



Kogito

Kogito Workshop

Ing. Adriel Paredes <aparedes@redhat.com>

Senior Software Engineer @ Business Systems And Intelligence Group, Red Hat Inc.

KIE Group Core Developer

Table of Contents

Kogito Workshop

Introducción Teórica

- Sistemas Expertos

- Paradigma Lógico

Manos a la Obra

- Caso de Estudio (COVID-19)

- Herramientas

- Entendiendo el Dominio

- DRL (Drools Rules Language)

- Decision Table

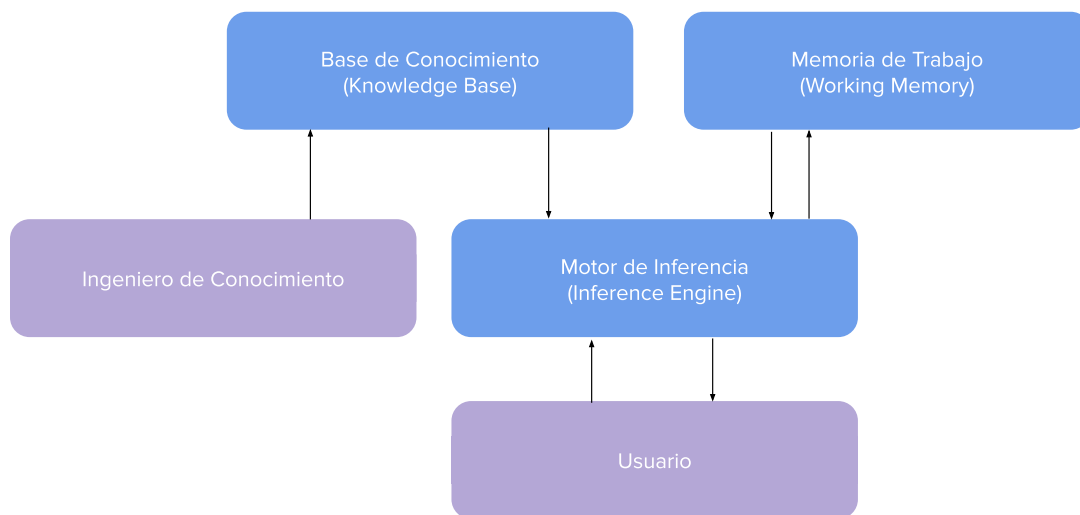
- DMN (Decision Model and Notation)

- Links útiles

Introducción Teórica

Sistemas Expertos

Un sistema experto es un sistema que emula el razonamiento humano actuando tal y como lo haría un experto. Es una rama de la inteligencia artificial que se encarga de simular ese comportamiento extrayendo ese conocimiento y almacenándolo en forma de reglas para que luego, mediante un motor de inferencia y la base de conocimiento se pueda resolver el problema en cuestión.



Estos sistemas están compuestos de 3 grandes componentes:

- La base de conocimientos
- La memoria de trabajo
- El motor de inferencia

En conjunto forman lo que llamamos sistema experto. Sin embargo para que esto funcione no solamente se necesita el software, sino que como dijimos previamente necesitamos al Ingeniero en Conocimiento que es un factor muy importante para la extracción y traducción del conocimiento en computaciones que el motor comprenda.

Base de Conocimiento

Es el encargado de almacenar el conocimiento del experto. El ingeniero de conocimiento encargado de extraer la información del experto deberá traducirlas a un lenguaje que el sistema comprenda para así poder dar solución al problema en cuestión. En el caso de Drools existen varios tipos de opciones diferentes para escribir ese conocimiento: en forma de reglas, en forma de tablas de decisión, DMN, etc.

Memoria de Trabajo

Es el lugar donde residen los hechos o las entidades sobre las cuales las reglas van a inferir resultados o nuevos hechos. Son básicamente los datos.

Motor de Inferencia

Es el encargado de realizar el procesamiento. Utiliza la base de conocimiento y la memoria de trabajo para inferir nuevos hechos. En el caso de Kogito/Drools, utiliza el algoritmo PHREAK con dicho objetivo. Si quieren leer más pueden buscar información en este blog: <http://blog.athico.com/2014/01/which-rule-engine-algorithm-is-faster.html>

Ingeniero en Conocimiento

Es la conjunción de dos tareas. Comprender el dominio sobre el que se va a trabajar y conocer como funciona el motor a utilizar. Esto significa que el ingeniero en conocimiento va a tener que realizar tanto tareas de análisis, diseño y refinación del conocimiento como la de traducir dicho conocimiento en reglas, tablas, gráficos que el motor comprenda y que refleje el dominio que se intenta simular.

Paradigma Lógico

Comprender el paradigma lógico es una parte fundamental para poder escribir las reglas ya que es la base de los sistemas expertos. Escribir relaciones entre hechos mediante predicados que me permitan declarar el *qué* sin pensar en el *cómo* ya que de esto último se va a encargar el motor de inferencia. El paradigma lógico es declarativo y como un ejemplo de lo que dije anteriormente:

- Si yo tengo un conjunto de `Personas` y quiero saber cuales se llaman `Juan` de primer nombre. Solo me concentro en preguntar:
- `/personas(primer_nombre == "Juan")` y no en:

```
List<Persona> resultado = new ArrayList<>();
for (Persona persona: personas){
    if("Juan".equals(persona.getPrimerNombre())){
        resultado.add(persona);
    }
}
return resultado;
```

JAVA

Allí pueden ver la diferencia entre procedural imperativo, donde yo tengo que declarar explícitamente como va a ser el procedimiento de búsqueda y algo declarativo, donde yo solo pregunto al motor de inferencia quienes se llaman `Juan`.

Manos a la Obra

Caso de Estudio (COVID-19)

CS criterio 1

Toda persona de cualquier edad que presente dos o más de los siguientes síntomas:

- Fiebre (37.5°C o más)
- Tos
- Odinofagia (Dolor al tragar alimentos o líquidos)
- Dificultad respiratoria
- Anosmia/disgeusia de reciente aparición (Pérdida parcial o completa del sentido del olfato/gusto)
- Cefalea
- Diarrea y/o vómitos
- Este criterio incluye toda enfermedad respiratoria aguda severa sin otra etiología que explique completamente la presentación clínica

CS criterio 2

Toda persona que:

- Sea trabajador de salud
- Resida o trabaje en instituciones cerradas o de internación prolongada.^[1]
- Sea Personal esencial.^[2]
- Resida en barrios populares o pueblos originarios.^{[3][4]}
- Sea contacto estrecho de caso confirmado de COVID-19, que dentro de los 14 días * posteriores al contacto:
- Presente 1 o más de estos síntomas:

- fiebre (37.5°C o más)
- Tos
- Odinofagia
- dificultad respiratoria
- pérdida repentina del gusto o del olfato.

Herramientas

Kogito

Automatización de negocio Cloud-Native para la construcción de aplicaciones inteligentes. Te da la posibilidad de crear aplicaciones (utilizando Quarkus, Spring boot, KNative) embebiendo directamente los motores de reglas y procesos, con lo cual se pueden escribir las reglas de negocio en un proyecto java, y Kogito va a autogenerar los endpoints para poder consumir ese dominio particular dentro del motor a través de un servicio REST.

Como crear un proyecto Kogito

<https://kogito.kie.org/>

```
mvn archetype:generate \
  -DarchetypeGroupId=org.kie.kogito \
  -DarchetypeArtifactId=kogito-quarkus-archetype \
  -DgroupId=org.kogito -DartifactId=covid-19 \
  -DarchetypeVersion=0.16.0 \
  -Dversion=1.0-SNAPSHOT
```

BASH

Extensiones a instalar:

- `jim-moody.drools`
- `kie-group.vscode-extension-kogito-bundle`
- `redhat.vscode-quarkus`
- `redhat.java`

Una vez finalizado este proceso ya estamos listos para comenzar a trabajar.

Clonar el proyecto

Van a necesitar hacer un git clone del proyecto que se encuentra en: <https://github.com/adrielparedes/kogito-workshop>

Allí hay 2 carpetas:

- Starter
- Final

Starter es la que tienen que utilizar de base y Final es el workshop terminado.

Estructura de directorios

- `src/main/java`: Modelos de la aplicación
- `src/main/resources`: Toma de decisión

Entendiendo el Dominio

Para poder comenzar con un proyecto de reglas lo primero que hay que hacer es comprender el dominio para poder planificar que tipo de herramientas se utilizará en la construcción de las reglas. No es lo mismo hacer reglas en DRL, que en una tabla de decisión o en un DMN. Primero tenemos que entender que implica cada una de las herramientas para luego poder asociar el dominio con las distintas herramientas que tenemos.

DRL (Drools Rules Language)

https://docs.jboss.org/kogito/release/latest/html_single/

Estructura de una regla

```
package 1

import 2

function // Optional 3

declare // Optional 4

global // Optional 5

rule "rule name" 6
    // Attributes
    when 7
        // Conditions
    then 8
        // Actions
end
```

MVEL

- 1 Simplemente un nombre, un espacio de trabajo.
- 2 Los paquetes que se van a necesitar importar para poder trabajar dentro del las reglas
- 3 Sirve para escribir código sin tener que ponerlo en una clase Java. Se recomienda este método solo para cosas simples.
- 4 En esta sección se pueden declarar nuevos tipos de datos, en vez de hacerlos en una clase Java. Para mejor reutilización es preferible que sea una clase Java.
- 5 Global sirve para guardar variables de manera global dentro de la working memory.
- 6 El nombre específico de la regla. Tiene que ser único para poder reconocerla.
- 7 LHS (Left Hand Side). La cláusula condicional de la regla. Va a determinar el valor de verdad de la misma.
- 8 RHS <Right Hand Side). La acción de la regla. Va a ejecutar lo que se encuentre allí siempre y cuando el LHS sea verdadero.

Rule Unit

Es un *módulo* para reglas y una unidad de ejecución. Esto significa que este conjunto de reglas se va a ejecutar todo junto. Por lo general todas las reglas que se van a ejecutar todas juntas se guardan en el mismo módulo. Y también sirve como *Namespace*

Declare

Drools permite declarar entidades dentro del archivo DRL en vez de hacerlo en archivos Java separados. Siempre es recomendable la clase Java para mayor reutilización

```
declare Person
    name: string
end
```

DRL

Data Sources

Los Rule Units tiene que estar subscriptos a data sources para poder reaccionar a los cambios en la memoria de trabajo, para ello hay 3 tipos de data sources

Data Stream

Solo se podrán almacenar nuevos hechos dentro de este data source . Las entidades dentro de el no se podrán remove.

```
dataSouce.append(new Entity());
```

Data Store

En este tipo de data source se podrá tanto agregar como quitar elementos.

```
dataSouce.add(new Entity());
dataSouce.remove(anEntity);
```

JAVA

Singleton Store

En este almacenamiento solamente se puede asignar o borrar un único valor.

```
dataSouce.set(new Entity());
dataSouce.clear();
```

JAVA

Query

Las queries buscan dentro de la Memoria de Trabajo los hechos que concuerdan con la regla escrita dentro de ese archivo DRL. Al mismo tiempo Kogito crea por cada Query un endpoint con el nombre de la query.

```
query highSeverity
    alerts : /alertData[ severity == "HIGH" ]
end
```

DRL

En este caso el endpoint creado es: /high-severity

Decision Table

Es simplemente una tabla (como si fuese un excel) donde voy a poder escribir en cada columna las condiciones que va a tener la regla de negocio y por cada fila, el conjunto de valores que esa regla debería tomar para que sea verdadero.

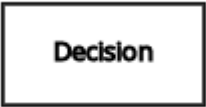
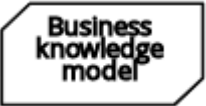


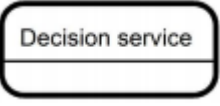
DMN (Decision Model and Notation)

Es un estándar establecido por la OMG (Object Management Group) para describir y modelar deciciones operacionales. DMN esta definido mediante un XML lo cual es posible utilizarlo en diferentes vendors siempre y cuando implementen dichos componentes.

Componentes de DMN

Elementos

Notación	Nombre	Descripción
----------	--------	-------------

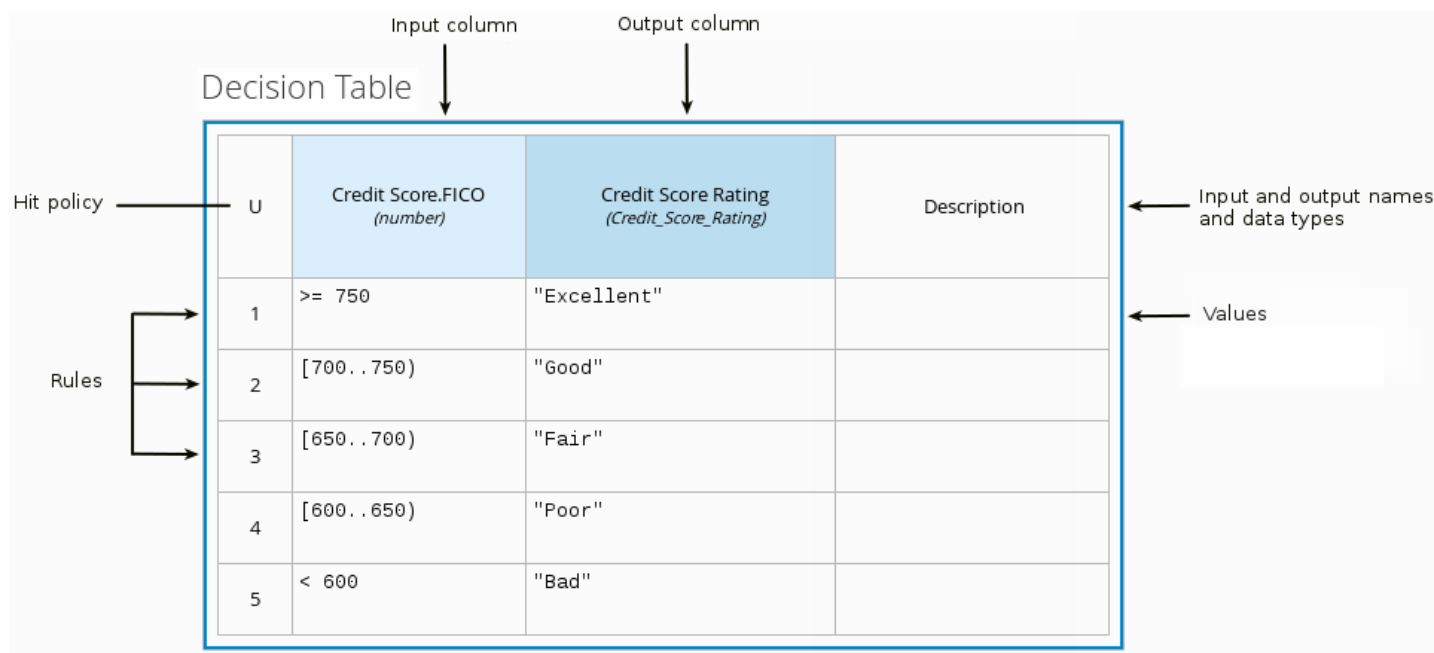
Notación	Nombre	Descripción
	Decision	Nodo donde uno o más elementos de entrada determinan una salida basado en la decisión lógica definida
	Business Knowledge Model	Función reutilizable con uno o más elementos de decisión. Tiene la misma lógica que los nodos de decisión.
	Knowledge Source	Elementos externos, documentos, comités o políticas que regulan la decisión o Business Knowledge Model. Son referencias a factores reales más que a reglas de negocios ejecutables.
	Input data	Información que se utilizará en el nodo de decisión o Business Knowledge Model.
	Decision Service	Decisión de alto nivel que contendrá un conjunto de decisiones reutilizables publicadas como servicio para su invocación.

FEEL

Significa Friendly Enough Expression Language y es un lenguaje definido por la OMG dentro de la especificación de DMN. Sirve para definir lógica de decision dentro de los modelos de DMN.

Tablas de Decision

Es una representación visual de una decision en formato de tabla. El contenido de los inputs o los outputs pueden ser tanto valores como expresiones FEEL.



Funciones

A diferencia de los nodos de decisión las funciones son elementos reutilizables que reciben parametros y que pueden ser invocadas dentro de los nodos de decisión. Las funciones pueden contener tanto expresiones FEEL, como decisiones más complejas o invocaciones a código Java.

Flight Capacity (Function)

F	Flight Capacity (boolean)
	(flight, rebooked list)
	<code>flight.Capacity > count(rebooked list[Flight Number = flight.Flight Number])</code>

Links útiles

- <http://learn-dmn-in-15-minutes.com/>

1. penitenciarias, residencias de adultos mayores, instituciones neuropsiquiátricas, hogares de niñas y niños
2. se considera personal esencial: Fuerzas de seguridad y Fuerzas Armadas Personas que brinden asistencia a personas mayores
3. Se considera barrio popular a aquellos donde la mitad de la población no cuenta con título de propiedad, ni acceso a dos o más servicios básicos.
4. Fuente: Registro Nacional de Barrios Populares