GENERACION AUTOMATICA DE PATRONES EPL A PARTIR DE MODELOS DMN
AUTOMATIC GENERATION OF EPL PATTERNS FROM DMN MODELS

Trabajo de Fin de Máster
Curso 2022–2023

Autor
Paola Andrea Bedoya Herrera

Director
Mercedes García Merayo

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

GENERACION AUTOMATICA DE PATRONES EPL A PARTIR DE MODELOS DMN
AUTOMATIC GENERATION OF EPL PATTERNS FROM DMN MODELS

Trabajo de Fin de Máster en Ingeniería Informática

Departamento de Sistemas Informáticos y Computación

Autor

Paola Andrea Bedoya Herrera

Director

Mercedes García Merayo

Convocatoria: *Febrero 2023*

Calificación: *7,5 (Notable)*

Máster en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

13 de febrero de 2023

Agradecimientos

A mis padres y hermanas por su comprensión y apoyo durante toda mi vida. A mi abuela y tía que viven lejos en Colombia porque siempre me animaron a continuar y cumplir mis objetivos. Y por último y muy importante a mi directora por la orientación, ayuda y paciencia que me brindó para la realización de este proyecto.

Resumen

GENERACION AUTOMATICA DE PATRONES EPL A PARTIR DE MODELOS DMN

La aplicación de técnicas informáticas en diversas áreas de negocios ha ido en aumento en los últimos años. Uno de los principales problemas en la actualidad viene dado por el gran volumen de información que hay que gestionar, lo que requiere conocimientos técnicos avanzados, en particular cuando la información viene dada en forma de eventos cuyo análisis deberá facilitar la toma de decisiones en tiempo real. A raíz de esto aparecen nuevas herramientas, dispositivos y lenguajes que dan apoyo al procesamiento de la información. Entre ellas destacan las tablas de decisión que permiten definir reglas basadas en datos de entrada y generando las salidas de interés. Las tablas de decisión son representaciones gráficas similares a una matriz con filas y columnas separadas por datos de entrada y datos de salida, que indican condiciones con resultados asociados. No obstante, es necesario generar código ejecutable que permita procesar los eventos recibidos de forma automática en tiempo real, aplicando las reglas definidas. Los motores de procesamiento de eventos permiten llevar a cabo esta tarea, pero para ello es necesario que las reglas que rigen nuestro negocio se especifiquen mediante patrones en un lenguaje específico que requiere conocimientos técnicos.

Este trabajo propone una herramienta que transforme las reglas especificadas en tablas de decisión a patrones que puedan ser ejecutados en un motor de procesamiento de eventos.

Palabras clave

Eventos, EPL, tabla de decisión, Camunda, Esper, patrones.

Abstract

AUTOMATIC GENERATION OF EPL PATTERNS FROM DMN MODELS

The application of IT techniques in various areas of business has been increasing in recent years. One of the main problems at present is given by the large volume of information that must be managed, which requires advanced technical knowledge, particularly when the information is given in the form of events whose analysis should facilitate decision-making in real time. As a result of this, new tools, devices and languages appear that support information processing. Among them, the decision tables stand out, which allow defining rules based on input data and generating the outputs of interest. Decision tables are matrix-like graphical representations with separate rows and columns of input data and output data, indicating conditions with associated outcomes. However, it is necessary to generate executable code that allows the events received to be processed automatically in real time, applying the defined rules. Event processing engines allow this task to be carried out, but for this it is necessary that the rules that govern our business are specified by means of patterns in a specific language that requires technical knowledge.

This paper proposes a tool that transforms the rules specified in decision tables into patterns that can be executed in an event processing engine.

Keywords

Event, EPL, decision table, Camunda, Esper, patterns.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se introduce la motivación para realizar este trabajo y los objetivos que se pretenden alcanzar. También se indica la estructura de esta memoria.

1.1. Motivación

Actualmente, la necesidad de monitorizar los datos en tiempo real, con rapidez y con el menor error posible, hace cada vez más necesario el uso de herramientas que faciliten este proceso. El procesamiento de la información es un punto clave en la toma de decisiones de las organizaciones, y requiere la aplicación de métodos que deben ser ágiles y eficientes para poder soportar el gran volumen de datos que se generan actualmente.

El procesamiento de eventos complejos (en inglés, *Complex Event Processing*, CEP) es una técnica de procesamiento de eventos en tiempo real, que analiza las relaciones causa-efecto entre eventos relacionados y nos permite tomar decisiones eficaces en situaciones específicas. Las condiciones de los eventos son establecidas por un usuario en forma de *patrones* y codificadas en el lenguaje de procesamiento de eventos correspondiente.

Una persona sin conocimientos previos en el lenguaje se verá abrumado por la alta complejidad del diseño de dichos patrones. Es por esta razón que puede ser realmente interesante crear un generador de patrones automático, a partir de una especificación sencilla e intuitiva que sea manejable para un usuario sin conocimientos previos, con la ventaja de disponer de una interfaz gráfica. Esto podría acercar a las personas a realizar un modelado de las condiciones específicas que se deben considerar al analizar sus eventos y de las acciones que desean adoptar en caso de detectar que dichas condiciones se cumplen.

1.2. Objetivos

Este trabajo tiene como objetivo general la creación de una herramienta para la generación y despliegue automático de patrones en un lenguaje de procesamiento de eventos (en inglés *Event Processing Language*, EPL) a partir de reglas de negocio definidas en una plataforma *amigable* mediante el uso de un método de modelado de decisiones (en inglés *Decision Model and Notation*, DMN). Esta herramienta proporcionará la transformación automática de dichos modelos a patrones que podrán ser desplegados para su activación y chequeo de eventos. Con el fin de alcanzar este objetivo principal se han determinado los objetivos específicos siguientes:

- Analizar y crear modelos DMN.
- Generar automáticamente las consultas EPL a partir de los modelos DMN.
- Despliegue en un motor de procesamiento de eventos complejos.
- Evaluar la nueva herramienta con un caso de estudio.

1.3. Plan de trabajo

Para cumplir el objetivo principal de este trabajo es necesario planificar las tareas a realizar y dividir el desarrollo del proyecto en varias fases. Finalmente se han definido las siguientes fases:

1. Revisar información sobre DMNs disponibles para seleccionar el más adecuado.
    - Revisar y analizar el diseño de las tablas de decisión.
    - Analizar archivos que se obtienen para su explotación.
2. Diseño e implementación de un algoritmo para generación automática de patrones.
    - Revisar y analizar diferentes motores de procesamiento de eventos complejos.
    - Determinar la traducción de los modelos DMN a patrones en función de sus características.
    - Diseñar un algoritmo que genere patrones EPL para las diferentes políticas de aplicación del modelo DMN.
    - Realizar pruebas para chequear la funcionalidad implementada.
3. Diseño del despliegue de los patrones en el motor CEP seleccionado.
4. Herramienta

- Automatización del sistema, integrando la generación automática de los patrones y el despliegue.
- Diseño interfaz de usuario.

5. Desarrollo de un caso de estudio.

1.4. Estructura del documento

La estructura del documento es la siguiente:

- En el capítulo 2 se presenta la información esencial para entender el trabajo desarrollado acerca de modelos DMN, motor CEP y patrones EPL.
- En el capítulo 3 se describe el proyecto desarrollado.
- En el capítulo 4 se describe la interfaz de usuario desarrollada, así como los lenguajes utilizados.
- En el capítulo 5 se presenta el caso de estudio que se ha llevado a cabo.
- En el capítulo 6 se muestran las conclusiones y posibles ampliaciones futuras sobre esta herramienta.

1.5. Código del proyecto

Esta aplicación con sus ficheros fuente están publicados bajo licencia de código abierto y libre a través de los enlaces:
`https://github.com/paolabedoya/patrones_EPL_Backend`
`https://github.com/paolabedoya/Patrones_EPL_FrontEnd`

Capítulo 2

Preliminares

En este capítulo se presentan los conceptos necesarios para entender el trabajo desarrollado acerca de la generación de patrones para un motor CEP a partir de modelos DMN.

2.1. Decision Model and Notation

Decision Model and Notation (DMN) es una notación para modelar decisiones, de la familia de los estándares de *Business Process Management* (BPM) que es administrada por la (OMG) Object Management Group (Actualizado 2022).

DMN apoya de forma gráfica el modelado de procesos de decisiones para definir y administrar reglas de negocios. DMN genera un modelo basado en XML que permite interpretar y ejecutar las reglas mediante herramientas externas (Hitpass y Jakob Freund (2017)).

En la Figura 2.1 podemos ver los elementos del estándar de DMN que se detallan a continuación.

Tablas de decisión Las tablas de decisión son una representación gráfica donde se especifican las acciones a realizar cuando se cumplen determinadas condiciones. Las tablas de decisión se pueden implementar en el análisis y diseño de sistemas. Nos permiten representar una descripción completa y clara de una situación que se puede resolver por medio de una decisión con distintas alternativas tomadas en un momento específico del tiempo. Esta herramienta es de gran ayuda para los analistas, ya que permite integrar los datos recopilados de diversas fuentes o métodos, y permite representar de manera fácil la lógica de un problema cuando es complejo (Hitpass y Jakob Freund (2017)).

Existen varios modeladores donde se puede implementar una tabla de decisión como por ejemplo Camunda.org (2022b), *Kogito* Projects (Actualizado 2022) o *GbTEC* AG

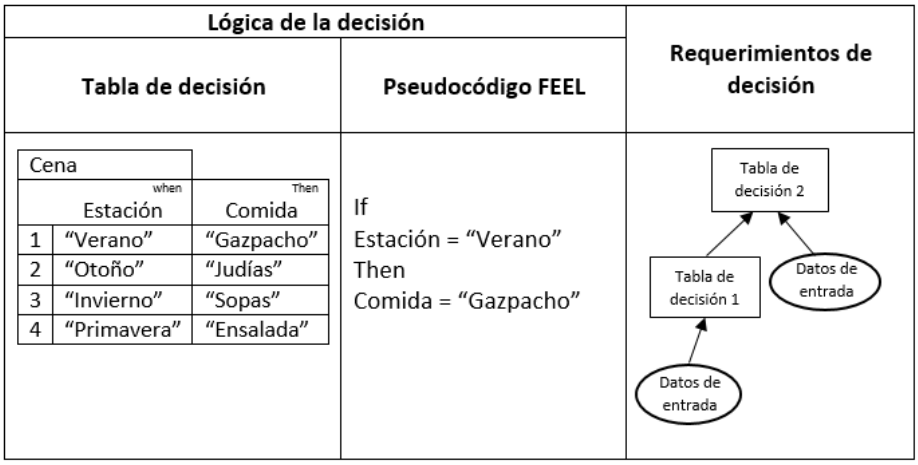

Figura 2.1: Elementos del estándar de DMN

(Actualizado 2022) entre otros. En este proyecto nos hemos decantado por *Camunda* aunque podría haberse utilizado cualquier otro modelador, pero este es el mejor documentado y de fácil manejo.

FEEL *(Friendly Enough Expression Language)* forma parte de la especificación DMN del OMG. Está diseñado para escribir expresiones para tablas de decisiones y expresiones literales de una manera que sea fácil y comprensible para los usuarios. En la Figura 2.2 podemos observar qué es el lenguaje FEEL y cómo lo interpreta una decisión.

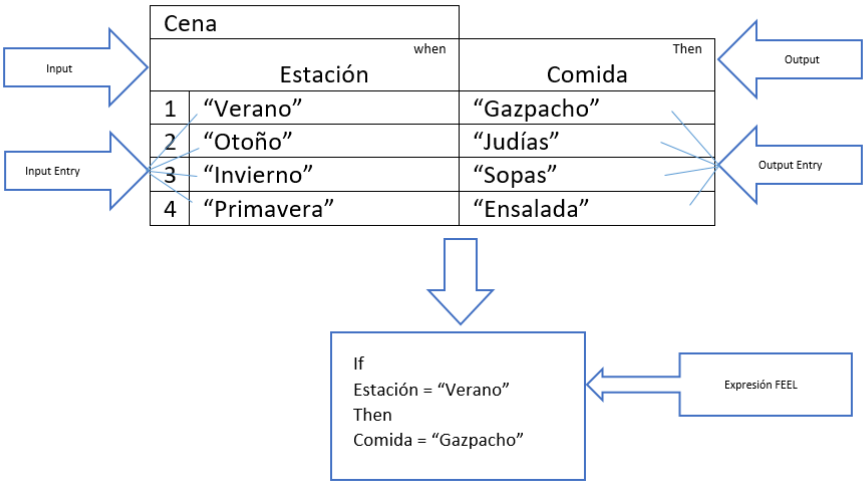

Figura 2.2: Traducción de elementos DMN a FEEL

Gráficos de Requerimientos de decisión Algunos procesos de decisión son más complejos, no se pueden expresar en una sola tabla de decisión. Un gráfico de requisitos de decisión permite modelar un dominio de toma de decisiones, mostrando los ele-

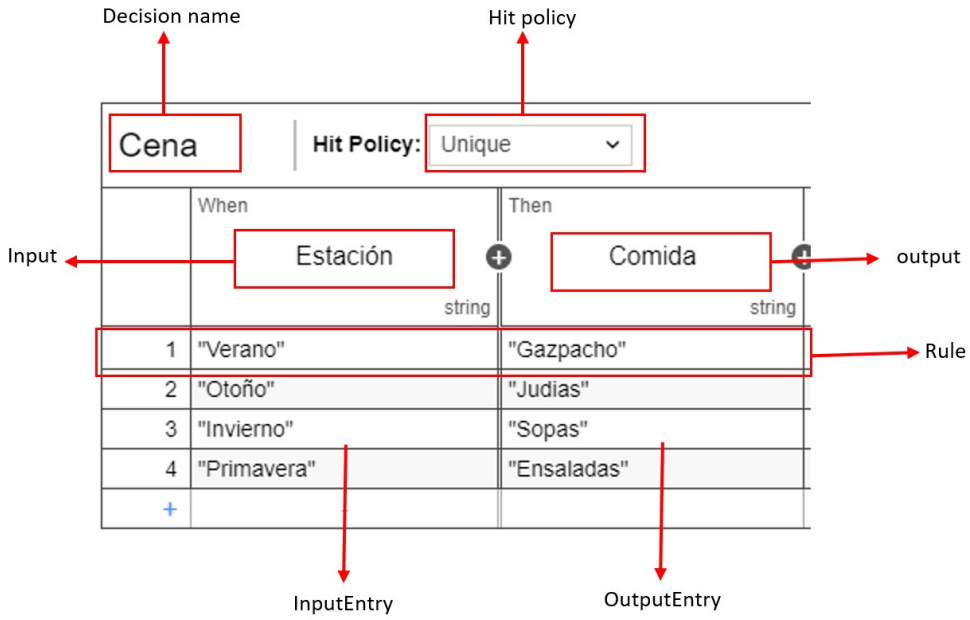

Figura 2.3: Elementos de una tabla DMN en *Camunda*

mentos más importantes que intervienen en él (tablas de decisión y fuentes de datos) así cómo las dependencias entre ellos.

2.2. Camunda DMN

Camunda es una plataforma de automatización de procesos que permite a los desarrolladores y analistas, diseñar, automatizar y gestionar procesos de forma rápida y ágil.

Camunda opera bajo el estándar de reglas DMN y está disponible bajo la licencia *bpmn.io*, donde las bibliotecas de terceros incluidas se distribuyen bajo sus respectivas licencias. Existen 2 versiones:

- **Camunda Modeler**. Aunque permite realizar el modelado de las tablas su principal inconveniente es no realizar las validaciones de las condiciones que se indican, tan solo permite generar las reglas exportándolas a un archivo XML y la validación debe realizarse externamente.

- **Enterprise Edition**. Tiene licencia comercial y está totalmente mantenida. Esta versión permite validar las condiciones de las reglas modeladas.

También cuenta con un simulador *online* donde se pueden ver ejemplos e interactuar con ellos Camunda.org (2021).

A continuación, se describen los distintos elementos que se deben considerar en el diseño de una tabla DMN en la plataforma *Camunda*.

La Figura 2.3 presenta la vista de una tabla DMN para la representación de la lógica de la toma de decisiones que se define mediante entradas o (*inputs*), salidas(*outputs*), reglas (*rules*) y *hit policies*.

Entradas o *inputs*

Se pueden incluir una o más entradas indicando un identificador único y requerido para cada una de ellas, una etiqueta que describa la entrada, una expresión que especifique cómo se genera el valor de la cláusula, una expresión para evaluar la entrada, y el tipo de datos.

Salidas o *outputs*

Una tabla de decisiones puede tener una o más salidas. Para cada una de ellas se debe definir el identificador único, la etiqueta que describe la salida y el tipo de datos.

Reglas o *rules*

Las tablas de decisión pueden incluir una o más reglas. Cada regla contiene datos de entrada y datos de salida. Los datos de entrada establecen las condiciones y los de salida la decisión de la regla. Si se cumple una condición, entonces se cumple la regla y se emitiría la decisión que contiene dicha regla.

Hit Policy

Una tabla de decisión tiene una *hit policy* asociada que especifica cómo gestionar las posibles decisiones de la evaluación de la reglas. La *hit policy* sirve para especificar cuántas reglas de una tabla de decisión pueden satisfacerse y cuáles de las decisiones de las reglas satisfechas se incluyen en el resultado de la decisión. Los tipos disponibles aparecen en la Tabla 2.1

Unique
First
Any
Collect (Sum, Min, Max, Count)
Rule Order

Tabla 2.1: Hit Policies en *Camunda*

En el caso de *Unique*, *Any* y *First* tan solo se incluirá en el resultado el output asociado a una de las reglas que se satisfagan y para *Rule Order* y *Collect* pueden devolver las salidas de múltiples reglas satisfechas. A continuación, se describe cada una de estas políticas.

- *Unique:* Solo puede satisfacer una regla o ninguna. El resultado de la tabla de decisión contiene el *output* de la regla satisfecha. Si se satisface más de una regla, se infringe la política *Unique*.

- *First:* En este caso se pueden satisfacer varias reglas, sin embargo, solo se considera el *output* de la primera regla satisfecha.

- *Any:* Permite satisfacer múltiples reglas. Sin embargo, todas las reglas satisfechas deben generar la misma salida. El *output* de la tabla de decisión contiene sólo la salida de una de las reglas satisfechas.

  Si se cumplen varias reglas que generan diferentes resultados, se infringe la política *Any*.

- *Collect*. Con esta política la salida puede estar condicionada por un campo llamado *aggregation* que puede tomar alguno de los valores indicados en la Tabla 2.2

Aggregation	Resultado
SUM	Suma de todos los valores de salida
MIN	Valor más pequeño de todos los valores de salida
MAX	Valor más grande de todos los valores de salida
COUNT	Cuenta número de valores de la salida

Tabla 2.2: Tipo *aggregation* para hit policy *Collect* Camunda.org (2022a).

- *Rule Order*. Se pueden satisfacer múltiples reglas. El resultado de la tabla de decisión contendrá la salida de todas las reglas satisfechas, con un orden de prioridad donde la primera regla tiene la prioridad más alta y descendiendo, hasta la última regla que tendrá la prioridad más baja.

2.3. Procesamiento de eventos complejos

El procesamiento de eventos complejos conocido como *Complex Event Processing* (CEP) Luckham (2001) es una de las tecnologías más comunes utilizadas en los sistemas dirigidos por eventos. Estos sistemas, que engloban un conjunto de técnicas y herramientas, reaccionan ante un flujo de eventos en tiempo real.

CEP permite analizar y relacionar grandes cantidades de datos para detectar situaciones concretas en tiempo real. Estas situaciones de interés corresponden a una secuencia de eventos que requiere una reacción inmediata. Un *evento simple* ocurre en un instante de tiempo, mientras que un *evento complejo* se deriva de un conjunto de eventos simples o de otros eventos complejos por medio de *patrones*. Los patrones permiten especificar las

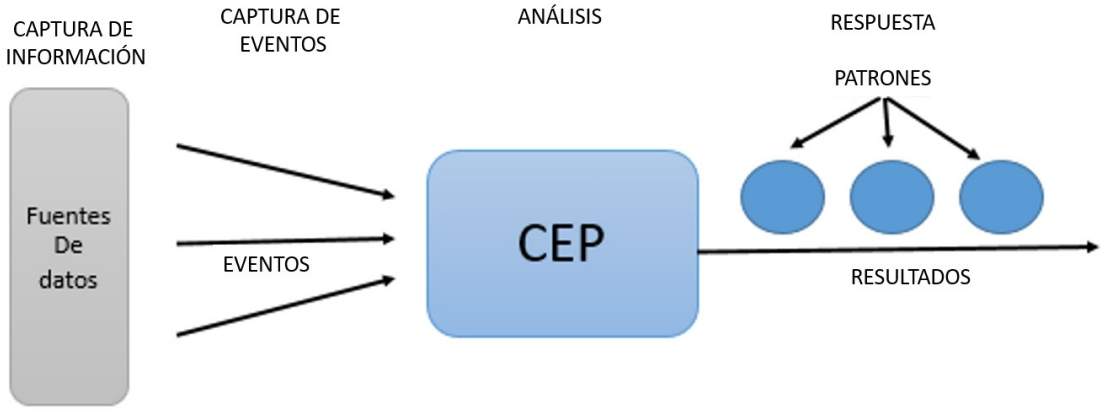

Figura 2.4: Esquema general de procesamiento de eventos

condiciones que deben cumplirse para detectar rápidamente determinadas situaciones de interés. Un motor CEP es el software utilizado para detectar estos patrones en tiempo real, así como para notificarlos a los usuarios y/o sistemas interesados.

La tecnología CEP proporciona alta velocidad de procesamiento de numerosos eventos, y ofrece un lenguaje específico de dominio para su procesado. Este lenguaje llamado *EPL (Event Processing Languaje)*, se trata de un lenguaje declarativo utilizado para implementar los patrones de interés. Los patrones de eventos complejos permiten agregar o filtrar datos de entrada con base en las reglas definidas.

Este proceso lo podemos explicar de manera general en tres etapas, que podemos visualizar en la Figura 2.4.

- Captura de eventos: Consiste en la recepción de los eventos que se analizan con la tecnología CEP a partir de diferentes fuentes.
- Análisis: A partir de los eventos definidos, el motor de CEP se encarga de procesar y relacionar la información en forma de eventos aplicando los patrones previamente definidos con el fin de detectar situaciones específicas.
- Respuesta: Al detectar una situación determinada, el sistema actúa en consecuencia y se procede a su notificación indicada en el patrón correspondiente.

  Esta tecnología ofrece las siguientes ventajas:

- Calidad y seguridad en las decisiones: Los computadores son capaces de gestionar más información por segundo, y se pueden contemplar más factores a la hora de tomar una decisión frente a las capacidades que un ser humano dispone.
- Respuesta en tiempo real: EL uso de CEP permite realizar automáticamente el análisis de eventos sin que intervenga ninguna persona, lo que conlleva la posibilidad de

dar una respuesta en tiempo real.

- Sobrecarga de información: Debido a la forma en que los sistemas CEP procesan la información, se reduce enormemente la cantidad de información, ya que solo mostrará las situaciones críticas o relevantes que se hayan descrito a través de los patrones de eventos.

- Esfuerzo humano: Al utilizar sistemas informáticos, reducimos el trabajo humano y tiempo necesario para analizar toda la información.

En este proyecto se optó por utilizar Esper, motor CEP de EsperThech, que utiliza el lenguaje Esper EPL (EsperTech (2022)).

Esper es un lenguaje compilador y *runtime* disponible para JAVA y .NET. Esper usa el lenguaje Esper EPL que cumple con el estándar SQL-92 que está extendido para analizar series de eventos a través del tiempo.

El compilador de Esper compila EPL en un código de bytes en un formato de archivo de tipo .jar para su distribución y ejecución.

El *runtime* de Esper se ejecuta sobre una *Java Virtual Machine* (JVM) donde ejecuta el código en bytes producido por el compilador de Esper.

La arquitectura de Esper es similar a la de otros lenguajes de programación que se compilan en código de bytes JVM, como Scala, Clojure o Kotlin. Sin embargo, Esper no es un lenguaje de programación imperativo.

El lenguaje Esper EPL está orientado al tratamiento de *streams*. Se trata de un lenguaje declarativo basado en *Structured Query Language* (SQL) y que utiliza sentencias y cláusulas como *SELECT*, *FROM*, *WHERE*, *GROUP BY* o *ORDER BY*.

También aplica los conceptos de correlación de datos por medio de sentencias *JOIN*, con subconsultas de tipo agregación y donde la cláusula *INSERT INTO* nos permite definir nuevos eventos, conocidos como eventos complejos, a partir de los eventos simples disponibles para su posterior procesamiento.

Capítulo 3

Generación de patrones a partir de modelos DMN

En este capítulo se detalla la propuesta para el desarrollo del proyecto, es decir la generación automática de patrones de eventos a partir de tablas DMN para su despliegue en un motor CEP.

La generación de patrones se realiza a partir de un modelo DMN creado con *Camunda* que proporciona un archivo XML de respuesta con las condiciones creadas.

En este capítulo se explican detalladamente las trasformaciones necesarias de los elementos incluidos en dicho fichero para obtener el código en lenguaje EPL que permita ejecutar las reglas descritas en la tabla DMN haciendo uso de un motor CEP. En la Figura 3.1 podemos ver una tabla DMN en la que se muestran los diferentes elementos que se pueden utilizar en las tablas de decisión y que se han descrito en el Capítulo 2. El fichero XML generado por *Camunda* correspondiente a dicha tabla, que se puede encontrar en el Listing 8.1 del Apéndice, será utilizado para ilustrar las diferentes transformaciones. El proceso de transformación se divide en dos partes que se detallan a continuación: Creación de los esquemas necesarios a partir de los datos que se manejan en la tabla y creación de los patrones haciendo uso de dichos esquemas y de las reglas incluidas en la misma.

3.1. Creación de esquemas

La creación de los esquemas es lo primero que se debe hacer para declarar tipos de eventos y las propiedades de dicho evento. La instrucción para crear los esquemas se corresponde con la siguiente sintaxis:

```
create schema schema_name
(property_name property_type [,property_name property_type [,...])
```

Datos — Hit Policy: Unique

	When	And	And	And	Then	And	And	And
	Dato String (string)	Dato Boolean (boolean)	Dato Integer (integer)	Dato Date (date)	Output String (string)	Output Boolean (boolean)	Output Integer (integer)	Output Date (date)
1	"ABC","DEF","GHI"	true	0	date and time("2021-05-11T00:00:00")	"Rule 1 String"	true	1	date and time("2021-05-11T00:00:00")
2	not("ABC","DEF")	false	< 0	< date and time("2021-05-11T00:00:00")	"Rule 2 String"	false	2	date and time("2021-05-12T00:00:00")
3	-	-	<= 0	> date and time("2021-05-11T00:00:00")	"Rule 3 String"		3	date and time("2021-05-13T00:00:00")
4	-	-	> 0	[date and time("2021-05-12T00:00:00")..date and time("2021-05-15T00:00:00")]	"Rule 4 String"		4	date and time("2021-05-14T00:00:00")
5	-	-	>= 0	-	"Rule 5 String"		-5	date and time("2021-05-15T00:00:00")
6	-	-	[1..10]	-	"Rule 6 String"		-6	
7	-	-	[0..20[	-	"Rule 7 String"		-7	
8	-	-	]-3..-1]	-	"Rule 8 String"		-8	
9	-	-	]-10..-5[	-				

Figura 3.1: Tabla DMN

En nuestra transformación el esquema recibe el nombre de la tabla DMN que en el archivo XML se puede obtener del parámetro *name* asociado a la etiqueta `<decision>`. En nuestro ejemplo

```
<decision id="Decision_0gip5xq" name="Datos">
```

La declaración de las propiedades se realiza con el valor contenido en la etiqueta `<input>`. La etiqueta interna `<text>` contiene el nombre de la propiedad y el tipo de la misma está en el parámetro *typeRef* de la etiqueta `<inputExpression>`. Un ejemplo de propiedad se encuentra en el siguiente fragmento de nuestro fichero XML.

```
<input id="InputClause_0kihd5b" label="Dato Boolean">
    <inputExpression id="LiteralExpression_0d47n2r" typeRef="boolean">
        <text>datoBoolean</text>
    </inputExpression>
</input>
```

Como resultado obtendremos la instrucción que nos permite crear los tipos de eventos utilizados en nuestras tablas DMN. La instrucción obtenida para nuestro ejemplo se muestra a continuación.

```
create schema Datos( id integer, datoString string, datoBoolean boolean,
                     datoInteger integer, datoDate date);
```

Como puede observarse el esquema contiene una propiedad, id, que no está incluida en nuestro fichero XML. Esta propiedad se incluye porque los motores CEP necesitan identificar cada evento recibido. Esta propiedad se utilizará como identificador único de dichos eventos.

3.2. Creación básica de patrones

Después de crear el esquema de la tabla, se generan los patrones asociados a las reglas incluidas en la tabla DMN. Un patrón básico se ajusta a la siguiente sintaxis.

```
@Name(rule_name)
insert into output_name
Select id,current_timestamp.format() as date_table_name,
output_value as output_name [,....]
from table_name [,....]
where condition [[and | or] condition] [and...]...]
```

Intuitivamente, este patrón generará un nuevo *stream* con el nombre `output_name` que almacenará eventos con las propiedades indicadas en la sentencia `Select` cuando los eventos recibidos del *stream* con nombre `table_name` cumplan las condiciones indicadas en la cláusula `where`.

Cada etiqueta `<rule>` que aparece en el fichero XML corresponde a una regla de la tabla DMN y dará lugar a un patrón. Los patrones se irán generando secuencialmente para cada etiqueta `<rule>`.

Los diferentes elementos que deben incluirse en el patrón se obtienen en unos casos del fichero XML y en otros se generan automáticamente. A continuación, se describe cómo se obtiene cada uno de ellos.

Dada una etiqueta `<rule>` generaremos automáticamente un nombre, ya que las reglas en las tablas DMN no tienen un nombre asociado. Este nombre se crea concatenando el nombre de la tabla (o esquema) con el *string* "_Rule" y el número de regla empezando desde 0 y en orden ascendente hasta la última regla. En nuestro ejemplo tenemos 9 reglas que recibirán los nombres Datos_Rule0, Datos_Rule1, Datos_Rule2, etc.

En la linea 2 el valor `output_name` se obtiene concatenando el nombre de la tabla con el *string* "_output".

La cláusula `Select` contendrá dos atributos iniciales ajenos a la definición de la tabla DMN, `id` y `current_timestamp.format()`, que almacenarán un identificador único y la fecha del sistema, respectivamente. A continuación, se incluirán tantos atributos como *outputs* tenga asociados la regla correspondiente. La información asociada a cada uno de ellos aparece en las etiquetas `<output>` y `<outputEntry>`. El `output_value` se generará a partir del valor de la etiqueta `<text>` asociada a la etiqueta `<outputEntry>`. El `output_name` corresponde al valor del parámetro *name* de la etiqueta `<output>`.

En la línea 4 el valor `table_name` se obtiene, al igual que el nombre del esquema, del parámetro *name* de la etiqueta `<decision>` del archivo XML.

Las condiciones de un patrón se generan utilizando los valores asociados a las etiquetas

Tipo	Representación con Camunda	Transformación EPL
Integer Long Double	[valor1, valor2]	input >= valor1 and input <= valor2
	]valor1, valor2[	input > valor1 and input < valor2
	[valor1, valor2[	input >= valor1 and input < valor2
	]valor1, valor2]	input > valor1 and input <= valor2
	< valor	input < valor
	> valor	input > valor
	<= valor	input <= valor
	>= valor	input >= valor
	valor	input = valor
Boolean	True	input
	False	Not input
Date	[date and time("fecha1")..date and time("fecha2")]	input. between(cast(fecha1, Date, dateformat: 'yyyy-MM-dd HH:mm:ss'),(cast(fecha2, Date, dateformat: 'yyyy-MM-dd HH:mm:ss'))
	> date and time("fecha")	input.after(cast(fecha, Date, dateformat: 'yyyy-MM-dd HH:mm:ss))
	< date and time("fecha")	input.before(cast(fecha, Date, dateformat: 'yyyy-MM-dd HH:mm:ss'))
	date and time("fecha")	(cast(input, fecha, dateformat: 'yyyy-MM-dd HH:mm:ss'))
String	Not ("valor1","valor2", ...)	Not in ('valor1', 'valor2',' ...')
	"valor1","valor2"...	In ('valor1', 'valor2','...')

Figura 3.2: Transformación de condiciones DMN a EPL

`<text>` asociados a las etiquetas `<input>` e `<inputEntry>` de cada regla.

En la Figura 3.2 podemos ver las condiciones que se pueden expresar en *Camunda* para los distintos tipos de datos que acepta y su transformación al lenguaje EPL a la hora de construir las condiciones de los patrones. En nuestro ejemplo, la primera regla contiene cuatro condiciones sobre datos de tipo *String* (`datoString`), *Boolean* (`datoBoolean`), Integer (`datoInteger`) y Date (`datoDate`). Los valores correspondientes son una lista de *strings* válidos `('ABC","DEF","GHI")`, el valor `true`, el intervalo [1,10] y fecha menor que `2021-05-11`. Estas condiciones se transformarán a un predicado con cláusulas unidas mediante el operador lógico AND que se obtendrán de la transformación indicada en la Figura 3.2. A continuación presentamos el fragmento del fichero XML generado por *Camunda* para nuestra tabla DMN.

```
<input id="Input_1" label="Dato String">
    <inputExpression id="InputExpression_1" typeRef="string">
        <text>datoString</text>
```

```
    </inputExpression>
 </input>
<input id="InputClause_0kihd5b" label="Dato Boolean">
    <inputExpression id="LiteralExpression_0d47n2r" typeRef="boolean">
        <text>datoBoolean</text>
    </inputExpression>
</input>

<input id="InputClause_14gr2kk" label="Dato Integer">
   <inputExpression id="LiteralExpression_12s2pj9" typeRef="integer">
      <text>datoInteger</text>
   </inputExpression>
</input>

<input id="InputClause_0vlrnnw" label="Dato Date">
    <inputExpression id="LiteralExpression_1tgb03t" typeRef="date">
        <text>datoDate</text>
    </inputExpression>
</input>

<output id="Output_1" label="Output String" name="outString"
                      typeRef="string" biodi:width="192" />
<output id="OutputClause_0jt21py" label="Output Boolean" name="outputBoolean"
                      typeRef="boolean" />
<output id="OutputClause_0shg3dn" label="Output Integer" name="outputInteger"
                      typeRef="integer" />
<output id="OutputClause_0i93ava" label="Output Date" name="outputDate"
                      typeRef="date" />

<rule id="DecisionRule_01ld593">
        <inputEntry id="UnaryTests_012wgio">
          <text>"ABC","DEF","GHI"</text>
        </inputEntry>
        <inputEntry id="UnaryTests_1ixtuq5">
          <text>true</text>
        </inputEntry>
        <inputEntry id="UnaryTests_01q5nh9">
          <text>[1..10]</text>
        </inputEntry>
        <inputEntry id="UnaryTests_1rxjy8t">
          <text>&lt; date and time("2022-05-11T00:00:00")</text>
        </inputEntry>
        <outputEntry id="LiteralExpression_0fwveuk">
          <text>"Rule 1 "</text>
        </outputEntry>
        <outputEntry id="LiteralExpression_1szvqyo">
          <text>true</text>
```

```
        </outputEntry>
        <outputEntry id="LiteralExpression_05kdn1s">
          <text>1</text>
        </outputEntry>
        <outputEntry id="LiteralExpression_10x7uis">
          <text>date and time("2022-05-11T00:00:00")</text>
        </outputEntry>
      </rule>
```

La regla da lugar al siguiente patrón mediante la aplicación del proceso indicado previamente.

```
@Name('Datos_Rule0')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos,
'Rule 1 ' as outString, true as outputBoolean, 1 as outputInteger,
(cast(('2022-05-11 00:00:00'), Date, dateformat: 'yyyy-MM-dd HH:mm:ss')))
as outputDate
from Datos
Where datoString in ('ABC','DEF','GHI') and datoBoolean and
datoInteger >= 1  and datoInteger <= 10
and datoDate.before(cast('2022-05-11 00:00:00',
Date, dateformat: 'yyyy-MM-dd HH:mm:ss'));
```

El conjunto de esquemas y patrones generados para nuestro ejemplo se encuentra en el Listing 8.2 del Apéndice.

3.3. Extensión patrones con *Hit Policy*

Los patrones con hit policy diferente de `UNIQUE` se identifican en la etiqueta `<decisionTable>` con el parámetro *hitPolicy*, como se puede ver a continuación.

```
<decisionTable id="DecisionTable_0lwxmib" hitPolicy="ANY">
```

Dependiendo de la *hit policy* los patrones generados presentarán diferentes variaciones en su construcción, tal y como se describe a continuación para cada uno de ellos.

ANY

Los patrones con *hit policy* `ANY` usan la sintaxis del patrón básico cambiando la forma de insertar las condiciones. Si los valores de salida de dos o más reglas son iguales, se agruparán todas las reglas en un único patrón que tendrá en la cláusula `where` todas las condiciones generadas para cada una de esas reglas unidas con el operador lógico OR.

```
where condition1 [[or] condition2] [or...]...]
```

RULE ORDER

Los patrones con *hit policy* `RULE ORDER`, utilizan la anotación `@Priority` para darle una prioridad a cada patrón. La prioridad se determina de forma descendente, donde el primer patrón encontrado tiene la prioridad más alta y el último patrón tiene la prioridad más baja. Para determinar la prioridad más alta se cuentan las reglas existentes de la tabla DMN en el archivo XML. En nuestro ejemplo hay 9 reglas, por lo que la prioridad más alta, correspondiente a la primera regla, se indicaría con @Priority(8), y desde esta se desciende hasta llegar a la última con la prioridad @Priority(0). Esta prioridad se indica a continuación del nombre de la regla en el patrón correspondiente.

```
@Name(rule_name)
@Priority(n)
```

FIRST

Los patrones con *hit policy* `FIRST` utilizan las anotaciones `@Priority` y `@Drop`. Cuando se cumplen las condiciones de varios patrones, se procesa tan solo el de mayor prioridad, ya que debido a la anotación @Drop, se interrumpe cualquier posible procesamiento del evento por otro patrón. Para calcular la prioridad utiliza la misma forma que el patrón `RULE ORDER`.

```
@Name(rule_name)
@drop
@Priority(n)
```

COLLECT

Los patrones con *hit policy* `COLLECT` están condicionados por el parámetro *aggregation*, que se encuentra en la etiqueta `<decisionTable>` y que puede tener los valores `SUM, MIN, MAX, COUNT`.

```
<decisionTable id="DecisionTable_0lwxmib" hitPolicy="COLLECT" aggregation="SUM">
```

En este caso se definirán dos tipos de patrones. Los primeros se generan siguiendo una versión abreviada del patrón básico, ya que la cláusula `Select` incluirá tan solo el

identificador único y un único atributo correspondiente al *output* asociado a la regla, que solo puede ser de tipo `int, long o double`. Si existe más de un output o tiene un tipo de dato diferente a los mencionados el patrón no se podrá generar.

El segundo patrón se construye después de haber generado los patrones de las diferentes reglas y sirve para calcular y resumir los valores generados por dichos patrones. La definición de este patrón se ajusta a la siguiente sintaxis.

```
@Name(COLLECT_aggregation)
Insert into result_output
Select id, current_timestamp.format(),
aggregation(output_name) as result_aggregation
from table_name
group by id;
```

El nombre del patrón se crea a partir de la concatenación de la cadena `COLLECT` y el valor asociado al parámetro *aggregation*.

En la cláusula `Insert` se utiliza siempre el nombre `result_output` para indicar el flujo de datos en el que se almacenarán los resultados de la ejecución del patrón.

La cláusula `Select` contendrá dos atributos iniciales ajenos a la definición de la tabla DMN, `id` y `current_timestamp.format()`, que almacenarán un identificador único y la fecha del sistema, respectivamente. A continuación, `aggregation(output_name)` se generará a partir del valor del parámetro *aggregation* de la etiqueta `<decisionTable>` concatenado con el parámetro `name` de la etiqueta`<output>` y `result_aggregation` se obtendrá de la concatenación del string ''result" y el valor del parámetro *aggregation*. La cláusula `from` hace referencia al nombre del flujo de datos que se ha indicado en el primer grupo de patrones. Finalmente, la cláusula `group by` permitirá agrupar los valores por el identificador `id`. A continuación, se presentan los patrones generados para una tabla DMN con tres reglas que establecen un bonus dependiendo del valor del input `year` de los eventos del esquema `collect_Sum`. Hay que tener en cuenta que varios patrones se pueden aplicar a un mismo evento, asignándole varios valores. En este caso el patrón de agregación se encargará de sumar todos esos valores para cada evento.

```
create schema collect_Sum(id Integer, year integer);

@Name('collect_Sum_Rule0')
insert into collectSum_Output
Select id,100 as bonus
from collectSum
Where year > 1;

@Name('collect_Sum_Rule1')
```

```
insert into collectSum_Output
Select id,200 as bonus
from collectSum
Where year > 2;

@Name('collect_Sum_Rule2')
insert into collectSum_Output
Select id,300 as bonus
from collectSum
Where year > 3;

@Name(collect_Sum)
insert into Result_output
Select id, current_timestamp.format(), SUM(bonus) as results_SUM
from collect_Sum_Output
group by id;
```

3.4. Patrones para tablas relacionadas

En algunas ocasiones es necesario analizar decisiones complejas que necesitan utilizar varias tablas DMN relacionadas entre sí. Esto hace que se deban generar patrones más complejos que los descritos en las secciones anteriores. En la Figura 3.3 se observa la relación de dos tablas DMN. Dicha relación se establece convirtiendo la salida de la tabla 1 en entrada de la tabla 2.

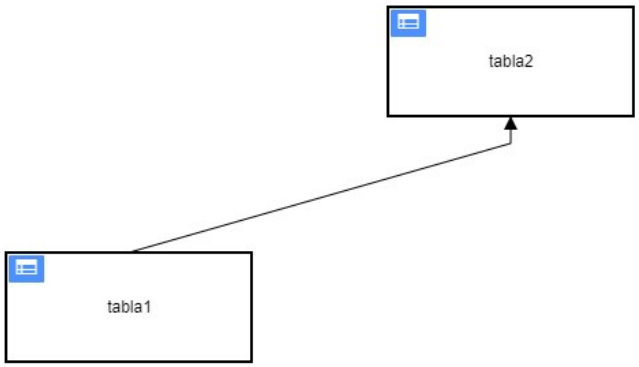

Figura 3.3: Relación entre tablas DMN

La Figura 3.4 muestra las tablas DMN correspondientes a este diagrama. Como se puede ver tabla2 tiene como entrada el valor salida generado por los patrones asociados a las reglas 1 y 2 de la tabla1.

Si se tiene una o más relaciones de tablas DMN, el archivo XML generado incluirá la etiqueta `<requiredDecision>` en las tablas que generan las salidas utilizadas en otras ta-

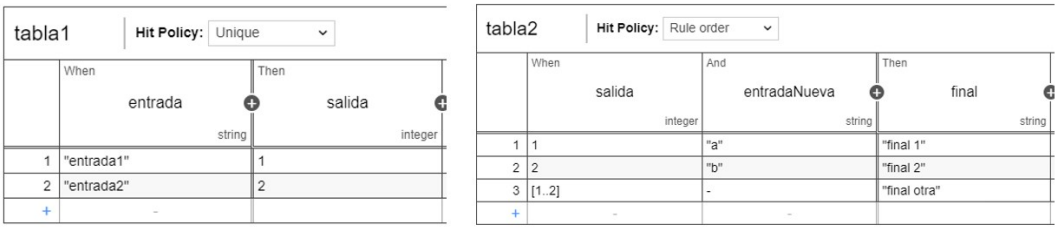

Figura 3.4: Tablas DMN de tabla 1 y tabla 2

blas para poder identificar que tabla o tablas tienen relación. La tabla principal no incluye esta etiqueta. Dicha etiqueta tiene un parámetro *href* que indica la tabla que utilizará la salida producida y que corresponde al parámetro *id* de la etiqueta `<decision>` de la misma. A continuación, se muestra el fichero XML generado de la definición de la relación de las tablas que aparecen en la Figura 3.4.

```
<decision id="Decision_1hd9ugh" name="tabla1">
   <decisionTable id="DecisionTable_0dqfqcr">
     <input id="Input_1" label="entrada" camunda:inputVariable="entrada">
       <inputExpression id="InputExpression_1" typeRef="string">
         <text>entrada</text>
       </inputExpression>
     </input>
     <output id="Output_1" label="salida" name="salida1" typeRef="integer" />
     <rule id="DecisionRule_0avw7hl">
       <inputEntry id="UnaryTests_0pwrcqf">
         <text>"entrada1"</text>
       </inputEntry>
       <outputEntry id="LiteralExpression_1im0z1f">
         <text>1</text>
       </outputEntry>
     </rule>
     <rule id="DecisionRule_17fz338">
       <inputEntry id="UnaryTests_0y99ziw">
         <text>"entrada2"</text>
       </inputEntry>
       <outputEntry id="LiteralExpression_1mzymj7">
         <text>2</text>
       </outputEntry>
     </rule>
   </decisionTable>
 </decision>
 <decision id="Decision_1j3halt" name="tabla2">
   <informationRequirement id="InformationRequirement_1bwcwx1">
     <requiredDecision href="#Decision_1hd9ugh" />
```

```
</informationRequirement>
<decisionTable id="DecisionTable_07acjqb" hitPolicy="RULE ORDER">
  <input id="InputClause_0skaxb2" label="salida"
  camunda:inputVariable="salida">
    <inputExpression id="LiteralExpression_0dyvdug" typeRef="integer">
      <text>salida</text>
    </inputExpression>
  </input>
  <input id="InputClause_0g90wuv" label="entradaNueva"
  camunda:inputVariable="entradaNueva">
    <inputExpression id="LiteralExpression_037660j" typeRef="string">
      <text>entradaNueva</text>
    </inputExpression>
  </input>
  <output id="OutputClause_0ebok2u" label="final" name="final"
  typeRef="string" />
  <rule id="DecisionRule_1u15j0g">
    <inputEntry id="UnaryTests_0cb44rc">
      <text>1</text>
    </inputEntry>
    <inputEntry id="UnaryTests_0z4veol">
      <text>"a"</text>
    </inputEntry>
    <outputEntry id="LiteralExpression_1585dbw">
      <text>"final 1"</text>
    </outputEntry>
  </rule>
  <rule id="DecisionRule_1nebo8c">
    <inputEntry id="UnaryTests_1kg5796">
      <text>2</text>
    </inputEntry>
    <inputEntry id="UnaryTests_1fos9jk">
      <text>"b"</text>
    </inputEntry>
    <outputEntry id="LiteralExpression_0m0bqy7">
      <text>"final 2"</text>
    </outputEntry>
  </rule>
  <rule id="DecisionRule_087wlog">
    <inputEntry id="UnaryTests_1rry47h">
      <text>[1..2]</text>
    </inputEntry>
    <inputEntry id="UnaryTests_00bnlsz">
      <text></text>
    </inputEntry>
    <outputEntry id="LiteralExpression_1atpai6">
      <text>"final otra"</text>
```

```
        </outputEntry>
      </rule>
    </decisionTable>
  </decision>
```

Para cada etiqueta `<decision>` encontrada en el archivo XML se debe de generar el esquema correspondiente. En nuestro ejemplo se observan dos tablas, el algoritmo generará dos esquemas con los atributos correspondientes. En este caso, el atributo `id` que se incluye en cada *schema* debe concatenarse con el nombre del mismo.

```
create schema tabla1(
id_tabla1 Integer,
entrada string
);

create schema tabla2(
id_tabla2 Integer,
salida Integer,
entradaNueva string
);
```

Después de crear los esquemas de las tablas, se generan los patrones asociados a las reglas de las tablas DMN. Estos patrones parten de un patrón básico realizando un ajuste en la cláusula `from` para indicar la relación entre las tablas.

En este caso la cláusula `from` incluirá todas las tablas que indiquen en el parámetro *href* de la etiqueta `<requiredDecision>` el identificador de la tabla cuyas reglas se estén tratando y que se corresponde con el valor del parámetro *id* de la etiqueta `<decision>`. El nombre de la tabla se obtiene del parámetro *name* de la etiqueta `<decision>` correspondiente. A continuación, se muestran los patrones generados para las diferentes reglas asociadas a las tablas de nuestro ejemplo. Los dos primeros patrones corresponden a las reglas de la tabla1 y se generan de acuerdo al formato de patrón básico. Sin embargo, en el caso de la tabla2 los datos de entrada corresponden a los generados por los patrones de la tabla1 y esto se indica en la cláusula `from` haciendo referencia al último evento producido por la tabla1. Para ello se utiliza el método `std:lastevent()` por Esper.

```
@Name('tabla1_Rule0')
insert into tabla1_Output
Select id_tabla1,current_timestamp.format() as date_tabla1, 1 as salida1
from tabla1
Where entrada ='entrada1';

@Name('tabla1_Rule1')
```

```
insert into tabla1_Output
Select id_tabla1,current_timestamp.format() as date_tabla1, 2 as salida1
from tabla1
Where entrada ='entrada2';

@Name('tabla2_Rule0')
insert into tabla2_Output
Select id_tabla2,current_timestamp.format() as date_tabla2,
'final salida 1' as final
from tabla2.std:lastevent() as tabla2, tabla1.std:lastevent() as tabla1
Where salida = 1 and entradaNueva='a';

@Name('tabla2_Rule1')
insert into tabla2_Output
Select id_tabla2,current_timestamp.format() as date_tabla2,
'final salida 2' as final
from tabla2.std:lastevent() as tabla2, tabla1.std:lastevent() as tabla1
Where salida = 2  and entradaNueva='b';

@Name('tabla2_Rule2')
insert into tabla2_Output
Select id_tabla2,current_timestamp.format() as date_tabla2,
'final salida 2' as final
from tabla2.std:lastevent() as tabla2, tabla1.std:lastevent() as tabla1
Where salida in (1,2);
```

Capítulo 4

Implementación

En este capítulo se describe la implementación de la herramienta de generación automática de patrones y las diferentes herramientas tecnológicas utilizadas para el desarrollo de la misma.

4.1. Arquitectura del sistema

Este proyecto se divide en tres módulos: patrones EPL, resultados Esper y vista. Estos elementos se conectan entre sí por medio de un controlador, para facilitar el manejo de errores y permitiendo que el sistema sea escalable. Los dos primeros módulos contienen las funcionalidades y datos básicos mientras que la vista maneja la información que envía o recibe el usuario. El controlador es el encargado de manejar las entradas del usuario y enviarlas al módulo correspondiente para realizar la ejecución.

4.1.1. Módulo de patrones EPL

Este módulo tiene la representación de los datos del archivo XML y sus transformaciones en patrones EPL. Este módulo no tiene conocimiento de la vista o del controlador. En la Figura 4.1 se muestran las clases que se emplean en el diseño del mismo.

La gestión del archivo XML se realiza por medio de la clase *FileXML* que contiene los métodos relativos para controlar los datos del archivo, lectura, transformaciones de datos y sintaxis EPL. Al finalizar la lectura y almacenamiento de la información guarda los esquemas y patrones en un archivo de texto (.txt).

La lectura del archivo XML se realiza con el método *parseXML* y almacena la información en una lista y con las clases objeto de Java se hace referencia a cualquier información almacenada en el archivo, donde cada una de estas clases tiene asignado el mismo nombre de la etiqueta XML correspondiente y los parámetros necesarios para realizar el patrón.

Los esquemas y patrones se generan por medio del método *crearTablaParse*. La primera

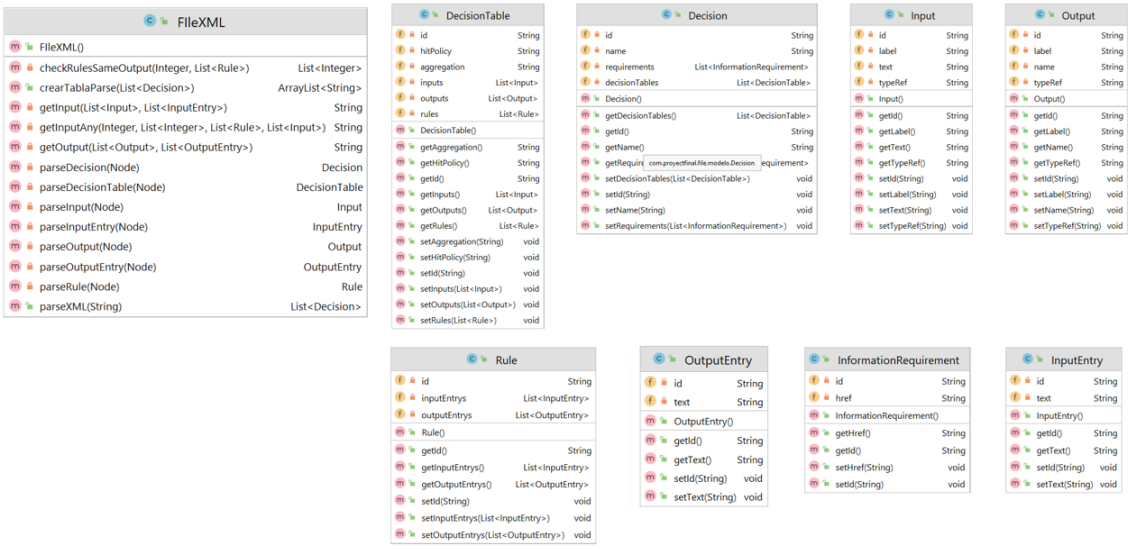

Figura 4.1: Clases del módulo de patrones EPL

Figura 4.2: Clases de módulo de resultado Esper

parte del método se encarga de seleccionar la información de la etiqueta `input` y por cada etiqueta `decision` existente crea el esquema correspondiente.

La segunda parte del método evalúa la *hit policy* asociada y de acuerdo a dicho parámetro selecciona la estructura adecuada para la creación del patrón.

Por último almacena la información en el fichero de texto con el esquema y reglas encontradas.

4.1.2. Módulo resultados Esper

Este módulo parte del archivo de texto creado por el módulo de patrones EPL y los datos fuente creados a partir de la información de cada esquema y almacenados en archivos csv.

En la Figura 4.2 se muestran las clases que se emplean en el desarrollo de este módulo de la aplicación.

La clase *classEsper* se encarga de compilar los esquemas y patrones del archivo por medio de las rutinas *EpCompiled*, *EPDeployment* y *EPStatement* de la librería de Esper mediante el siguiente código.

```
// compilamos todos los creates encontrados en los ficheros
```

```
for (String createsInArray : creates)
{
    epCompiled = compiler.compile(createsInArray, args1);
}
// compilamos todos los @Name encontrados en los ficheros
for (String namesInArray : names)
{
    epCompiled = compiler.compile(namesInArray, args1);
}
```

Cada compilación pasa por la interfaz destruir una declaración, así como obtener información de la declaración, como el nombre de la declaración, el texto de la expresión y el estado actual. Las declaraciones tienen 3 estados: iniciado, detenido y destruido.

En el estado iniciado, las declaraciones evalúan activamente los flujos de eventos de acuerdo con la expresión de la declaración. Las declaraciones iniciadas se pueden detener y destruir.

En estado detenido, las sentencias están inactivas. Las declaraciones detenidas se pueden iniciar, en cuyo caso comienzan a evaluar activamente los flujos de eventos, o se pueden destruir.

Las declaraciones destruidas han perdido todos los recursos de la declaración y no se pueden iniciar ni detener.

La clase *genericListener* implementa *UpdateListener* encargada de notificar eventos nuevos y antiguos. Por medio de la interfaz *EventBean*, permite consultar el tipo de evento, los valores de propiedad del evento y el objeto de evento subyacente, para generar correctamente cada respuesta a los diferentes patrones que está procesando.

4.1.3. Vista

Este módulo es el encargado de interactuar con el usuario y contará con las siguientes funcionalidades.

- Página principal (*Home*).
- Carga de archivos XML.
- Generación de patrones EPL.
- Carga de archivos de datos CSV.
- Generación de resultados.

La Figura 4.3 presenta la vista correspondiente a la página principal.

La carga de archivos XML, permite al usuario cargar los archivos con la información definida en las tablas DMN y solicitar la generación automática de los patrones que serán

Figura 4.3: Vista página principal

almacenados en un archivo de texto. La Figura 4.4 presenta la vista correspondiente a estas funcionalidades.

Para generar los patrones EPL el sistema recibe el archivo XML almacenándolo en una carpeta temporal mientras el algoritmo genera los patrones. Al finalizar, si el archivo se procesó correctamente muestra en pantalla el archivo generado, si no envía un mensaje de error, lo que indica que no ha podido procesarse el archivo XML.

Figura 4.4: Vista carga archivo XML

Al finalizar la generación de los patrones EPL, ya se encuentran dispuestos para procesar datos y generar las respuestas a los patrones. En la Figura 4.5 se observa la vista que permite subir uno o más archivos csv con la información a procesar por los patrones. Esta

vista primero permite cargar los archivos al controlador y si la carga es correcta activa el botón para la generación de respuestas de patrones, mostrando en la parte inferior la lista de ficheros que contendrán las respuestas generadas. Se debe tener en cuenta que por cada esquema se debe cargar un archivo fuente del mismo nombre y generará con el mismo nombre el archivo de respuesta.

Figura 4.5: Vista Resultados

4.1.4. Controlador

El controlador es el encargado de manejar las entradas del usuario realizadas en la vista e interactúa con los módulos de patrones EPL y Esper en función de la acción a realizar. Para finalizar pasa a la vista la información a desplegar al usuario. En la figura 4.6 podemos ver las funcionalidades que tiene este controlador.

La clase *FileController* es la encargada de gestionar la relación entre las entradas del usuario y la lógica necesaria para que se ejecuten los patrones. Se inicializa recibiendo el archivo XML y almacenándolo en una carpeta temporal. El controlador identifica el archivo XML y lo envía para generar los patrones y al finalizar la ejecución se encarga de enviar la respuesta a la vista de ejecución exitosa que activa la carga de archivos fuente de tipo CSV para ser aplicados a los patrones que generarán los datos de respuesta.

Para continuar, el controlador recibe nuevamente uno o varios archivos fuente con datos los almacena en una carpeta temporal, y los envía a la parte de generación de respuesta, dichos datos son interpretados por medio de la librería Esper y al finalizar la ejecución el controlador verifica los archivos de salida y envía la información nuevamente a la vista de los archivos creados.

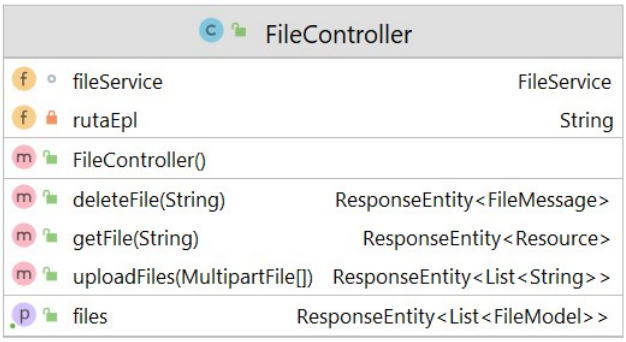

Figura 4.6: Clases del módulo controlador

4.2. Herramientas utilizadas

Para el desarrollo e implementación de la herramienta Web se utilizaron las siguientes tecnologías y *frameworks*.

4.2.1. HTML

HTML es un lenguaje de marcado de hipertexto que se usa para construir el contenido de las páginas web. Se basa en etiquetas, a través de las cuales se definen los documentos. El lenguaje HTML ha sido utilizado en el desarrollo de la aplicación para diseñar la interfaz. Gracias a este lenguaje de marcado, se ha distribuido los distintos elementos de la misma: en la parte izquierda un menú lateral, con el que se navega a todos los módulos de la aplicación, y en la parte central el contenido correspondiente a cada una de estos módulos.

4.2.2. Cascading Style Sheets

Cascade Style Sheets, más conocido como CSS, es un lenguaje de reglas de estilo que se utiliza para definir la presentación del contenido HTML. En este trabajo se ha utilizado para establecer los colores, tipos de letra y la maquetación de los elementos que componen los formularios.

4.2.3. Angular

Angular es un framework JavaScript potente, muy adecuado para el desarrollo de aplicaciones frontend modernas, de complejidad media o elevada. El framework Angular ofrece una base para el desarrollo de aplicaciones robustas, escalables y optimizadas, que promueve además las mejores prácticas y un estilo de codificación homogéneo y de gran modularidad. https://angular.io/

4.2.4. Framework Intellij IDEA

Es un entorno de desarrollo multilenguaje, se ha utilizado para la programación con java y angular, usando dependencias como Gradle y Mavem que son utilizadas en el desarrollo de la herramienta, permitiendo crear una aplicación sencilla y adaptable. https://www.jetbrains.com/es-es/lp/idea-extended-trial/

Capítulo 5

Experimentos

En este capítulo se presentan los resultados del experimento que se ha llevado a cabo para evaluar la funcionalidad de la propuesta. Por un lado, se analiza la carga del archivo DMN y los patrones generados de dicho archivo. Por otro lado, se comparan los patrones con datos de entrada obteniendo información con todas las combinaciones posibles que nos permitan las condiciones de los patrones anteriormente generados, permitiendo clasificar información importante y adecuada para cualquier problema.

El algoritmo se ha aplicado en un caso de estudio referente a un catálogo de servidores Valencia-Parra et al. (2021) construidos a partir de datos proporcionados por terceros. Las tablas de decisión establecen reglas que analizan los requisitos de calidad para tomar una decisión sobre cual servidor tiene la mejor calidad y uso.

En la Figura 5.1 podemos observar el diagrama de las 17 tablas de decisión relacionadas a las cuales se le aplicará el proceso de generación y ejecución de patrones EPL. Asimismo, las Figuras 5.2, 5.3 y 5.4 presentan las descripciones de las tablas DMN tal como han sido definidas en *Camunda*.

5.1. Descripción Experimentos

El experimento se divide en dos partes. En la primera parte ha llevado a cabo la lectura del archivo XML obtenido a través de *Camunda* y la generación de patrones EPL. En la segunda parte se ha realizado el experimento para la obtención de resultados de la ejecución de los patrones en el motor CEP de Esper a conjuntos de datos generados artificialmente.

El objetivo de este experimento es evaluar si el algoritmo es capaz de responder correctamente cuando tenemos varias tablas DMN relacionadas usando simultáneamente varias entradas procedentes de decisiones tomadas por la aplicación de reglas asociadas a otras tablas.

El algoritmo procesa el archivo XML generado por *Camunda* y analiza las tablas de

decisión para la creación de los esquemas. Este proceso se realiza automáticamente en orden ascendente para poder determinar las dependencias, desde la parte inferior izquierda y en orden ascendente hasta alcanzar la tabla que aparece en la posición superior del diagrama. Al finalizar la creación de los esquemas, el algoritmo comienza la creación de los patrones en el mismo orden que se crearon los esquemas. Los esquemas y patrones se pueden encontrar en el repositorio, a modo de ejemplo incluimos continuación los patrones correspondientes a la tabla DMN `DQAssessment`.

```
@Name('DQAssessment_Rule0')
@drop
@Priority(3)
insert into DQAssessment_Output
Select id, current_timestamp.format() as date_DQAssessment,
'suitable' as DQAssessment
from DQAssessment.std:lastevent() as DQAssessment,
Completeness.std:lastevent() as Completeness , Accuracy.std:lastevent()
as Accuracy, Consistency.std:lastevent() as Consistency
Where Completeness in ('adequatelycomplete') and Accuracy = 100 and
Consistency in ('consistent');

@Name('DQAssessment_Rule1')
@drop
@Priority(2)
insert into DQAssessment_Output
Select id, current_timestamp.format() as date_DQAssessment,
'enough quality' as DQAssessment
from DQAssessment.std:lastevent() as DQAssessment,
Completeness.std:lastevent() as Completeness , Accuracy.std:lastevent()
as Accuracy, Consistency.std:lastevent() as Consistency
Where Completeness in ('notcomplete') and Accuracy >=70 and Consistency
in ('consistent') ;

@Name('DQAssessment_Rule2')
@drop
@Priority(1)
insert into DQAssessment_Output
Select id, current_timestamp.format() as date_DQAssessment,
'bad quality' as DQAssessment
from DQAssessment.std:lastevent() as DQAssessment,
Completeness.std:lastevent() as Completeness , Accuracy.std:lastevent()
as Accuracy, Consistency.std:lastevent() as Consistency
Where Completeness in ('notcomplete') and Accuracy >=50 and Consistency
in ('consistent') ;
```

```
@Name('DQAssessment_Rule3')
@drop
@Priority(0)
insert into DQAssessment_Output
Select id, current_timestamp.format() as date_DQAssessment,
'non usable' as DQAssessment
from DQAssessment.std:lastevent() as DQAssessment,
Completeness.std:lastevent() as Completeness,
Accuracy.std:lastevent() as Accuracy,
Consistency.std:lastevent() as Consistency;
Where salida in (1,2);
```

Partiendo de los esquemas y patrones creados se realiza la compilación para el despliegue en el motor de Esper, que permitirá relacionar y analizar la información de los datos recibidos para emitir decisiones en base a las reglas definidas.

5.2. Resultados

Este experimento destaca por tener múltiples tablas de decisión y reglas. Esper generará archivos de resultados para cada uno de los patrones que produzcan resultados a partir de los eventos recibidos.

En este experimento se han utilizado tantos archivos de datos fuente como tablas aparecen en nivel inferior de nuestro diagrama. Hay que tener en cuenta que las tablas `Completeness, Accuracy, Consistency y DQAssessment` reciben información generada por otros patrones por lo que no se incluyen ficheros de datos para ellas. Cada fichero contiene información correspondiente a 100 eventos, cada uno de ellos ajustados al esquema correspondiente.

Al finalizar la compilación de los datos en el motor Esper, se obtienen los resultados obtenidos de los patrones que se hayan ejecutado. A continuación, a modo de ejemplo se muestran los resultados correspondientes a los patrones asociados a las reglas de la tabla `DQAssessment`. Como puede verse se muestra el `id` del evento que ha disparado el patrón, la fechas y hora, así como la salida producida por dicho patrón.

```
id: 1, date_DQAssessment: 12/01/23 15:24, DQAssessment:'suitable'
id: 4, date_DQAssessment: 12/01/23 15:24, DQAssessment:'suitable'
id: 6, date_DQAssessment: 12/01/23 15:24, DQAssessment:'suitable'
id: 32, date_DQAssessment: 12/01/23 15:24, DQAssessment:'suitable'

id: 2, date_DQAssessment: 12/01/23 15:24, DQAssessment:'enough quality'
id: 7, date_DQAssessment: 12/01/23 15:24, DQAssessment:'enough quality'
id: 16, date_DQAssessment: 12/01/23 15:24, DQAssessment:'enough quality'
id: 45, date_DQAssessment: 12/01/23 15:24, DQAssessment:'enough quality'
```

```
id: 22, date_DQAssessment: 12/01/23 15:24, DQAssessment:'bad quality'
id: 25, date_DQAssessment: 12/01/23 15:24, DQAssessment:'bad quality'
id: 78, date_DQAssessment: 12/01/23 15:24, DQAssessment:'bad quality'
id: 97, date_DQAssessment: 12/01/23 15:24, DQAssessment:'bad quality'

id: 3, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 5, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 8, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 9, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 10, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 11, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
id: 12, date_DQAssessment: 12/01/23 15:24, DQAssessment:'non usable'
...
```

El conjunto de todos los archivos que se han generado en este experimento los podemos encontrar en el siguiente enlace:
`https://drive.google.com/drive/folders/1ZbejbMzwznEnACxwn4ZwOJTtzwTc16gX`.

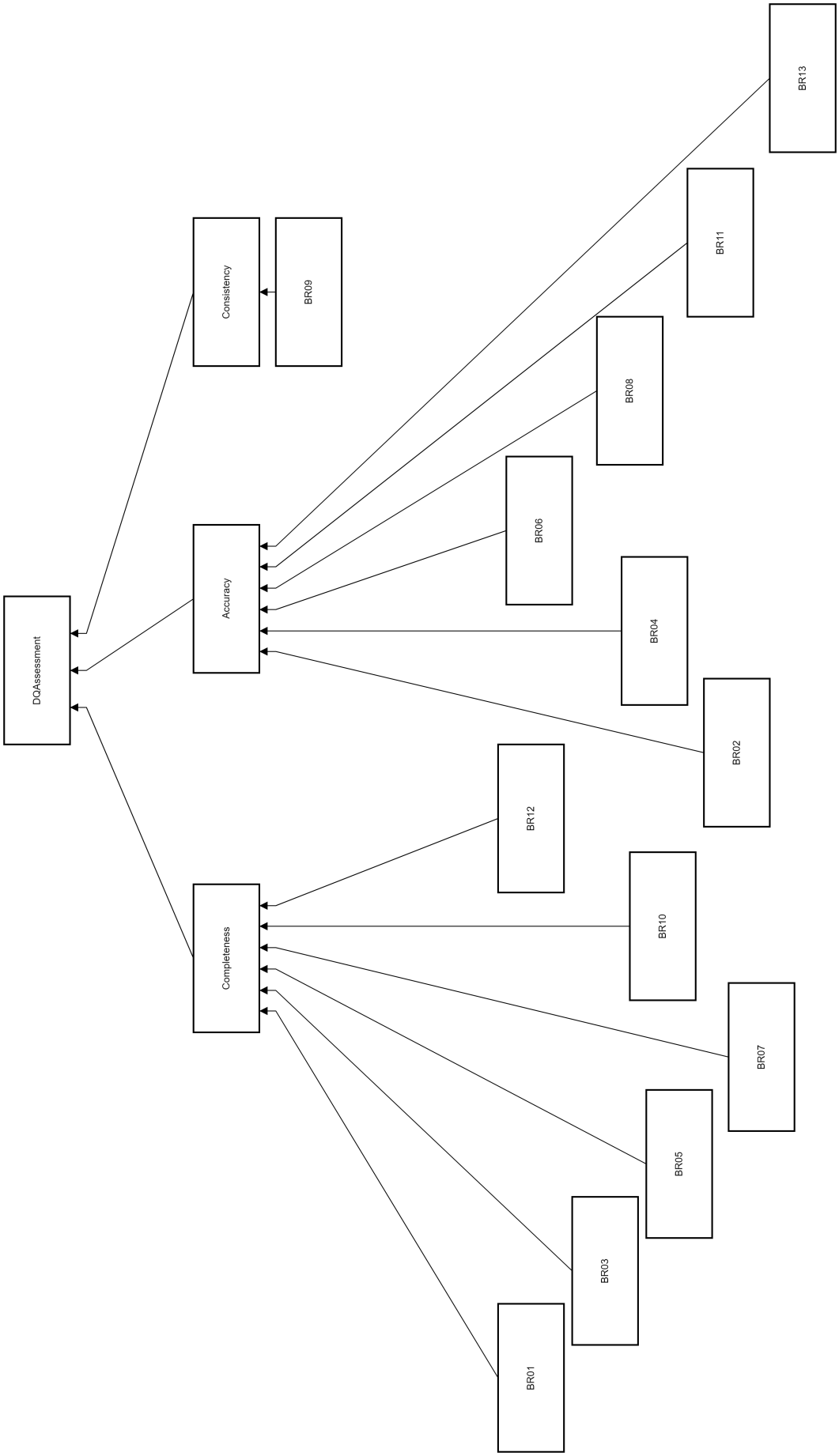

Figura 5.1: Tablas DMN del experimento

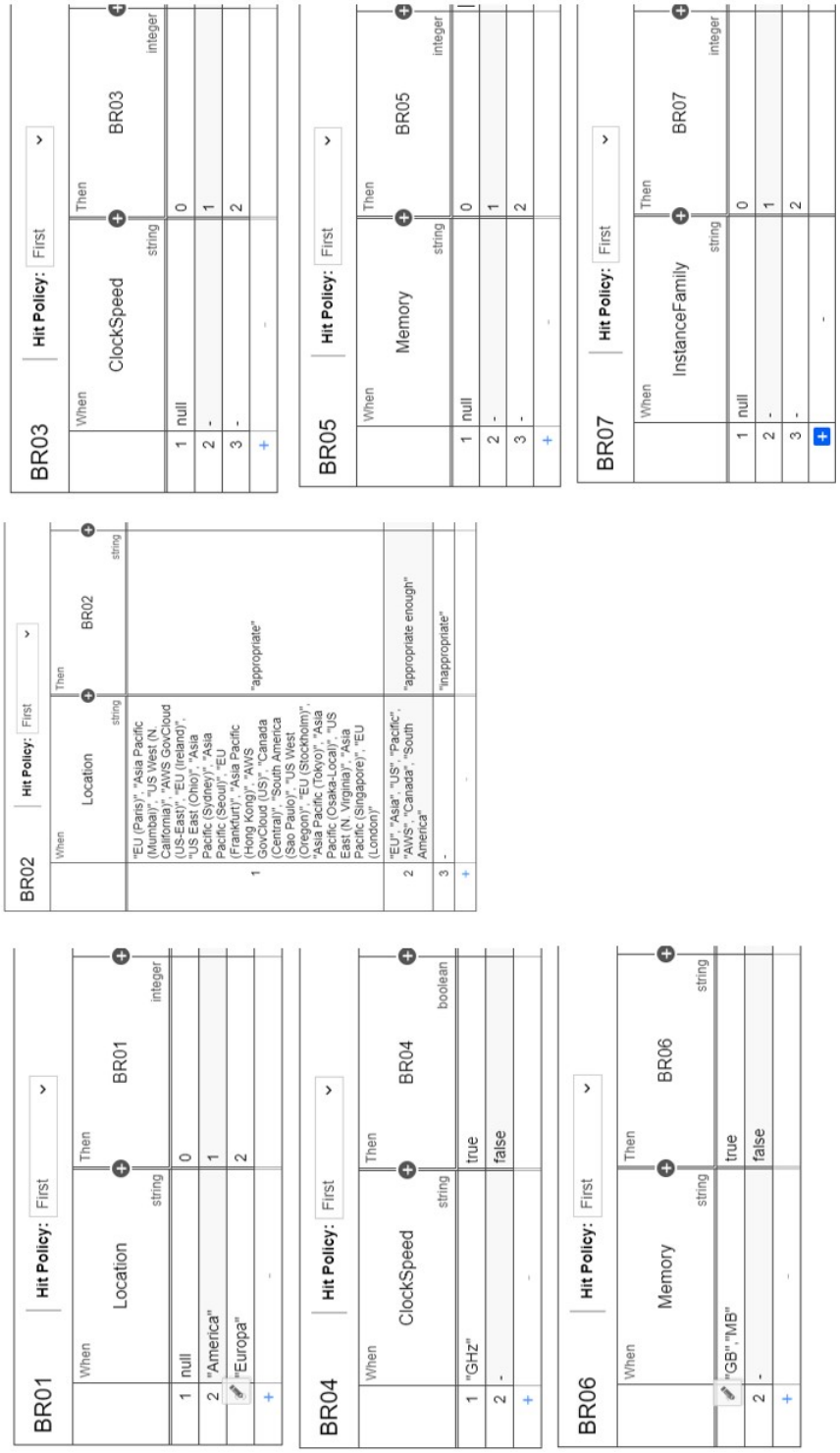

Figura 5.2: Tablas DMN del experimento (I)

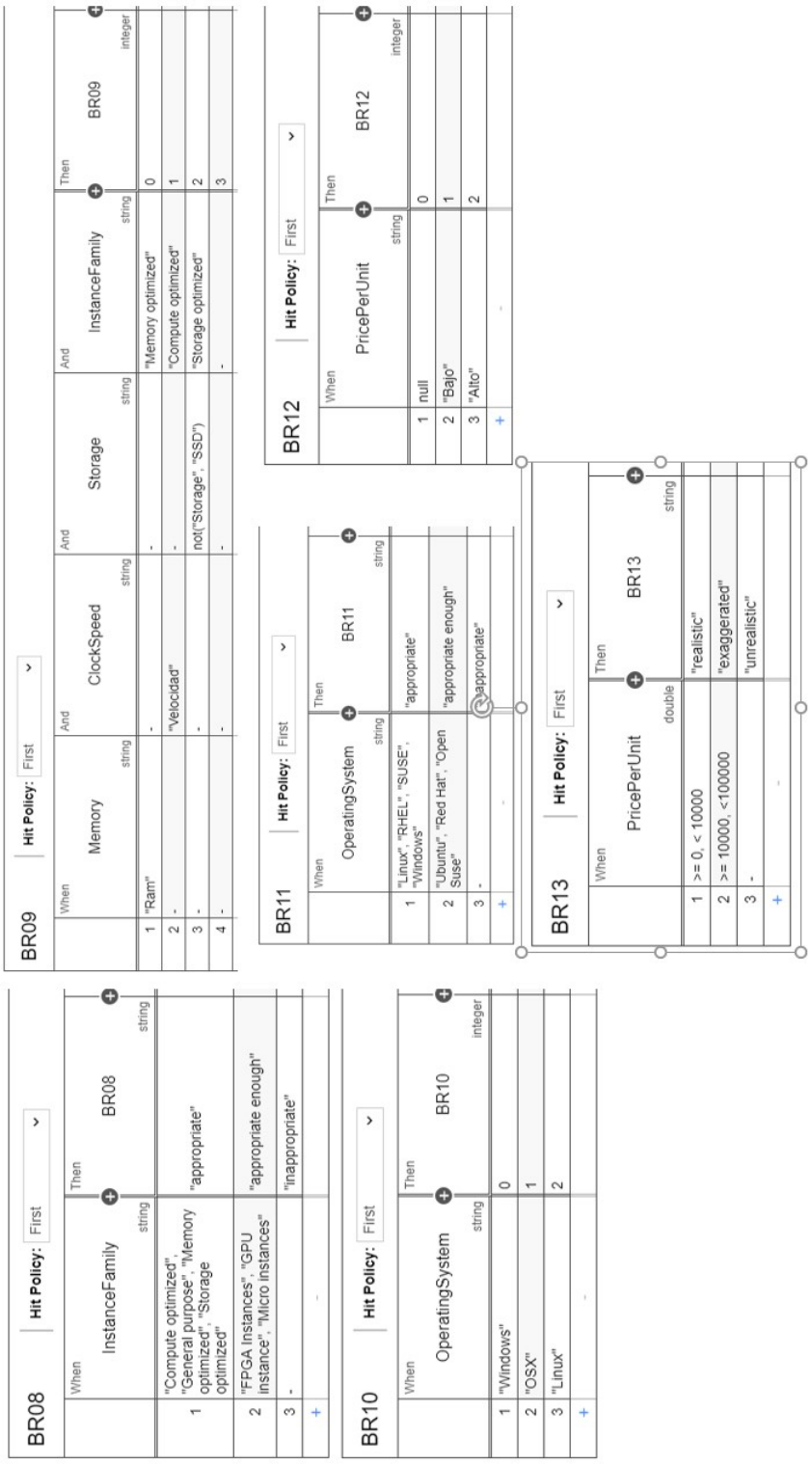

Figura 5.3: Tablas DMN del experimento(II)

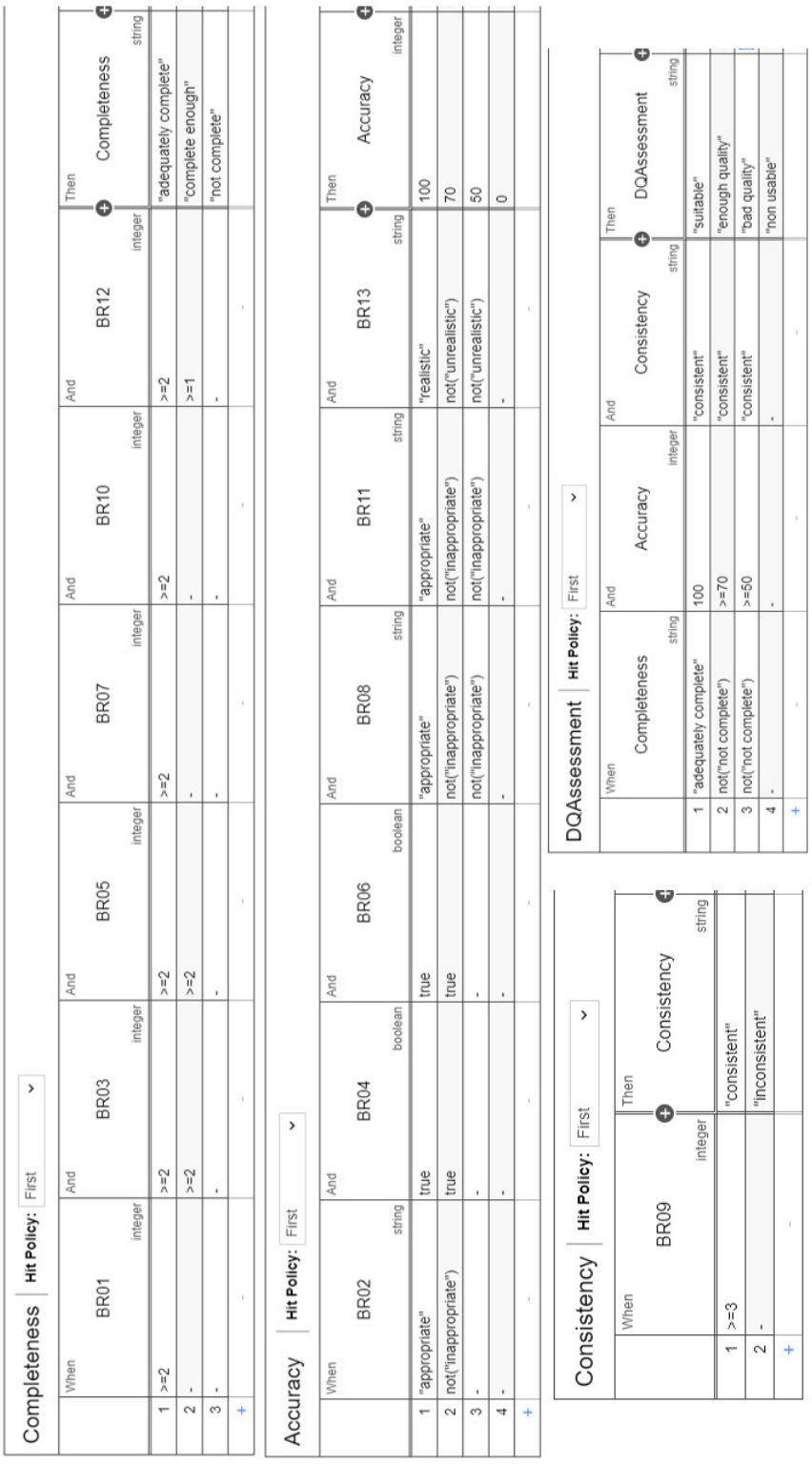

Figura 5.4: Tablas DMN del experimento (III

Capítulo 6

Conclusiones y Trabajo Futuro

En este trabajo hemos propuesto una herramienta de generación automática de patrones a partir de modelos DMN y se ha definido un conjunto de transformaciones específicas para ello. Se ha implementado una aplicación que permite realizar dichas trasformaciones a patrones en el lenguaje EPL de Esper y se ha verificado su funcionalidad con experimentos.

La herramienta se ha implementado para que genere los resultados cargando un archivo XML con la información de tablas de decisión de *Camunda*, se pueda leer y transformar la información de acuerdo a las especificaciones del lenguaje EPL. La herramienta desarrollada permite generar automáticamente los patrones de acuerdo a las políticas de aplicación que tiene cada tabla y determina la cantidad de patrones que se deben generar para las reglas definidas.

En cuanto al alcance del trabajo, se puede decir que se han cumplido los objetivos específicos planteados: generación automática de patrones EPL a partir de tablas DMN y despliegue del motor CEP de Esper para procesar y clasificar la información en base a los patrones generados. Además se ha realizado la evaluación de los resultados obtenidos de la aplicación desarrollada mediante un caso de estudio que abarca varias tablas y reglas.

Como trabajo futuro se podrían aplicar dos mejoras. La primera que se pudieran recibir archivos de tablas de decisión diseñadas en plataformas diferentes a *Camunda*, como por ejemplo *Kogito*. La segunda mejora sería aplicar el despliegue de resultados a otros motores CEP diferentes de Esper.

Esta aplicación con sus ficheros fuente están publicados bajo licencia de código abierto y libre a través de los enlaces: Patrones EPL BackEnd y Patrones EPL FrontEnd.

Chapter 7

Introduction

This chapter introduces the motivation for this work and the objectives to be achieved. It also indicates the structure of this report.

7.1. Motivation

Nowadays, the need to monitor data in real time, quickly and with as little error as possible, makes it increasingly necessary to use tools that can be the use of tools that facilitate this process is becoming increasingly necessary. process. The processing of information is a key point in the decision making process of organisations and requires the application of tools that facilitate this process that must be agile and efficient in order to be able to support the large volume of data currently generated.

Complex Event Processing (CEP) is a technique for processing events in real time which analyses cause-effect relationships between related events and allows us to make effective decisions in specific situations. Event conditions are established by a user in the form of patterns are encoded in the corresponding event processing language. A person with no prior knowledge of the language will be overwhelmed by the high complexity of the design of such patterns. It is for this reason that it can be really interesting to create an automatic pattern generator from a simple and intuitive specification that is manageable and intuitive for a user with no previous knowledge, having a graphical interface. This could bring people closer to modelling the specific conditions to be considered when analysing their events and and the actions they wish to take in case of detecting that these conditions are met.

7.2. Objectives

The general objective of this work is to create a tool for the automatic generation and deployment of patterns in an Event Processing Language (EPL) from business rules

defined in a user-friendly platform through the use of a Decision Model and Notation (DMN) method. This tool will provide the automatic transformation of these models into patterns that can be deployed for activation and event checking. In order to achieve this main objective, the following specific objectives have been determined.

- Analyse and create DMN models
- Automatically generate EPL queries from the DMN models
- Deployment in a complex event processing engine
- Evaluate the new tool with a case study

7.3. Work plan

In order to fulfil the main objective of this work it is necessary to plan the tasks to be carried out and to divide the development of the project in several phases. Finally, the following phases have been defined:

1. Review information on available DMNs to select the most suitable one.
    - Review and analyse the design of the decision tables
    - Analysing files obtained for exploitation.
2. Design and implement an algorithm for automatic pattern generation
    - Review and analyse different complex event processing engines
    - Determine the translation of DMN models into patterns according to their characteristics
    - Design an algorithm to generate EPL patterns for the different application policies of the DMN model
    - Perform tests to check the implemented functionality
3. Design the deployment of the patterns in the selected CEP engine
4. Tool
    - Automation of the system by integrating the automatic generation of the patterns and the deployment
    - User interface design
5. Development of a case study

7.4. Structure of the document

The structure of the document is as follows:

- Chapter 2 presents the essential information to understand the work developed on DMN models, CEP engine and EPL patterns
- Chapter 3 describes the development of the project
- Chapter 4 describes the implementation of the project
- Chapter 5 explains the case study that has been carried out
- Chapter 6 shows the conclusions and possible future extensions of this tool

Chapter 8

Conclusions and Future Work

In this work we have proposed a tool for the automatic generation of patterns from DMN models and a set of specific transformations has been defined for this purpose.

An application has been implemented to perform the transformations to patterns in the Esper EPL language and its functionality has been verified with experiments.

The application has been implemented so that, from an XML file with the information from the Camunda decision tables, it allows reading and transforming the information according to the specifications of the EPL language. The developed tool allows the automatic generation of the patterns according to the application policies that each table presents and determines the number of patterns to be generated for the defined rules.

In terms of the scope of the work, it can be said that the specific objectives have been fulfilled: automatic generation of EPL patterns from DMN tables and deployment of the Esper CEP engine to process and classify the information based on the generated patterns. In addition, the results obtained from the developed application have been evaluated by means of a case study covering several tables and rules.

Two improvements can be proposed as future work. The first one would be to deal with decision tables from platforms other than *Camunda*, such as *Kogito*. The second improvement would be to apply the deployment of results to other CEP engines different from Esper.

The tool with its source files are published under open source license and free through the links: EPL BackEnd Patterns and EPL FrontEnd Patterns.

Apéndice

Listing 8.1: Fichero XML de tabla DMN de ejemplo de Capítulo 3

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="https://www.omg.org/spec/DMN/20191111/MODEL/"
    xmlns:dmndi="https://www.omg.org/spec/DMN/20191111/DMNDI/"
    xmlns:dc="http://www.omg.org/spec/DMN/20180521/DC/"
    xmlns:biodi="http://bpmn.io/schema/dmn/biodi/2.0" id="Definitions_11m1wo9"
    name="DRD" namespace="http://camunda.org/schema/1.0/dmn"
    exporter="Camunda Modeler" exporterVersion="4.6.0">
  <decision id="Decision_0gip5xq" name="Datos">
    <decisionTable id="DecisionTable_0lwxmib">
      <input id="Input_1" label="Dato String">
        <inputExpression id="InputExpression_1" typeRef="string">
          <text>datoString</text>
        </inputExpression>
      </input>
      <input id="InputClause_0kihd5b" label="Dato Boolean">
        <inputExpression id="LiteralExpression_0d47n2r" typeRef="boolean">
          <text>datoBoolean</text>
        </inputExpression>
      </input>
      <input id="InputClause_14gr2kk" label="Dato Integer">
        <inputExpression id="LiteralExpression_12s2pj9" typeRef="integer">
          <text>datoInteger</text>
        </inputExpression>
      </input>
      <input id="InputClause_0vlrnnw" label="Dato Date">
        <inputExpression id="LiteralExpression_1tgb03t" typeRef="date">
          <text>datoDate</text>
        </inputExpression>
      </input>
      <output id="Output_1" label="Output String" name="outString"
      typeRef="string" biodi:width="192" />
```

```
<output id="OutputClause_0jt21py" label="Output Boolean"
name="outputBoolean" typeRef="boolean" />
<output id="OutputClause_0shg3dn" label="Output Integer"
name="outputInteger" typeRef="integer" />
<output id="OutputClause_0i93ava" label="Output Date"
name="outputDate" typeRef="date" />
<rule id="DecisionRule_011d593">
  <inputEntry id="UnaryTests_012wgio">
    <text>"ABC","DEF","GHI"</text>
  </inputEntry>
  <inputEntry id="UnaryTests_1ixtuq5">
    <text>true</text>
  </inputEntry>
  <inputEntry id="UnaryTests_01q5nh9">
    <text>[1..10]</text>
  </inputEntry>
  <inputEntry id="UnaryTests_1rxjy8t">
    <text>&lt; date and time("2022-05-11T00:00:00")</text>
  </inputEntry>
  <outputEntry id="LiteralExpression_0fwveuk">
    <text>"Rule 1 "</text>
  </outputEntry>
  <outputEntry id="LiteralExpression_1szvqyo">
    <text>true</text>
  </outputEntry>
  <outputEntry id="LiteralExpression_05kdn1s">
    <text>1</text>
  </outputEntry>
  <outputEntry id="LiteralExpression_10x7uis">
    <text>date and time("2022-05-11T00:00:00")</text>
  </outputEntry>
</rule>
<rule id="DecisionRule_17jkw46">
  <inputEntry id="UnaryTests_0x227z1">
    <text>not("ABC","DEF")</text>
  </inputEntry>
  <inputEntry id="UnaryTests_1tfosc1">
    <text>false</text>
  </inputEntry>
  <inputEntry id="UnaryTests_08w53ki">
    <text>&lt; 0</text>
  </inputEntry>
  <inputEntry id="UnaryTests_0xx1xid">
    <text>date and time("2022-05-11T00:00:00")</text>
  </inputEntry>
  <outputEntry id="LiteralExpression_0w6idql">
    <text>"Rule 2"</text>
```

```
      </outputEntry>
      <outputEntry id="LiteralExpression_0oei0yo">
        <text>false</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_142oy0h">
        <text>2</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_0wndye6">
        <text>date and time("2022-05-12T00:00:00")</text>
      </outputEntry>
    </rule>
    <rule id="DecisionRule_0vpxerv">
      <inputEntry id="UnaryTests_08l58dz">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_1b47ga1">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_1rp7ea8">
        <text>&lt;= 0</text>
      </inputEntry>
      <inputEntry id="UnaryTests_0aw7mrh">
        <text>&gt; date and time("2022-05-11T00:00:00")</text>
      </inputEntry>
      <outputEntry id="LiteralExpression_0g125gx">
        <text>"Rule 3"</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_1h9jd4j">
        <text></text>
      </outputEntry>
      <outputEntry id="LiteralExpression_08463lr">
        <text>3</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_15ppax4">
        <text>date and time("2022-05-13T00:00:00")</text>
      </outputEntry>
    </rule>
    <rule id="DecisionRule_0m5kmh9">
      <inputEntry id="UnaryTests_1rchjtt">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_1bmqliy">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_026lrsh">
        <text>&gt; 0</text>
      </inputEntry>
```

```
      <inputEntry id="UnaryTests_1hkiqcn">
        <text>[date and time("2022-05-12T00:00:00")..date and
        time("2022-05-15T00:00:00")]</text>
      </inputEntry>
      <outputEntry id="LiteralExpression_1jtx5rh">
        <text>"Rule 4"</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_0cjg201">
        <text></text>
      </outputEntry>
      <outputEntry id="LiteralExpression_05jny8h">
        <text>4</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_04nd9vj">
        <text>date and time("2022-05-14T00:00:00")</text>
      </outputEntry>
    </rule>
    <rule id="DecisionRule_1n1voop">
      <inputEntry id="UnaryTests_1efhi7y">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_09bkv07">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_1gleg5i">
        <text>&gt;= 0</text>
      </inputEntry>
      <inputEntry id="UnaryTests_1uii2py">
        <text></text>
      </inputEntry>
      <outputEntry id="LiteralExpression_0e8tbey">
        <text>"Rule 5"</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_0erg33o">
        <text></text>
      </outputEntry>
      <outputEntry id="LiteralExpression_11g67ht">
        <text>-5</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_00le1xx">
        <text>date and time("2022-05-15T00:00:00")</text>
      </outputEntry>
    </rule>
    <rule id="DecisionRule_0zba79n">
      <inputEntry id="UnaryTests_048hxeg">
        <text></text>
      </inputEntry>
```

```
      <inputEntry id="UnaryTests_1yvgiu7">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_05lwiy6">
        <text>0</text>
      </inputEntry>
      <inputEntry id="UnaryTests_084871p">
        <text></text>
      </inputEntry>
      <outputEntry id="LiteralExpression_0zz19iu">
        <text>"Rule 6"</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_0smf0xf">
        <text></text>
      </outputEntry>
      <outputEntry id="LiteralExpression_1lyr613">
        <text>-6</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_15i3yvd">
        <text></text>
      </outputEntry>
    </rule>
    <rule id="DecisionRule_1vyskmh">
      <inputEntry id="UnaryTests_0kstbtt">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_03tcf1f">
        <text></text>
      </inputEntry>
      <inputEntry id="UnaryTests_087g31h">
        <text>[0..20[</text>
      </inputEntry>
      <inputEntry id="UnaryTests_0drg2o6">
        <text></text>
      </inputEntry>
      <outputEntry id="LiteralExpression_1i5fv64">
        <text>"Rule 7"</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_1txn0lf">
        <text></text>
      </outputEntry>
      <outputEntry id="LiteralExpression_1s11g37">
        <text>-7</text>
      </outputEntry>
      <outputEntry id="LiteralExpression_0hr2d6n">
        <text></text>
      </outputEntry>
```

```
</rule>
<rule id="DecisionRule_1wqv1p9">
  <inputEntry id="UnaryTests_087xqqo">
    <text></text>
  </inputEntry>
  <inputEntry id="UnaryTests_0fqykjs">
    <text></text>
  </inputEntry>
  <inputEntry id="UnaryTests_0kfwmxk">
    <text>]-3..-1]</text>
  </inputEntry>
  <inputEntry id="UnaryTests_05mrpd9">
    <text></text>
  </inputEntry>
  <outputEntry id="LiteralExpression_0dg9llm">
    <text>"Rule 8"</text>
  </outputEntry>
  <outputEntry id="LiteralExpression_0kr6xdd">
    <text></text>
  </outputEntry>
  <outputEntry id="LiteralExpression_1wdym58">
    <text>-8</text>
  </outputEntry>
  <outputEntry id="LiteralExpression_069sqly">
    <text></text>
  </outputEntry>
</rule>
<rule id="DecisionRule_0rb18aq">
  <inputEntry id="UnaryTests_0fulsj6">
    <text></text>
  </inputEntry>
  <inputEntry id="UnaryTests_08a7wia">
    <text></text>
  </inputEntry>
  <inputEntry id="UnaryTests_1wnbc1w">
    <text>]-10..-5[</text>
  </inputEntry>
  <inputEntry id="UnaryTests_077wvg5">
    <text></text>
  </inputEntry>
  <outputEntry id="LiteralExpression_1d0nym4">
    <text></text>
  </outputEntry>
  <outputEntry id="LiteralExpression_17cndxw">
    <text></text>
  </outputEntry>
  <outputEntry id="LiteralExpression_04f1y32">
```

```
          <text></text>
        </outputEntry>
        <outputEntry id="LiteralExpression_03jx73v">
          <text></text>
        </outputEntry>
      </rule>
      <rule id="DecisionRule_1k2s74t">
        <inputEntry id="UnaryTests_0k8gi63">
          <text></text>
        </inputEntry>
        <inputEntry id="UnaryTests_0711ua7">
          <text></text>
        </inputEntry>
        <inputEntry id="UnaryTests_16cggum">
          <text></text>
        </inputEntry>
        <inputEntry id="UnaryTests_105kfjy">
          <text></text>
        </inputEntry>
        <outputEntry id="LiteralExpression_01chpit">
          <text></text>
        </outputEntry>
        <outputEntry id="LiteralExpression_0g5as0y">
          <text></text>
        </outputEntry>
        <outputEntry id="LiteralExpression_0eebs0w">
          <text></text>
        </outputEntry>
        <outputEntry id="LiteralExpression_0zc5ah1">
          <text></text>
        </outputEntry>
      </rule>
    </decisionTable>
  </decision>
  <dmndi:DMNDI>
    <dmndi:DMNDiagram>
      <dmndi:DMNShape dmnElementRef="Decision_0gip5xq">
        <dc:Bounds height="80" width="180" x="160" y="100" />
      </dmndi:DMNShape>
    </dmndi:DMNDiagram>
  </dmndi:DMNDI>
</definitions>
```

Listing 8.2: Patrones EPL generados a partir del archivo XML del Listing 8.1

```
create schema Datos(
id Integer,
datoString string,
datoBoolean boolean,
datoInteger integer,
datoDate date);


@Name('Datos_Rule0')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 1 String' as outString,
true as outputBoolean,1 as outputInteger,(cast(('2021-05-11 00:00:00'), Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'))) as outputDate
from Datos
Where datoString in ('ABC','DEF','GHI') and datoBoolean and datoInteger = 0 and
datoDate = (cast('2021-05-11 00:00:00', Date, dateformat: 'yyyy-MM-dd HH:mm:ss')) ;

@Name('Datos_Rule1')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 2 String' as outString,
false as outputBoolean,2 as outputInteger,(cast(('2021-05-12 00:00:00'), Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'))) as outputDate
from Datos
Where datoString not in (('ABC','DEF')) and Not datoBoolean and datoInteger < 0 and
datoDate.before(cast('2021-05-11 00:00:00', Date, dateformat: 'yyyy-MM-dd HH:mm:ss')) ;

@Name('Datos_Rule2')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 3 String' as outString,
3 as outputInteger,(cast(('2021-05-13 00:00:00'), Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'))) as outputDate
from Datos
Where datoInteger <= 0 and
datoDate.after(cast('2021-05-11 00:00:00', Date, dateformat: 'yyyy-MM-dd HH:mm:ss')) ;

@Name('Datos_Rule3')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 4 String' as outString,
4 as outputInteger,(cast(('2021-05-14 00:00:00'), Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'))) as outputDate
from Datos
Where datoInteger > 0 and
datoDate.between(cast('2021-05-12 00:00:00', Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'),(cast('2021-05-15 00:00:00', Date,
dateformat: 'yyyy-MM-dd HH:mm:ss')) ;
```

```
@Name('Datos_Rule4')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 5 String' as outString,
-5 as outputInteger,(cast(('2021-05-15 00:00:00'), Date,
dateformat: 'yyyy-MM-dd HH:mm:ss'))) as outputDate
from Datos
Where datoInteger >= 0 ;

@Name('Datos_Rule5')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 6 String' as outString,
-6 as outputInteger
from Datos
Where datoInteger >= 1 and datoInteger <= 10 ;

@Name('Datos_Rule6')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 7 String' as outString,
-7 as outputInteger
from Datos
Where datoInteger >= 0 and datoInteger < 20 ;

@Name('Datos_Rule7')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos, 'Rule 8 String' as outString,
-8 as outputInteger
from Datos
Where datoInteger > -3 and datoInteger <= -1 ;

@Name('Datos_Rule8')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos,
from Datos
Where datoInteger < -10 and datoInteger > -5 ;

@Name('Datos_Rule9')
insert into Datos_Output
Select id,current_timestamp.format() as date_Datos,
from Datos;
```

Bibliografía

AG, G. S. Bic platform the digital transformation suite. Actualizado 2022. Disponible en `https://www.gbtec.com/software/` (último acceso, Mayo, 2022).

Camunda.org. Simulador camunda. 2021. Disponible en `https://consulting.camunda.com/dmn-simulator/` (último acceso, mayo 2022).

Camunda.org. decision camunda. 2022a. Disponible en `https://docs.camunda.io/docs/components/modeler/dmn/decision-table/` (último acceso, mayo, 2022).

Camunda.org. Dmn decision engine. 2022b. Disponible en `https://camunda.com/platform/decision-engine/` (último acceso, Mayo 2022).

EsperTech. Esper. `http://www.espertech.com/esper/`, 2022.

Hitpass, B. y Jakob Freund, B. *BPMN Manual de Referencia y Guía Práctica 5a Edición: Con una introducción a CMMN y DMN*. BPM Center, 2017. ISBN 9789563451825.

Luckham, D. C. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Berlin, Heidelberg, 2001. ISBN 0201727897.

Object Management Group, S. D. O. Dmn decision model and notation. Actualizado 2022. Disponible en `https://www.omg.org/spec/DMN/` (último acceso, Mayo, 2022).

Projects, R. H. M. C. Kogito ergo automate. Actualizado 2022. Disponible en `https://kogito.kie.org/` (último acceso, Mayo, 2022).

Valencia-Parra, A., Parody, L., Varela-Vaca, A., Caballero, I. y Gómez-López, M. Dmn4dq: When data quality meets dmn. *Decision Support Systems*, vol. 141, página 113450, 2021. ISSN 0167-9236.