Transaction Check** を作成します。



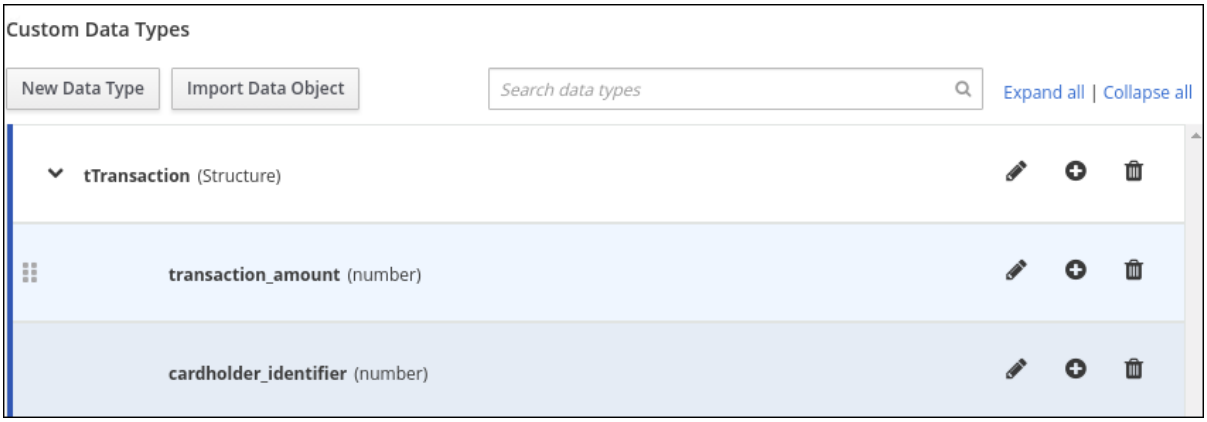
a. プロジェクトウィンドウに戻り、ブレッドクラムの **Credit Card Dispute** をクリックします。

b. **Add Asset** をクリックします。

c. アセットライブラリーで **DMN** をクリックします。

d. **Create new DMN** ダイアログで、**Name** ボックスに **Dispute Transaction Check** と入力し、**Package** メニューから **com** を選択して **OK** をクリックします。
DMN モデル **Dispute Transaction Check** で DMN エディターが開きます。

5. **tTransaction** カスタムデータタイプを作成します。



a. Data Types タブをクリックします。

b. Add a custom Data Type をクリックします。

c. Name ボックスに、**tTransaction** を入力します。

d. Type メニューから Structure を選択します。

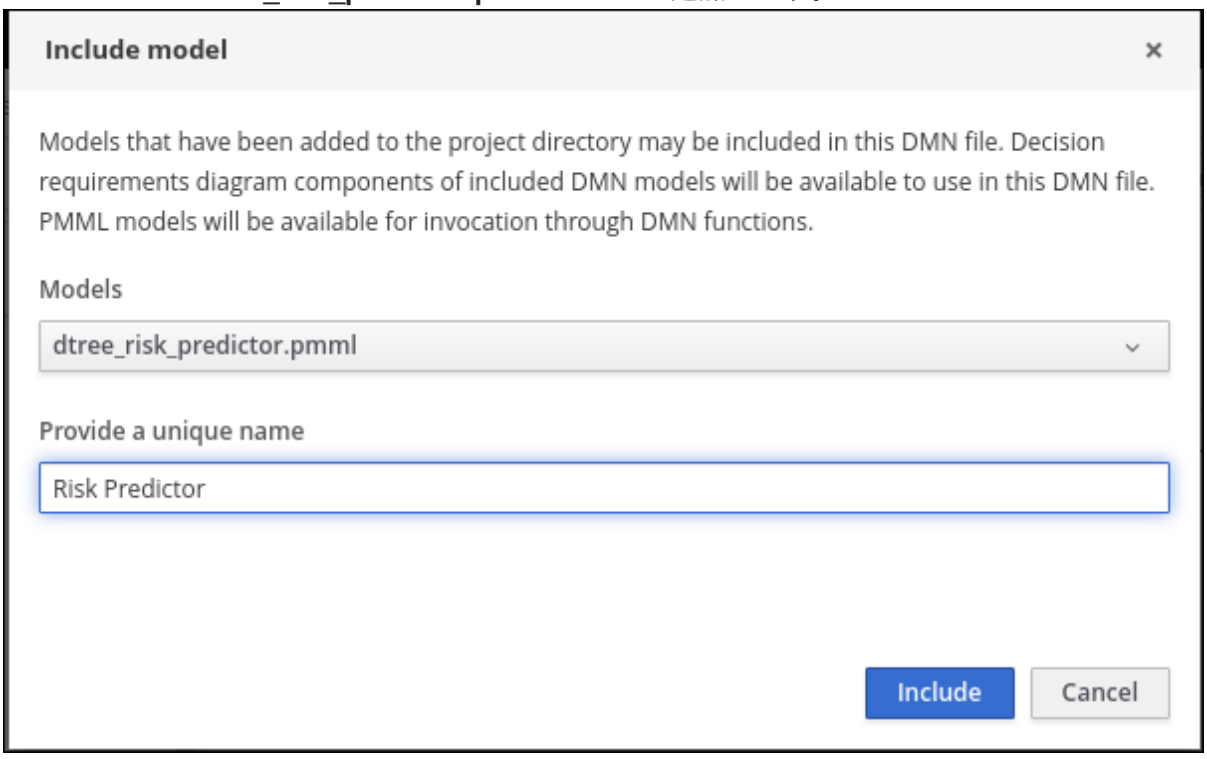
e. データ型を作成するには、チェックマークをクリックします。
tTransaction カスタムデータタイプは、1 つの変数行で表示されます。

f. 変数行の Name フィールドに **transaction_amount** と入力し、Type メニューから Number を選択してからチェックマークをクリックします。

g. 新しい変数行を追加するには、**transaction_amount** 行のプラス記号をクリックします。新しい行が表示されます。

h. Name フィールドに **cardholder_identifier** と入力し、Type メニューから Number を選択してからチェックマークをクリックします。

6. Risk Predictor **dtree_risk_predictor.pmml** モデルを追加します。

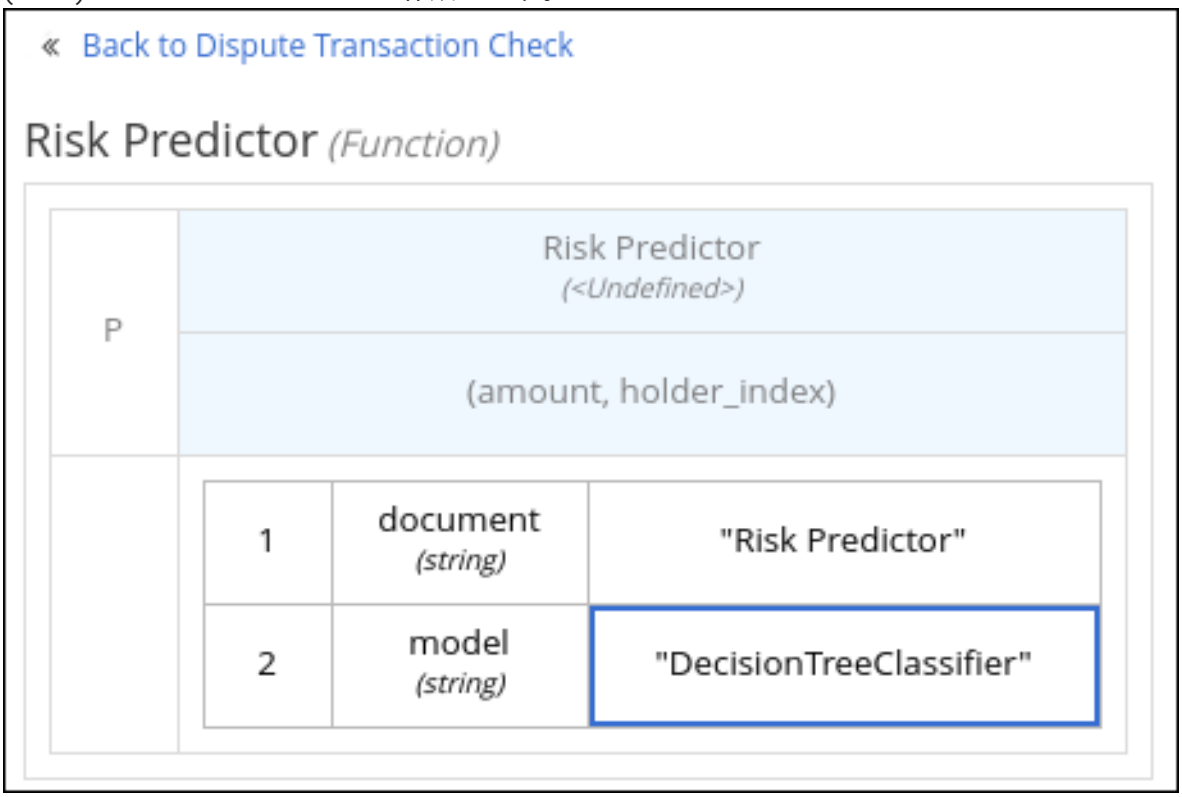


a. DMN エディターの Included Models ウィンドウで Include Model をクリックします。

b. **Include Model** ダイアログで、**Models** メニューから **dtree_risk_predictor.pmml** を選択します。

c. **Provide a unique name** ボックスに **Risk Predictor** と入力し、**OK** をクリックします。

7. **Risk Predictor** モデルおよび **DecisionTreeClassifier** モデルで、Business Knowledge Model (BKM) ノード **Risk Predictor** を作成します。



a. DMN エディターの **Model** ウィンドウで、BKM ノードを DMN エディターパレットにドラッグします。

b. **Risk Predictor** ノードの名前を変更します。

c. ノードの左側にあるごみ箱の配下にある編集アイコンをクリックします。

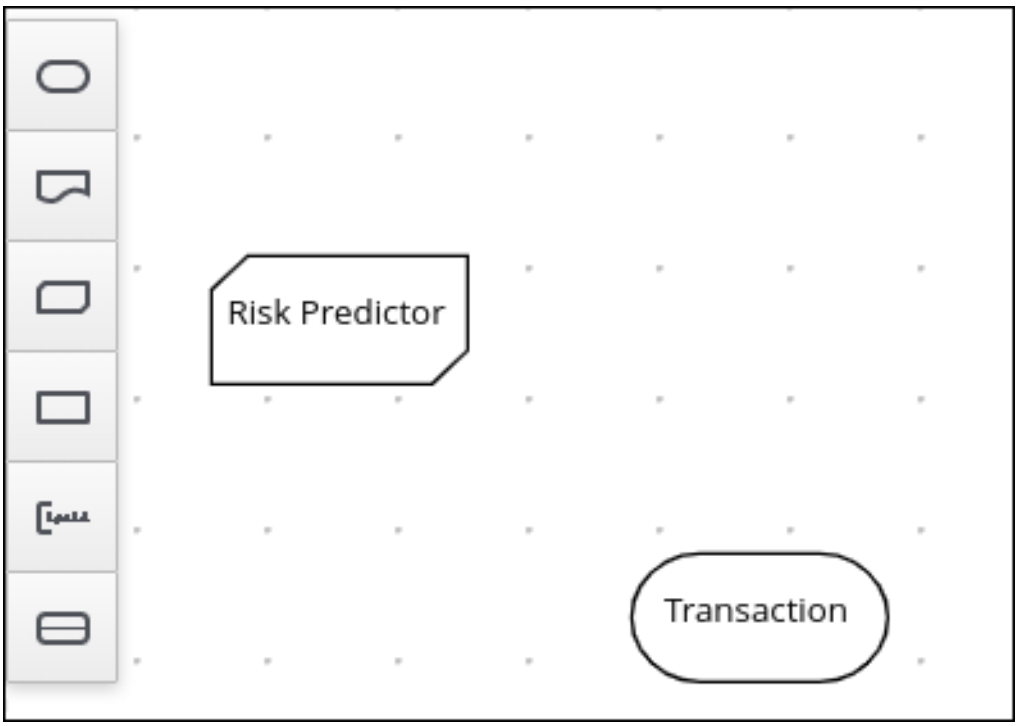
d. **Risk Predictor** ボックスで **F** をクリックし、**Select Function Kind** メニューから **PMML** を選択します。**F** が **P** に変わります。

e. **First select PMML document** ボックスをダブルクリックして、**Risk Predictor** を選択します。

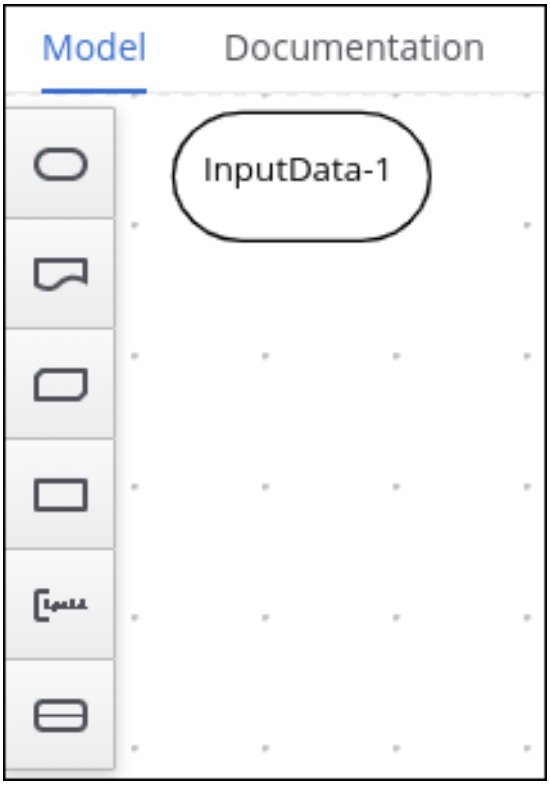
f. **Second select PMML model** ボックスをダブルクリックして、**DecisionTreeClassifier** を選択します。

g. DMN エディターパレットに戻るには、**Back to Dispute Transaction Check**をクリックします。

8. データ型 **tTransaction** で **Transaction** 入力データノードを作成します。



a. DMN エディターの **Model** ウィンドウで、入力データノードを DMN エディターパレットにドラッグします。

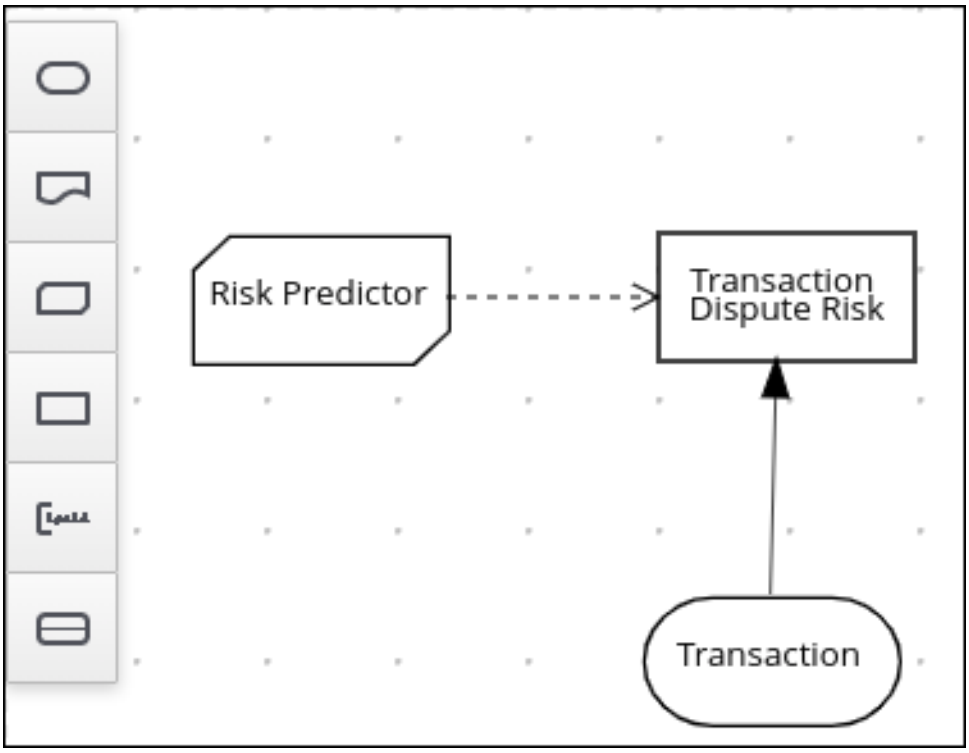


b. **トランザクション** ノードの名前を変更します。

c. ノードを選択して、ウィンドウの右上隅にあるプロパティーの鉛筆アイコンをクリックします。

d. **Properties** パネルで **Information Item → Data type → tTransaction**を選択し、パネルを閉じます。

9. **Transaction Dispute Risk**デシジョンノードを作成し、データ入力用の **Transaction** ノードと、関数の **Risk Predictor** ノードを追加します。



a. DMN エディターの **Model** ウィンドウで、デシジョンデータノードを DMN エディターパレットにドラッグします。

b. ノードの名前を **Transaction Dispute Risk** に変更します。

c. **Risk Predictor** ノードを選択し、ノードの右上にある矢印を **Transaction Dispute Risk** ノードをドラッグします。

d. **Transaction** ノードを選択し、ノードの右上にある矢印を **Transaction Dispute Risk** にドラッグします。

10. **Transaction Dispute Risk** ノードで、**Risk predictor** 予測関数を作成します。

#	Transaction Dispute Risk *(<Undefined>)*	
	Risk Predictor	
1	amount *(number)*	Transaction.transaction_amount
2	holder_index *(number)*	Transaction.cardholder_identifier

a. **Transaction Dispute Risk** ノードを選択し、ノードの左側にある編集アイコンをクリックします。

b. **Select expression** をクリックし、メニューから **Invocation** を選択します。

c. **Enter function** ボックスに **Risk Predictor** と入力します。

d. **P1** をクリックします。

e. **Edit Parameter** ダイアログボックスで、**Name** ボックスに **amount** と入力し、**Data Type** メニューから **number** を選択して、Enter キーを押します。

f. **Select expression** をクリックし、メニューから **Literal expression** を選択します。

g. **amount** の横にあるボックスに **Transaction.transaction_amount** を入力します。

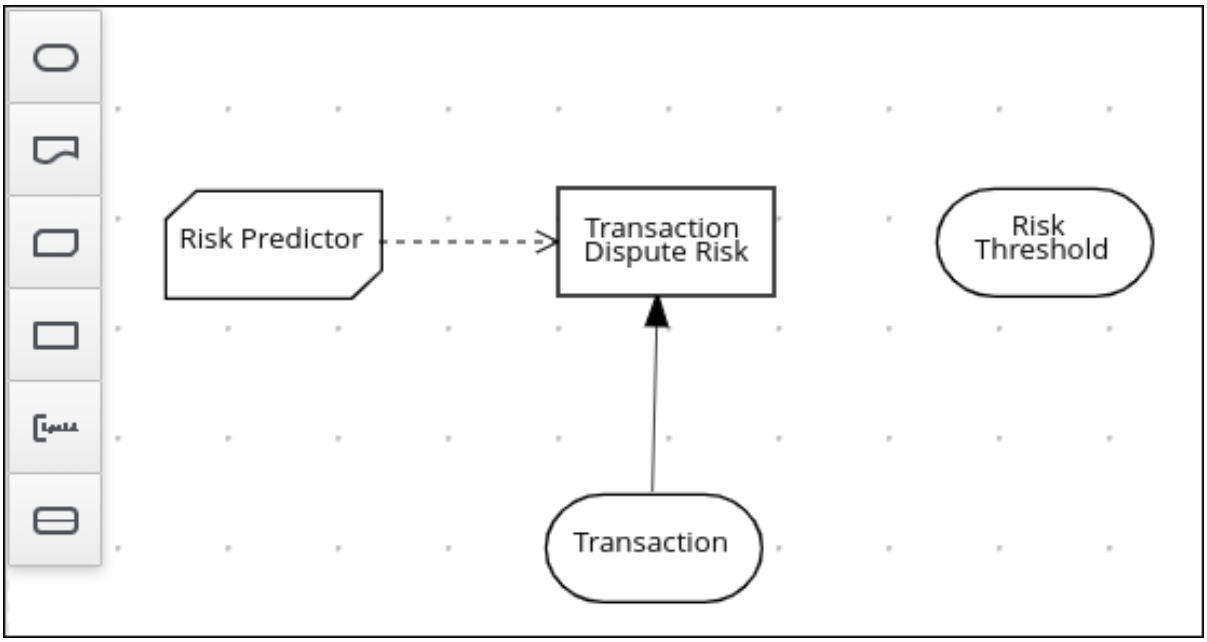
h. **1** を右クリックして、**Insert below** を選択します。パラメーターの **Edit Parameter** が開きます。

i. **Name** ボックスに **holder_index** と入力し、**Data Type** メニューから **number** を選択し、Enter キーを押します。

j. 行 **2** で **Select expression** をクリックし、メニューから **Literal expression** を選択します。

k. **amount** の横にあるボックスに **Transaction.cardholder_identifier** と入力します。

11. データ型 **number** で、入力データノード **Risk Treshold** を作成します。



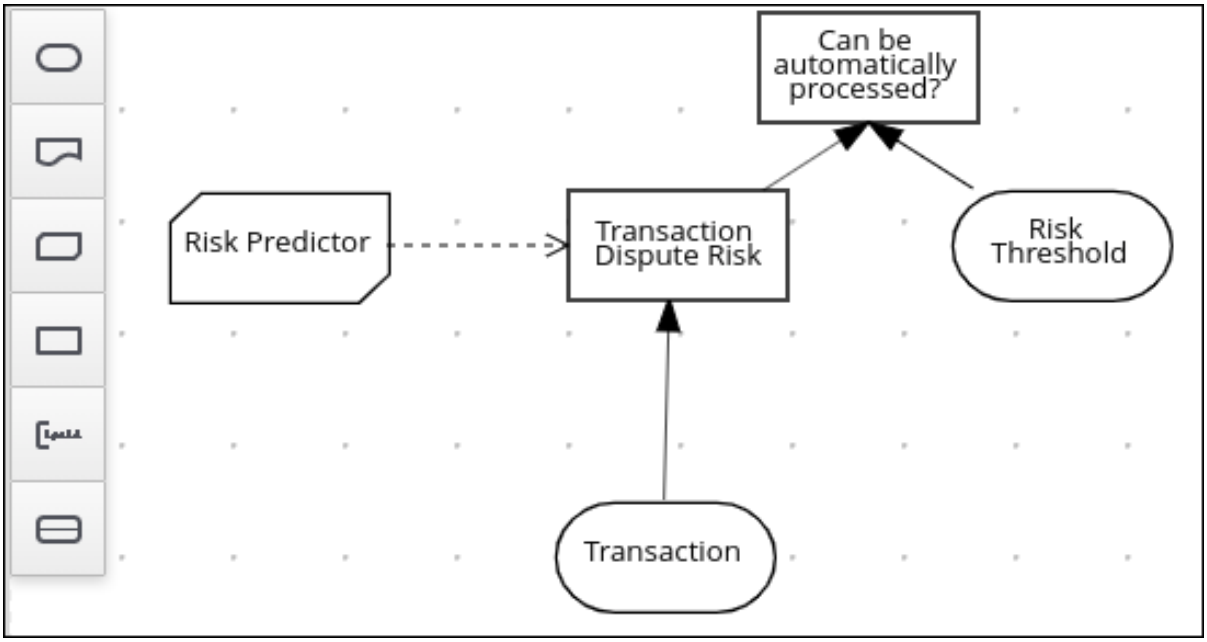
a. DMN エディターの **Model** ウィンドウで、入力データノードを DMN エディターパレットにドラッグします。

b. **Risk Threshold** ノードの名前を変更します。

c. ノードを選択して、ウィンドウの右上隅にあるプロパティーの鉛筆アイコンをクリックします。

d. **Properties** パネルで **Information Item → Data type → number**の順に選択し、パネルを閉じます。

12. **Can be automatically processed?** デシジョンノードは、**Transaction Dispute Risk** ノードおよび **Risk threshold** ノードの入力として取ります。



a. デシジョンノードを DMN エディターパレットにドラッグし、名前を **Can be automatically processed?** に変更します。

b. ノードを選択し、ノードの左上にある編集アイコンをクリックします。

c. **Select expression** をクリックしてから、メニューから **Literal expression** を選択します。

d. ボックスに **Transaction Dispute Risk.predicted_dispute_risk < Risk Threshold** と入力します。

e. **Transaction Dispute Risk** ノードを選択し、ノードの左上にある矢印を **Can be automatically processed?** にドラッグします。

f. **Risk Threshold** ノードを選択し、ノードの左上にある矢印を **Can be automatically processed?** ノードにドラッグします。

13. モデルを保存し、プロジェクトをビルドします。

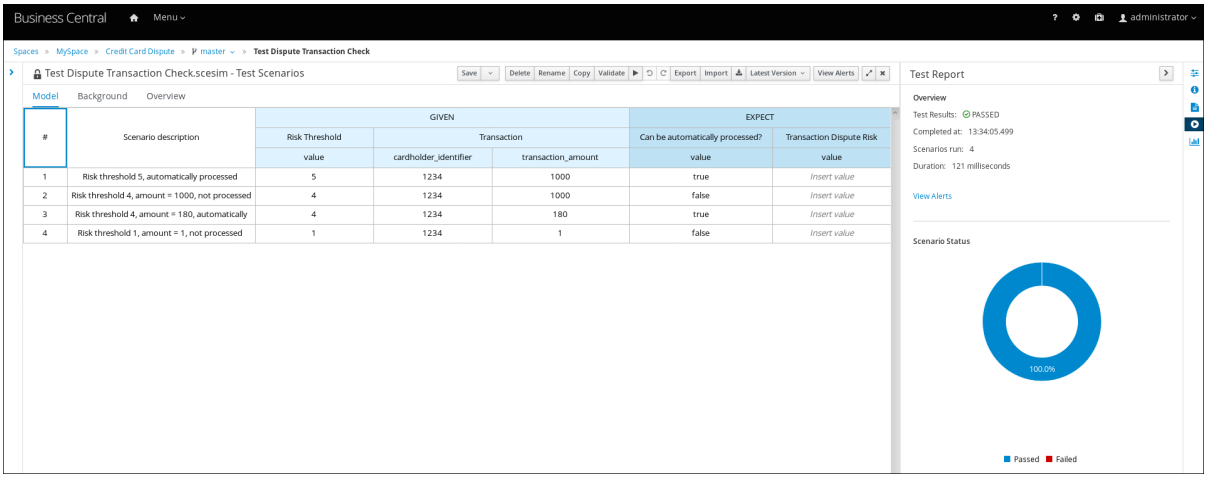
a. DMN エディターで **Save** をクリックします。

b. 必要な場合は、表示されるエラーを修正します。

c. プロジェクトウィンドウに戻り、ブレッドクラムの **Credit Card Dispute** をクリックします。

d. **Build** をクリックします。プロジェクトが正常にビルドされるはずです。

14. テストシナリオを追加して実行します。



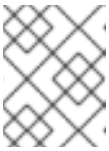
a. **Add Asset** をクリックします。

b. **Test Scenario** を選択します。

c. **Create new Test Scenario** ダイアログで、**Test Dispute Transaction Check** の名前を入力し、**Package** メニューから **com** を選択して **DMN** を選択します。

d. **Choose a DMN asset** メニューから **Dispute Transaction Check.dmn** を選択し、**OK** をクリックします。テスト用テンプレートがビルドされます。

e. 以下の値を入力して **Save** をクリックします。



注記

Transaction Dispute risk 列に値を追加しないでください。この値は、テストシナリオにより決定されます。

表92.1テストシナリオパラメーター

説明	リスクのしきい値	cardholder_identifier	transaction_amount	自動的に処理可能か
リスクしきい値5、自動的に処理される	5	1234	1000	true
リスクしきい値4、合計 = 1000、未処理	4	1234	1000	false
リスクしきい値4、合計 = 180、自動処理	4	1234	180	true
リスクしきい値1、合計 = 1、未処理	1	1234	1	false

f. テストを実行するには、**Validate** の右側にある **Play** ボタンをクリックします。結果は、画面の右側の **Test Report** パネルに表示されます。

92.2. クレジットカード取引紛争の例 PMML ファイル

以下の XML コンテンツを使用して、「PMML モデルと DMN モデルを使用して、クレジットカード取引の異議申し立てを解決」 演習で **dtree_risk_predictor.pmml** ファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML xmlns="http://www.dmg.org/PMML-4_2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd">
  <Header copyright="Copyright (c) 2018 Software AG" description="Default Description">
    <Application name="Nyoka" version="4.3.0" />
    <Timestamp>2020-10-09 14:27:26.622723</Timestamp>
  </Header>
  <DataDictionary numberOfFields="3">
    <DataField name="amount" optype="continuous" dataType="double" />
    <DataField name="holder_index" optype="continuous" dataType="double" />
    <DataField name="dispute_risk" optype="categorical" dataType="integer">
      <Value value="1" />
      <Value value="2" />
      <Value value="3" />
      <Value value="4" />
      <Value value="5" />
    </DataField>
  </DataDictionary>
  <TreeModel modelName="DecisionTreeClassifier" functionName="classification" missingValuePenalty="1.0">
    <MiningSchema>
      <MiningField name="amount" usageType="active" optype="continuous" />
      <MiningField name="holder_index" usageType="active" optype="continuous" />
      <MiningField name="dispute_risk" usageType="target" optype="categorical" />
    </MiningSchema>
    <Output>
      <OutputField name="probability_1" optype="continuous" dataType="double" feature="probability" value="1" />
      <OutputField name="probability_2" optype="continuous" dataType="double" feature="probability" value="2" />
      <OutputField name="probability_3" optype="continuous" dataType="double" feature="probability" value="3" />
      <OutputField name="probability_4" optype="continuous" dataType="double" feature="probability" value="4" />
      <OutputField name="probability_5" optype="continuous" dataType="double" feature="probability" value="5" />
      <OutputField name="predicted_dispute_risk" optype="categorical" dataType="integer" feature="predictedValue" />
    </Output>
    <Node id="0" recordCount="600.0">
      <True />
      <Node id="1" recordCount="200.0">
        <SimplePredicate field="amount" operator="lessOrEqual" value="99.94000244140625" />
        <Node id="2" score="2" recordCount="55.0">
          <SimplePredicate field="holder_index" operator="lessOrEqual" value="0.5" />
          <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
```

```
        <ScoreDistribution value="2" recordCount="55.0" confidence="1.0" />
        <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
        <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
        <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
       </Node>
       <Node id="3" score="1" recordCount="145.0">
        <SimplePredicate field="holder_index" operator="greaterThan" value="0.5" />
        <ScoreDistribution value="1" recordCount="145.0" confidence="1.0" />
        <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
        <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
        <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
        <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
       </Node>
      </Node>
      <Node id="4" recordCount="400.0">
       <SimplePredicate field="amount" operator="greaterThan" value="99.94000244140625" />
       <Node id="5" recordCount="105.0">
        <SimplePredicate field="holder_index" operator="lessOrEqual" value="0.5" />
        <Node id="6" score="3" recordCount="54.0">
         <SimplePredicate field="amount" operator="lessOrEqual" value="150.4550018310547" />
         <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="3" recordCount="54.0" confidence="1.0" />
         <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
        </Node>
        <Node id="7" recordCount="51.0">
         <SimplePredicate field="amount" operator="greaterThan" value="150.4550018310547" />
         <Node id="8" recordCount="40.0">
          <SimplePredicate field="amount" operator="lessOrEqual" value="200.00499725341797" />
          <Node id="9" recordCount="36.0">
           <SimplePredicate field="amount" operator="lessOrEqual" value="195.4949951171875" />
           <Node id="10" recordCount="2.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="152.2050018310547" />
            <Node id="11" score="4" recordCount="1.0">
             <SimplePredicate field="amount" operator="lessOrEqual" value="151.31500244140625"
/>
             <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
             <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
            <Node id="12" score="3" recordCount="1.0">
             <SimplePredicate field="amount" operator="greaterThan" value="151.31500244140625"
/>
             <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
             <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
           </Node>
           <Node id="13" recordCount="34.0">
            <SimplePredicate field="amount" operator="greaterThan" value="152.2050018310547" />
            <Node id="14" recordCount="20.0">
```

```
              <SimplePredicate field="amount" operator="lessOrEqual" value="176.5050048828125"
/>
              <Node id="15" recordCount="19.0">
               <SimplePredicate field="amount" operator="lessOrEqual" value="176.06500244140625"
/>
               <Node id="16" score="4" recordCount="9.0">
                <SimplePredicate field="amount" operator="lessOrEqual" value="166.6449966430664"
/>
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="9.0" confidence="1.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
               <Node id="17" recordCount="10.0">
                <SimplePredicate field="amount" operator="greaterThan" value="166.6449966430664"
/>
                <Node id="18" score="3" recordCount="1.0">
                 <SimplePredicate field="amount" operator="lessOrEqual"
value="167.97999572753906" />
                 <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
                 <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                </Node>
                <Node id="19" score="4" recordCount="9.0">
                 <SimplePredicate field="amount" operator="greaterThan"
value="167.97999572753906" />
                 <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="4" recordCount="9.0" confidence="1.0" />
                 <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                </Node>
               </Node>
              </Node>
              <Node id="20" score="3" recordCount="1.0">
               <SimplePredicate field="amount" operator="greaterThan" value="176.06500244140625"
/>
               <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
               <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
               <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
               <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
               <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
              </Node>
             </Node>
             <Node id="21" score="4" recordCount="14.0">
              <SimplePredicate field="amount" operator="greaterThan" value="176.5050048828125"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="4" recordCount="14.0" confidence="1.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
             </Node>
```

```
          </Node>
         </Node>
         <Node id="22" recordCount="4.0">
          <SimplePredicate field="amount" operator="greaterThan" value="195.4949951171875" />
          <Node id="23" score="3" recordCount="1.0">
           <SimplePredicate field="amount" operator="lessOrEqual" value="195.76499938964844"
/>
           <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
           <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
          <Node id="24" recordCount="3.0">
           <SimplePredicate field="amount" operator="greaterThan" value="195.76499938964844"
/>
           <Node id="25" score="4" recordCount="1.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="196.74500274658203"
/>
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
           </Node>
           <Node id="26" recordCount="2.0">
            <SimplePredicate field="amount" operator="greaterThan" value="196.74500274658203"
/>
            <Node id="27" score="3" recordCount="1.0">
             <SimplePredicate field="amount" operator="lessOrEqual" value="197.5800018310547"
/>
             <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
             <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
            <Node id="28" score="4" recordCount="1.0">
             <SimplePredicate field="amount" operator="greaterThan" value="197.5800018310547"
/>
             <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
             <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
           </Node>
          </Node>
         </Node>
        </Node>
        <Node id="29" score="5" recordCount="11.0">
         <SimplePredicate field="amount" operator="greaterThan" value="200.00499725341797" />
         <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
```

```
          <ScoreDistribution value="5" recordCount="11.0" confidence="1.0" />
         </Node>
        </Node>
       </Node>
       <Node id="30" recordCount="295.0">
        <SimplePredicate field="holder_index" operator="greaterThan" value="0.5" />
        <Node id="31" score="2" recordCount="170.0">
         <SimplePredicate field="amount" operator="lessOrEqual" value="150.93499755859375" />
         <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="2" recordCount="170.0" confidence="1.0" />
         <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
         <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
        </Node>
        <Node id="32" recordCount="125.0">
         <SimplePredicate field="amount" operator="greaterThan" value="150.93499755859375" />
         <Node id="33" recordCount="80.0">
          <SimplePredicate field="holder_index" operator="lessOrEqual" value="2.5" />
          <Node id="34" recordCount="66.0">
           <SimplePredicate field="amount" operator="lessOrEqual" value="199.13500213623047" />
           <Node id="35" score="3" recordCount="10.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="155.56999969482422"
/>
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="10.0" confidence="1.0" />
            <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
           </Node>
           <Node id="36" recordCount="56.0">
            <SimplePredicate field="amount" operator="greaterThan" value="155.56999969482422"
/>
            <Node id="37" score="2" recordCount="1.0">
             <SimplePredicate field="amount" operator="lessOrEqual" value="155.9000015258789"
/>
             <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
             <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
             <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
            <Node id="38" recordCount="55.0">
             <SimplePredicate field="amount" operator="greaterThan" value="155.9000015258789"
/>
             <Node id="39" recordCount="31.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="176.3699951171875"
/>
              <Node id="40" recordCount="30.0">
               <SimplePredicate field="amount" operator="lessOrEqual"
value="175.72000122070312" />
               <Node id="41" recordCount="19.0">
                <SimplePredicate field="amount" operator="lessOrEqual"
value="168.06999969482422" />
                <Node id="42" recordCount="6.0">
                 <SimplePredicate field="amount" operator="lessOrEqual" value="158.125" />
                 <Node id="43" score="3" recordCount="5.0">
```

```
                    <SimplePredicate field="amount" operator="lessOrEqual"
value="157.85499572753906" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="5.0" confidence="1.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                   </Node>
                   <Node id="44" score="2" recordCount="1.0">
                    <SimplePredicate field="amount" operator="greaterThan"
value="157.85499572753906" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                    <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                   </Node>
                  </Node>
                  <Node id="45" score="3" recordCount="13.0">
                   <SimplePredicate field="amount" operator="greaterThan" value="158.125" />
                   <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="3" recordCount="13.0" confidence="1.0" />
                   <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                 </Node>
                 <Node id="46" recordCount="11.0">
                  <SimplePredicate field="amount" operator="greaterThan"
value="168.06999969482422" />
                  <Node id="47" score="2" recordCount="1.0">
                   <SimplePredicate field="amount" operator="lessOrEqual"
value="168.69499969482422" />
                   <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                   <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                   <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                  <Node id="48" recordCount="10.0">
                   <SimplePredicate field="amount" operator="greaterThan"
value="168.69499969482422" />
                   <Node id="49" recordCount="4.0">
                    <SimplePredicate field="holder_index" operator="lessOrEqual" value="1.5" />
                    <Node id="50" score="2" recordCount="1.0">
                     <SimplePredicate field="amount" operator="lessOrEqual"
value="172.0250015258789" />
                     <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                     <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                    <Node id="51" score="3" recordCount="3.0">
                     <SimplePredicate field="amount" operator="greaterThan"
value="172.0250015258789" />
```

```
                        <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="3" recordCount="3.0" confidence="1.0" />
                        <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                       </Node>
                      </Node>
                      <Node id="52" score="3" recordCount="6.0">
                       <SimplePredicate field="holder_index" operator="greaterThan" value="1.5" />
                       <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                       <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                       <ScoreDistribution value="3" recordCount="6.0" confidence="1.0" />
                       <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                       <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                      </Node>
                     </Node>
                    </Node>
                   </Node>
                   <Node id="53" score="2" recordCount="1.0">
                    <SimplePredicate field="amount" operator="greaterThan"
value="175.72000122070312" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                    <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                   </Node>
                  </Node>
                  <Node id="54" recordCount="24.0">
                   <SimplePredicate field="amount" operator="greaterThan" value="176.3699951171875"
/>
                   <Node id="55" score="3" recordCount="16.0">
                    <SimplePredicate field="amount" operator="lessOrEqual" value="192.0999984741211"
/>
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="16.0" confidence="1.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                   </Node>
                   <Node id="56" recordCount="8.0">
                    <SimplePredicate field="amount" operator="greaterThan" value="192.0999984741211"
/>
                    <Node id="57" score="2" recordCount="1.0">
                     <SimplePredicate field="amount" operator="lessOrEqual"
value="192.75499725341797" />
                     <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                     <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                     <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                    <Node id="58" score="3" recordCount="7.0">
                     <SimplePredicate field="amount" operator="greaterThan"
value="192.75499725341797" />
                     <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
```

```
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="7.0" confidence="1.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                   </Node>
                  </Node>
                 </Node>
                </Node>
               </Node>
               <Node id="59" recordCount="14.0">
                <SimplePredicate field="amount" operator="greaterThan" value="199.13500213623047" />
                <Node id="60" score="5" recordCount="10.0">
                 <SimplePredicate field="holder_index" operator="lessOrEqual" value="1.5" />
                 <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="5" recordCount="10.0" confidence="1.0" />
                </Node>
                <Node id="61" score="4" recordCount="4.0">
                 <SimplePredicate field="holder_index" operator="greaterThan" value="1.5" />
                 <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                 <ScoreDistribution value="4" recordCount="4.0" confidence="1.0" />
                 <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                </Node>
               </Node>
              </Node>
              <Node id="62" recordCount="45.0">
               <SimplePredicate field="holder_index" operator="greaterThan" value="2.5" />
               <Node id="63" score="2" recordCount="37.0">
                <SimplePredicate field="amount" operator="lessOrEqual" value="199.13999938964844" />
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="37.0" confidence="1.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
               <Node id="64" score="4" recordCount="8.0">
                <SimplePredicate field="amount" operator="greaterThan" value="199.13999938964844" />
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="8.0" confidence="1.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
              </Node>
             </Node>
            </Node>
           </Node>
          </Node>
         </TreeModel>
        </PMML>
```

第93章 関連情報

- **ケース管理の使用ガイド**
- **デシジョンサービスのスタートガイド**
- **DMN モデルを使用したデシジョンサービスの作成**
- **Red Hat Process Automation Manager でのソルバーの開発**
- Predictions 2019: Expect A Pragmatic Vision Of AI

付録A バージョン情報

本書の最終更新日: 2022 年 3 月 8 日 (火)

付録B お問い合わせ先

Red Hat Process Automation Manager のドキュメントチーム: brms-docs@redhat.com