UNIDAD 5: TEORÍA DE DECISIONES

Trabajo Práctico 5: Reglas del Negocio (Decisiones Programadas)

RECURSOS:

Sitio de Drools: https://www.drools.org/

Documentación: https://www.drools.org/learn/documentation.html

Tutorial: http://www.tutorialspoint.com/drools/

Libros:

Introducción a Drools

En la mayoría de las empresas la lógica de negocio se encuentra dispersa por diferentes sitios: en el código de las aplicaciones, hojas de cálculo, en las mentes de los expertos en la materia, etc... Este hecho hace que, a la mayoría de las personas de la organización les sea complejo consultar y comprender las reglas que constituyen la base del negocio.

Los sistemas de gestión de reglas de negocio (**BRMS-Business Rules Management System)** como **Drools**, surgen ante la necesidad de **centralizar y gestionar la lógica de negocio**. Para ello, se codifica dicha lógica en forma de reglas de negocio, que sirven para tomar decisiones dentro de un contexto. Estas reglas se ejecutan dentro de un motor de reglas (**BRM-Business Rule Managemento BRE-Business Rules Engine**). Las reglas, por tanto, no sirven únicamente para representar la lógica de negocio, sino también para ejecutarla. Además, el hecho de que la lógica se encuentre codificada en una regla, hace más fácil su comprensión que cuando se encuentra en el código de una aplicación, sobre todo para el personal no técnico.

Fuente: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IntroduccionDrools

Desde una perspectiva de Sistemas de Información, las reglas de negocio cambian más frecuentemente que el código de una aplicación.

En resumen, los motores de reglas de negocio (Rules Engines o Inference Engines) son componentes de software que separan las reglas de negocio del código de aplicación, permitiendo que los usuarios modifiquen las mismas frecuentemente sin la necesidad de intervención de una persona con conocimientos técnicos, de manera de permitir que las aplicaciones sean más flexibles y adaptables.

La **Administración de reglas de negocio** surge como una disciplina de las áreas de Inteligencia Artificial y Sistemas Expertos. Típicamente, los **Sistemas Expertos** se construyen en torno a un **Motor de Inferencia** que aplica mecanismos como la deducción e inducción a partir de un conjunto de reglas de inferencia para llegar a una solución. Forma parte de una clase de sistemas inteligentes a los que se les denominan sistemas basados en reglas. A finales de 1990 surgen los primeros sistemas inteligentes denominados motores de evaluación de reglas, aportando su valor al externalizar de las aplicaciones cierta lógica del negocio a través de términos, hechos y reglas. **Un motor de evaluación de reglas es un tipo específico de Sistema Experto en donde el conocimiento se representa de la forma de reglas, generalmente como sentencias : (If <condición> then <sentencias>).** A mediados de la década del 2000, surge la necesidad de automatizar y administrar las reglas de negocio y decisiones; aparecen los **Gestores de reglas del negocio**.

Las reglas del negocio

Una regla del negocio es una clase de instrucción o comando que se aplica en una situación determinada; generalmente se pueden escribir con sentencias de la forma "when-then" (cuando/entonces). Cuando se cumple una condición, se desencadena una acción (que puede ser cualquier cosa). Ejemplo: *"Cuando un cliente que realiza un pedido es VIP, entonces se le realiza un descuento del 10%"*.

A la sección "when" se le denomina parte izquierda (LHS - Left-Hand Side), predicado o premisa, que consta de una serie de patrones que especifican los hechos (o datos) que causan que la regla sea aplicable.

A la sección "then" se le denomina parte derecha (RHS - Right-Hand Side), acciones o conclusiones, que detalla las acciones que se deben realizar en caso de que se satisfagan las premisas (sección "when").

Estructura de una Regla

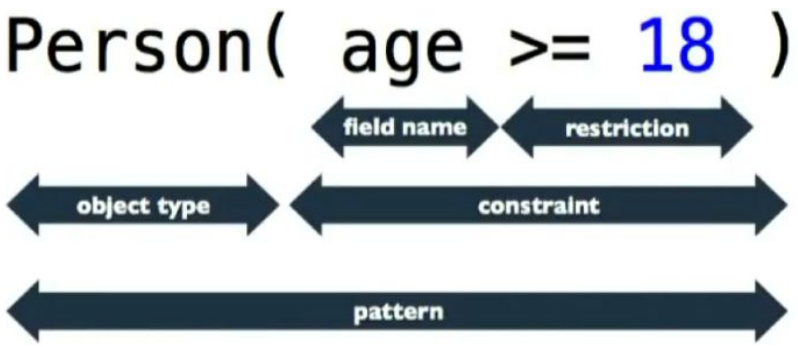
```
rule <rule_name>
    <attribute><value>
    when
        <conditions>
    then
        <actions>
end
```

```
rule "My Rule"
    <attributes>
    when <LHS>
      Person(name == "John") <CEs>
    then  <RHS>
      System.out.println("Hi John!");
      <Actions>
end
```

Ejemplo de Regla

```
rule "Infer Adult"
    when
        $p : Person( age >= 18 )
    then
        System.out.println("Adult!")
end
```

Pattern Matching (Coincidencia de Patrón)



Ejemplo de Reglas

```
rule "Infer Child"
    when
        Person( age < 18 )
    then
        //code
end
```

```
rule "Infer Baby"
    when
        Person( age <= 2 )
    then
        //code
end
```

```
rule "Infer Adult"
    when
        Person( age >= 18 )
    then
        //code
end
```

¿Con qué regla hay Pattern Matching si ingresamos a la memoria de trabajo un objeto Person con atributo age con el valor 22?

Ejemplo Licencia de Conducir

```
package com.company.license

rule "Is of valid age"
when
    $a : Applicant( age < 18 )
then
    $a.setValid( false );
end
```

```
public class Applicant {

    private String name;

    private int age;

    private boolean valid;

    // getter and setter methods here

}
```

Diferencia entre un método y una regla

- Los métodos son llamados directamente.
- Se pasan instancias específicas de objetos.
- Una llamada resulta en una simple ejecución.

```
public void helloWorld(Person person) {
    if ( person.getName().equals( "Chuck" ) ) {
        System.out.println( "Hello Chuck" );
    }
}
```

- Las **reglas** se ejecutan cuando hay **coincidencia con los objetos que se insertan en la memoria de trabajo**.
- Las reglas nunca son llamadas directamente.
- Instancias específicas no pueden ser pasadas a una regla.
- Dependiendo de las coincidencias, **una regla se puede disparar una o varias veces, o nunca.**

```
rule "Hello World"
    when
        Person( name == "Chuck" )
    then
        System.out.println( "Hello Chuck" );
        end
```

Drools y los sistemas de gestión de reglas de negocio

Las reglas de negocio de una empresa se suelen encontrar dispersas, ya sea codificadas en las aplicaciones, en una hoja de cálculo o en las mentes de los expertos del negocio.

El propósito de un sistema de gestión de reglas de negocio (BRMS) es centralizar todas esas reglas de negocio que se suelen encontrar dispersas en una empresa y que, a su vez, constituyen la verdadera inteligencia del negocio. Esto permitirá **poder acceder con suma facilidad a ellas y poder gestionar los cambios que se produzcan en éstas con mayor rapidez**.

Normalmente estos sistemas de gestión de reglas, también conocidos como BRMS, suelen contar con una interface gráfica que permite trabajar de forma sencilla con las reglas del negocio. Esta característica es especialmente interesante ya que permite que los **expertos del negocio** (y no los técnicos) puedan gestionarlas de manera eficiente.

Los BRMS cuentan con un motor de reglas (BRM), que es el componente donde se ejecutan las reglas. Las ejecución de esas reglas, en base a unos datos de entrada, desencadenará una acción. Como dijimos en el punto anterior, lo ideal sería que la regla decidiese qué acciones hay que tomar en base a los datos analizados.

El BRM puede ser utilizado incorporándolo directamente en el código de nuestra aplicación. Sin embargo, también puede ser expuesto como un **servicio de toma de**

decisiones, que probablemente sea su mejor enfoque. Lo que es lo mismo, un componente en nuestra arquitectura en el que se apoyen los distintos componentes de nuestra plataforma para saber cómo deben actuar en determinados casos. Si hablamos de una arquitectura SOA, podríamos decir que el servicio de toma de decisiones sería **el cerebro de nuestro sistema**.

Ventajas

- **Mayor modularidad**: Separar las reglas de negocio permite su reutilización desde cualquier aplicación. Se aseguran servicios modulares y reutilizables.
- **Mayor consistencia**: Todas las reglas que se relacionan en forma lógica pueden organizarse y mantenerse en una ubicación centralizada, minimizando la duplicación de código y aumentando la consistencia.
- **Simplicidad:** Se captura la regla de negocio en un formato que el analista puede entender. Se pueden definir diferentes formas de representación de reglas, como por ejemplo árboles y tablas de decisión.
- **Auto-descriptiva y fácil de Entender:** Una tabla de precios en una base de datos nunca podrá comunicar el escenario completo tan bien como una tabla de decisión. Se aumenta la colaboración entre las áreas de negocio y el departamento de sistemas al lograr un lenguaje común de comunicación.
- **Test independiente:** Dado que las reglas se prueban en forma separada, además de asegurar la consistencia minimiza el esfuerzo a realizar cuando se pone en producción la aplicación.

Desventajas

- El **uso de esta tecnología no garantiza la calidad de las reglas** por sí mismas. La calidad de las reglas dependerá del conocimiento del experto seleccionado para la implementación del sistema experto en cuestión.
- No es óptimo para todos los tipos de problemas.
- Se requiere de conocimiento considerable para que no se desperdicie el uso de esta tecnología.
- Puede ser saboteado por usuarios sin conocimiento que definan reglas equivocadas o que entren en conflicto con reglas creadas previamente.
- La performance de los motores de inferencias puede verse seriamente afectada por el volumen de datos en la base de conocimiento.

¿Qué es Drools?

Drools es un sistema de gestión de reglas de negocio (BRMS, por las siglas en inglés de business rule management system) open source, con un motor de reglas basado en encadenamiento hacia adelante y hacia atrás, más correctamente conocido como sistema de reglas de producción.

Drools 6.x, y 7.x implementan un algoritmo perezoso llamado PHREAK.

Sus principales componentes son:

- **Motor de reglasdel Negocio:** Mediante el algoritmo PHREAK, permite que las reglas se ejecuten de manera eficiente. Se llama "**Drools Expert**" (business rules engine).

    - Es el núcleo de Drools.
    - Contiene un Motor de Inferencia.
    - Utiliza el algoritmo PHREAK para la coincidencia de patrones (pattern matching)
    - Provee al usuario un modo para crear una base de conocimiento (knowledge base)
    - Provee al usuario una API simple para interactuar con el motor de inferencia.

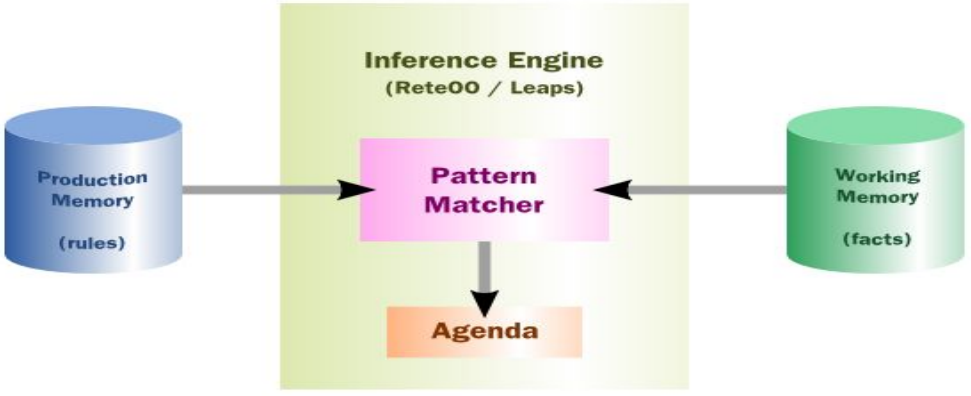
- **Lenguaje DRL:** que permite la creación de reglas de manera sencilla. Permite también la creación de reglas en lenguaje específico usando DSL (Lenguaje Específico del Dominio). Permite el tratamiento de hojas de cálculo.

- **BRMS:** Permite gestionar las reglas de manera centralizada con una interface web. Tiene funcionalidades de: gestión de reglas, herramientas de análisis, colaboración, etc**. Se puede acceder mediante servicios REST.**

¿Cuándo usar Drools?

Pues el escenario ideal para usar Drools, y en general cualquier motor de reglas, es cuando queremos **separar la lógica de negocio de las aplicaciones**. Cuando queremos que esa lógica de negocio esté centralizada y sea gestionada por los expertos del negocio y no por el personal técnico. Si encima esa **lógica de negocio es muy cambiante** (sistema de incentivos, promociones, descuentos, etc...) el sistema de gestión de reglas es perfecto, ya que permite realizar cambios de forma realmente ágil mediante: la creación de nuevas reglas, su modificación, flujos de reglas, etc...

Componentes de Drools-Expert (Importante)



- **El motor de evaluación de reglas (Motor de Inferencia) provee los mecanismos necesarios para ejecutar reglas (también denominadas producciones) y alcanzar algún objetivo.**

- **La Base de Reglas (Memoria de Producción) contiene el conjunto de reglas por el cual se rige el negocio. Es decir, contiene todas las reglas que el sistema conoce.**

- **La memoria de trabajo (working memory o fact base), contiene toda la información (conocimiento) con la cual un motor de inferencia trabaja. Un hecho es la unidad de información más pequeña con la que se puede trabajar en la memoria de trabajo.**

Típicamente, un evaluador de reglas contiene cientos o miles de reglas y es **probable que en un momento dado se activen varias reglas**. La **lista de reglas que son potencialmente ejecutables** se almacenan en lo que se denomina **Agenda**.

Al **conjunto de reglas activadas que tienen posibilidad de ser ejecutadas** se les denomina **grupo conflictivo de reglas (conflict set).**

Al **proceso de ordenar las reglas para ser disparadas** se le denomina **resolución de conflictos**; el resultado de la resolución de conflictos es una lista ordenada de activaciones de reglas que es la Agenda propiamente dicha.

El motor de inferencia debe decidir qué reglas se disparan y cuándo según el contexto de ejecución, es decir, los hechos que se encuentren definidos en la memoria de trabajo.

El propósito del componente que compara patrones (pattern matcher) es :

«decir cuándo aplicar cuáles reglas según el contenido de la memoria de trabajo».

Generalmente éste es un punto complejo en el diseño de los motores de evaluación de reglas dado que el pattern matcher deberá buscar en miles o millones de combinaciones de hechos para encontrar aquellos hechos que satisfagan las reglas

Resolución de Conflictos

Dado que el disparo de las consecuencias de las reglas manipula el conocimiento que existe en un momento dado en el motor de evaluación de reglas de negocio, **es importante definir el orden en el cual se van a disparar las reglas**.

Como se mencionó anteriormente, aplicar un orden a las diferentes activaciones que se produzcan se conoce como **resolución de conflictos**.

Algunas **estrategias** que se pueden utilizar son:

- **Salience: a cada regla se le puede definir un atributo entero especificando su prioridad; es decir aquellas reglas con los valores más altos tendrán mayor prioridad por lo que se ejecutarán primero.**

- **Recency:** se verifica el contador que se asigna a cada hecho en la activación; aquellas activaciones con el mayor contador son los primeros en la agenda.

- **Primary:** se verifica el contador que se asigna para cada hecho en la activación; aquellas activaciones con el menor contador son los primeros en la agenda.

- **FIFO o depth:** las reglas activadas recientemente son las que serán disparadas primero.

- **LIFO o breath:** las reglas se disparan según el orden de activación, la última activación va a ser disparada en primer lugar.

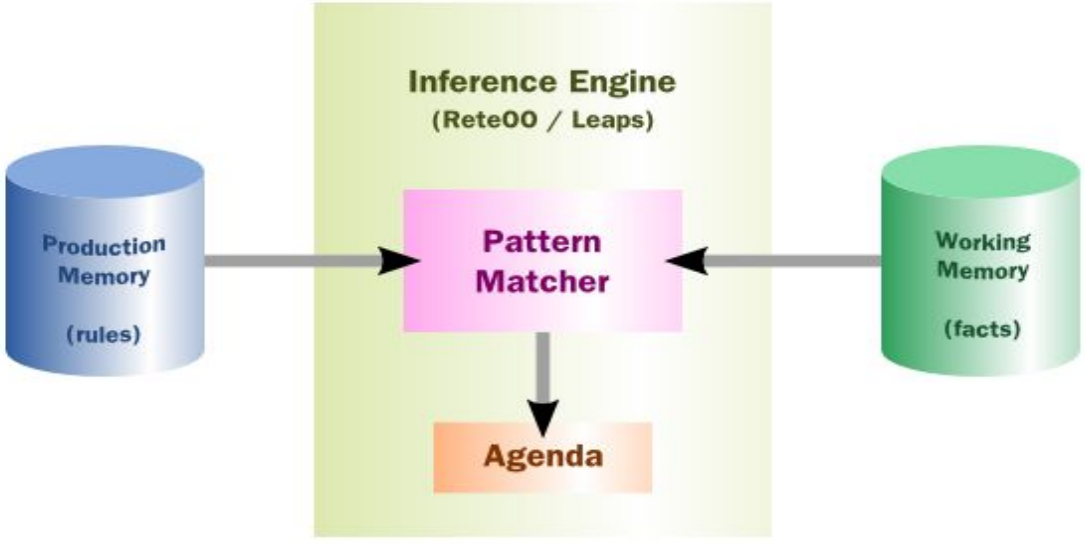
- **Complexity:** se toma en cuenta la complejidad de las reglas en conflicto; cuanto más compleja la regla (más condiciones tiene), más específica será; las activaciones con la mayor complejidad son las primeras en la agenda.

- **Simplicity:** se toma en cuenta la simplicidad de las reglas en conflicto; cuanto más simple es la regla (menos condiciones tiene) mayor prioridad tendrá, por lo que serán las primeras en la agenda.

- **Orden de carga:** cuando una regla se agrega al conjunto de reglas se le asigna un número de carga único; aquellas reglas con mayor contador serán las primeras en la agenda.

- **Random:** las activaciones se agregan en forma aleatoria en la agenda.

¿Cómo funciona el motor de inferencias?



1_ Se aplican las reglas a la memoria de trabajo.

2_ Se comparan los diferentes patrones existentes; se comparan todas las reglas con la memoria de trabajo utilizando el comparador de patrones (*pattern matcher*) para decidir cuáles reglas deben activarse en este ciclo. Como ya se mencionó

anteriormente, a ésta lista desordenada de reglas que se activan se le denomina grupo conflictivo de reglas (*conflict set*).

3_ Se crea una Agenda con las activaciones; el grupo conflictivo de reglas es ordenado para formar la agenda con la lista de reglas que serán ejecutadas.

4_ Se ejecutan las activaciones en un motor de ejecución. La primera regla de la agenda es disparada, por lo que se ejecutan todas las operaciones de la sección de acciones de la misma. Es posible que en las propias acciones de la regla se actualice la memoria de trabajo del motor de ejecución; esto implica que todo el proceso se vuelva a repetir. Es decir, es posible que una regla cambie un objeto y éste cambio implique que nuevas reglas sean factibles de ser ejecutadas, que serán agregadas a la agenda. Este proceso continúa hasta que no hay reglas en la Agenda.

5_ Si bien esta repetición implica una cantidad importante de retrabajo, la mayoría de los motores de evaluación de reglas utilizan técnicas sofisticadas para evitar este retrabajo. En particular, los resultados de comparar patrones en las condiciones de las reglas y la resolución de conflictos para crear la Agenda será preservado entre los diferentes ciclos de ejecución, de manera que solo lo esencial será re-ejecutado.

6_ Se devuelve el control a la aplicación.

Métodos de Ejecución para Sistemas de Reglas

- **Progresivo ó Encadenamiento hacia delante**. Las inferencias se realizan desde los antecedentes hacia los consecuentes.

- **Regresivo ó Encadenamiento hacia atrás**. Las inferencias se realizan partiendo desde los consecuentes hacia los antecedentes.

Encadenamiento hacia Adelante

- Los Motores de inferencia de tipo *Forward-chaining* comienzan el mecanismo de inferencia con los datos disponibles y utilizan reglas para extraer más datos hasta que se alcanza un objetivo.

- Cuando el **motor de inferencia** procesa un hecho en la memoria de trabajo, verifica su base de reglas para ver si alguna de las condiciones es verdadera. En caso afirmativo, las acciones que detallan las reglas son procesadas, pudiendo cambiar la información (hechos/Objetos) de la memoria de trabajo.

- Cuando se completa la acción, el motor chequea si la condición de alguna otra regla es verdadera y el proceso se repite hasta que no existan reglas que sean verdaderas.

Encadenamiento hacia adelante

Algoritmo Rete

El algoritmo **Rete** es un algoritmo de reconocimiento de patrones eficiente para implementar un sistema de producción de reglas. Fue creado por el **Dr. Charles L. Forgy** en la Carnegie Mellon University. Su primera referencia escrita data de 1974, y apareció de forma más detallada en su tesis doctoral (en 1979) y en un artículo científico de 1982.

Una implementación simple de un sistema experto basado en reglas comprobaría cada regla con los hechos de la base de conocimiento activando la regla si corresponde, y pasando a evaluar la siguiente. Este algoritmo, incluso para un número bajo de reglas y hechos, tiene un tiempo de ejecución muy alto (haciéndolo inadecuado para sistemas de producción reales).

El algoritmo Rete es la base de diversas implementaciones más eficientes de sistemas expertos. Un sistema experto basado en Rete construye una red de nodos, donde cada uno de ellos (excepto el nodo raíz) representa un patrón que aparece en la parte izquierda (el condicional) de una regla. Por lo tanto, el camino desde el nodo raíz a una hoja define la parte condicional entera de una regla. Cada nodo tiene una memoria de hechos que satisfacen su patrón.

A medida que se añaden o modifican hechos, se propagan los cambios por la red, haciendo que los nodos que se activan con el patrón se activen. Cuando un hecho o un conjunto de ellos hace que todos los patrones de una regla se satisfagan, se llega a un nodo hoja y la regla es activada.

Básicamente, el algoritmo Rete sacrifica memoria para incrementar velocidad de procesamiento.

Drools utiliza el Algoritmo PHREAK

Drool utiliza hasta la versión 5 el algoritmo ReteOO. El cual tiene las siguientes características:

- **Es la adaptación del algoritmo Rete para que interactúe con lenguajes orientados a objetos.**
- **Los hechos son objetos Java.**
- **Existen relaciones entre objetos en lugar de tuplas como en Rete.**

Drools 6.x introduce un nuevo algoritmo, que trata de abordar algunos de los temas centrales de RETE. El algoritmo incorpora todo el código existente de ReteOO, y todos sus accesorios. El nuevo algoritmo se llama: Phreak, es una evolución del algoritmo RETE, ya no se clasifica como una implementación de RETE.

Preguntas de Autoevaluación

A esta altura, se debe ser capaz de responder las siguientes preguntas:

1. ¿Qué es un motor de reglas de negocio?
2. ¿Cuándo es necesario usar un motor de reglas de negocio?
3. ¿Cuál es la diferencia entre un método de un objeto y una regla con respecto al procesamiento de objetos?
4. ¿Cómo funcionan: el motor de reglas, la memoria de trabajo y la memoria de producción?
5. ¿Qué es la Agenda y para qué sirve?
6. ¿Cuándo se genera un grupo conflictivo de reglas?. Dar un ejemplo.
7. ¿Cómo funcionan los Motores de inferencia de tipo encadenamiento hacia adelante?