Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIO: jBPM Y CAMUNDA

Lázaro Daniel Alvarez Pérez
Asesorado por el Ing. Pablo Daniel Girón Baldizón

Guatemala, noviembre de 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIO: jBPM Y CAMUNDA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

LÁZARO DANIEL ALVAREZ PÉREZ
ASESORADO POR EL ING. PABLO DANIEL GIRÓN BALDIZÓN

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Oscar Humberto Galicia Nuñez
VOCAL V	Br. Carlos Enrique Gómez Donis
SECRETARIA	Ing. Lesbia Magalí Herrera Lopez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. José Ricardo Morales Prado
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
EXAMINADOR	Ing. César Rolando Batz Saquimux
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIO: jBPM Y CAMUNDA

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 11 de abril de 2018.

Lázaro Daniel Alvarez Pérez

Guatemala, 12 de julio de 2018

Ingeniero Carlos Azurdia
Facultad de Ingeniería USAC

Por medio de la presente hago de su conocimiento que doy por finalizado y aprobado el informe final correspondiente al tema: **"COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIO: JBPM Y CAMUNDA"**, elaborado por el estudiante Lázaro Daniel Alvarez Pérez, quien se identifica con número de carné 200511932 de la Universidad de San Carlos de Guatemala.

Sin otro particular, me es grato suscribirme.

Ingeniero Pablo Daniel Giron Baldizón
Asesor

Ing. Pablo D. Girón B
Colegiado No 11 076

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 9 de agosto de 2018

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **LÁZARO DANIEL ALVAREZ PÉREZ** con carné **200511932 y CUI 1653 13986 0101,** titulado **"COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIO: JBPM Y CAMUNDA"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIOS: jBPM Y CAMUNDA",*** *realizado por el estudiante, LÁZARO DANIEL ALVAREZ PÉREZ aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Türk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 14 de noviembre de 2018

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 505.2018

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas al Trabajo de Graduación titulado: **"COMPARATIVA Y APLICACIÓN DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DE PROCESOS DE NEGOCIOS: JBPM Y CAMUNDA"** presentado por el estudiante universitario: **Lázaro Daniel Alvarez Pérez** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala noviembre de 2018.

/echm

ACTO QUE DEDICO A:

Dios	Por darme la vida, unos maravillosos padres y permitirme alcanzar mis metas.
Mis padres	Lázaro Alvarez y Matilde Pérez, por su amor y su apoyo durante toda la carrera.
Mis hermanos	Ana Yolanda, Virna Julissa, Heber Raúl y Mónica Lisseth Alvarez Pérez, por el apoyo incondicional durante toda la carrera.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por permitir mi formación profesional.
Facultad de Ingeniería	Por brindarme los conocimientos necesarios para ser un profesional.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
s/s	Solicitudes por segundo.

GLOSARIO

API	Interfaz de programación de aplicaciones.
Componente	Elemento que brinda nuevas propiedades y funcionalidades, disponibles a través de interfaces para acceder a ellas.
Dominio	Área a la que pertenecen varios objetos o entidades que comparten características en común.
Driver	Componente de software que permite comunicar dos sistemas distintos de software como de hardware.
Evento	Mensaje que se utiliza para indicar que existió un cambio en el sistema.
Interfaz	Intermediario para la comunicación entre sistemas distintos, componentes o usuarios y ordenadores.
JSP	Java Server Page, paginas HTML con código java incrustado.
Metadata	Datos que proveen información acerca de datos.

Plugin	Aplicación que puede ser adaptada a otro sistema para extenderla de alguna funcionalidad.
Servlet	Clases java que renderizan una página HTML desde el servidor de aplicaciones.
Tipado dinámico	También llamado débilmente tipado, son lenguajes de programación que permiten alterar el tipo de dato de una variable en tiempo de ejecución.
Web	Más conocida como la World Wide Web, es la red mundial de comunicación.

RESUMEN

Inicialmente se fijan los objetivos generales y específicos a los que se desea llegar con esta investigación, posteriormente se presenta información acerca de conceptos importantes involucrados en las herramientas de BPM, como lo son: los lenguajes para la implementación de acciones en las actividades de un flujo de BPM, los factores a considerar al momento de implementar herramientas BPM y configuraciones iniciales antes de utilizar e implementar las mismas.

Posteriormente a los conceptos fundamentales que involucra las herramientas de jBPM se detalla la implementación de un flujo de BPM para el caso de una empresa aseguradora de vehículos, en la que se ponen en práctica los conceptos y fundamentos de las herramientas de BPM, y con base en estas implementaciones en cada una las plataformas jBPM y Camunda se realizan pruebas de rendimiento para brindar comparaciones y conclusiones.

Por último, se detallan algunas gráficas y tablas comparativas entre las herramientas con base en la implementación y pruebas de rendimiento, también se brindan algunas recomendaciones que pueden ser de utilidad y como anexo se muestra una gráfica que indica el crecimiento y demanda de las herramientas jBPM y Camunda para la implementación de procesos de negocio, obtenida de la página de Trending de Google[1].

[1] Trending Google. https://trends.google.com/trends/. Consulta: 25 de abril de2018.

OBJETIVOS

General

Realizar un análisis con base en la información que el trabajo recaba de las distintas fuentes e implementaciones, identificando las características y determinando las ventajas, desventajas y similitudes en el uso de la aplicación de las herramientas jBPM y Camunda, con el fin de orientar al desarrollador y personas de negocio que incursionan en esta tendencia para la administración de procesos de negocio.

Específicos

1. Especificar los conceptos importantes en la implementación de procesos de negocio, a través de información obtenida en páginas web y libros, con el objetivo de brindar la información básica e importante que rodea a las herramientas de BPM.

2. Enumerar los factores a tomar en cuenta en la implementación del proceso de negocio a través de información recolectada en páginas web y libros, con el objetivo de brindar los factores o características a tomar en cuenta antes de implementar procesos de negocio con una herramienta BPM.

3. Detallar las configuraciones iniciales que se deben realizar y tomar en cuenta para la implementación de jBPM y Camunda, a través de información obtenida en páginas web y libros, con el objetivo de dar a

conocer cómo debe estar configurado el ambiente, a nivel de hardware, software y arquitectura para la implementación de proyectos BPM.

4. Analizar y ejemplificar la implementación de flujos de proceso en cada una de las herramientas, poniendo a prueba cada una de las características que el trabajo describe, así como las reglas de negocio, DMN y BPMN, a través de implementaciones en cada una de las herramientas y documentando los resultados para posteriormente analizar los datos obtenidos.

5. Proporcionar un cuadro comparativo exponiendo los resultados obtenidos según las fuentes de información e implementaciones que se realizaron durante el desarrollo del trabajo de investigación, para una mejor comprensión y toma de decisión al momento de elegir entre jBPM y Camunda.

INTRODUCCIÓN

Con el tiempo se han vuelto más populares las herramientas de administración de procesos, como se puede ver en las tendencias de búsquedas en Google. Cada vez son más las personas y empresas que implementan estas herramientas, las cuales se basan en el modelo de notación de procesos de negocio (BPMN, por sus siglas en inglés) para la creación de aplicaciones empresariales.

Esta misma popularidad ha permitido que con el tiempo aparezcan más herramientas para su implementación, una de ellas y de las más antiguas es jBPM, la cual ha logrado mantenerse con el tiempo, volviéndose una de las herramientas más robustas para implementar procesos de negocio, con una gran comunidad. En cada una de sus versiones lanza más y mejores características, sin mencionar que tiene el respaldo de RedHat en la versión empresarial de esta herramienta.

Por otra parte, Camunda, aunque no es relativamente tan antigua como jBPM, ofrece muchas características de integración y adaptación a arquitecturas modernas de implementación de software, como la arquitectura orientada a microservicios.

1. CONCEPTOS IMPORTANTES EN LA IMPLEMENTACIÓN DE PROCESOS DE NEGOCIO

1.1. Lenguajes de programación en la implementación de procesos de negocios

Los lenguajes de programación son un factor muy importante en la implementación de procesos de negocio, estos determinan los prerrequisitos a nivel de hardware, software y conocimientos del programador antes de iniciar a implementar una de estas herramientas.

1.1.1. JAVA

Es uno de los lenguajes aceptados por jBPM para la escritura de *scripts* dentro de las acciones de entrada y salida de actividades, así como cada una de las actividades en las que puede utilizarse un lenguaje de programación para escribir rutinas más complicadas dentro de un flujo de BPM.

1.1.1.1. Definición

Lenguaje de programación orientado a objetos, muy parecido al lenguaje C++ pero con un modelo de objetos mucho más simple, el cual puede ser ejecutado en cualquier tipo de hardware debido a su independencia de la plataforma y un recolector de basura que permite liberar la memoria de los objetos muertos sin necesidad de programar los destructores para cada objeto.

1.1.1.2. Historia

Una de las primeras incursiones del lenguaje java en los entornos web fueron los *applets,* los cuales eran aplicaciones que se incrustaban en las páginas HTML que eran ejecutadas por el navegador con la ayuda de la máquina virtual de java (JVM, por sus siglas en inglés) y un *plugin*, luego vino el sistema servidor de java, con el cual aparecieron las especificaciones de los Servlets y también las páginas java del servidor (JSP, por sus siglas en inglés), que son una forma alternativa y simplificada de crear Servlets[2].

1.1.1.3. jBPM y Camunda

Este lenguaje es el lenguaje principal del motor y ejecución de flujos dentro de la plataforma de Camunda, dado que se vale de APIs codificados en lenguaje Java para ejecutar las acciones del flujo diseñado en esta plataforma.

También es uno de los lenguajes aceptados por jBPM para la escritura de *scripts* dentro de las acciones de entrada y salida de actividades, así como cada una de las actividades en las que puede utilizarse un lenguaje de programación para escribir rutinas más complicadas dentro de un flujo de BPM.

[2]Java (Lenguaje de programación). http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n). Consulta: 25 de abril de 2018.

1.1.2. **Javascript**

Lenguaje aceptado por jBPM para la escritura de *scripts* dentro de las acciones de entrada y salida de actividades, así como cada una de las actividades en las que puede utilizarse un lenguaje de programación para escribir rutinas más complicadas dentro de un flujo de BPM

1.1.2.1. Definición

Lenguaje de programación interpretado mayormente orientado al paradigma de la programación orientada a objetos, implementado en el lado del cliente, lo que permite una mejora en la interfaz del usuario y la interacción con el mismo, sin embargo, también existe una forma de JavaScript del lado del servidor. JavaScript utiliza una sintaxis similar a la del lenguaje de programación C, por lo que para los desarrolladores de este lenguaje no es muy diferente, actualmente todos los navegadores web tienen la capacidad de interpretar código JavaScript en las páginas web. El código JavaScript va incrustado en las páginas web dentro del documento HTML y básicamente se encarga de realizar acciones en el lado del cliente, como pedir información, mostrar mensajes, comprobar campos, etc. JavaScript es sensible a las mayúsculas y minúsculas, por lo que se debe tener cuidado al hacer referencia a una palabra reservada.

1.1.2.2. Características

Las variables en JavaScript pueden tener dos ámbitos globales o locales en donde una variable global puede ser accedida desde la etiqueta <script> y una variable local únicamente en la función que se ha definido. JavaScript reconoce seis tipos de variables, los cuales son números, booleanos, cadenas, objetos, nulos e indefinidos, como muchos lenguajes de *scripting* es de tipado dinámico o

débilmente tipado, el cual valida los tipos de datos a nivel de ejecución y no de compilación[3].

1.1.2.3. jBPM y Camunda

Este lenguaje es otro de los lenguajes aceptados por jBPM para la escritura de *scripts* dentro de las acciones de entrada y salida de actividades, así como cada una de las actividades en las que puede utilizarse un lenguaje de programación para escribir rutinas más complicadas dentro de un flujo de BPM.

1.1.3. MVEL

Lenguaje de gran importancia si se desea realizar reglas de negocio a nivel de archivo drl para la plataforma jBPM, es por eso por lo que se amplía un poco más este lenguaje y su sintaxis. Muchas veces se escriben expresiones de comparaciones dentro de los archivos drl.

1.1.3.1. Definición

Lenguaje para *scripting* basado en la sintaxis Java y de tipeado dinámico, es decir que la validación de los tipos de datos no es verificada en tiempo de compilación, sino en tiempo de ejecución, como lo hace la mayoría de *scripting.* Este lenguaje es utilizado para expresiones y condiciones dentro de la definición de reglas de negocio en archivos “drl”, cabe resaltar que las acciones dentro de las reglas de negocio son en lenguaje JAVA.

[3] JavaScript. http://es.wikipedia.org/wiki/Javascript. Consulta: 25 de abril de 2018.

1.1.3.2. Sintaxis básica

Este lenguaje de *scripting* maneja separadores de sentencia, al igual que el lenguaje Java, este separador es el punto y coma (;)[4] ejemplo:

```
objeto.propiedad = 1; objeto.propiedadSimple = 2; …;
```

Para el acceso a las propiedades no se utiliza get y/o set, únicamente se hace referencia al nombre de la propiedad, suponiendo una POJO de clase Persona, y su propiedad nombre, se puede acceder a una propiedad de una instancia de la clase Persona de la siguiente forma:

```
persona.nombre = “daniel”;
variableNombre = persona.nombre;
```

De esta misma forma, si el objeto al que se está accediendo contiene otros objetos y se desea acceder a las propiedades de estos objetos se representaría de la siguiente forma: suponiendo la clase Solicitud, la cual contiene una propiedad Cliente, la cual a su vez está compuesta por otras propiedades, se puede acceder al CUI del cliente desde una instancia de la clase Solicitud de la siguiente manera:

```
solicitud.cliente.cui = 163458987540101;
variableCui = solicitud.cliente.cui;
```

Para comparar propiedades, por ejemplo, de tipo cadena, no es necesario acceder a una propiedad de este tipo, como se hace con Java, sino que se debe

[4] Documentación MVEL. http://mvel.documentnode.com/. Consulta: 25 de abril de 2018.

utilizar los mismos operadores para comparar datos de tipo numérico, por ejemplo:

if(persona.nombre==”Daniel”)

1.1.3.3. jBPM y Camunda

Este lenguaje es importante conocerlo si se desea realizar reglas de negocio a nivel de archivo drl para la plataforma jBPM, es por eso por lo que se amplía un poco más este lenguaje y su sintaxis. Muchas veces se escriben expresiones de comparaciones dentro de los archivos drl, como estas:

When
persona:Persona(nombre==”Daniel)
Then
System.out.println(“Persona encontrada”);

Esto se debe a que los archivos drl reciben “facts” o hechos, por su traducción al español, los cuales son objetos Java que serán evaluados.

Estas son algunas de las características más importantes y más utilizadas del lenguaje utilizadas dentro de la plataforma jBPM, para mayor información acerca de este lenguaje revisar la documentación oficial[5].

[5] Lenguajes jBPM. http://mvel.documentnode.com/#language-guide-for-2.0. Consulta: 2018.

1.1.4. **Apache Groovy**

Lenguaje de *scripting* utilizado en la plataforma de Camunda, por esta razón que se hace un poco más de énfasis y se detallan algunas características de su sintaxis.

1.1.4.1. Definición

Es un lenguaje de *scripting* y como los últimos lenguajes de *scripting* que se han mencionado es un lenguaje dinámico que permite tipificar las variables o no, por lo que al no definir el tipo de variables se determina el tipo en tiempo de ejecución y no de compilación, además es un lenguaje liviano que permite realizar las operaciones básicas y, por supuesto, con una sintaxis muy parecida a Java, constituyendo una ventaja para los programadores que utilizan dicho lenguaje[6].

1.1.4.2. Sintaxis básica

A diferencia del lenguaje de *scripting* MVEL, este lenguaje no necesita un separador de sentencias, por lo que un grupo de sentencias en Groovy pueden verse algo así:

```
Persona persona
println “Name of the person: “ + persona.nombre
```

Si se desea realizar comentarios se debe colocar al inicio de la línea un signo de numeral # de la siguiente manera:

```
#Este es un comentario
println “hola mundo”
```

[6] Apache Groovy. http://groovy-lang.org/. Consulta: 25 de abril de 2018.

Una forma de definir variables dentro de este lenguaje de *scripting* es de la siguiente manera:

def nombre
def apellido

Una característica muy interesante de este lenguaje es que permite referenciar a las propiedades al igual que MVEL sin necesidad de *getters* y *setters*, pero además de esto permite utilizar comilla simple o doble comilla para acceder a las propiedades, con la finalidad de poder utilizar caracteres especiales en el nombre de las propiedades, por ejemplo, espacios vacíos en los nombres de las propiedades.

solicitud.persona.nombre
solicitud.persona.”primer apellido”
solicitud.persona.’segundo apellido’

Otra de las características de Groovy es que las cadenas se pueden interpolar y hacer referencias a nombre de variables utilizando la instrucción $ y entre llaves {} colocar el nombre de la variable, un ejemplo de esto sería de la siguiente forma:

```
String saludar(String nombre){
println “Hola ${nombre}”
}
```

Dentro de estas llaves también es posible evaluar expresiones como se muestra a continuación:

println “La suma de 1 mas 2 es: ${1 + 2}”
y se obtiene como resultado “La suma de 1 mas 2 es 3”

Para realizar condiciones es muy similar a MVEL, no se necesita utilizar métodos adicionales de comparación entre cadenas, sino que se compara tal y como si fuera un valor de tipo numérico, adicionalmente a esto se pueden comparar expresiones de la siguiente manera:

if("${peresona.nombre} Alvarez" == "Daniel Alvarez")
prirntln "Hola Daniel Alvarez"

1.1.4.3. jBPM y Camunda

Apache Groovy es un lenguaje de *scripting* utilizado en la plataforma de Camunda, es por esta razón que se hizo un poco más de énfasis y se detallaron algunas características de su sintaxis. Estas son algunas de las características más importantes y más utilizadas del lenguaje, utilizadas para definir acciones en los flujos de BPM utilizados por Camunda, para mayor información acerca de este lenguaje revisar la documentación oficial[7].

1.2. Reglas de negocio en la implementación de procesos de negocio

Las reglas de negocio determinan las acciones a tomar según algunos criterios, para esto se implementan nuevas herramientas y/o estándares para la implementación de estas reglas.

1.2.1. BRMS

Por sus siglas en inglés: Sistema de Administración de Reglas de Negocio, es el nombre que se le da a todo sistema que sea capaz de procesar de una manera controlable, configurable, robusta y de fácil mantenimiento reglas

[7] Lenguajes Camunda. http://groovy-lang.org/documentation.html. Consulta: 2018.

definidas por el negocio, las cuales están completamente separadas del núcleo de sus sistemas de procesamiento, esto permite que sean mantenidas por separado y cambiar en cualquier momento, sin que esto represente un mayor esfuerzo a nivel económico, tiempo y recurso humano[8].

1.2.2. **Drools**

Drools es un BRMS, el cual permite evaluar reglas de negocio a través de archivos drl y recientemente implementó la capacidad de procesar archivos DMN, que es el estándar definido para la representación de reglas de negocio.

Drools permite ejecutar reglas independientemente del sistema, dado que puede comunicarse a través de un KIE Server, el cual expone servicios REST para la ejecución de las reglas, además de este tipo de arquitectura define un API para la ejecución de reglas dentro de proyectos Java.

Drools es una plataforma que va de la mano con la plataforma jBPM de flujo de procesos, pero no es exclusiva dado que Camunda permite conectarse a este tipo de plataforma de ejecución de reglas[9].

Algunas de las características importantes de Drools son:

- Permite la realización de pruebas unitarias para probar las reglas definidas.
- Proporciona *plugins* para eclipse para el diseño, ejecución, testeo y *debug* de reglas.

[8] Sobre Groovy. http://groovy-lang.org/documentation.html. Consulta: 07 de mayo de 2018.
[9] Drools. https://www.drools.org/. Consulta: 07 de mayo de 2018.

- Proporciona una plataforma web para el diseño, ejecución, testeo y *debug* de reglas.
- Se integra a un servidor de ejecución de reglas que proporciona un interfaz REST para la ejecución de estas.
- Permite crear archivo de reglas drl a través de tablas guiadas, las cuales pueden ser editadas por personas de negocio para ajustar los valores de validación de las reglas.

Ejecución de reglas a través de archivos Excel con un formato definido para la definición de reglas de negocio.

1.3. Estándares involucrados en la implementación de procesos de negocio

Es importante tener en cuenta los estándares alrededor de las herramientas de implementación de procesos de negocio, ya que, junto a los lenguajes de programación son prerrequisitos antes de iniciar con cualquier herramienta. Estos estándares tratan de mantener homogéneas las distintas herramientas en el mercado para la implementación de flujos de BPM.

1.3.1. BPMN

Es un estándar para el modelado de procesos de negocio, en el cual se describe la notación gráfica para representar procesos de negocio, las tareas que interactúan y colaboran entre sí y la forma en que cada una de las actividades de un proceso se comunican[10].

Este estándar es de suma importancia para el conocimiento de todas las personas involucradas en proceso de migración de un proceso empresarial a esta

[10] BPMN. http://www.bpmn.org/. Consulta: 29 de mayo de 2018.

metodología, desde las personas de negocio hasta el programador, quien implementa la solución tecnológica.

Aunque la nomenclatura estándar para la representación de negocios es extensa, al momento de la ejecución o implementación de herramientas solamente una pequeña parte de la nomenclatura es utilizada. Esta porción de objetos utilizados de la nomenclatura varía según el fabricante de la herramienta.

1.3.1.1. Nomenclatura

La nomenclatura del BPMN detalla cada una de las entidades que interactúan en un flujo de BPM, su comportamiento e interacción con otras entidades dentro del flujo.

1.3.1.1.1. Tareas

Las tareas son las entidades atómicas o no divisibles en más actividades, las cuales representan una acción e interacción con otra entidad, esta actividad puede llevarse a cabo en repetidas ocasiones durante los procesos de algún negocio, un ejemplo de esto es: la validación de documentos como tareas humanas y la consulta a servicios externos como servicios REST.

1.3.1.1.2. Procesos

Un proceso es un conjunto de actividades que interactúan entre sí en una secuencia ordenada. Al igual que las actividades, tienen un inicio y un fin, pero para llegar del inicio al fin no es una sola su ejecución, sino que hay muchos caminos para poder llegar al final del proceso.

Todas las actividades tienen un inicio y fin definidos, es decir que solo se ejecuta y allí termina su ciclo de vida. Una actividad puede tener los siguientes estados:

- Lista
- Reclamada
- Iniciada
- En progreso
- Finalizada

1.3.1.1.3. Actividades

Las actividades son representadas por un rectángulo con las esquinas curvadas, las cuales tienen ciertas propiedades, como el nombre, tipo, acciones al entrar y acciones al salir de la actividad, estas actividades regularmente son escritas en verbo-sustantivo, por ejemplo: “Realizar análisis”, “Evaluar solicitud”, “Verificar papelería”. Los tipos de actividades que se pueden encontrar dentro del estándar son las siguientes:

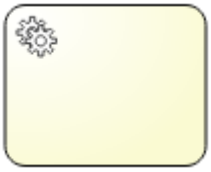
- Tareas humanas: estas tareas son representadas por cuadros rectangulares como toda actividad con un icono de una persona. Estas tareas representan la actividad a realizar de una persona, por ejemplo, si se desea autorizar la venta de un auto y esta autorización debe realizarla el director, el cual debe revisar el caso e indicar con una acción en la pantalla, como presionar un botón de “Aceptar”, esta es una tarea naturalmente humana dentro de un flujo de proceso empresarial.

Figura 1. **Tarea humana**

Fuente: elaboración propia, empleando jBPM Workbench.

- Tareas de servicio: estas tareas representan un proceso automatizado en el cual no existe intervención humana, un ejemplo de esto es el llamado a un sistema externo de validación de estado de habitaciones para un proceso de reserva de hotel.

Figura 2. **Tarea de servicio**



Fuente: elaboración propia, empleando jBPM Workbench.

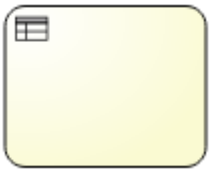
- Tareas Script: estas tareas son capaces de ejecutar código de algún lenguaje de *scripting* para realizar algunas tareas básicas, las cuales se desean realizar dentro de un proceso. El lenguaje de *scripting* dependerá mucho del motor utilizado para la ejecución de flujos de BPM, por ejemplo: Java, Javascript, Groovy, entre otros.

Figura 3. **Tarea de Script**

Fuente: elaboración propia, empleando jBPM Workbench.

- Tareas de reglas de negocio: estas tareas aparecieron hasta la versión 2.0 del estándar de BPMN y son tareas muy importantes dentro de un flujo de bpm, debido a que son capaces de ejecutar reglas de negocio, evaluando objetos que pueden ser definidos dentro de la misma plataforma, o exportados como dependencias *maven.*

Figura 4. **Tarea de validación de reglas de negocio**

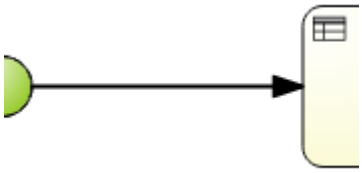


Fuente: elaboración propia, empleando jBPM Workbench.

1.3.1.1.4. Flujos de secuencia

Son conectores en forma de flechas que se utilizan para indicar el camino a seguir, normalmente luego de realizar una actividad o los caminos que es posible seguir luego de la evaluación de una condición con información proveída por una tarea humana, un servicio o alguna tarea de *script*, la cual ejecuta código, como se verá más adelante, y puede generar algún resultado que pueda tomarse en una decisión de flujo.

Figura 5. **Conector de flujo de secuencia**



Fuente: elaboración propia, empleando jBPM Workbench.

1.3.1.1.5. Eventos

Son representados gráficamente como círculos, los cuales describen una acción que puede desviar el flujo de un proceso de su flujo normal, un ejemplo de esto es una excepción en algún servicio que se esté ejecutando, un tiempo de espera para una tarea que no se quiere que esté en espera de ejecución más de un tiempo determinado.

- Eventos de inicio: estos eventos ayudan a denotar el inicio de un proceso y son representados por los siguientes iconos:

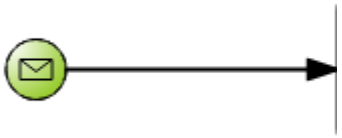
    - Básico: el evento de inicio básico es lanzado desde la ejecución con el API, es decir que es realizado manualmente desde una señal externa al proceso.

Figura 6. **Evento de inicio básico**

Fuente: elaboración propia, empleando jBPM Workbench.

- Por mensaje: los eventos de inicio por mensaje significan que son lanzados desde una solicitud externa por un evento definido dentro de un proceso externo al que se quiere iniciar.

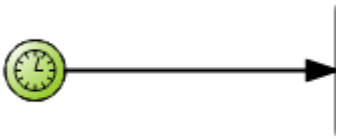
Figura 7. **Evento de inicio por mensaje**



Fuente: elaboración propia, empleando jBPM Workbench.

- Por temporizador: este evento de inicio especifica que cada cierto tiempo se lanzará un evento, este tiempo es configurable e inicia un proceso que está calendarizado periódicamente y no se necesita de intervención humana para iniciar el flujo.

Figura 8. **Evento de inicio por temporizador**



Fuente: elaboración propia, empleando jBPM Workbench.

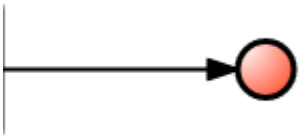
- Múltiple: es un evento de inicio que puede ser lanzado por diferentes fuentes o *swimlanes* diferentes, actúa como un *gateway* exclusivo, dado que tiene diferentes entradas y al final inicia el flujo.

- Eventos de fin: estos eventos ayudan a denotar el fin de un proceso o subproceso, muchas veces, aunque el evento de fin tiene el mismo significado, es recomendable utilizar diferentes eventos de fin para

diferentes finales de un subproceso, por ejemplo, se finaliza un proceso porque transcurrió con normalidad el mismo, pero si finaliza por alguna excepción se deberá representar por otro evento de fin que denote la condición por la que fue finalizado el proceso.

- Básico: un evento básico de fin, son aquellos a los que se llega desde un flujo normal o por una excepción en el flujo, este evento de fin es el mismo cada vez que se elija, por lo que puede utilizarse uno para todo el flujo o uno en cada situación que el flujo necesite finalizar.

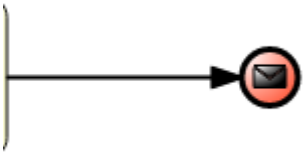
Figura 9. **Evento de fin básico**



Fuente: elaboración propia, empleando jBPM Workbench.

- Por mensaje: este evento es lanzado al momento de alcanzar el fin de un flujo y se desea notificar a otro flujo, por ejemplo, a otro *swimlane*, el cual debería realizar alguna otra acción con la finalización del proceso que ha lanzado el mensaje de finalización.

Figura 10. **Evento de fin por mensaje**



Fuente: elaboración propia, empleando jBPM Workbench.

- Terminador: este evento de fin, a diferencia del básico, finaliza el proceso actual que finaliza los subprocesos.

- Eventos de envío: son eventos que se accionan a través de un mensaje.

Figura 11. **Evento de envío**

Fuente: elaboración propia, empleando jBPM Workbench.

- Eventos de recepción: son eventos que se reciben para realizar una acción a través de un mensaje.

Figura 12. **Evento de recepción**

Fuente: elaboración propia, empleando jBPM Workbench.

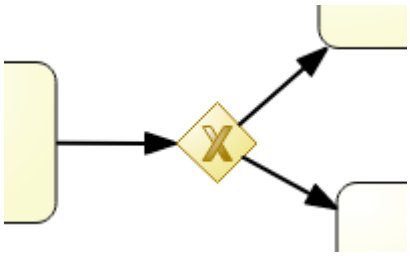
1.3.1.1.6. Gateways

Estas entidades son para determinar caminos a seguir dada una condición, son figuras en forma de diamante y son comúnmente llamadas “compuertas” que representan bifurcaciones en el diagrama, es decir, son puntos en los que se toman decisiones y se decide con base en estas últimas continuar con el flujo normal o el de una excepción, las excepciones no son más que caminos alternos al flujo normal de un proceso exitoso de negocio. Por ejemplo, en un proceso de solicitud de préstamo, si una persona realiza una solicitud de préstamo y cuenta con los requerimientos necesarios, el préstamo es otorgado y desembolsado al

solicitante, en otro caso, si no cumple con todos los requerimientos, o si casi cumple con todos los requerimientos, existen flujos alternos que son iniciados a partir de los *gateways* definidos en un diagrama de flujo de un proceso empresarial.

- Compuerta exclusiva: esta compuerta es utilizada cuando se sigue un solo camino de los múltiples caminos que proporciona esta compuerta, el cual se determina por la evaluación de una condición.

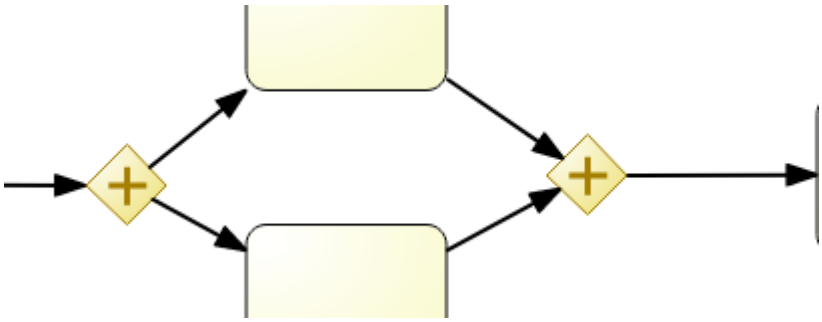
Figura 13. **Compuerta exclusiva**



Fuente: elaboración propia, empleando jBPM Workbench.

- Compuerta paralela: es utilizada para realizar diferentes actividades en paralelo y/o unir actividades o flujos de actividades para que un flujo no pueda continuar hasta que estas dos actividades en flujos distintos se hayan completado, esta compuerta es llamada compuerta de división AND o unión AND. Esta opción en el flujo es representada por un diamante con una cruz en el centro de la siguiente manera:

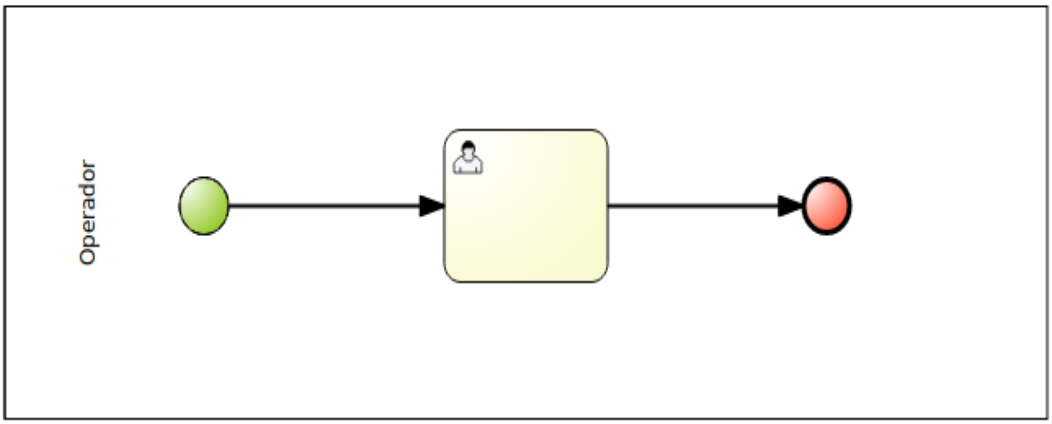
Figura 14. **Compuertas paralelas**



Fuente: elaboración propia, empleando jBPM Workbench.

- *Swimlanes:* los *swimlanes* son cajas en las que es posible agrupar actividades, que serán ejecutadas por grupos específicos de una empresa: operadores, supervisores, auditores, entre otros. Incluso permite representar un sistema que ejecutará ciertos servicios los cuales proveerán información al proceso, en la mayoría de las ocasiones son utilizadas únicamente para actividades humanas, sin embargo, es posible agrupar actividades autónomas.

Figura 15. **Swimlane**



Fuente: elaboración propia, empleando jBPM Workbench.

1.3.1.1.7. Subprocesos

Los subprocesos también son considerados actividades, debido a que dentro de un flujo también representan las actividades de una tarea, con la diferencia que esta actividad está compuesta por muchas más actividades, las cuales también llevan un orden de ejecución y sus posibles excepciones.

Los subprocesos en un diagrama BPMN 2.0 son representados de dos formas: en diagramas distintos, en donde el proceso padre llama al proceso hijo a través de un enlace; o representando el subproceso en el mismo diagrama a través de una caja que agrupa las actividades del subproceso, en muy pocas ocasiones se utiliza esta última opción dado que proporciona mayor ruido en el flujo de BPM.

1.3.1.1.8. Procesos AD-HOC

Son procesos en los cuales no se tiene una secuencia definida para la ejecución del proceso, es decir que cada actividad determina la siguiente tarea a ejecutar. Estos procesos comúnmente no son representados con el estándar de BPMN, aunque si es posible representarlos, gráficamente no tienen mucho sentido dado que no llevan una secuencia definida.

Estos tipos de procesos difícilmente son realizados o representados en un diagrama de BPM, los procesos *ad hoc* sirven para ejecutar tareas en paralelo, mayormente son utilizados como subprocesos dado que permiten ejecutar un conjunto de actividades sin flujo de secuencia, el cual al completarse puede continuar con el flujo del proceso padre que sí tiene una secuencia de actividades por realizar.

1.3.1.1.9. Participantes externos

Los participantes externos son las personas que interactúan externamente a un flujo de proceso, por ejemplo un cliente quien es el que realiza la reservación de una habitación, a nivel de flujo el cliente es el que inicia la solicitud y realiza proceso de reservación, sin embargo, se desconoce sus acciones sobre la página, el flujo de procesos desconoce cómo elige las opciones, al flujo de reservación le interesa la información final para realizar una reservación, estos participantes externos son representados por *swimlanes* vacíos sin flujo y únicamente hay flechas de dirección para indicar mensajes entre los *swimlanes.*

Muchas veces estos participantes externos no son representados en un diagrama de ejecución, sin embargo, muchas veces es importante agregarlos en un diagrama conceptual, y debería ir en la parte superior del diagrama para que permita una mejor visibilidad de cómo interactúan los participantes externos como los clientes en el sistema.

1.3.1.1.10. Flujos de mensajes

Los mensajes pueden ser de envío o recepción, estos mensajes son ejecutados por el flujo cuando son de envío y cuando son de recepción son capaces de ejecutar una acción y/o iniciar algún flujo de excepción. Las flechas del flujo de mensajes son representadas como líneas intermitentes y el evento en sí de envío o recepción de mensajes es representado por un círculo con un sobre dentro de él.

1.3.2. **DMN**

Es el estándar de notación y modelado para decisiones, las cuales son repetibles en los procesos de un negocio. Esta notación y modelado puede ser utilizada individualmente, pero básicamente es un complemento para los estándares de BPMN y CMMN.

El estándar nació como una forma de delegar la implementación de decisiones empresariales a la gente de negocio, a través de este estándar que permitiera de una forma sencilla y entendible al negocio definir las reglas, dado que aparte del estándar DMN se utilizan los BRMS, los cuales permiten definir reglas de negocio, pero las reglas se necesitan pasar del negocio y traducirlas a un lenguaje máquina para que puedan ser interpretadas, lo cual conlleva una serie de revisiones cada vez que se realice un cambio o una nueva regla. Con el estándar DMN se pretende que la gente del negocio pueda definir las reglas, las cuales son pasadas directamente como fueron definidas al sistema que las valida.

El DMN tiene tres elementos importantes: diagramas de decisión de requerimientos, llamados también por sus siglas en inglés DRD (Decisión Requirments Diagram), estos diagramas especifican cómo interactúan los elementos que conforman el diagrama de decisiones. Otro elemento es el contexto del negocio, el cual busca identificar las métricas que serán evaluadas para tomar las decisiones, las cuales son representadas como tablas de decisión en donde se define cada una de las reglas que serán validadas, y, por último y no menos importante, el lenguaje que será utilizado para evaluar las reglas de un negocio[11].

[11] DMN Wikipedia. https://en.wikipedia.org/wiki/Decision_Model_and_Notation. Consulta: 03 de mayo de 2018.

1.3.3. **CMMN**

Es el estándar de notación y modelado para administración de casos, al igual que el DMN este estándar complementa el estándar de BPMN, describe cómo representar gráficamente métodos de trabajo en casos de un negocio, casos los cuales tienen actividades asociadas[12]. A diferencia de un flujo de BPM, en que se tiene un flujo guiado de trabajo, los métodos de trabajo muchas veces no lo tienen, es por eso que cuando se quieren simular estos métodos de trabajo en BPM siempre se terminan representando proceso *ad-hoc*, ya que estos no tienen una guía programada de trabajo, sino que pueden ejecutarse todas al mismo tiempo o alguna de las actividades puede llamar a cualquiera de las otras. Aunque tiene muchas entidades en su diagrama muy parecidos al BPM, como lo son las tareas humanas y eventos temporizados, actualmente son utilizados para usos distintos, aunque, como se mencionó anteriormente, el BPM podría simularse con procesos *ad-hoc*, actualmente estos dos estándares se complementan y pueden ser utilizados en conjunto.

1.3.4. **Herramientas de modelado en la implementación de BPM**

Las herramientas de modelado en cada una de las plataformas son diversas, algunas implementan el estándar, otras además de implementarlo lo extienden y existen algunas que crean su propia estructura de proyectos para la implementación de flujos de negocio.

1.3.4.1. jBPM Workbench

Es la herramienta proporcionada por la comunidad jBPM para implementar y, hasta la versión 6.5, ejecutar los flujos de BPM realizados en el

[12] CMMN Wikipedia. https://en.wikipedia.org/wiki/CMMN. Consulta: 03 de mayo de 2018.

modelador de flujos BPM. jBPM Workbench es una plataforma web que permite realizar:

- Administrar usuarios y roles, los usuarios se dividen en dos tipos: aplicación y *web server*, los usuarios de aplicación son los que contienen roles para la utilización de la herramienta web, y los usuarios de *web server* son los que administran las fuentes de datos, configuración general y administración de otras herramientas que proporciona la plataforma web.
- Administrar repositorios de fuentes permite visualizar y administrar los repositorios de GIT en donde están versionadas las fuentes de los productos que se desarrollen.
- Administrar proyectos, todas las fuentes que se generen se guardan en un proyecto Java-Maven, dentro de estos proyectos es en donde se almacenan los flujos de BPM, y es posible definir:
    - Archivos BPMN para el flujo de procesos.
    - Objetos de datos Java para la creación de clases que pueden instalarse y utilizarse para la validación de reglas y/o dentro del flujo de BPM.
    - Archivos drl para la creación y validación de reglas de negocio, estos archivos drl pueden ser creados en texto plano o con tablas guiadas que permiten crear archivos drl de una forma gráfica y amigable para personas del negocio.
    - Escenarios de pruebas que permiten crear pruebas unitarias para las reglas de negocio definidas.

- Administrar los despliegues de proyectos para su ejecución.
- Administrar las instancias de procesos.
- Administrar las tareas humanas de las instancias de procesos.
- Visualizar *dashboard* de los procesos, proyectos, despliegues y tareas.

La plataforma web jBPM Workbench no es la única opción para diagramar los flujos de BPM para el motor jBPM. Existe un *plugin* para eclipse, el cual permite crear el mismo tipo de proyecto Java-Maven para que puedan ser desplegados dentro de la plataforma web jBPM Workbench y poder utilizarlos.

1.3.4.2. Camunda Modeler

Camunda Modeler[13] es la herramienta que proporciona Camunda para la realización de diagramas BPM, y además de diagramas BPMN permite crear diagramas y tablas DMN y diagramas CMMN. La herramienta permite crear flujos BPM que pueden ser desplegados y ejecutados por otros motores de BPM, dado que trabaja directamente con el XML estándar, proporciona plantillas precargadas de tareas customizadas, por ejemplo: servicio de envío de correo, consumo de servicios y otros.

Permite adicionar *plugins* para extender su funcionalidad, estos *plugins* pueden descargarse del GIT[14] de Camunda e instalarse para su utilización, alguno de los que pueden ser descargados son los siguientes:

- Simulación de *token*, con este *plugin* se puede visualizar un *token* por donde pasa la simulación del flujo de proceso.

- Asistente Darlene, con este *plugin* se puede tener un asistente para el desarrollo de modelos.

[13] Modeler Camunda. https://camunda.com/products/modeler/. Consulta: 09 de mayo de 2018.

[14] Plugins de Camunda. https://github.com/camunda/camunda-modeler-plugins#camunda-modeler-plugins-electric_plug. Consulta: 07 de mayo de 2018.

- Ajustado de color, con este plugin es posible agregarle un color personalizados a las entidades dentro de un diagrama.

- Renombrado, con este *plugin* es posible renombrar de forma automática los IDs de las entidades dentro del archivo fuente XML, debido a que el modelador le genera ID que quizás no sean de acorde al nombre de la entidad.

Adicionalmente a la herramienta de escritorio, Camunda proporciona una herramienta *online* en la que es posible generar los diagramas de BPMN, DMN y CMMN, además de proporcionar una simulación para cada uno de los diagramas en tiempo real para demostrar su funcionamiento.

2. FACTORES A CONSIDERAR EN LA IMPLEMENTACIÓN DE PROCESOS DE NEGOCIO

2.1. DMN y BRMS

El estándar DMN para la implementación de reglas de negocio y las tablas de decisión son dos elementos importantes en la implementación de flujos de negocios, permiten definir e implementar reglas de negocio, estas reglas permiten tomar decisiones dentro de un flujo de BPM para tomar un camino alterno a excepciones en el flujo normal. La importancia de estas herramientas de implementación de negocios radica en que muchas veces es más fácil trasladar las reglas de negocio que define la gente del negocio a un lenguaje dedicado únicamente para este fin. Si no se contara con estas herramientas, la definición de reglas de negocio para el ejemplo de una solicitud de préstamo sería la siguiente:

Dadas las reglas:

- Si el solicitante es menor de 18 años o mayor a 70 años será denegada la solicitud de préstamo.
- Si los ingresos del solicitante son menores al salario mínimo, será denegada la solicitud de préstamo.
- Si el solicitante tiene menos de un año de laborar en su lugar de trabajo, será denegada la solicitud de préstamo.

Una porción de código en lenguaje Java sería el siguiente:

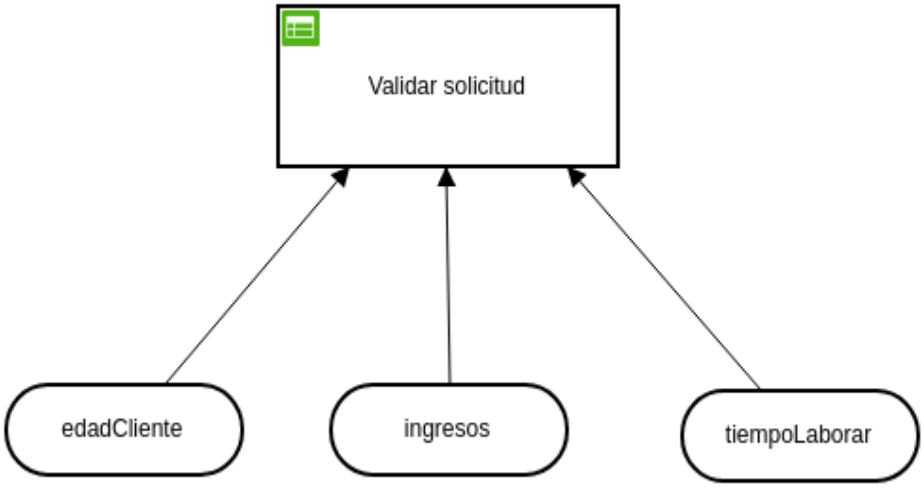
```
if(solicitud.edadCliente < 18 || solicitud.edadCliente > 69){
  return false;
}
if(solicitud.ingresos < 2500.00){
  return false;
}
if(solicitud.tiempoLaborar < 365){
  return false;
}
```

Se debe escribir una condición *if* por cada regla, además de ser tedioso, el mantenimiento del código es muy difícil, dado que, si se desea modificar o agregar una nueva regla, estas deben ser modificadas directamente en el código, lo que implicaría tener un tiempo necesario para realizar todo el ciclo de desarrollo y pruebas de una aplicación, además de que es necesario crear una ventana de tiempo para aplicar el cambio en un ambiente de producción.

Por otro lado, si se implementan estas mismas reglas con el estándar DMN se tiene que realizar de la siguiente forma:

- Definir el diagrama de decisión de requerimientos, con las variables a evaluar (Ver figura 16).
- Definir cada una de las reglas en la tabla de decisión (Ver figura 17).

Figura 16. **Diagrama DRD para una solicitud de crédito**



El rectángulo en la parte superior representa las reglas que se aplican sobre las propiedades u objetos de datos representados en forma de óvalos.

Fuente: elaboración propia, empleando jBPM Workbench.

Figura 17. **Tabla de decisión para la solicitud de préstamo**

Input +			Output +
Edad de cliente `edadCliente`	Ingresos `ingresos`	Tiempo de laborar `tiempoLaborar`	aprobado
integer	double	integer	string
< 18	< 2500.00	< 365	-
> 70	-	-	-

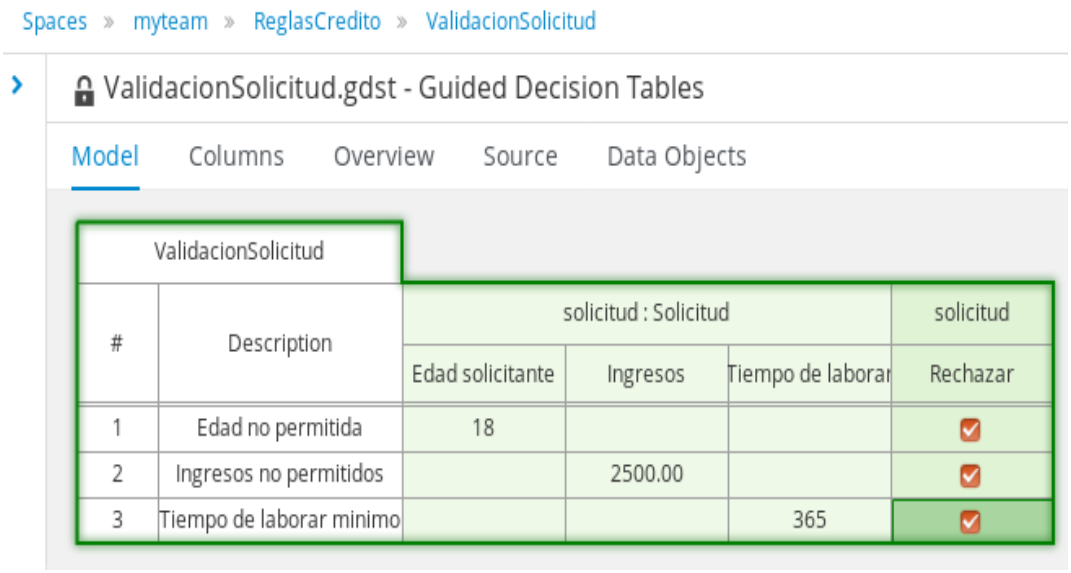
Dividida en dos secciones: entradas (*input*) representando las propiedades a validar y salidas (*output*) con los resultados de la evaluación de las reglas.

Fuente. elaboración propia.

Al implementarlo con un BRMS, específicamente el Drools Workbench de la misma organización que jBPM, la forma de implementar la regla dada es definir las reglas en un archivo drl o con la ayuda de tablas guiadas generar este mismo archivo a partir de una tabla gráfica en la que se definen las reglas por condiciones y acciones y se genera un archivo drl, el cual puede ser utilizado por el motor de reglas para su validación.

Las reglas se definen con base en las propiedades que son definidas en una clase según las necesidades, en el caso de la regla presentada, es necesario crear una clase con las propiedades de “edad”, “ingresos” y “tiempoLaborar”, para poder crear las reglas que validen estas propiedades.

Figura 18. **Tabla guiada de Drools BRMS**



Dividida en tres secciones, la primera representa una descripción de la regla, la segunda de las propiedades a ser evaluadas y la tercera el resultado de la evaluación de las reglas.

Fuente: elaboración propia, empleando jBPM Workbench.

Figura 19. **Código MVEL generado de la tabla guiada**

Fuente: elaboración propia, empleando jBPM Workbench.

2.2. Curva de aprendizaje en la implementación del estándar BPMN

Del estándar BPMN existen las versiones 1.0, 1.1, 1.2 y la versión actual 2.0, de la versión 1.2 a la versión 2.0 no ha sido mayor el cambio, la razón del cambio radical de la versión es a nivel interno, con la serialización de XML, para tratar de generalizar un estándar de comunicación entre herramientas distintas,

pero a nivel de entidades para la representación de modelos de procesos, únicamente fueron agregadas una que otra entidad o evento.

El BPMN permite una integridad de elementos en el flujo, dado que especifica que cada una de las entidades debe estar definida dentro del diagrama de flujo desde que inicia la primera actividad hasta cada una de las posibles actividades finales, sin dejar cabos sueltos, dado que es posible representar entidades externas al flujo en la parte de conceptos importantes en el capítulo 1, por tal razón no se debe obviar ningún elemento que esté involucrado en un flujo de proceso de negocio.

El BPMN permite desarrollar un diagrama claro, incluso para personas del negocio, el BPMN no define una metodología para implementar los flujos de negocio, pero la nomenclatura permite definir flujos con claridad definiendo cada uno de los procesos y subprocesos, en donde es posible visualizar los subprocesos a detalle, cada una de las excepciones que permita el flujo deben ser diagramadas sin excepción alguna, primero para que cumpla y satisfaga las necesidades del negocio, y además para que sea entendible para las personas que no son técnicas en el uso de tecnologías para representar flujos de negocio.

Algunos pasos importantes para definir un flujo de procesos son:

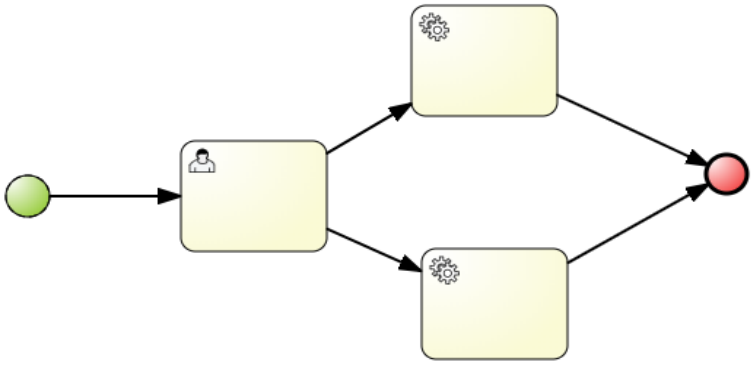
- Definir el alcance y ámbito del proceso, es importante determinar el alcance del proceso para determinar en dónde comienza y en dónde termina, y así determinar cuáles no son las responsabilidades del flujo a definir.
- Definir desde qué punto va a ser iniciado el flujo, esto puede ser desde una llamada telefónica, un evento de una página web, un proceso *batch,* entre otros.

- Definir cada una de las actividades que interactúan en el proceso.
- Definir el proceso macro, es importante definir todo el flujo a un nivel macro, en donde los subprocesos únicamente son representados por una actividad de subproceso que encapsula el detalle de ese subproceso.
- Definir el detalle de los subprocesos, luego de definir el alcance, las actividades y los subprocesos que interactúan en el proceso que se está definiendo, como último punto es útil definir cada uno de los subprocesos que se diagramaron.

Consideraciones importantes:

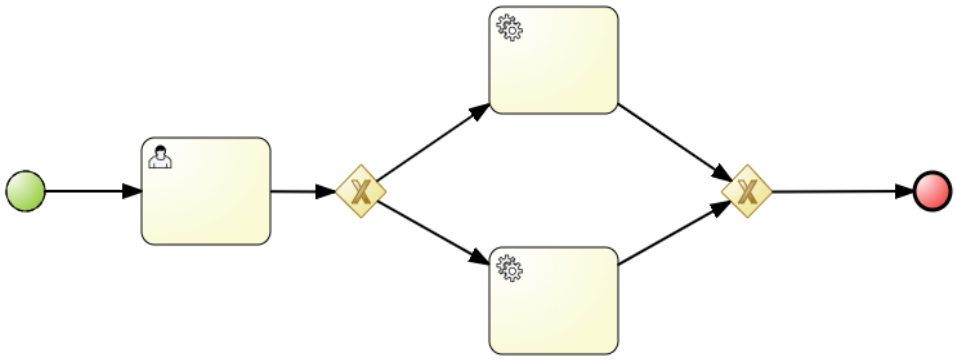
- Conectores de flujo: cuando es preciso realizar la conexión de una actividad con otras actividades es necesario la utilización de un *gateway* exclusivo o paralelo según lo que se necesite realizar, los conectores nunca deben salir de una actividad hacia las otras actividades directamente, como se ve a continuación:

Figura 20. **Forma incorrecta de conexiones entre actividades**



Fuente: elaboración propia, empleando jBPM Workbench.

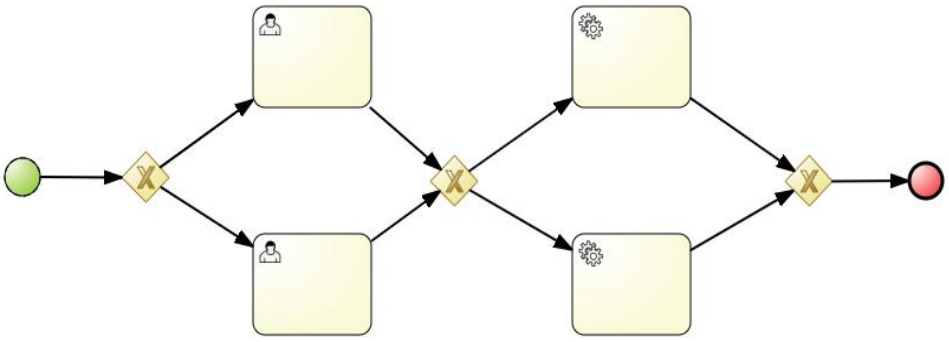
Figura 21. **Forma correcta de conexiones hacia o desde múltiples actividades**



Fuente: elaboración propia, empleando jBPM Workbench.

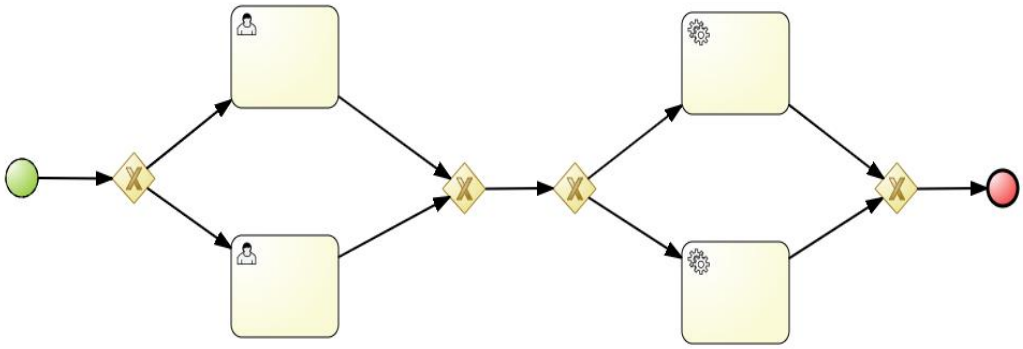
- Utilización de gateways: cuando se utilizan los gateways, aunque sirven para distribuir los conectores hacia otras tareas, únicamente se es capaz de realizar una tarea, distribuir o recibir conectores para un camino y no los dos al mismo tiempo.

Figura 22. **Forma incorrecta de unir múltiples actividades**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 23. **Forma correcta de unir múltiples actividades**



Fuente: elaboración propia, empleando jBPM Workbench.

Otras consideraciones son:

- Los procesos deben tener por lo menos una actividad.
- Los eventos de inicio no pueden tener un conector entrante, ya sea un evento básico, de mensaje o señal.
- Los eventos de fin no pueden tener un conector saliente, ya sea un evento básico, de mensaje o señal.
- Toda entidad dentro del diagrama que tenga conectores debe estar conectada con alguna otra entidad, no pueden quedar conectores sin unir.
- Una secuencia de actividades dibujada en el diagrama no puede quedar fuera del subproceso o *swimlane* en el que inicia su actividad.
- Los eventos intermedios deben tener conexión hacia alguna otra actividad, no pueden quedarse sin conectar.

2.3. Curva de aprendizaje en las plataformas

Inicialmente muchas herramientas implementaron su propia nomenclatura para representar procesos de negocio, esto fue cambiando a medida que el

estándar se fue volviendo mucho más robusto y eficiente que las propias implementaciones de una herramienta determinada. Cada empresa desarrolladora de las herramientas fue adoptando las entidades del estándar que necesitaron para mejorar sus implementaciones, incluso en algunas ocasiones se basaron en el estándar para realizar algunas otras modificaciones, es por esto que muchas veces un modelo BPMN 2.0 no necesariamente será funcional en otra herramienta que implemente la misma versión del estándar.

Es importante hacer la distinción entre herramientas de diagramado y herramientas de ejecución, la primera puede implementar u ofrecer todo el estándar definido en el BPMN 2.0 y hay muchas de ellas, incluso en línea, pero no es posible ejecutar los diagramas definidos en ellas, es por esta razón que existen las herramientas de ejecución, las cuales contienen motores de ejecución de flujos BPM, estos soportan algún lenguaje de programación, realizan actividades de comunicación con otros sistemas, un ejemplo de esto es el envío de correos y envío de solicitudes a *endpoints* de sistemas externos al BPM y otros.

La curva de aprendizaje en cada plataforma dependerá del nivel de complejidad con que se desee implementar los flujos de BPM, cada una de las plataformas trae muchas de sus configuraciones por defecto con las que fácilmente se pueden implementar aplicaciones, servicios, páginas, si así se desea, que implementen flujos de BPM, sin embargo, muchas de las configuraciones por defecto en la vida real o en entornos empresariales no necesariamente son las más útiles.

Muchas de las configuraciones por defecto incluyen:

- Administración y configuración de usuarios

- Reglas de negocio
- Servidores de despliegue
- Arquitectura
- Repositorio de fuentes
- Bases de datos

2.3.1. **jBPM**

La plataforma jBPM tiene una interfaz web y proporciona el modelador dentro de la misma, por lo que se necesita alguna otra herramienta para diagramar el flujo BPM, sin embargo, es posible realizar diagramas compatibles con la plataforma a través del *plugin* para el entorno de desarrollo Eclipse. A continuación, se definen algunas de las características especiales que es posible hacer dentro de la plataforma y que quizás no sean necesarias para empezar a implementar flujos de BPM, pues podrían complicar la curva de aprendizaje de la herramienta.

2.3.1.1. Repositorio de fuentes

En el capítulo anterior se mencionó que la plataforma tiene la capacidad de trabajar con el repositorio de fuentes GIT, por defecto la plataforma internamente cuenta con su repositorio de fuentes para los proyectos que se deseen desarrollar localmente, pero, si se desea trabajar de forma colaborativa como actualmente se hace en la mayoría de empresas y organizaciones, jBPM tiene la opción de clonar repositorios, es muy importante mencionar que la forma de trabajar con GIT es haciendo bifurcaciones (Fork[15] en inglés), por lo que no es posible

[15] Sobre repositorios. https://help.github.com/articles/fork-a-repo/. Consulta: 07 de mayo de 2018.

visualizar las fuentes de los proyectos, dado que están almacenados en binario dentro del repositorio copia que se genera del repositorio original. Para obtener las fuentes del proyecto o del repositorio la plataforma proporciona una opción de descargar una copia del proyecto seleccionado o el repositorio completo.

2.3.1.2. BRMS

jBPM a partir de la versión 7.x añade la opción de externalizar las reglas de negocio hacia un proyecto creado en un BRMS, específicamente el Drools Workbench, en versiones anteriores únicamente se podían consumir reglas de negocio desde un flujo de BPM a través de tareas de reglas de negocio (Rule Business Task), esto se logra a través de una nueva tarea de reglas de negocio, la cual permite indicar dentro de sus parámetros de entrada el contendor asociado al proyecto que contiene las reglas de negocio. Iniciar con la externalización de reglas es muy complicado, se debe tener ciertos conocimientos de los KIE Server, utilización y configuración con la plataforma jBPM, por lo que al inicio no es recomendable utilizar la externalización de reglas, pues con la utilización de archivos drl y/o tablas guiadas es suficiente para enlazar reglas de negocio a los flujos de BPM, pero posteriormente es muy recomendado utilizarlos debido a que permiten configurar las reglas sin necesidad de modificar, compilar y desplegar de nuevo el proyecto que contiene el flujo de procesos empresariales.

2.3.1.3. KIE Server

Es un servidor de despliegue de aplicaciones jBPM, el cual se registra al controlador de jBPM para que se puedan desplegar los proyectos generados en la plataforma y estén disponibles a través de este servidor, el cual funciona como alta disponibilidad en el servicio, dado que es posible configurar diferentes KIE

Servers para un mismo controlador jBPM y este puede enviar la solicitud a cualquiera de los disponibles[16].

A estos servidores se les llama servidores de contenedores, debido a que cada despliegue que se hace en el servidor genera un contenedor de aplicaciones, desde la versión 7.x en adelante fue obligatorio utilizar un servidor de aplicaciones KIE Server para el despliegue de los proyectos generados en la plataforma jBPM Workbench.

La combinación del KIE Server y el controlador de jBPM se puede lograr configurando la URL del controlador jBPM y el usuario y contraseña del usuario de jBPM con el rol “kie-server” de dos maneras, como propiedades de configuración al levantar la aplicación y configurando el archivo de configuración: standalone.xml o standalone-full.xml.

2.3.1.4. Kie Base

Todas las aplicaciones que se realizan y se despliegan a través del BPM utilizan un Kie Base por defecto, por lo que no se debe configurar nada, incluso al momento de ejecutar ciertas tareas no es necesario indicar el Kie Base que se desea utilizar cuando no se ha configurado ninguno. Los Kie base son utilizados para encapsular recursos, como los flujos de BPM, reglas de negocio y objetos de datos, estos recursos son ejecutados durante el tiempo de vida de una sesión, la importancia en la implementación de Kie Bases por grupo de recursos radica en que permiten mejorar el rendimiento de las aplicaciones que engloban sus recursos, haciendo que no utilicen memoria innecesaria al momento de ejecutar

[16] Drools documentation. https://docs.jboss.org/drools/release/6.4.0.CR2/drools-docs. Consulta : 08 de junio2018.

un despliegue con muchos recursos que no involucran a la implementación que se requiera. Esta configuración se realiza en el archivo de configuración por proyecto *kmodule.xml*[17].

2.3.2. **Camunda**

La plataforma de Camunda tiene una interfaz web y, a diferencia de jBPM, no proporciona un modelador dentro de la misma plataforma, por lo que necesita de la herramienta modeladora proporcionada por la misma Camunda para la realización de los flujos de BPMN. A continuación, se definen algunas de las características especiales que es posible hacer dentro de la plataforma.

2.3.2.1. Multiarrendamiento

Es una característica de Camunda que permite aislar la información de los procesos como si fueran diferentes servidores[18], esta característica es una alternativa a la forma típica de instalar distintos servidores para clientes distintos, por lo tanto, al hablar de clientes distintos es claramente una opción cuando el flujo de BPM diseñado e implementado es proporcionado como un SaaS (Software as a Service). Esta característica se logra implementar a través de la creación de inquilinos (*tenants*) desde la administración de la plataforma de Camunda. La implementación de esta característica se puede lograr de dos maneras, detalladas a continuación:

[17] Redhat Documentation. https://access.redhat.com/documentation. Consulta: 07 de junio de junio 2018.

[18] Multi Tenanncy - Camunda. https://docs.camunda.org/manual/7.5/user-guide/process-engine/multi-tenancy/. Consulta: 09 de junio 2018.

- Servidor por inquilino: en esta forma de desplegar proyectos se utilizan distintos servidores por cada inquilino que se desea implementar, incluso utilizando su propia base de datos, para los procesos que utilicen esta conexión y almacenen los datos respectivos. La ventaja de esto es tener todos los recursos aislados a otros procesos de los que no se quiere tener compartida la información y una de las desventajas más importante es que, al utilizar su propia base de datos, es un poco costoso en términos de rendimiento, dado que se necesita saltar entre base de datos a través de enlaces.
- Esquema por inquilino: esta manera para desplegar los proyectos permite utilizar un esquema distinto por cada inquilino, esto se logra configurando en su archivo de configuración (bpm-platform.xml) el esquema que utilizará el inquilino que se esté definiendo.

2.3.2.2. Multimotor

Es una característica de Camunda que permite crear distintos motores de Camunda para la ejecución de flujos BPM asociados a un inquilino, los cuales están asociados a diferentes despliegues de flujos BPM y estos pueden aprovecharse para configurar inquilinos por esquema para mantener la data separada por cada inquilino.

2.3.2.3. Administración de usuarios

La administración de usuarios la puede llevar la misma plataforma, pero permite la opción de conectarse a un LDAP para sistemas en distribuciones GNU/Linux y también permite conectarse a un Active Directory de Microsoft.

Para esto Camunda proporciona el servicio de identidad (IdentityService), el cual permite la administración de grupos y usuarios, de LDAP y Active Directory. Es importante mencionar que aunque la plataforma es configurable para distintos motores de identificación, el motor de BPM no valida los usuarios en tiempo de ejecución para la asignación de tareas, es decir que el motor no tiene la habilidad de conocer si un usuario al cual se ha asignado una tarea humana existe, o si está activo. Esto se logra desde la plataforma configurando el *plugin* para proveer identidad (IDAP Identity Provider) en el archivo bpm-platform.xml.

2.3.2.4. Reglas de negocio

A diferencia de jBPM, que tenía distintas formas de validar reglas de negocio, desde archivos drl hasta la externalización de reglas a través de un BRMS, Camunda únicamente proporciona el estándar DMN para la configuración e implementación de reglas de negocio.

3. CONFIGURACIONES INICIALES EN LA IMPLEMENTACIÓN DE JBPM Y CAMUNDA

3.1. Preparación de la plataforma para la implementación del lenguaje

La preparación de la plataforma antes de implementar procesos de negocio empresarial conlleva una serie de configuraciones las cuales varían según la herramienta seleccionada. Estas configuraciones pueden ser desde fuentes de datos, administración de usuarios y servidores de aplicaciones.

3.1.1. jBPM

jBPM y Drools (BRMS) se distribuyen bajo una licencia Apache 2.0, la cual permite usar de forma libre el software bajo la condición de mantener el *copyright* y la exención de responsabilidad. Además de la plataforma jBPM ofrece un API Java para la ejecución de procesos dentro de la plataforma, así como fuera de ella, esto es, en el caso que únicamente se quiera ejecutar un único flujo de BPM para algún proyecto, se puede desarrollar el flujo de BPM con el *plugin* de eclipse[19] y desde el mismo proyecto ejecutar el proceso con el motor proveído por el API.

Para la utilización de la plataforma jBPM es posible instalarlo de dos maneras, la primera forma es la convencional, se descarga un archivo .zip de la página oficial[20] y se despliega en el servidor web que se prefiera, algunos ejemplos de servidores web en los que se pueden desplegar son los siguientes:

[19] jBPM Eclipse. https://docs.jboss.org/jbpm/v6.0/userguide/jBPMEclipseJBPM.html. Consulta: 06 mayo de 2018.

[20] jBPM. https://www.jbpm.org/. Consulta: 06 de mayo de 2018.

- Wildfly
- EAP (Red Hat Enterprise Application Platform)
- Tomcat
- Glassfish
- Payara

Para este tipo de instalación es necesario tener el ambiente configurado con una instalación de JDK y Ant, debidamente configuradas las variables de entorno para poder utilizar el lenguaje Java sin ningún problema y poder generar el despliegue de la aplicación. Otra de las herramientas necesarias a ser instaladas es GIT, para la administración de las fuentes que se desarrollen en estas plataformas (jBPM y Drools).

Dado que es una aplicación web Java, esta puede ser desplegada en cualquier sistema operativo que soporte Java y tenga un servidor web para poder desplegar la aplicación en formato WAR. Otra de las alternativas para utilizar la plataforma sin necesidad de instalar los prerrequisitos antes mencionados es utilizando una imagen Docker de jBPM[21], Drools[22] y/o KIE Server[23], estas imágenes no solo traen los requisitos de la aplicación, sino que ya viene desplegada la misma únicamente para que sea configurada a nivel de usuarios y base de datos.

Los usuarios que la aplicación puede manejar son usuarios locales, los cuales son configurados a nivel de consola y la aplicación los administra y la otra manera de configurar usuarios es a través de LDAP, se configura en la aplicación

[21] jBPM workbench. https://hub.docker.com/r/jboss/jbpm-workbench/. Consulta: 09 de mayo de 2018.

[22] Drools workbench. https://hub.docker.com/r/jboss/drools-workbench/. Consulta: 09 de mayo de 2018.

[23] KIE Server. https://hub.docker.com/r/jboss/kie-server/. Consulta: 09 de mayo de 2018.

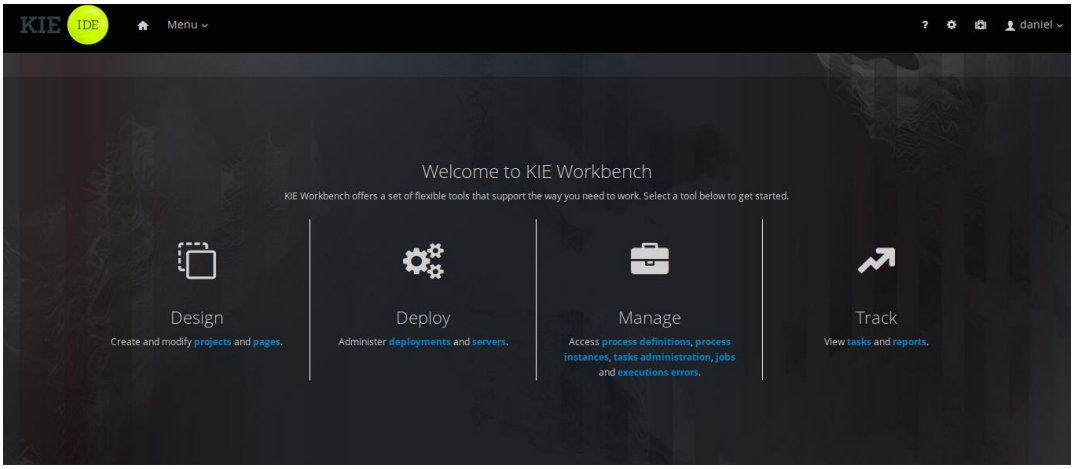
en su archivo de configuración standalone-full.xml el servidor y credenciales de LDAP para que pueda trabajar con los usuarios de LDAP.

La base de datos también es configurable, por defecto utiliza la base de datos db2, pero estas son algunas de las bases de datos a las que es posible configurar para su conexión:

- MySQL
- Oracle
- Postgres
- SQL Server

Para más información de cómo configurar la administración de usuarios y las bases de datos, es posible consultar la documentación oficial[24].

Figura 24. **Pantalla inicial de jBPM**



Dividida en cuatro secciones: diseño: engloba los proyectos que pueden ser implementados, despliegue: lista de aplicaciones desplegadas en los servidores de ejecución, administración: administración de instancias y tareas de procesos y, por último, una sección de reportería. Fuente: elaboración propia, empleando jBPM Workbench.

[24] Configuración de jBPM. https://docs.jboss.org/jbpm/release/7.7.0.Final/jbpm-docs/html_single/. Consulta: 09 de mayo de 2018.

3.1.2. **Camunda**

Existen dos versiones de Camunda: una empresarial, que además de brindar soporte técnico, ofrece mayores características que se pueden ver en la página oficial[25], y la versión de la comunidad de Camunda, que se distribuye bajo una licencia Apache 2.0, la cual permite usar de forma libre el software bajo la condición de mantener el *copyright* y la exención de responsabilidad.

Al igual que jBPM ofrece un API Java para la ejecución de procesos dentro y fuera de la plataforma, a diferencia de jBPM no distribuye un *plugin* para algún IDE, sino que se utiliza la misma herramienta modeladora y se importa la fuente al proyecto. Otra diferencia considerable al API de jBPM es que además de proporcionar el motor de ejecución de procesos, proporciona una pequeña plataforma para monitorear el proceso que se ejecuta dentro de la aplicación en la cual fue incrustado el flujo. Más detalles del uso de la plataforma pueden encontrase en su documentación[26] y para la utilización dentro de proyectos también visitar su documentación[27].

Para la utilización de la plataforma de Camunda es posible instalarla de dos maneras, al igual que jBPM, la forma convencional en la que se descarga un archivo .zip para sistemas Windows y un tar.gz para sistemas para distribuciones Linux de la página oficial[28], y se despliega en el servidor web que se prefiera, algunos ejemplos de servidores web en los que se puede desplegar son los siguientes:

[25] Productos Camunda. https://camunda.com/products/cockpit/#/features. Consulta: 08 de mayo de 2018.

[26] Configurar Camunda. https://docs.camunda.org/get-started/bpmn20/. Consulta: 08 de mayo de 2018.

[27] Manual de Camunda. https://docs.camunda.org/manual/7.6/user-guide/process-engine/process-engine-bootstrapping/. Consulta: 08 de mayo de 2018.

[28] Descargar Camunda. https://camunda.com/download/. Consulta: 08 de mayo de 2018.

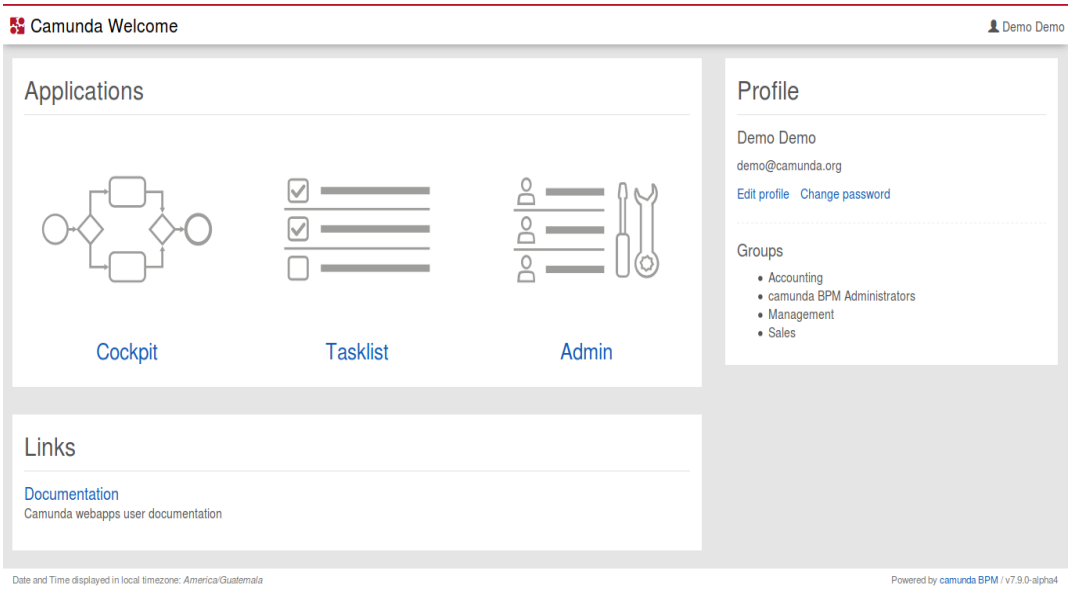
- Wildfly
- Tomcat
- Glassfish
- Payara

Los prerrequisitos para la utilización de la plataforma son que el servidor *host* tenga instalador y esté debidamente configurado el JDK, adicionalmente se recomienda la instalación de GIT como administrador de fuentes y Maven como administrador de dependencias para los proyectos Java.

Otra de las alternativas para utilizar la aplicación sin necesidad de instalar los prerrequisitos antes mencionados es utilizando una imagen de la plataforma de Camunda[29], estas imágenes no solo traen los requisitos de la aplicación, sino que ya viene desplegada la misma únicamente para que sea configurada a nivel de usuarios y base de datos.

[29] Plataforma Camunda. https://hub.docker.com/r/camunda/camunda-bpm-platform/. Consulta: 08 de mayo de 2018.

Figura 25. **Pantalla inicial de la plataforma Camunda**



Dividida en tres secciones: administración de flujos BPM (*cockpit)*, administración de tareas humanas de instancias de procesos(*tasklist)* y administración de usuarios, fuentes de datos(*Admin)*.

Fuente: elaboración propia, empleando jBPM Workbench.

3.2. Arquitectura para la utilización de implementación de un BPM con jBPM y Camunda

Cada una de las herramientas implementan una arquitectura mínima para poder funcionar, y pueden llegar a ser implementadas en arquitecturas complejas para una mayor robustez de la aplicación. La arquitectura también dependerá en algunas ocasiones de las herramientas que se deseen agregar y que interactúan alrededor de flujos de negocio, como lo es la implementación de reglas de negocio.

3.2.1. **jBPM**

Al momento de implementar esta herramienta es posible hacerlo de distintas maneras, empezando por que la herramienta permite realizar flujos sin necesidad de tener instalada toda la plataforma, sino como librerías es posible ejecutar su motor de procesos como lo que se dio a conocer en capítulos anteriores.

Utilizando la plataforma, la arquitectura se vuelve un poco más compleja, dependiendo del grado de complejidad que se le desee para una mayor robustez de la implementación. Y esto va a depender mucho de la utilización de reglas de negocio, local o extendido en otro servidor, y la utilización de un KIE Server para la distribución y alta disponibilidad de las reglas de negocio y/o los despliegues de los flujos BPM.

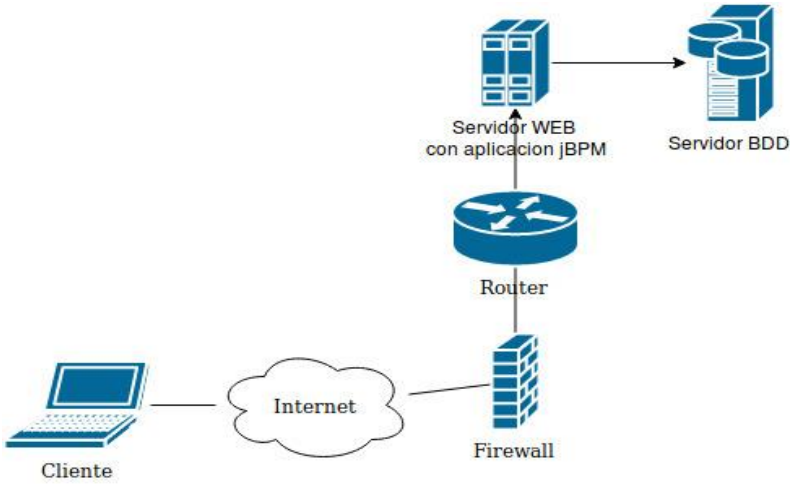
Así que se detallarán algunas formas en las que se puede definir la arquitectura según los requerimientos que se tengan.

- Arquitectura con motor de flujos incrustado: con este tipo de utilización del motor de jBPM podrían existir muchas arquitecturas debido a que, al final, se obtiene una aplicación como cualquier otra, por lo que todas las arquitecturas existentes para un proyecto de escritorio y web podrían ser aplicadas a esta forma de utilizar el motor jBPM. Debido a que hoy en día la mayoría de las aplicaciones son web o de aplicaciones para celulares, a continuación, se muestra la típica arquitectura cliente-servidor para las aplicaciones web.

    - Cliente web
    - *Firewall*
    - *Router*

- Servidor web
- Servidor de base de datos

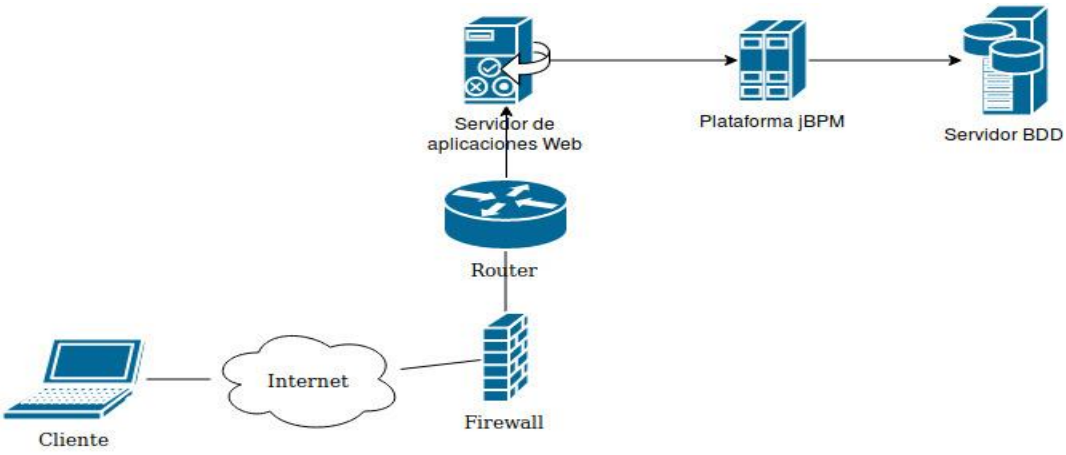
Figura 26. **Arquitectura cliente-servidor**



Fuente: elaboración propia, empleando draw.io.

- Arquitectura con plataforma sin servidor de despliegues: en esta arquitectura únicamente se agrega la plataforma al diagrama anterior y es cuando las reglas de negocio son implementadas dentro de la plataforma de BPM. Es importante mencionar que para versiones inferiores a jBPM 7.0 no era necesario utilizar un KIE Server para desplegar los proyectos y la plataforma brindaba un KIE Server incrustado en la misma.

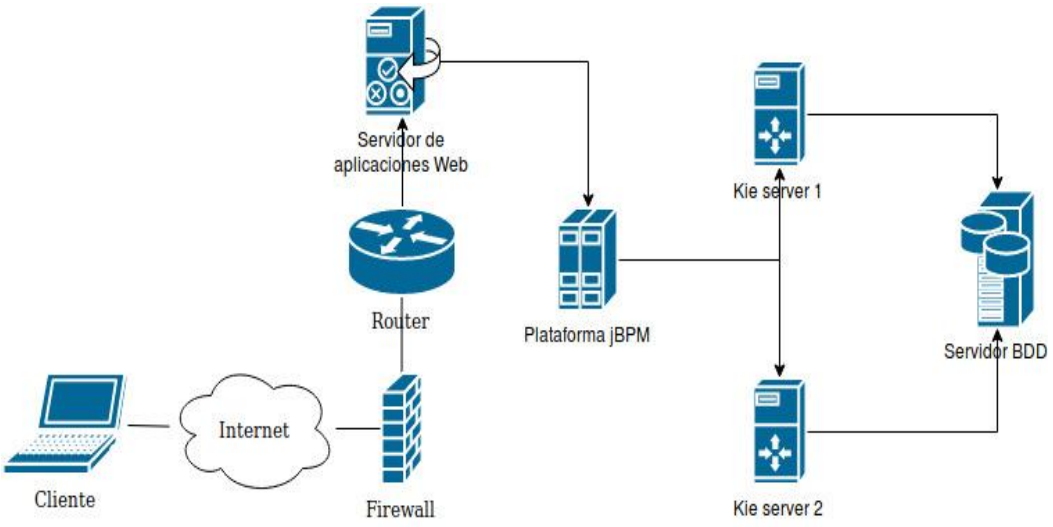
Figura 27. **Arquitectura con plataforma y sin KIE Server**



Fuente: elaboración propia, empleando draw.io.

- Arquitectura con plataforma y servidor de despliegues: en esta arquitectura únicamente se agrega la plataforma y los servidores de despliegues (KIE Server). Para las nuevas versiones de jBPM, la plataforma obliga a utilizar un servidor de despliegues, en este caso uno o más KIE Server, que se registran al controlador que contiene la plataforma para proporcionar los despliegues que se hayan realizado en ellos, estos despliegues son los proyectos Maven que contienen las fuentes de los flujos de BPM.

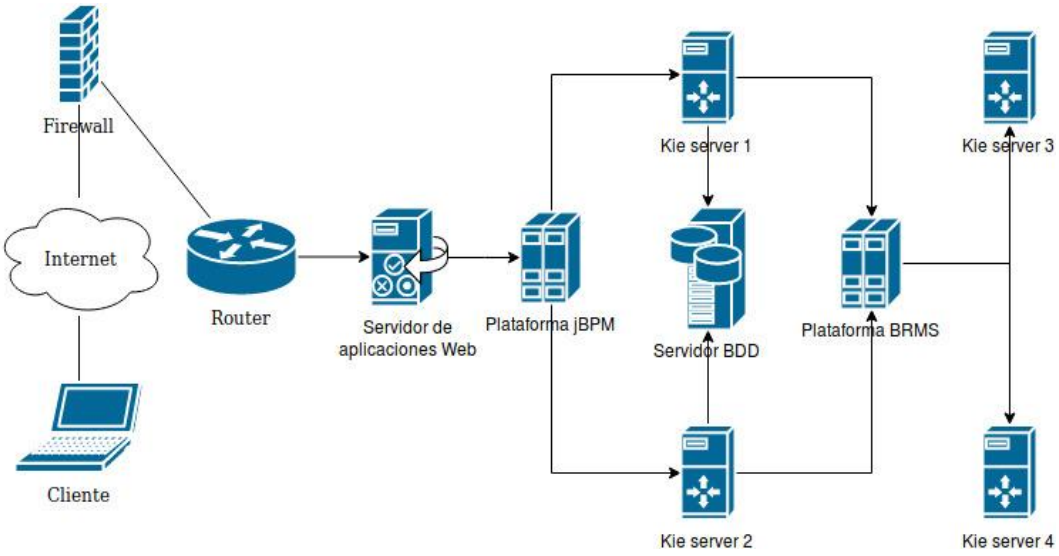
Figura 28. **Arquitectura de plataforma jBPM con KIE Server**



Fuente: elaboración propia, empleando draw.io.

- Arquitectura con plataforma utilizando BRMS y servidores de despliegue: por último, esta arquitectura es la más compleja debido a que se externaliza la función de las reglas de negocio a un Sistema de Administración de Reglas de Negocio (BRMS por sus siglas en inglés), sistema que también es capaz de utilizar servidores de despliegue para exponer las reglas de negocio. En esta arquitectura, al igual que en la última vista, la cantidad de servidores de despliegue dependerá de cada implementador.

Figura 29. **Arquitectura de jBPM con BRMS y KIE Server**



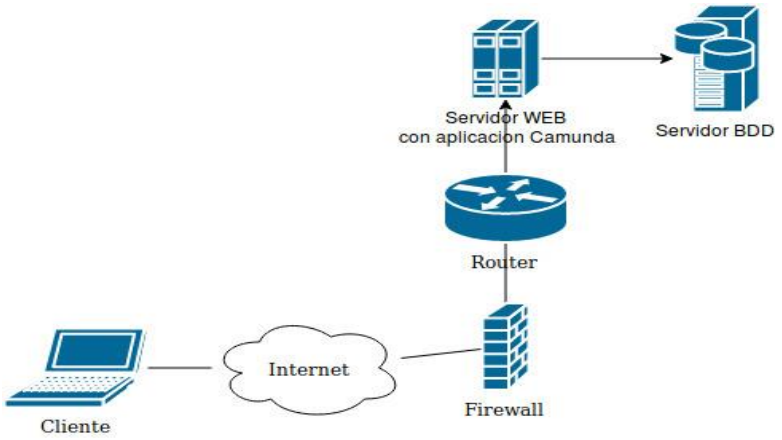
Fuente: elaboración propia, empleando draw.io.

3.2.2. **Camunda**

Al igual que jBPM, Camunda ofrece las dos modalidades de utilización de su motor de flujos de BPM, el cual puede ser utilizado dentro de un proyecto a través de librerías o utilizando el motor a través de una plataforma que se encarga de administrar todas las instancias de todos los flujos que están desplegados en ella.

- Arquitectura con motor de flujos incrustado: es una arquitectura exactamente igual como la que se puede implementar con jBPM al utilizar un motor incrustado por librerías. El ciclo de vida de los procesos que lleva esta modalidad es el mismo que lleva una aplicación cuando es iniciada y terminada.

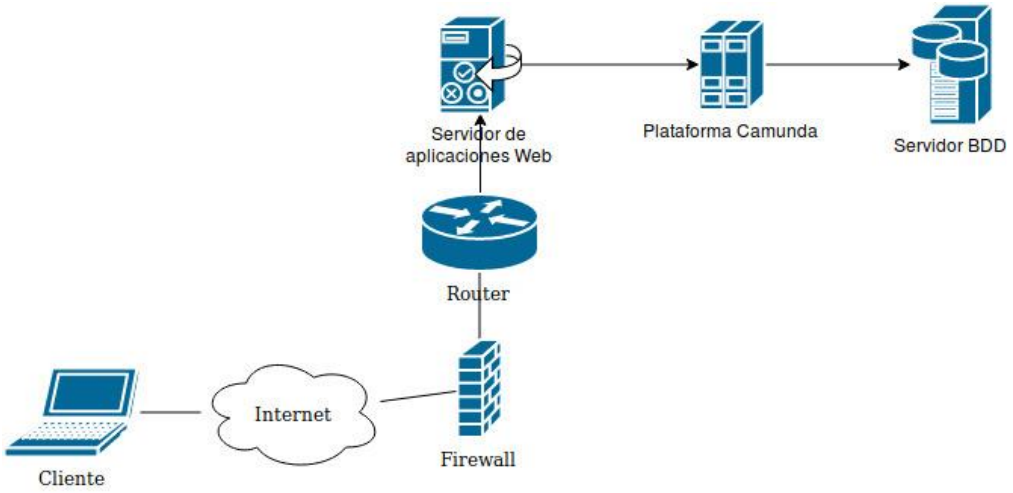
Figura 30. **Arquitectura Camunda como aplicación web independiente**



Fuente: elaboración propia, empleando draw.io.

- Arquitectura con plataforma de Camunda: cuando se utiliza Camunda a través de una plataforma que administra las instancias la arquitectura sería la siguiente:

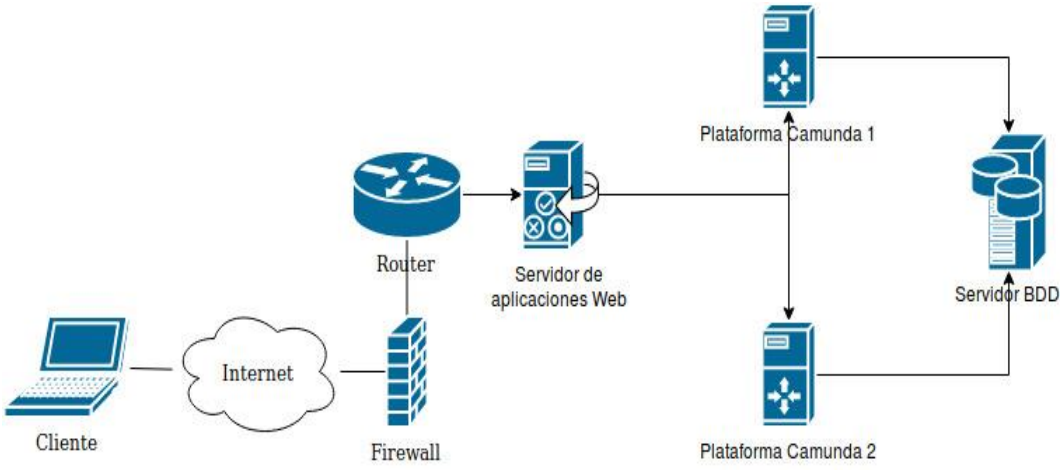
Figura 31. **Arquitectura Camunda como plataforma**



Fuente: elaboración propia, empleando draw.io.

- Arquitectura con plataforma de Camunda en *clúster*: Cuando se utiliza Camunda a través de una plataforma en *clúster.*

Figura 32. **Arquitectura Camunda como plataforma en *clúster***



Fuente: elaboración propia.

3.3. Preparación de las herramientas para el modelado de los procesos

La optima configuración de las herramientas es de vital importancia antes de iniciar a crear flujos de BPM, ya que depende de la configuración inicial el correcto funcionar al momento de ejecutar los procesos de negocio.

3.3.1. jBPM

Antes de implementar flujos de procesos empresariales en la herramienta jBPM es importante preparar el ambiente con los prerrequisitos mínimos para iniciar a crear flujos de procesos, los requisitos mínimos varian según las necesidades, si se desea conectar a una base de datos especifica como lo es MySQL o Postgres, deberán configurase los drivers y JDBC para conectarse a estas base de datos, de igual manera, si se desea autenticarse a través de LDAP hay que configurar el archivo de propiedades para aceptar estas características.

3.3.1.1. Configuración de la fuente de datos

Cuando se desea utilizar una base de datos diferente a la que viene configurada H2, es necesario agregar el *driver* del sistema de base de datos y a continuación se debe configurar el mismo, toda esta configuración no es parte de la plataforma jBPM sino del servidor web en el que se desea alojar la aplicación. A continuación, se presenta la configuración de un sistema de base de datos diferente al H2 por defecto por un el sistema de base de datos MySQL en un servidor web de aplicaciones Wildfly:

- Wildfly 10.0
- MySQL Server 8.0

Primero se debe configurar un nuevo módulo para el servidor Wildfly, en este caso se debe agregar un módulo MySQL en el siguiente directorio:

/opt/jboss/wildfly/modules/system/layers/base/com/mysql/main/

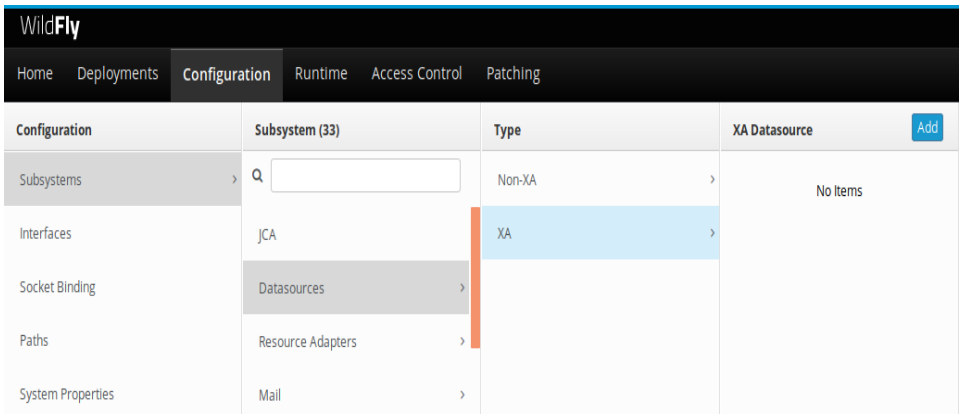
Si el directorio no existe, se debe crear y agregar un archivo de configuración llamado *module.xml* con el siguiente contenido:

```
<module xmlns="urn:jboss:module:1.5" name="com.mysql">
  <resources>
        <resource-root path="mysql-connector-java-8.0.11.jar" />
  </resources>
  <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Y el *driver* especial para la base de datos a la que se desea conectar, en el caso de MySQL en la versión 8.0 el nombre del *driver* es: mysql-connector-java-8.0.11.jar. Luego de esta adición del nuevo módulo ya es posible agregar nuevas fuentes de datos para conexiones MySQL. A continuación, se muestran las imágenes para la configuración de la fuente de datos para la plataforma jBPM que se utilizará.

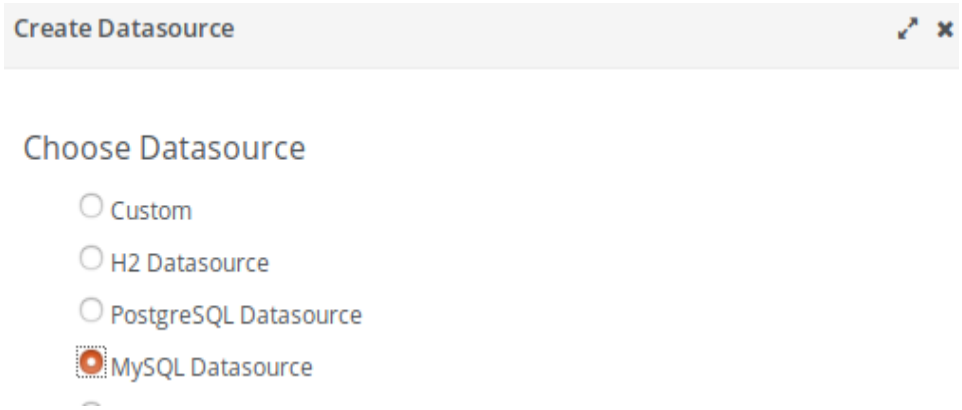
Las fuentes de datos XA son utilizadas para sistemas transacciones en donde se tiene un *clúster* de base de datos y las no XA son utilizadas para sistemas transaccionales, pero no en *clúster*, es decir que únicamente la fuente de datos únicamente se conectará a una base de datos.

Figura 33. **Configuración de una fuente de datos desde la administración de Wildfly**



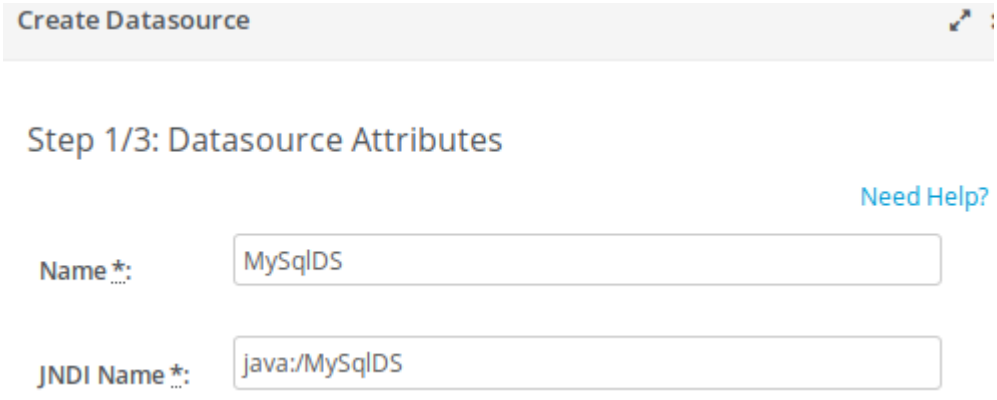
Fuente: elaboración propia, empleando Wildfly 11.0.

Figura 34. **Selección del tipo de fuente de datos**



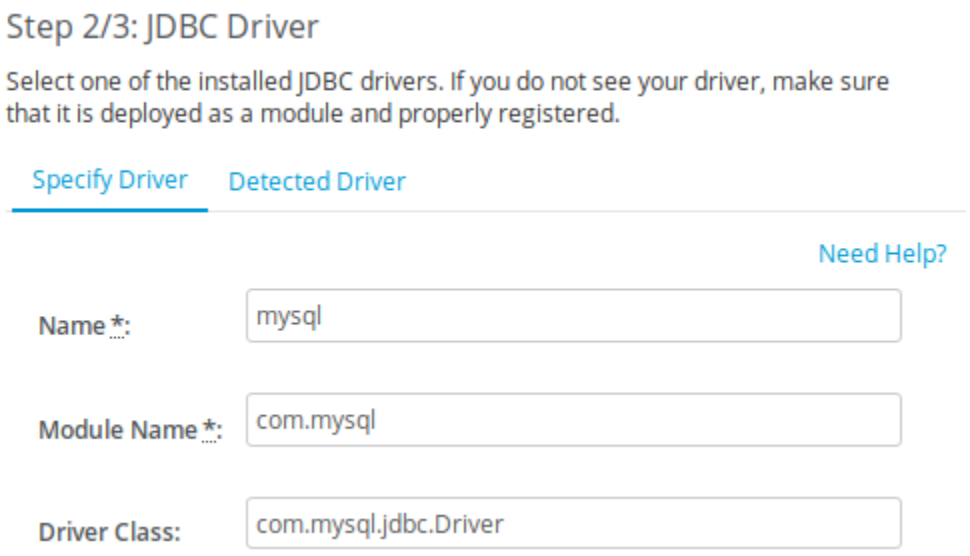
Fuente elaboración propia, empleando Wildfly 11.0.

Figura 35. **Configuración de nombre de fuente de datos**



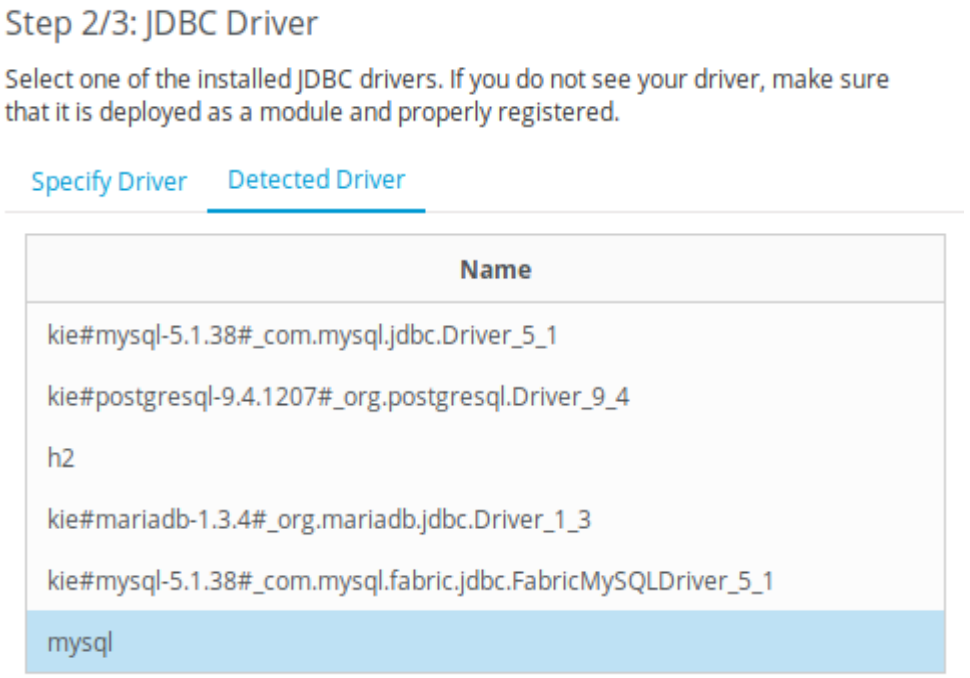
Fuente: elaboración propia, empleando Wildfly 11.0.

Figura 36. **Configuración del controlador de la fuente de datos**



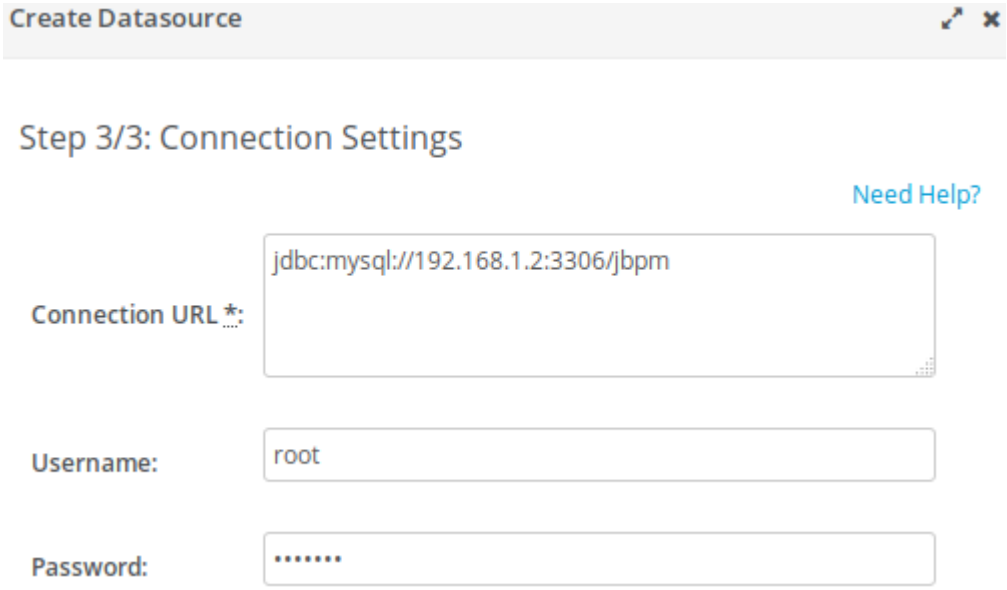
Fuente: elaboración propia, empleando Wildfly 11.0.

Figura 37. **Selección de *driver* MySQL configurado**



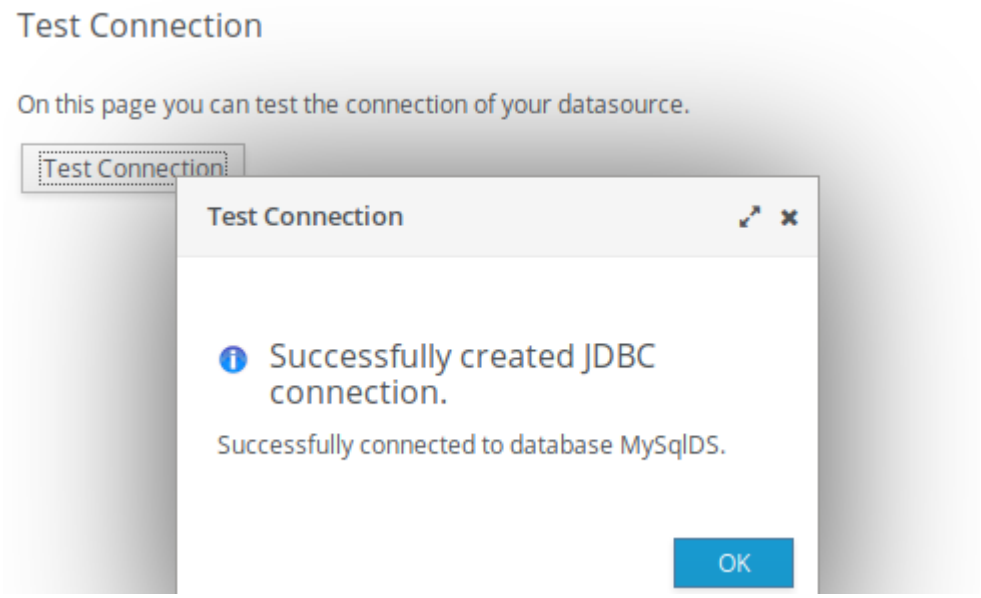
Fuente: elaboración propia, empleando Wildfly 11.0.

Figura 38. **Ingreso de credenciales para la fuente de datos configurada**



Fuente: elaboración propia, empleando Wildfly 11.0.

Figura 39. **Prueba de comunicación para nueva fuente de datos**



Fuente: elaboración propia, empleando Wildfly 11.0.

- Consideraciones importantes:

    - Para agregar por *default* la nueva fuente de datos, se debe configurar la sección “default-bindings” con el nuevo jndi-name :

      <default-bindings
      context-service="java:jboss/ee/concurrency/context/default"
      datasource="java:jboss/datasources/ExampleDS"
      jms-connection-factory="java:jboss/DefaultJMSConnectionFactory"
      managed-executor-service="java:jboss/ee/concurrency/executor/default"
      managed-scheduled-executor-
      service="java:jboss/ee/concurrency/scheduler/default"
      managed-thread-factory="java:jboss/ee/concurrency/factory/default"/>

    - La configuración de la fuente de datos debe realizarse tanto en la plataforma jBPM Workbench como en el servidor *KIE Server*, debido a que este último es el que genera los procesos y los registrará en la base de datos. Un dato importante que tomar en cuenta es que en la versión que se utiliza para desarrollar este documento, la versión 7.7 tiene un error en el que no es posible cambiar el JNDI del *datasource*, dado que el módulo jBPM que extiende la capacidad de *KIE Server* de administrar flujos de BPM utiliza la JNDI por defecto llamada ExampleDS, por lo que cualquier fuente de datos que se configure debe llevar el mismo nombre.

    - Si no se tiene configurado un directorio de usuarios como LDAP, la aplicación maneja sus propios usuarios para administración (Wildfly) y para la aplicación (Workbench). Para configurar estos usuarios se debe acceder al directorio: */opt/jboss/wildfly/bin*, y ejecutar el add-user.sh, un usuario de aplicación seleccionar la

opción (b) y a continuación ingresar las credenciales para el nuevo usuario, el rol correspondiente:

- admin
- analyst
- developer
- manager
- user

Si se piensa utilizar el usuario para hacer llamadas al servicio REST que expone el controlador de jBPM, es necesario agregar un rol llamado: “rest-all”.

3.3.1.2. Configuración del KIE Server

Esta configuración se basa en una imagen de Docker, en la cual ya viene instalado el KIE Server sobre un servidor web Wildfly 10.0, por lo que únicamente se procede a configurar para que se registre al controlador Workbench antes configurado.

Debe crearse un usuario de aplicación de la misma forma que se explicó en la configuración de jBPM Workbench, con la única diferencia que ahora debe agregarse un nuevo rol llamado: “kie-server”, de igual manera debe crearse un usuario dentro del controlador jBPM Workbench para que el KIE Server pueda autenticarse y registrarse en la aplicación. El usuario puede ser el mismo.

Teniendo los dos usuarios creados en los dos servidores, se procede a configurar el inicio del KIE Server para que se registre, para esto se debe utilizar los siguientes valores:

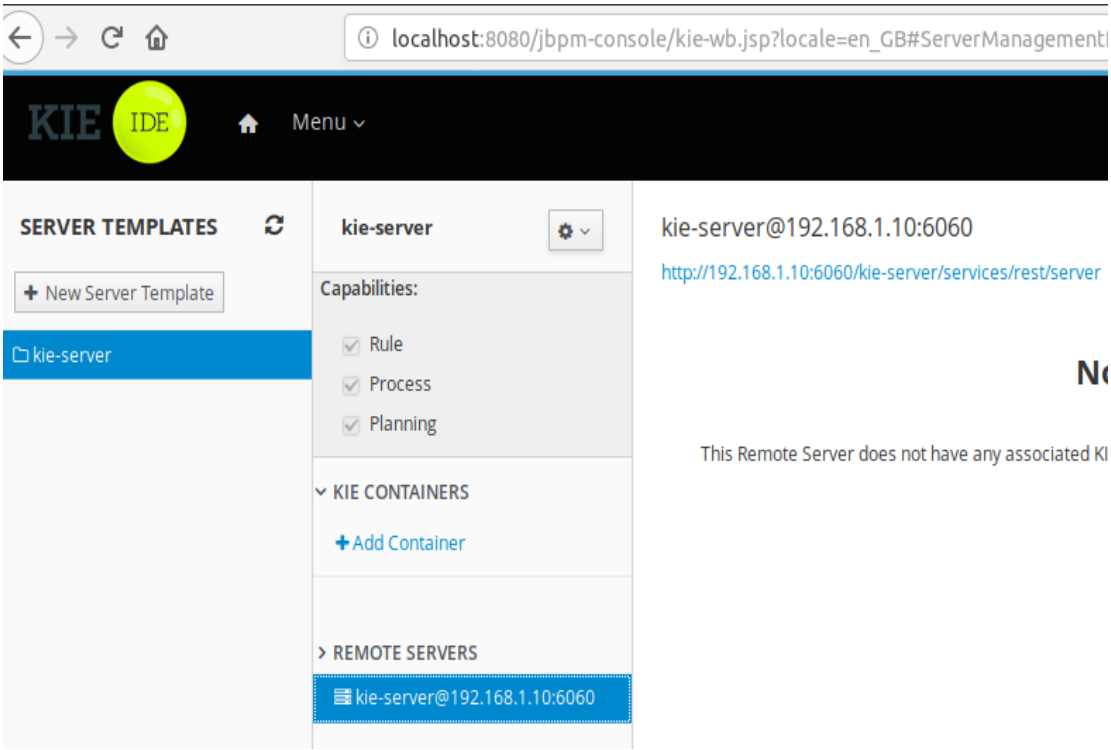
- org.kie.server.id: nombre del servidor con el que se registra al controlador.
- org.kie.server.user: usuario dentro de aplicación dentro del servidor de KIE Server.
- org.kie.server.pwd: contraseña del usuario dentro de aplicación dentro del servidor de KIE Server.
- org.kie.server.controller: URL del controlador de jBPM Workbench.
- org.kie.server.controller.user: usuario de aplicación configurado dentro del servidor de jBPM Workbench.
- org.kie.server.controller.pwd: contraseña del usuario de aplicación configurado dentro del servidor de jBPM Workbench.
- org.kie.server.location: URL del servicio REST del servidor KIE Server.

Quedando una línea de configuración con los parámetros antes configurados de la siguiente forma:

exec ./standalone.sh
-b $JBOSS_BIND_ADDRESS
-c standalone-full.xml
-Dorg.kie.server.location=http://192.168.1.10:6060/kie-server/services/rest/server
-Dorg.kie.server.id=kie-server
-Dorg.kie.server.user=kieserver
-Dorg.kie.server.pwd=kieserver1!
-Dorg.kie.server.controller=http://192.168.1.10:8080/jbpm-console/rest/controller -Dorg.kie.server.controller.user=kieserver
-Dorg.kie.server.controller.pwd=kieserver1!

Y de esta manera aparecerá un servidor KIE Server para desplegar las aplicaciones que se generen con el jBPM Workbench.

Figura 40. **KIE Server registrado dentro de la plataforma jBPM**



Fuente: elaboración propia, empleando jBPM Workbench.

3.3.1.3. Configuración del repositorio de software

Esta configuración se debe realizar tanto en el servidor de jBPM como en el servidor de KIE Server, debido a que en el jBPM únicamente se trabaja el proyecto y al momento de desplegar el proyecto se debe generar el JAR en un repositorio de software. Este repositorio debe ser centralizado para que también sea accesible desde el KIE Server, dado que utiliza este mismo repositorio para exponer los flujos disponibles a través de un servicio REST. Como bien es sabido

la sección “distribution management” dentro del archivo POM.xml indica el servidor de repositorio de software al que se debe llegar para registrar los nuevos proyectos, muchas veces se cambiará de ambiente entre desarrollo, pruebas y producción, por lo que se recomienda utilizar perfiles que se configuran dentro del ../.m2/settings.xml, y en estos configurar propiedades, las cuales pueden ser utilizadas dentro del archivo POM.xml de los proyectos. La forma de hacer referencia a estas propiedades es:

${propiedad.configurada.perfil}

Un ejemplo del archivo de configuración Settings.xml sería el siguiente:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd">
<localRepository/>
<servers>
<server>
<id>internal</id>
<username>admin</username>
<password>alvarez1</password>
<configuration/>
</server>
</servers>
<profiles>
<profile>
<id>main</id>
<repositories>
<repository>
<id>internal</id>
<name>internal</name>
<url>http://192.168.1.10:5050/repository/internal/</url>
```

</repository>
</repositories>
<properties>
<distribution.management>http://192.168.1.10:5050/repository/internal/</distribution.manageme
nt>
</properties>
</profile>
</profiles>
<activeProfiles>
<activeProfile>main</activeProfile>
</activeProfiles>
</settings>

Y el archivo POM.xml del proyecto sería el siguiente:

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<modelVersion>4.0.0</modelVersion>
<groupId>com.test</groupId>
<artifactId>flow</artifactId>
<version>1.0.1</version>
<name>Test</name>
<description>Test</description>
<distributionManagement>
<repository>
<id>internal</id>
<url>${distribution.management}</url>
</repository>
</distributionManagement>
<repositories>
<repository>
<id>guvnor-m2-repo</id>
<name>Guvnor M2 Repo</name>

<url>file:///opt/jboss/wildfly/bin/repositories/kie/global</url>
</repository>
</repositories>
</project>

3.3.2. **Camunda**

Al igual que jBPM en la herramienta Camunda es importante preparar el ambiente con los prerrequisitos mínimos para iniciar a crear flujos de procesos, los requisitos mínimos varian según las necesidades, si se desea conectar a una base de datos especifica como lo es MySQL o Postgres, deberán configurase los drivers y JDBC para conectarse a estas base de datos, de igual manera, si se desea autenticarse a través de LDAP hay que configurar el archivo de propiedades para aceptar estas características.

3.3.2.1. Configuración de la fuente de datos

Al igual que la plataforma de jBPM, la base de datos configurada por defecto dentro de la imagen de Docker proporcionada por Camunda es una H2. Para cambiar la base de datos por defecto es necesario agregar el *driver* del sistema de base de datos y posteriormente se debe configurar el mismo. Esta configuración no es parte de la plataforma de Camunda sino del servidor web en el que se encuentra instalada la aplicación.

A continuación, se presenta la configuración de un sistema de base de datos diferente al H2 por defecto, por un sistema de base de datos MySQL en un servidor web de aplicaciones Tomcat:

- Apache Tomcat 9.0

- MySQL Server 8.0

Antes de configurar la fuente de datos se debe configurar la base de datos con la estructura utilizada por la plataforma de Camunda, por lo que se debe ejecutar los archivos *mysql_engine_7.9.0.sql* y *mysql_identity_7.9.0.sql* para el caso de utilizar una base de datos MySQL, de lo contrario dentro del mismo directorio se encuentran los *scripts* de Lengua de Definición de Datos (DDL por sus siglas en inglés) para distintas bases de datos. Estos archivos se encuentran dentro de la carpeta de instalación[30] de Camunda en el directorio */sql/create*.

Posteriormente a la creación de la estructura dentro de la base de datos, se debe agregar el *driver* de MySQL 8.0 al directorio de librerías de Apache Tomcat.

/camunda/lib/mysql-connector-java-8.0.11.jar

Posteriormente se debe configurar una nueva fuente de datos dentro de la sección de recursos en el archivo de configuración de Apache Tomcat.

/camunda/conf/server.xml

Dentro de la sección *GlobalNamingResources* se debe agregar el siguiente contenido, el cual contiene el nombre del recurso, nombre del *driver*, credenciales y URL de la base de datos:

```
<Resource name="jdbc/CamundaMySQL"
auth="Container"
type="javax.sql.DataSource"
maxTotal="100"
maxIdle="30"
maxWaitMillis="10000"
```

[30] Descargar Camunda. https://camunda.com/download/. Consulta: 12 de mayo de 2018.

username="root"
password="alvarez"
driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://192.168.1.10:3306/camunda"/>

Luego de realizar esta configuración, solamente hace falta agregar esta nueva fuente de datos MySQL al archivo de configuración de Camunda, la cual se encuentra en siguiente directorio:

/camunda/conf/bpm-platform.xml

Y en la sección *ProcessEngine* se debe configurar el nombre del recurso configurado en el servidor Apache Tomcat.

<datasource>java:jdbc/CamundaMySQL</datasource>

4. IMPLEMENTACIÓN DE PROCESOS DE NEGOCIO: JBPM Y CAMUNDA

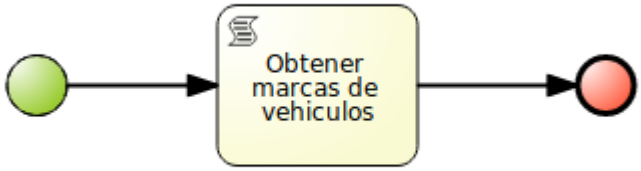
4.1. Implementación de los distintos lenguajes en las herramientas de BPM

A continuación, se toma como ejemplo el flujo de procesos para una empresa aseguradora de automóviles, en la que se requiere que se ingrese una lista de marcas de vehículos, que irá dentro de una cadena de texto separado por comas (,), por lo tanto, antes de procesar los vehículos se debe recorrer para determinar la cantidad de autos a asegurar.

4.1.1. jBPM

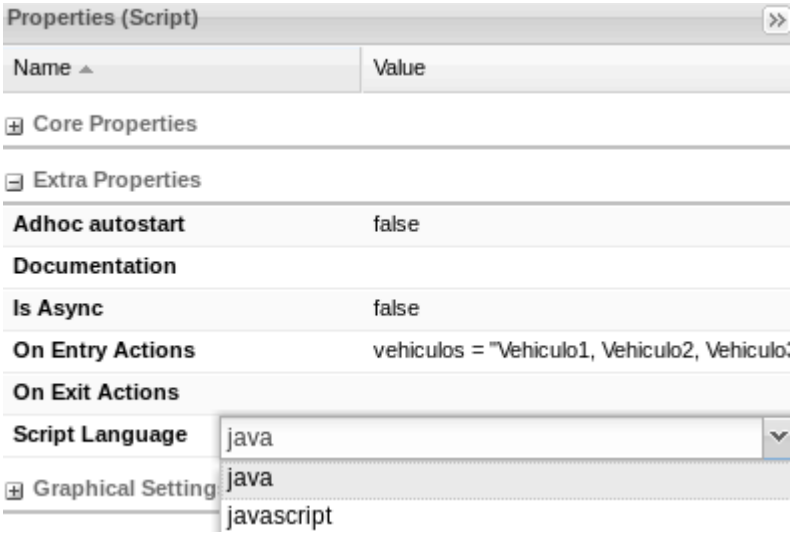
El flujo es el mismo para las dos implementaciones en jBPM (ver figura 41), las que únicamente contienen una actividad inicial, esta actividad es de tipo Script, por lo que se puede añadir código Java o JavaScript. Para cambiar de lenguaje basta con seleccionar el lenguaje dentro de la sección “Extra Properties” (ver figura 42) y la codificación para cada lenguaje debe implementarse según el lenguaje seleccionado (ver figuras 43 y 44).

Figura 41. **Tarea Script en digarama BPMN elaborado en jBPM**



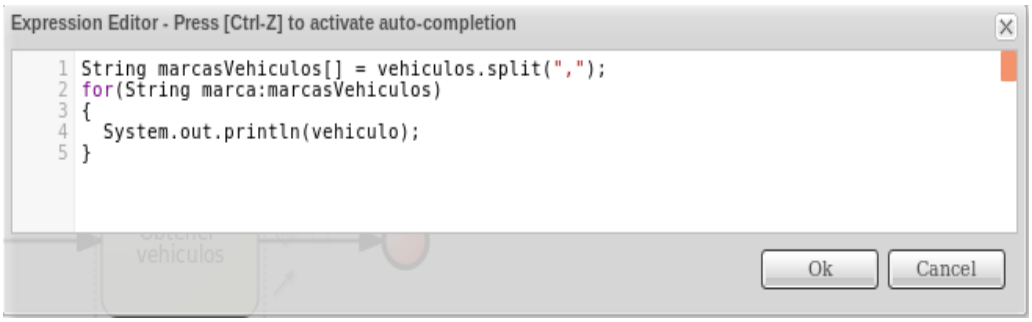
Fuente: elaboración propia, empleado jBPM Workbench.

Figura 42. **Cambio de lenguaje en tarea Script en jBPM**



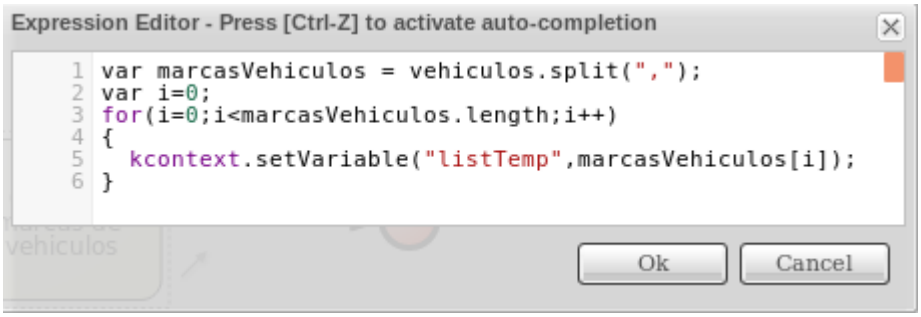
Fuente: elaboración propia, empleando jBPM Workbench.

Figura 43. **Código Java en tarea Script**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 44. **Código JavaScript en tarea tipo Script**



Fuente: elaboración propia, empleando jBPM Workbench.

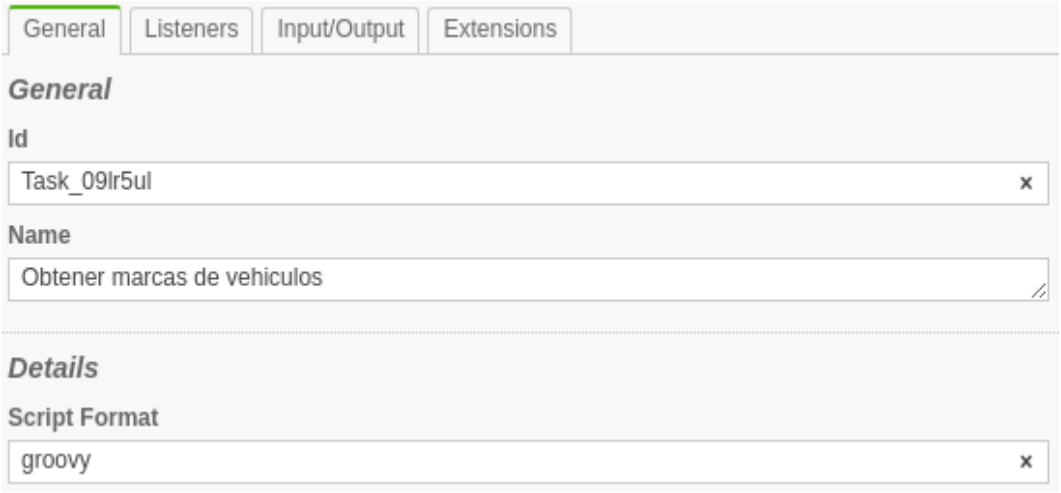
4.1.2. **Camunda**

El flujo es el mismo para Camunda (ver figura 45), la que únicamente contiene una actividad inicial, esta actividad es de tipo Script, por lo que se puede añadir código Apache Groovy o JavaScript. Para cambiar de lenguaje basta con seleccionar el lenguaje dentro de la pestaña “General” y sección “Details” (ver figura 46) y la codificación para cada lenguaje debe implementarse según el lenguaje seleccionado (ver figuras 47 y 48).

Figura 45. **Tarea Script en diagrama BPMN en plataforma Camunda**

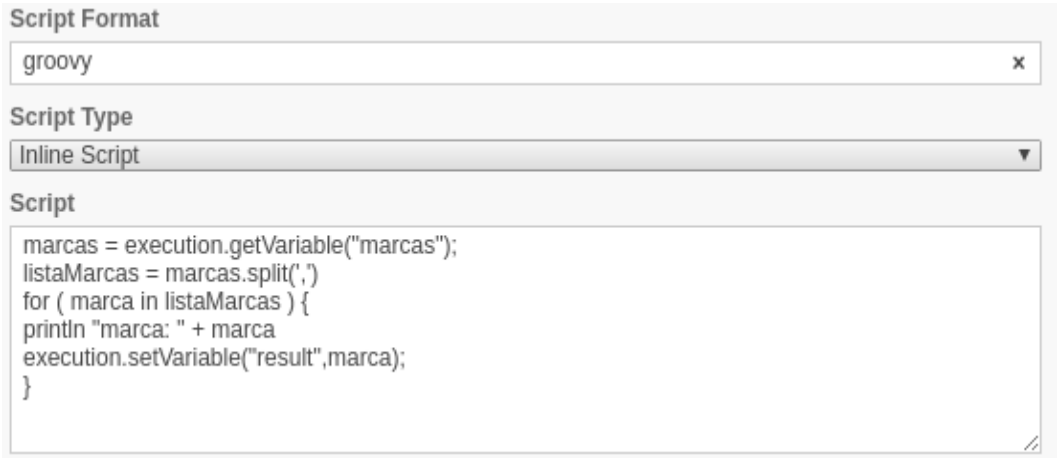
Fuente: elaboración propia, empleando Camunda Modeler.

Figura 46. **Cambio de lenguaje en tarea Script en Camunda Modeler**



Fuente: elaboración propia, empleando Camunda Modeler.

Figura 47. **Código Groovy para recorrer una lista**



Fuente: elaboración propia, empleando Camunda Modeler.

Figura 48. **Código JavaScript para recorrer una lista**

Fuente: elaboración propia, empleando Camunda Modeler.

4.2. Implementación de DMN y tablas de decisión

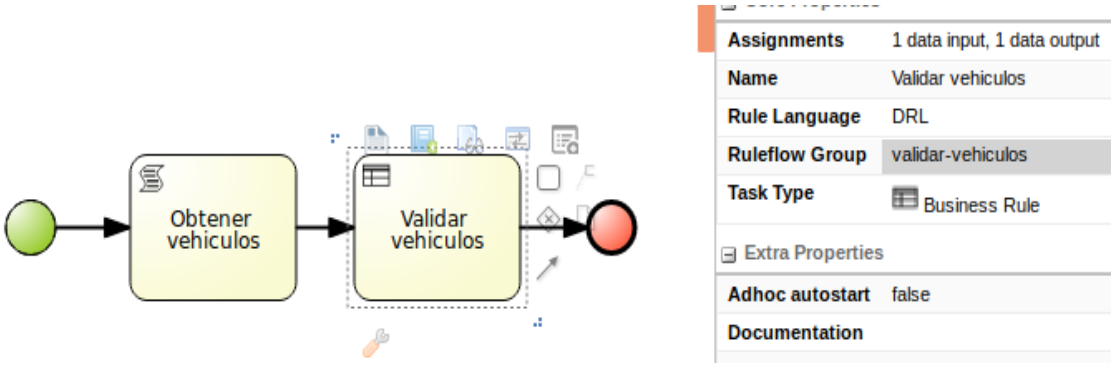
Siguiendo el ejemplo de la aseguradora de automóviles, luego de haber obtenido las marcas de vehículos a ser asegurados a través de una tarea de Script, se procedió a validar las marcas a través de reglas de negocio, en las que para el ejemplo una marca no permitida para ser asegurada es la marca Porsche (según la regla de negocio de la empresa aseguradora de ejemplo).

Estas reglas de negocio son implementadas en jBPM a través de tablas guiadas, las cuales generan archivos drl y pueden ser interpretadas por el motor de jBPM. Por otra parte, Camunda permite implementar reglas de negocio a través del estándar DMN.

4.2.1. jBPM y tablas de decisión

Una de las formas de implementar reglas de negocio en la plataforma de jBPM es la utilización de tablas guiadas, es una forma gráfica de generar archivos drl, los cuales el motor puede interpretar para aplicar una regla de negocio, estos archivos drl pueden generarse manualmente sin necesidad de utilizar tablas guiadas.

Figura 49. **Flujo BPM con tarea de reglas de negocio**



Fuente: elaboración propia, empleando jBPM Workbench.

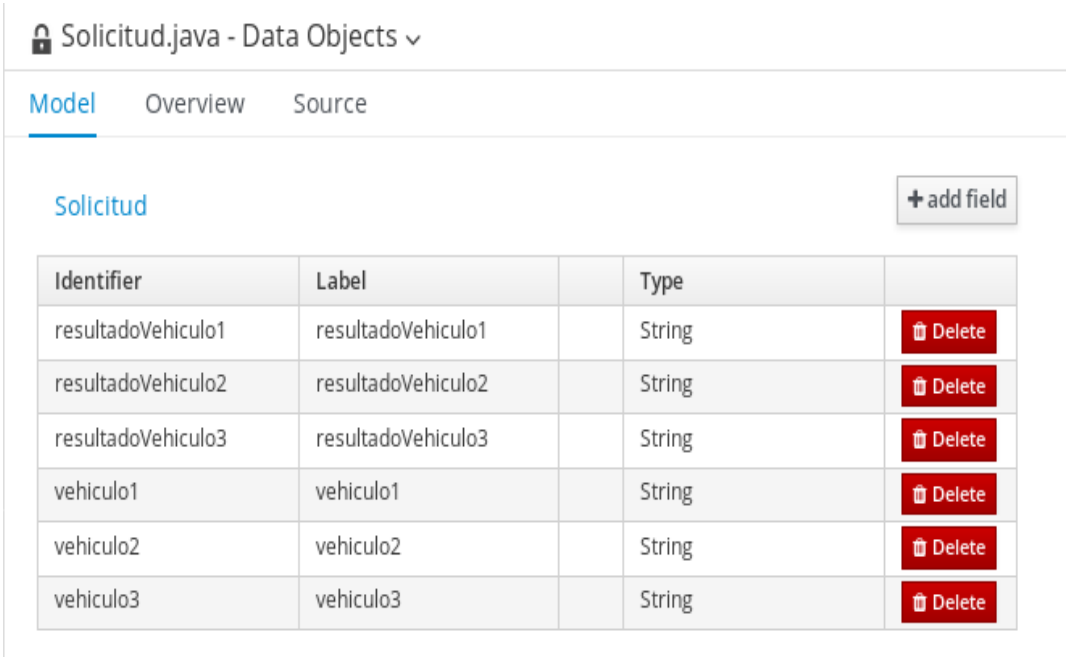
4.2.1.1. Definición de los objetos de datos

Antes de iniciar a definir las reglas dentro de la tabla guiada es importante crear un objeto de datos que sirva como una fuente de datos para definir cada una de las reglas de negocio, por ejemplo, para el caso de estudio se necesita una clase a la cual se le llama “Solicitud” (ver figura 50), la cual contiene las propiedades de los vehículos que serán evaluados y la regla de negocio indica que no se podrán asegurar vehículos de marca: Porsche, Maserati y Ferrari. La clase contiene las siguientes propiedades:

- Resultado vehículo1: propiedad utilizada para escribir el resultado de la evaluación del primer vehículo.
- Resultado vehículo2: propiedad utilizada para escribir el resultado de la evaluación del segundo vehículo.
- Resultado vehículo3: propiedad utilizada para escribir el resultado de la evaluación del tercer vehículo.
- Vehículo 1: propiedad utilizada para describir la marca del primer vehículo.

- Vehículo 2: propiedad utilizada para describir la marca del segundo vehículo.
- Vehículo 3: propiedad utilizada para describir la marca del tercer vehículo.

Figura 50. **Objeto de datos utilizado para la implementación de reglas de negocio**



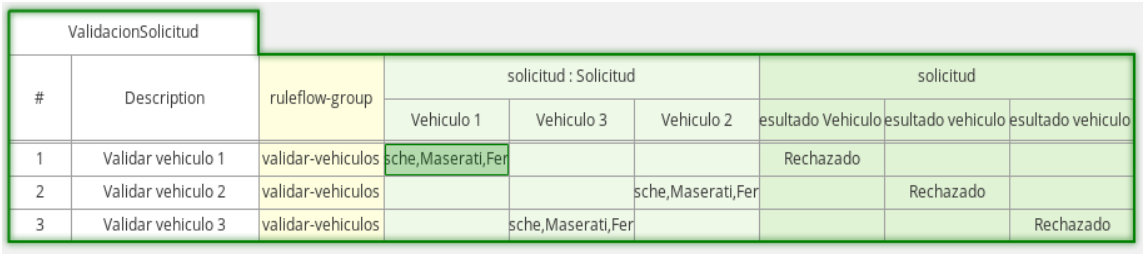
Fuente: elaboración propia, empleando jBPM Workbench.

4.2.1.2. Definición de tablas guiadas

Definido el objeto de datos con las propiedades que pueden conformar una regla de negocio se realizará la tabla guiada (ver figura 51), esta tabla se conforma de varias secciones o columnas (ver figura 52), las más importantes a utilizar para crear reglas de negocio son:

- “Metadata”: sección en la cual se definen propiedades aplicables a las reglas definidas dentro de la tabla, un ejemplo es la propiedad “ruleflow-group”, que permite definir un nombre que será utilizado dentro de la tarea de regla de negocio en el flujo de BPM para ejecutar las reglas que tengan esta propiedad.
- Descripción: campo utilizado para describir la regla definida por el negocio.
- Condiciones: secciones en las cuales se agregan cada una de las condiciones que definen las reglas de negocio, para el caso de ejemplo de una aseguradora de vehículos se define la regla para vehículos de marca Porsche, Maserati y Ferrari, en cuyos casos será rechazada la solicitud (ver figura 53).
- Acciones: esta sección es utilizada para definir el resultado de la evaluación de cada una de las reglas.

Figura 51. **Tabla guiada para la validación de marcas de vehículos**

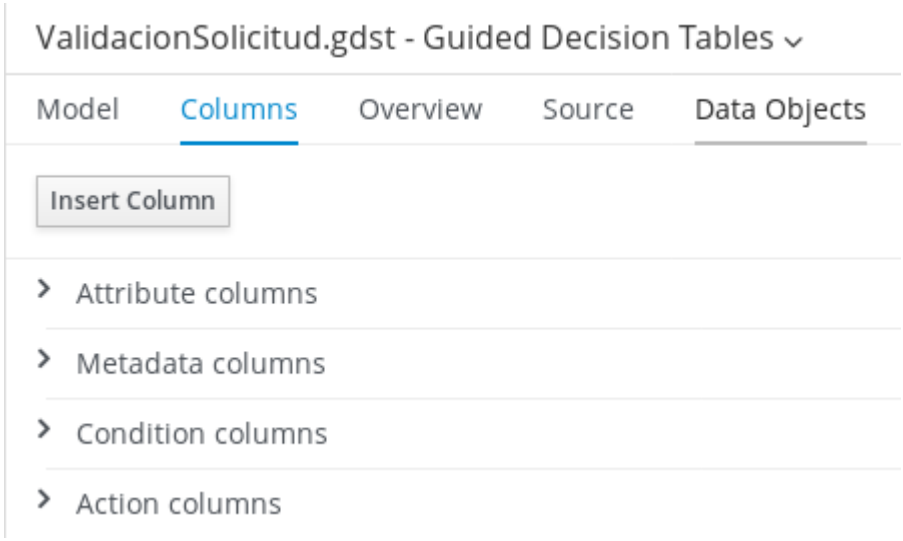


ValidacionSolicitud

#	Description	ruleflow-group	solicitud : Solicitud			solicitud		
			Vehiculo 1	Vehiculo 3	Vehiculo 2	esultado Vehiculo	esultado vehiculo	esultado vehiculo
1	Validar vehiculo 1	validar-vehiculos	sche,Maserati,Fer			Rechazado		
2	Validar vehiculo 2	validar-vehiculos			sche,Maserati,Fer		Rechazado	
3	Validar vehiculo 3	validar-vehiculos		sche,Maserati,Fer				Rechazado

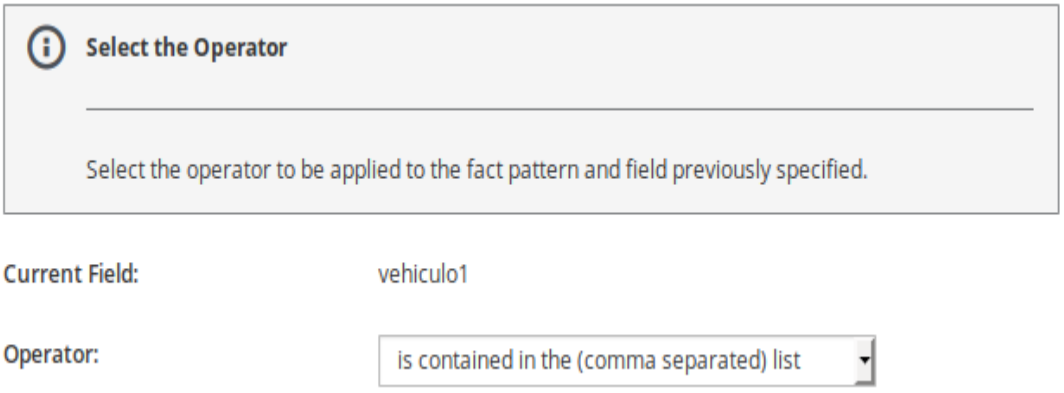
Fuente: elaboración propia, empleando jBPM Workbench.

Figura 52. **Selección de columnas en tabla guiada para la validación de vehículos**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 53. **Definición de condiciones para la validación de reglas de negocio**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 54. **Código MVEL generado por tabla guiada de validación de vehículos**

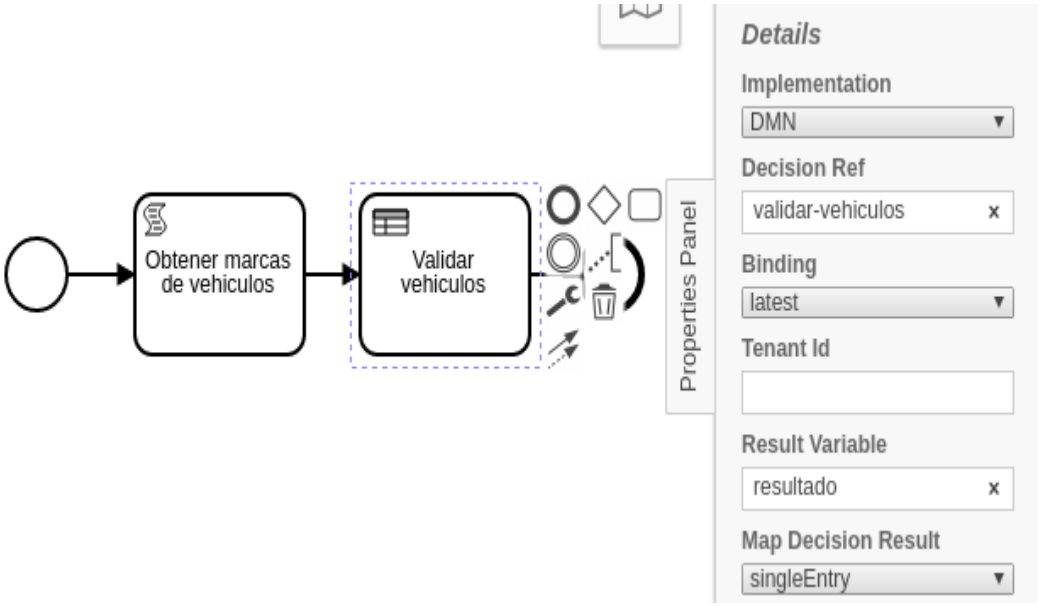
```
1  package com.test.flow;
2
3  //from row number: 1
4  //Validar vehiculo 1
5  rule "Row 1 ValidacionSolicitud"
6      ruleflow-group "validar-vehiculos"
7      dialect "mvel"
8      when
9          solicitud : Solicitud( vehiculo1 in ( "Porsche", "Maserati", "Ferrari" ) )
10     then
11         solicitud.setResultadoVehiculo1( "Rechazado" );
12 end
13
14 //from row number: 2
15 //Validar vehiculo 2
16 rule "Row 2 ValidacionSolicitud"
17     ruleflow-group "validar-vehiculos"
18     dialect "mvel"
19     when
20         solicitud : Solicitud( vehiculo2 in ( "Porsche", "Maserati", "Ferrari" ) )
21     then
22         solicitud.setResultadoVehiculo2( "Rechazado" );
23 end
24
25 //from row number: 3
26 //Validar vehiculo 3
27 rule "Row 3 ValidacionSolicitud"
28     ruleflow-group "validar-vehiculos"
29     dialect "mvel"
30     when
31         solicitud : Solicitud( vehiculo3 in ( "Porsche", "Maserati", "Ferrari" ) )
32     then
33         solicitud.setResultadoVehiculo3( "Rechazado" );
34 end
```

Fuente: elaboración propia, empleando jBPM Workbench.

4.2.2. Camunda y estándar DMN

La única forma de implementar reglas de negocio en la plataforma de Camunda es la utilización del estándar DMN. Como se mencionó en capítulos anteriores, el estándar DMN está conformado por una tabla que define las reglas y un diagrama en el que se representan los objetos a validar y las tablas de reglas de negocio. Posteriormente a esto se puede utilizar la tabla definida para enlazar las reglas al flujo de BPM (ver figura 55).

Figura 55. **Flujo BPM de validación de marcas de vehículo empleando Camunda**



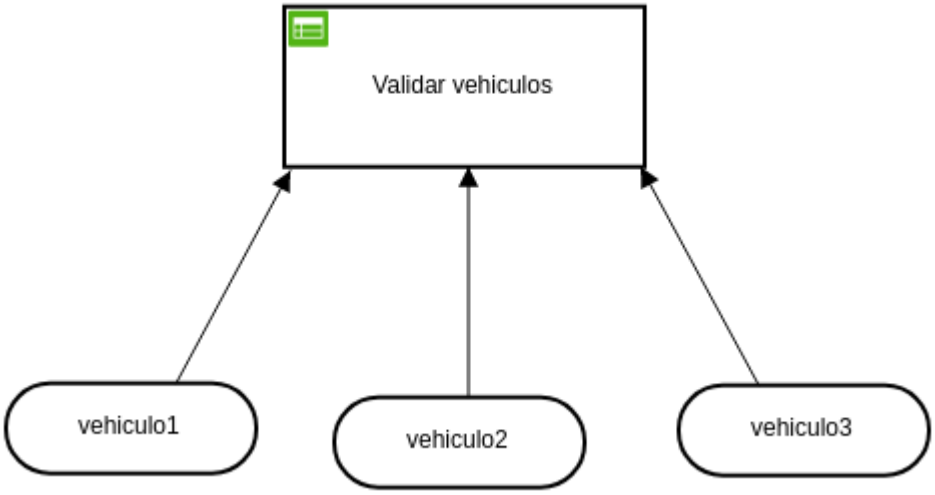
Fuente: elaboración propia, empleando jBPM Workbench.

4.2.2.1. Definición del modelo

A diferencia de jBPM, Camunda no obliga a crear un objeto de datos, pueden datos como variables de tipo cadena o numéricos, dado que el modelo se guía del nombre de la instancia para crear condiciones y definir cada una de las reglas de negocio, por ejemplo, para el caso de estudio, únicamente se necesita crear variables de los vehículos que serán evaluados para aplicar la regla de negocio que indica que no se podrán asegurar vehículos de marca Porsche. El modelo contiene las siguientes fuentes de datos:

- Vehículo 1: propiedad utilizada para describir la marca del primer vehículo.
- Vehículo 2: propiedad utilizada para describir la marca del segundo vehículo.
- Vehículo 3: propiedad utilizada para describir la marca del tercer vehículo.

Figura 56. **Modelo de validación de marcas de vehículos en estándar DMN**



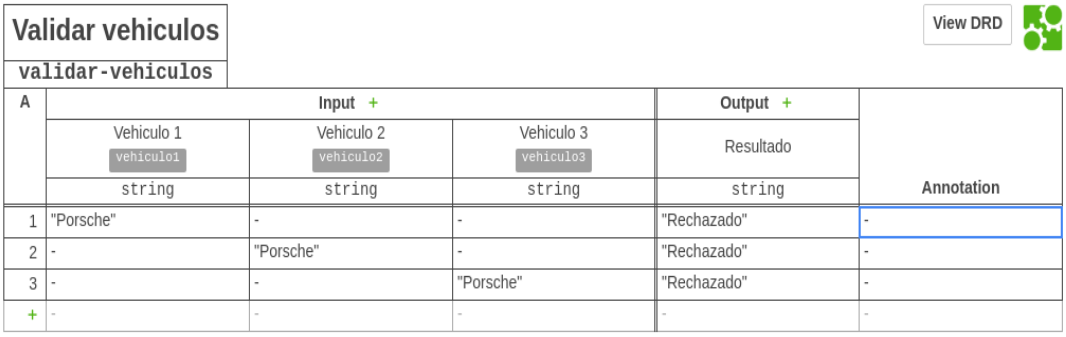
Fuente: elaboración propia, empleando jBPM Workbench.

4.2.2.2. Definición de reglas

Al tener definido el modelo con las propiedades que pueden conformar una regla de negocio se procede a realizar la tabla (ver figura 57), esta está conformada por varias secciones o columnas:

- Entradas (*inputs*): sección en la cual se agrega cada una de las variables de entrada y las condiciones sobre estas que definen las reglas de negocio, para el caso de ejemplo de una aseguradora de vehículos se define la regla para vehículos de marca Porsche, cuyas solicitudes serán rechazadas (ver figura 58).
- Salidas (*outputs)*: sección utilizada para definir el resultado de la evaluación de cada una de las reglas.
- Anotaciones (*annotations*): campo utilizado para describir la regla definida por el negocio.

Figura 57. **Modelo para la validación de marcas de vehículos en estándar DMN**



Validar vehiculos

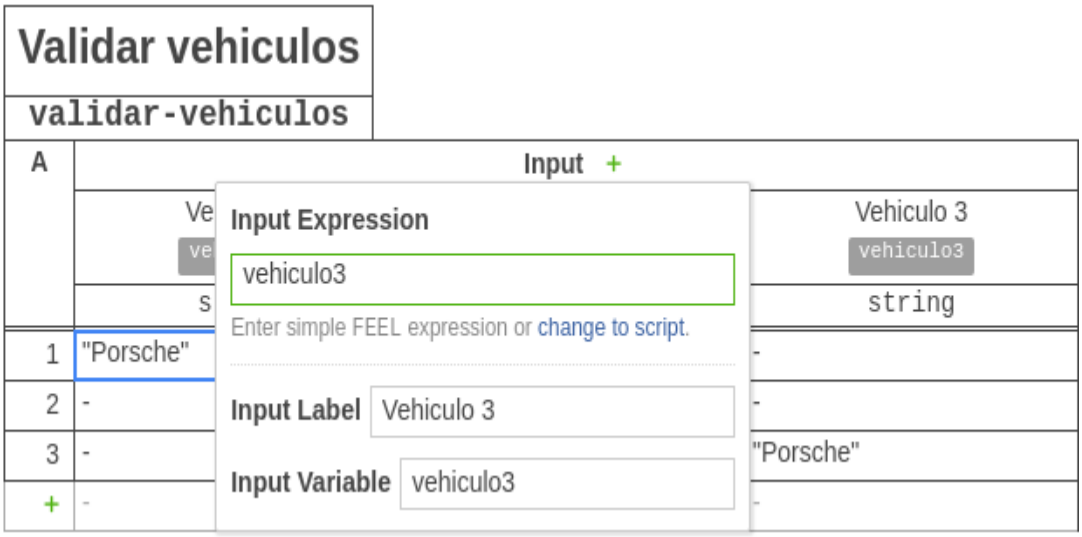
`validar-vehiculos`

View DRD

A	Input +			Output +	
	Vehiculo 1 `vehiculo1`	Vehiculo 2 `vehiculo2`	Vehiculo 3 `vehiculo3`	Resultado	Annotation
	string	string	string	string	
1	"Porsche"	-	-	"Rechazado"	-
2	-	"Porsche"	-	"Rechazado"	-
3	-	-	"Porsche"	"Rechazado"	-
+	-	-	-	-	-

Fuente: elaboración propia, empleando jBPM Workbench.

Figura 58. **Definición de condiciones para variable tipo cadena llamada de vehículo 3**

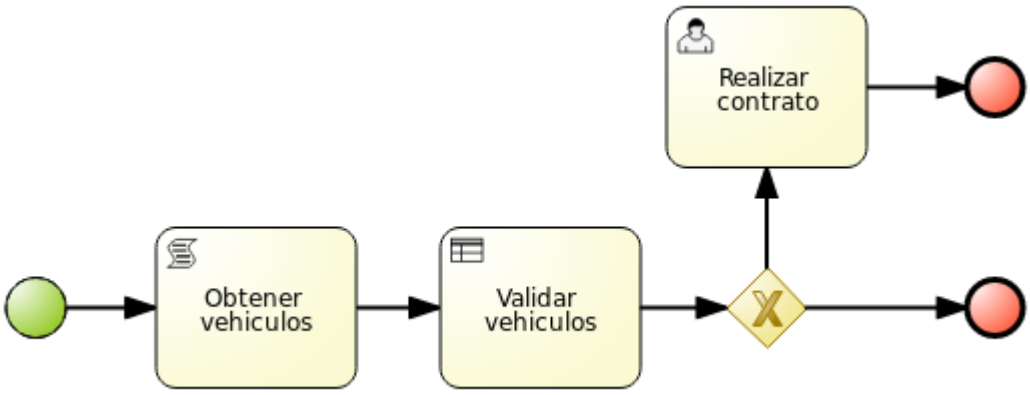


Fuente: elaboración propia, empleando jBPM Workbench.

4.3. Implementación de *gateways* y tareas humanas de BPM en cada herramienta de modelado

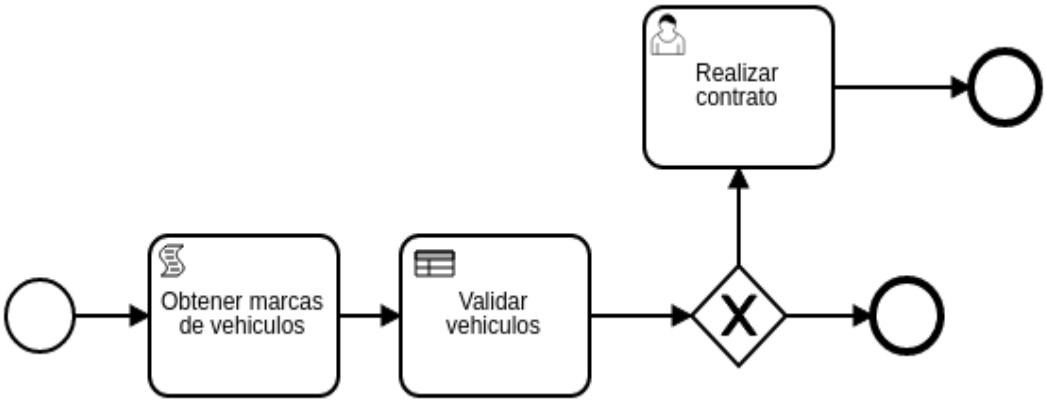
Siguiendo el ejemplo de la aseguradora de automóviles, luego de haber obtenido las marcas de vehículos a ser asegurados a través de una tarea de Script y ejecutar las reglas de negocio para validar las marcas de los vehículos, se procede a utilizar *gateways* para determinar con base en el resultado de la validación qué camino tomar en un flujo de BPM, en el que una solicitud de seguro será rechazada completamente si no cumple con las reglas de negocio satisfactoriamente y, en caso contrario, pasará a un ejecutivo para otorgar el seguro de los vehículos. Estas condiciones de flujo son implementadas a través de *gateways* y las acciones del ejecutivo por tareas humanas.

Figura 59. **Flujo con implementación de *gateway* y tarea humana para el aseguramiento de vehículos**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 60. **Flujo con implementación de *gateway* y tarea humana para el aseguramiento de vehículos**



Fuente: elaboración propia, empleando jBPM Workbench.

4.3.1. **jBPM**

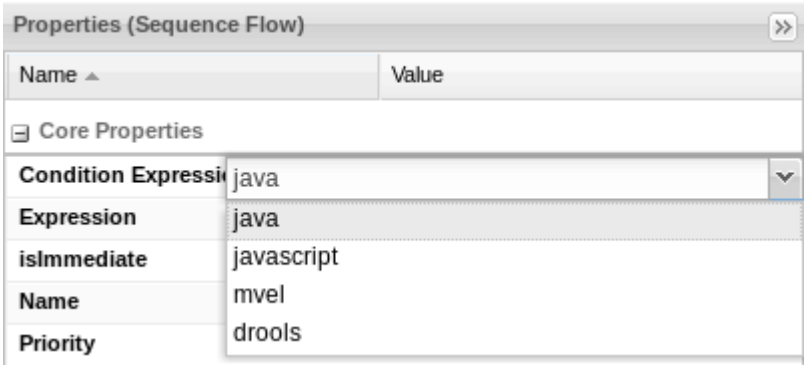
La implementación en la herramienta jBPM para el caso de estudio de la aseguradora consiste en agregar al *gateway* dos caminos, en los que se necesita definir las condiciones que harán que se vaya a un camino o a otro. Las condiciones en jBPM a partir de la versión 7.x se pueden escribir en distintos lenguajes (ver figura 61), para ver estas opciones se debe seleccionar la flecha del camino al que se quiere definir y en la sección de propiedades se podrá ingresar la condición para el camino seleccionado (ver figura 62).

4.3.1.1. Gateways

Para el caso de estudio, si ninguna marca de vehículo fue rechazada, se irá a una tarea humana en la que un operador realizará el contrato respectivo y finalizará el proceso (ver condición en figura 62), en caso de que alguna de las

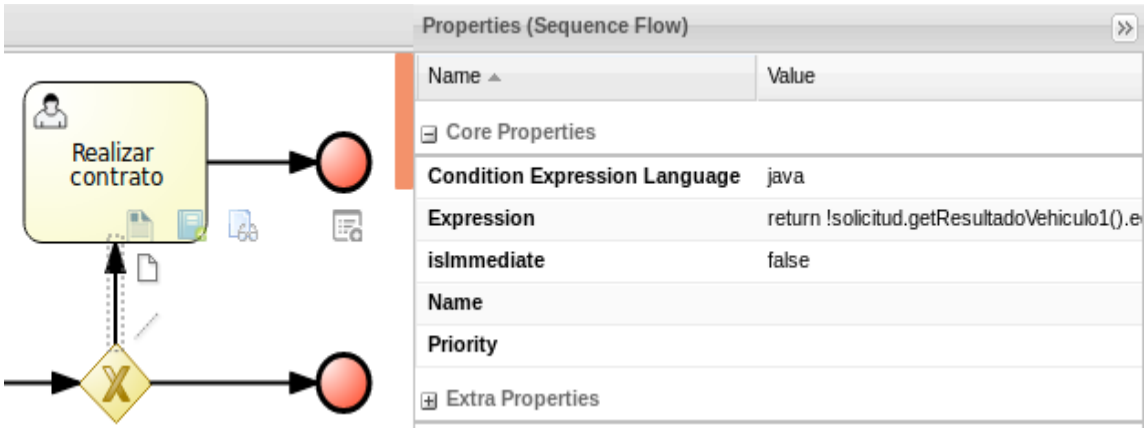
marcas de los vehículos fuera rechazada se rechazará y finalizará la gestión (ver condición en figura 63).

Figura 61. **Selección de lenguaje para condiciones en *gateway***



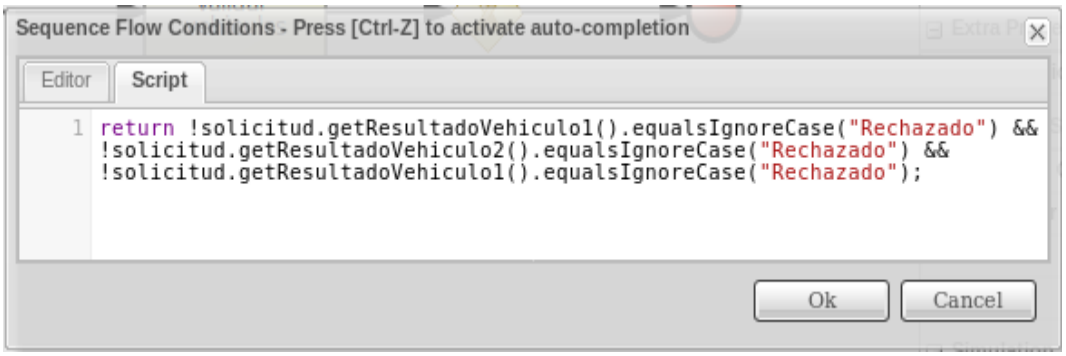
Fuente: elaboración propia, empleando jBPM Workbench.

Figura 62. **Propiedades de uno de los caminos del *gateway* empleando jBPM Workbench**



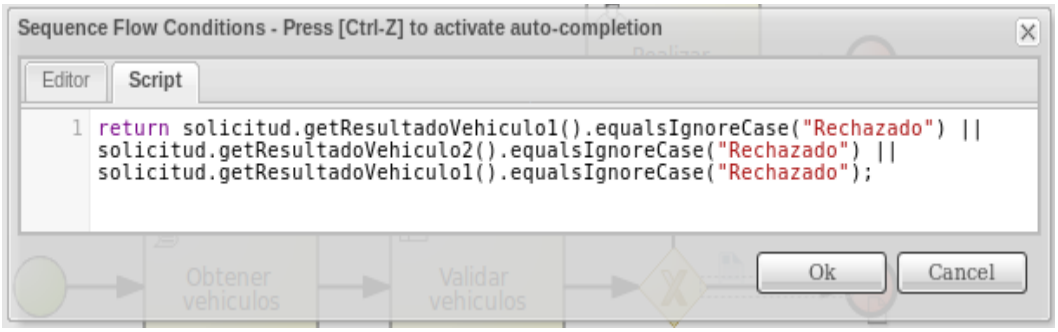
Fuente: elaboración propia, empleando jBPM Workbench.

Figura 63. **Condición para el envío de la gestión a un operador como tarea humana**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 64. **Condición para rechazar y finalizar la gestión por marcas de vehículos no permitidos**

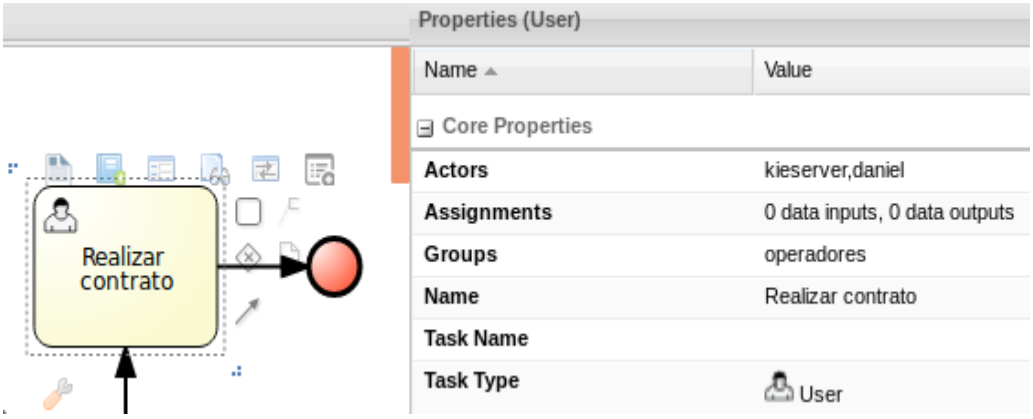


Fuente: elaboración propia, empleando jBPM Workbench.

4.3.1.2. Tareas humanas

La tarea humana para el caso de estudio está dada para un operador que realizará el contrato de aseguramiento de los automóviles ingresados. Como se mencionó en capítulos anteriores, las tareas humanas pueden ser asignadas a personas específicas (autores) o a un conjunto de personas que pertenecen a un rol específico (grupos) (ver figura 65).

Figura 65. **Tarea humana de operador para crear contrato de seguro, asignando usuarios y a un grupo**



Fuente: elaboración propia, empleando jBPM Workbench

4.3.2. Camunda

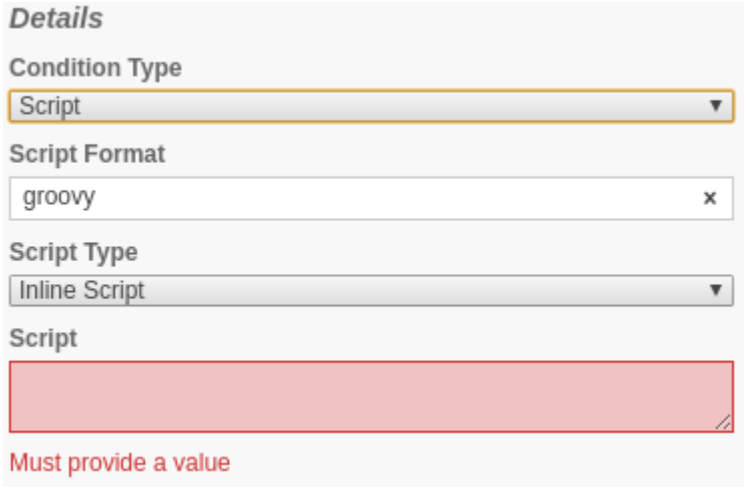
La implementación en Camunda para el caso de estudio de la aseguradora consiste en agregar al *gateway* dos caminos en los que se necesita definir las condiciones que harán que se vaya a un camino o a otro. Las líneas de los *gateways* en Camunda permiten definir condiciones a través de expresiones utilizando las variables de procesos, comparadores y *scripts* en distintos lenguajes (ver figura 66). Para ver estas opciones se debe seleccionar la flecha del camino que se quiere definir y en la sección de propiedades se podrá ingresar la condición para el camino seleccionado (ver figura 67).

4.3.2.1. Gateways

Para el caso de estudio, si ninguna marca de vehículo fue rechazada, se irá a una tarea humana en la que un operador realizará el contrato respectivo y finalizará el proceso (ver condición en figura 68). En caso de que alguna de las

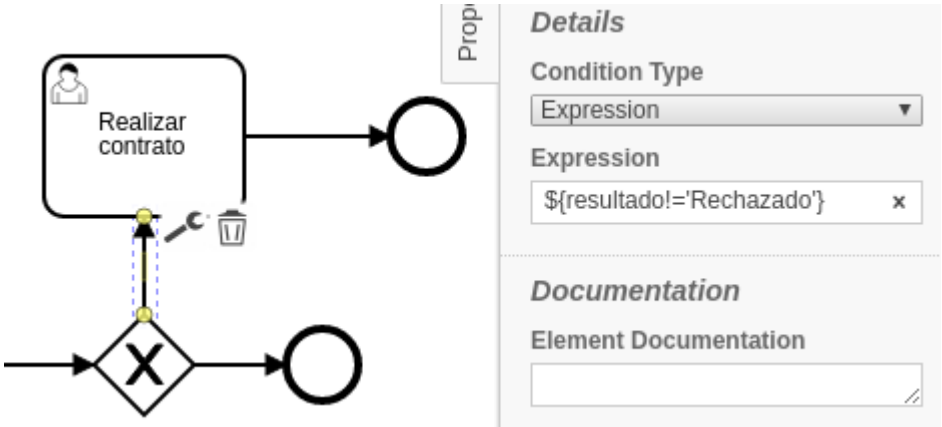
marcas de los vehículos fuera rechazada, ser rechazará y finalizará la gestión (ver condición en figura 69). Para las dos condiciones se utilizaron expresiones.

Figura 66. **Selección de lenguaje de *script* para definir condición en flujos de un *gateway* Camunda**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 67. **Propiedades de uno de los caminos de un gateway**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 68. **Condición para el envío de la gestión a un operador como tarea humana Camunda**

Fuente: elaboración propia, empleando jBPM Workbench

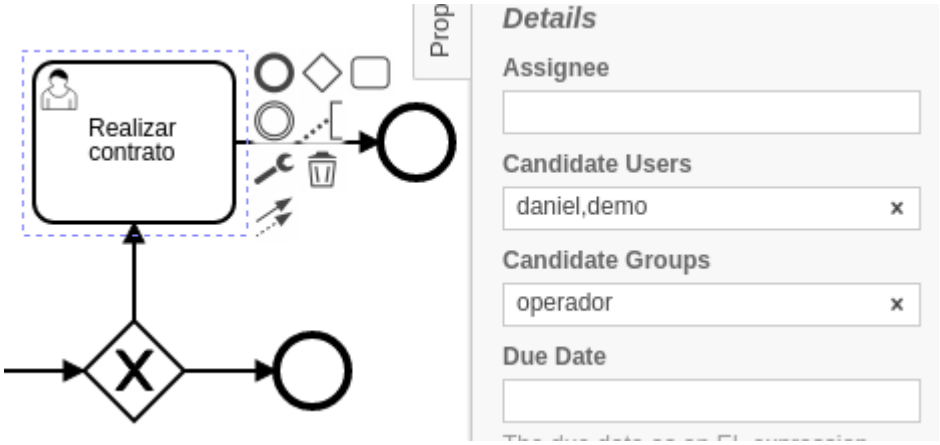
Figura 69. **Condición para rechazar y finalizar la gestión por marcas de vehículos no permitidos**

Fuente: elaboración propia, empleando jBPM Workbench.

4.3.2.2. Tareas humanas

La tarea humana para el caso de estudio está dada para un operador que realizará el contrato de aseguramiento de los automóviles ingresados. Como se mencionó en capítulos anteriores, las tareas humanas pueden ser asignadas a personas específicas (autores) o a un conjunto de personas que pertenecen a un rol específico (grupos) (ver figura 70). A diferencia de jBPM, las tareas humanas tienen propiedades adicionales en las que se puede definir un tiempo límite de la tarea y asignar una prioridad.

Figura 70. **Tarea humana de operador para crear el contrato de seguro, asignando usuarios y un grupo**



Fuente: elaboración propia, empleando jBPM Workbench.

4.4. Implementación de simulaciones de flujos de procesos

La simulación de flujos de proceso empresariales es un tema importante si se quiere obtener una estadística del funcionamiento del proceso implementado y/o visualizar los flujos por los que se pueden pasar dentro de un proceso determinado.

4.4.1. jBPM

jBPM Workbench proporciona en su modelador una opción de simulación. Esta simulación consiste en que, a medida que se va desarrollando el flujo a cada actividad, se le asigne valores de simulación (ver figura 71).

4.4.1.1. Configuración

Las propiedades de simulación en la herramienta de jBPM son diversas, algunas de estas pueden ser:

- De costos por unidad de tiempo: propiedad que indica el costo que tiene la ejecución de la tarea por la unidad de tiempo configurada, por ejemplo, si la tarea humana de “Realizar Contrato” es externalizada por otra empresa y cobran el tiempo por hora en que se tardan en realizar un contrato, este costo es el que debe colocarse para tener un aproximado, al final de la simulación, de lo que podría incurrir en costos la empresa al ejecutar cierto número de trámites. Para el caso de estudio se dio un costo de 50 quetzales por hora.
- Tiempo máximo de ejecución: tiempo máximo por unidad de tiempo que se cree se tarde en ejecutar la tarea a la cual se le está configurando la propiedad de simulación. Para el caso de estudio se colocó 10 minutos.
- Tiempo mínimo de ejecución: tiempo mínimo por unidad de tiempo que se cree se tarde en ejecutar la tarea a la cual se le está configurando la propiedad de simulación. Para el caso de estudio se colocó 5 minutos.
- Disponibilidad de personal: configuración propia de tareas humanas, representa un porcentaje de la disponibilidad del operador para ejecutar la tarea, por ejemplo, asumiendo que aparte de realizar contratos a los que dedica un 50 % de su tiempo, el otro 50 % lo dedica para archivar papelería. Para el caso de estudio se ha configurado al 100 %, indicando que el operador estará el 100 % del tiempo completo dedicado a realizar contratos.
- Horas de trabajo: configuración propia de tareas humanas que representa el tiempo de trabajo por día de un operador. Para el caso de estudio se ha configurado en 8 horas por día.

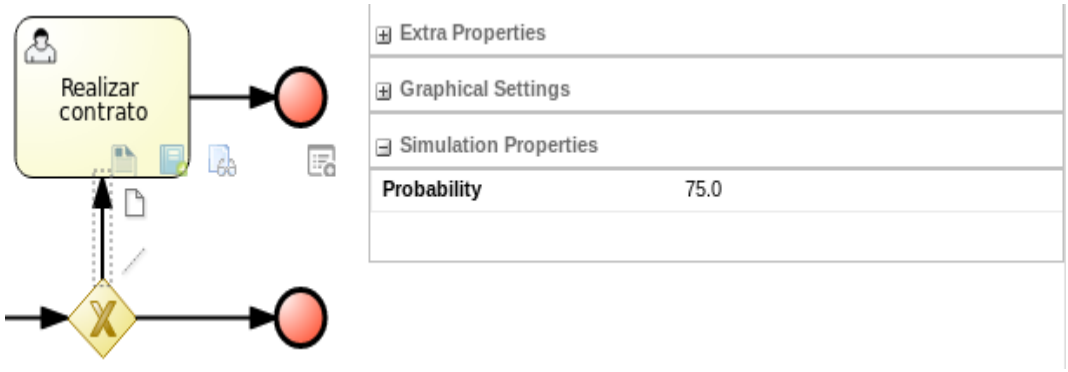
Figura 71. **Configuración de propiedades de simulación para una actividad humana**

Simulation Properties	
Cost per time unit	50.0
Distribution Type	uniform
Processing time (max)	10.0
Processing time (min)	5.0
Staff availability	1.0
Working Hours	8.0

Fuente: elaboración propia, empleando jBPM Workbench.

Para las flechas de flujo que salen de un *gateway* cambian las propiedades de configuración de simulación, únicamente tienen una propiedad y es la de "Probabilidad" (ver figura 72), que es un porcentaje de ocurrencia en el que se supone que sigue un camino del flujo determinado y en que la suma de todos estos porcentajes dados por cada línea que sale del *gateway* debe dar un total de 100 %. Para el caso de estudio se tiene que un 75 % de los casos son aprobados y se realiza un contrato y un 25 % de las solicitudes son rechazadas.

Figura 72. **Configuración de propiedades de simulación para una flecha de flujo**

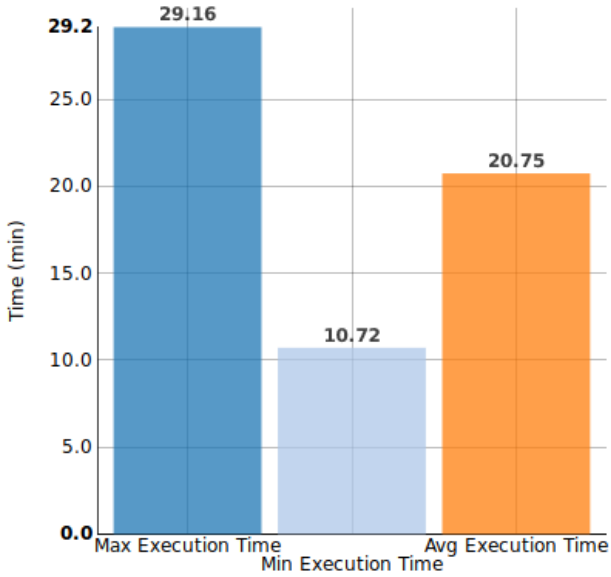


Fuente: elaboración propia, empleando jBPM Workbench.

4.4.1.2. Resultados

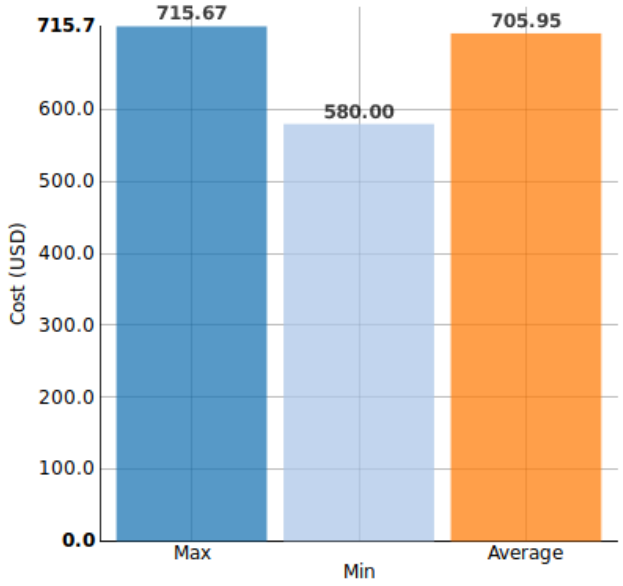
Los resultados de la simulación se pueden observar en la pestaña de resultados de simulación, algunos son:

Figura 73. **Gráfica de tiempo de ejecución para 100 trámites procesados**



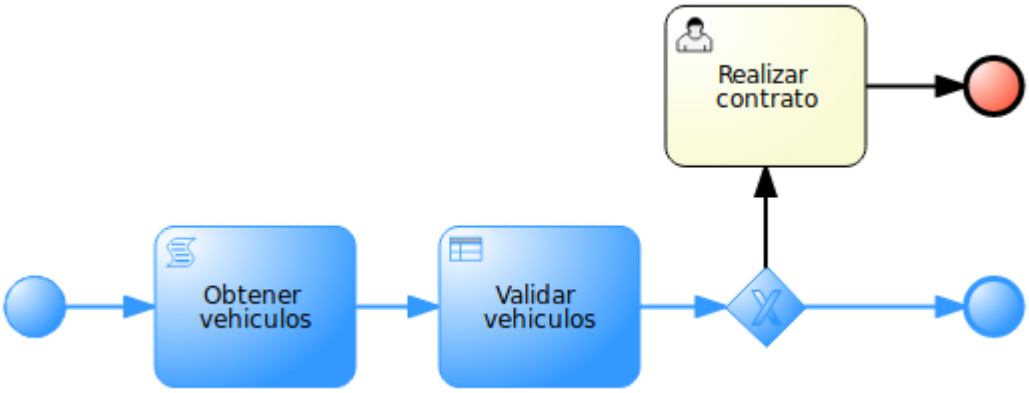
Fuente: elaboración propia, empleando jBPM Workbench.

Figura 74. **Gráfica de costos de ejecución para 100 trámites procesados**



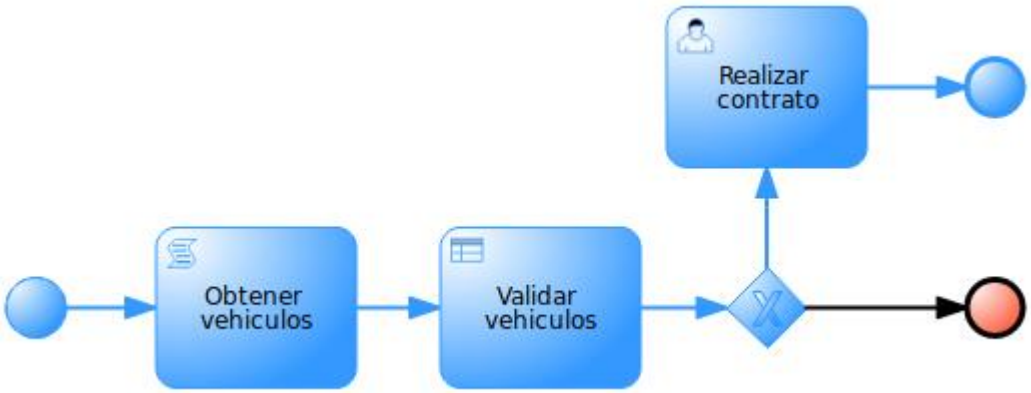
Fuente: elaboración propia, empleando jBPM Workbench

Figura 75. **Camino con 25 % de probabilidad para las solicitudes de seguro**



Fuente: elaboración propia, empleando jBPM Workbench.

Figura 76. **Camino con 75 % de probabilidad para las solicitudes de seguro**



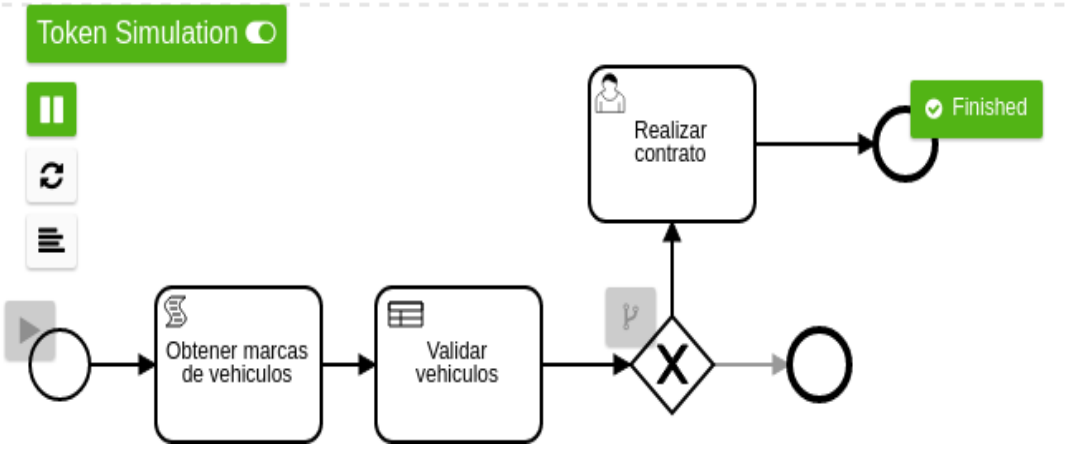
Fuente: elaboración propia, empleando jBPM Workbench.

4.3.1 Camunda

La simulación de procesos en Camunda es bastante gráfica y únicamente permite ver los caminos que un flujo de BPM puede tomar como las últimas figuras de los pasos según los porcentajes dados en el *gateway*, solo que de una forma dinámica. Esto se logra utilizando el *plugin*[31] para simulación de procesos y automáticamente aparece la opción de simulación en el modelador de Camunda (ver figura 77).

[31] Simulación de plugin. https://github.com/philippfromme/bpmn-js-token-simulation-plugin. Consulta: 11 de junio de 2018.

Figura 77. **Simulación del flujo de procesos para asegurar autos en el modelador de Camunda**



Fuente: elaboración propia, empleando jBPM Workbench.

5. COMPARACIÓN DE LAS HERRAMIENTAS: JBPM Y CAMUNDA

5.1. Comparación según el lenguaje de implementación

Según los lenguajes que implementa cada herramienta se puede estimar la conveniencia de utilizar una herramienta de la otra.

5.1.1. Comparación por lenguaje

Dado el lenguaje utilizado por cada una de las herramientas modeladoras se obtiene el siguiente cuadro:

Tabla I. **Comparación de lenguajes permitidos por herramienta de diseño**

	Java	**Javascript**	**Groovy**
jBPM	x	x	-
Camunda	-	x	x

Fuente: elaboración propia.

5.1.2. Comparación según su usabilidad

Al momento de implementar el código dentro de las herramientas es importante destacar las usabilidades en cada una de las mismas, las cuales se detallan en la siguiente tabla:

Tabla II. **Comparación de usabilidad por herramienta de diseño**

	Auto-completado	**Color Formato**	**Externa-lización de recursos**	**Intellisense**	**Validación de sintaxis**
jBPM	-	x	-	-	x
Camunda	-	-	x	-	-

Fuente: elaboración propia.

5.1.3. **Comparación según su rendimiento**

Para calcular el rendimiento se utilizó la herramienta Apache JMeter[32], la cual permite realizar pruebas de rendimiento utilizando las URL de ejecución para crear procesos de cada una de las plataformas jBPM y Camunda. Por cada implementación BPM se realizó una carga de 2 000 solicitudes, con los resultados obtenidos[33] se obtienen las siguientes tablas y gráficas.

Con base en los datos obtenidos se calcula el promedio y la media dadas por las respuestas en milisegundos de cada una de las solicitudes que se han realizado, como límite inferior se tiene el tiempo mínimo en responder una solicitud y como límite superior se tiene el tiempo máximo en que se tardó una de las solicitudes en ser respondida. Con base en esta información se tiene un resultado de rendimiento de solicitudes por segundo (ver tabla 3).

[32] Sobre Apache JMeter. https://jmeter.apache.org/. Consulta: 15 de junio de 2018.

[33] Rendimiento de JMeter. https://github.com/danialvarez2005/tesis-jmeter-rendimiento.git. Consulta: 15 de junio de 2018.

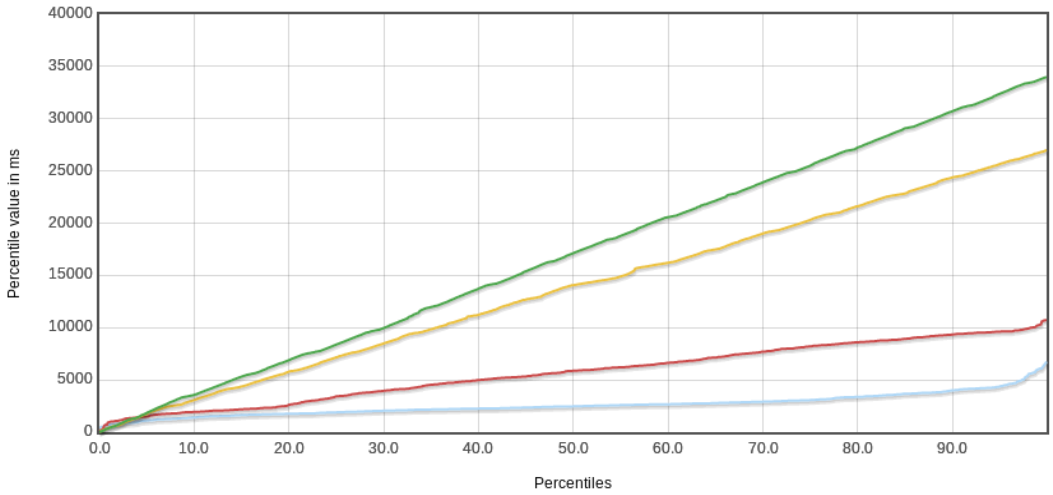
Tabla III. **Comparación de rendimiento por lenguaje y herramienta de diseño.**

	Solicitu-des	Promedio	Media	T. Mínimo	T. Máximo	Rendimiento
Camunda Groovy	2000	2589	2467	422	6747	188.44 s/s
Camunda Javascript	2000	5707	5872	124	10726	148.41 s/s
jBPM Java	2000	13732	14065	45	26992	68.98 s/s
jBPM Javascript	2000	17081	17133	45	33991	55.86 s/s

Rendimiento dado por solicitudes por segundo (s/s)

Fuente: elaboración propia.

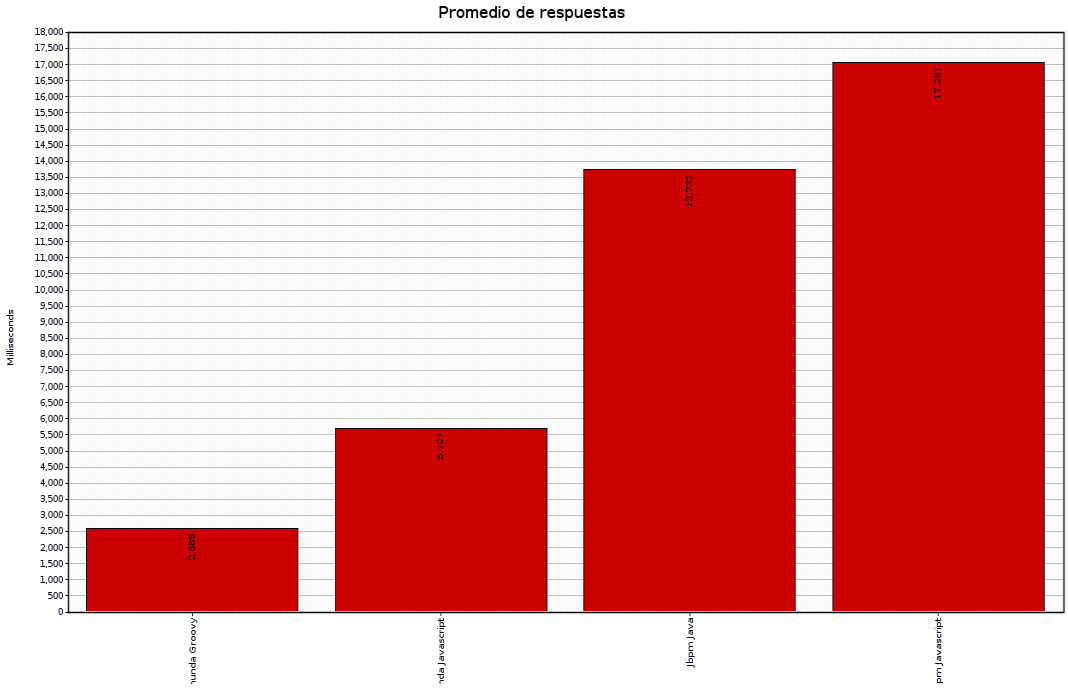
Figura 78. **Comparación de tiempos de respuesta por percentiles**



Camunda Groovy (celeste), Camunda Javascript (rojo), jBPM Java (amarillo) y jBPM Javascript (verde).

Fuente: elaboración propia, empleando la herramienta Apache JMeter.

Figura 79. **Comparación de tiempos de respuesta**



En orden de izquierda a derecha: Camunda Groovy, Camunda Javascript, jBPM Java y jBPM Javascript.

Fuente: elaboración propia con ayuda de la herramienta Apache JMeter.

5.2. Comparación según la implementación de reglas de negocio en cada herramienta

Según la implementación de flujos de procesos de negocio empresarial en cada herramienta se pueden determinar las características importantes de cada una a tomar en cuenta al momento de decidir que herramienta utilizar, por lo que se la compara en base a reglas de negocio, lenguaje de programación, usabilidad y rendimiento.

5.2.1. Reglas de negocio

Dada la implementación de reglas de negocio utilizadas por cada una de las herramientas modeladoras se tiene el siguiente cuadro:

Tabla IV. **Comparación de implementación de reglas de negocio por herramienta de diseño**

	Tablas guiadas	**DMN**
jBPM	x	-
Camunda	-	x

Fuente: elaboración propia.

5.2.2. Lenguajes para la implementación de condiciones

Dado el lenguaje que puede ser utilizado para la definición de condiciones por cada una de las herramientas modeladoras se tiene el siguiente cuadro:

Tabla V. **Comparación de lenguajes permitidos por herramienta de diseño**

	MVEL	**Java**	**Javascript**	**Groovy**	**Python**
jBPM	x	x	-	-	-
Camunda	-	-	x	x	x

Fuente: elaboración propia.

5.2.3. **Usabilidad**

Al momento de implementar las reglas de negocio dentro de las herramientas es importante destacar las usabilidades en cada una de las herramientas, las cuales se detallan en la siguiente tabla:

Tabla VI. **Comparación de usabilidad por herramienta de diseño**

	Asistente de reglas	**Validación de reglas**
jBPM	x	x
Camunda	-	-

El asistente de reglas es la ayuda que se tiene para crear nuevas reglas de negocio y la validación de reglas es la utilidad que valida si una regla está bien formada y no hace conflicto con otras.

Fuente: elaboración propia.

5.2.4. **Rendimiento**

Para calcular el rendimiento se utilizó la herramienta Apache JMeter[34], la cual permite realizar pruebas de rendimiento utilizando las URL de ejecución para crear procesos de cada una de las plataformas jBPM y Camunda. Por cada implementación BPM se realizó una carga de 1 000 solicitudes, con los resultados obtenidos[35] se obtienen las siguientes tablas y gráficas:

[34] Sobre Apache JMeter. https://jmeter.apache.org/. Consulta: 15 de junio de 2018.

[35] Rendimiento de JMeter. https://github.com/danialvarez2005/tesis-jmeter-rendimiento.git. Consulta: 2018.

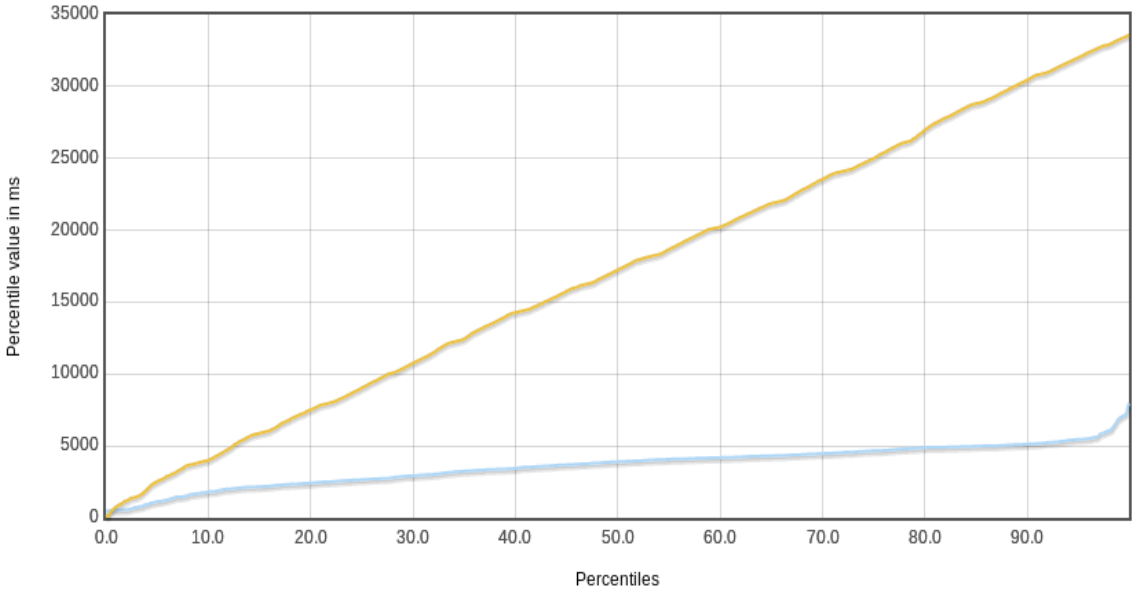
Tabla VII. **Comparación de rendimiento por ejecución de reglas de negocio y herramienta de diseño**

	Solicitu-des	Promedio	Media	T. Minímo	T. Máximo	Rendimiento
Camunda	1000	3659	3903	440	7972	89.00 s/s
jBPM	1000	17209	17261	74	33572	28.00 s/s

Rendimiento dado por solicitudes por segundo (s/s).

Fuente: elaboración propia.

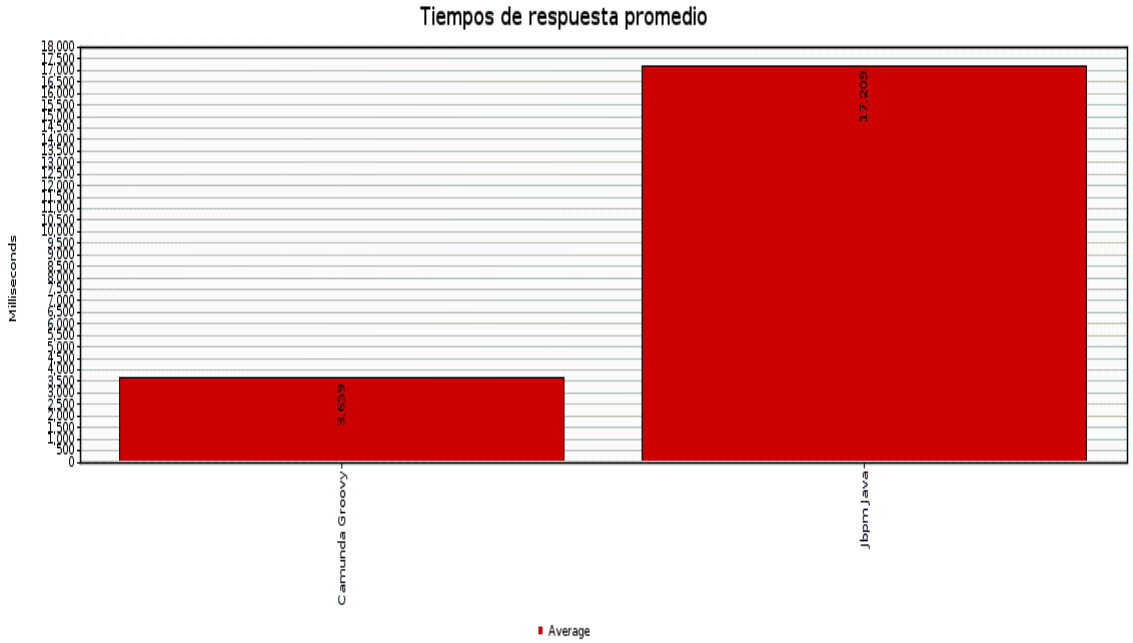
Figura 80. **Comparación de tiempos de respuesta por percentiles**



Camunda (celeste), jBPM (amarillo).

Fuente: elaboración propia con ayuda de la herramienta Apache JMeter.

Figura 81. **Comparación de tiempos de respuesta**



En orden de izquierda a derecha: Camunda, jBPM Java.

Fuente: elaboración propia con ayuda de la herramienta Apache JMeter.

5.3. Comparación según el editor de modelos BPM de cada herramienta

Los editores de cada una de las herramientas de modelado tienen sus propias características, por lo que se procede a comparar según sus características más sobresalientes.

5.3.1. Definición de objetos

El jBPM Workbench permite crear muchos más objetos en su modelador, como se puede ver a continuación:

- Estándares
  - Flujos BPM

    - Diagramas CMMN
- Objetos de decisión
    - Archivos de reglas (.drl)
    - Tablas guiadas
    - Reglas guiadas
    - Tablas de decisión por archivos Excel
- Modelo de datos
    - Definir clases
    - Definir enumeradores

A diferencia del modelador de Camunda, permite crear los siguientes objetos en su plataforma:

- Estándares
    - Flujos BPM
    - Tablas y diagramas DMN
    - Diagramas CMMN
- Objetos de decisión
    - Tablas y diagramas DMN

5.3.2. **Usabilidad**

Una de las características más importantes de los modeladores es su usabilidad, por lo que se procede a comparar esta característica por cada una de las herramientas.

Tabla VIII. **Comparación de usabilidad por herramienta de diseño**

	Repositorio de fuentes	**Despliegue**	**Validador de objetos**
jBPM Workbench	x	x	x
Modelador Camunda	-	x	-

Fuente: elaboración propia.

5.3.3. **Paleta de objetos en diagramas BPM**

El jBPM Workbench ofrece muchas más actividades para agregar a un flujo de BPM, como se puede ver a continuación:

- Tareas:
  - Humanas
  - Reglas de negocio
  - Envío de mensajes
  - Recepción de mensajes
  - De servicio
  - De *script*
- Subprocesos
  - Reutilizable
  - Múltiples instancias
  - Incrustados
  - Procesos *ad-hoc*
  - Por eventos
- Captura de eventos intermedios

    - Mensajes
    - Temporizadores
    - Escalamiento
    - Señales
    - Excepciones
- Lanzamiento de eventos intermedios
    - Mensajes
    - Escalamiento
    - Señales
- Otros
    - *Swimlanes*

Mientras que Camunda únicamente ofrece las siguientes actividades:

- Tareas:
    - Humanas
    - Reglas de negocio
    - Envío de mensajes
    - Recepción de mensajes
    - De servicio
    - De *script*
- Subprocesos
    - Reutilizables
    - Expandidos
- Otros
    - *Pools*

5.4. Comparación según la simulación de los flujos de procesos

Con base en la simulación en cada uno de los modeladores se obtuvo la siguiente tabla comparativa:

Tabla IX. **Comparación de simulación de flujos de proceso por herramienta de diseño**

	Estimación de caminos	**Estimación de costos**	**Estimación de tiempos de procesamiento**
jBPM Workbench	x	x	x
Modelador Camunda	x	-	-

Fuente: elaboración propia.

CONCLUSIONES

1. Camunda, aunque es muy joven en el mundo de las herramientas de BPM, tiene un motor bastante optimizado, como se pudo notar en las pruebas de rendimiento con tiempos de respuesta menores a los de jBPM.

2. La curva de aprendizaje para la utilización de jBPM es mucho más grande que la de Camunda, debido a que se necesitan muchas más configuraciones que Camunda para empezar a operar.

3. jBPM ofrece muchas más herramientas para ayudar al programador a realizar de mejor manera un flujo de BPM, uno de los puntos a destacar es la validación de los objetos que se crean para detectar errores desde la definición, a diferencia de Camunda, en que se pueden visualizar los errores hasta que se ejecuta el flujo diseñado.

4. Camunda utiliza e implementa únicamente estándares, a diferencia de jBPM, que utiliza e implementa algunos estándares, los extiende y agrega más funcionalidades.

5. De los lenguajes para *script* que se pueden utilizar, los más optimizados son Groovy para Camunda y Java para jBPM, como se pudo demostrar en la tabla de resultados.

6. La simulación de procesos de BPM en el modelador de Camunda es bastante limitado, a diferencia de jBPM, que además de calcular mucha más información presenta un informe detallado con gráficas.

RECOMENDACIONES

1. Si no se cuenta con mucho tiempo para el aprendizaje de una herramienta y no se piensa escribir flujos muy complejos, Camunda es una buena opción para comenzar a levantar flujos de procesos de negocio.

2. Antes de empezar a trabajar con una plataforma determinada se debe evaluar detenidamente todas sus características, además de las que el trabajo menciona, con el objetivo de reducir el tiempo de desarrollo y obtener un mejor desempeño que satisfaga las necesidades del negocio.

3. Aunque el rendimiento en los tiempos de respuesta de Camunda es mucho mejor, si el flujo que se desea desarrollar no es altamente transaccional, en donde existen muchos hilos de un mismo flujo ejecutándose a la vez, este no es un factor importante para no elegir utilizar jBPM.

4. Considerar el costo de las licencias si en algún momento se desea tener un soporte profesional con cada una de las versiones de la plataforma.

BIBLIOGRAFÍA

1. *Business Process Model and Notation (BPMN). Version 2.0.* [en línea]. <http://www.omg.org/spec/BPMN/2.0/>. [Consulta: mayo de 2018].

2. *Camunda. 7.8 User Guide.* [en línea]. <https://docs.camunda.org/manual/7.8/user-guide//>. [Consulta: 9 de marzo de 2018].

3. *Decision Model and Notation (DMN). V1.1.* [en línea]. <http://www.omg.org/spec/DMN/1.1/>. [Consulta: mayo de 2018].

4. DE MAIO, Mariano Nicolás; SALANTINO, Mauricio; ALIVERTI, Esteban. *jBPM6 Developer Guide.* Estados Unidos: Packt Publishing Ltd.,2014. 310p.

5. *Documentación Groovy.* [en línea] <http://groovy-lang.org/documentation.html/>. [Consulta: 12 de junio de 2018].

6. *Documentación MVEL.* [en línea] <http://mvel.documentnode.com/>. [Consulta: 10 de junio de 2018].

7. *Documentación Wildfly.* [en línea] <https://docs.jboss.org/author/display/WFLY10/Documentation/>. [Consulta: 15 de junio de 2018].

8. *Drools. 7.0.6 Final User Guide.* [en línea]. <https://docs.jboss.org/drools/release/7.6.0.Final/drools-docs/html_single/index.html/>. [Consulta: 9 de marzo de 2018].

9. *jBPM. 7.0.6 Final User Guide.* [en línea]. <https://docs.jboss.org/jbpm/release/7.6.0.Final/jbpm-docs/html_single/>. [Consulta: 9 de marzo de 2018].

10. *Manual Apache JMeter.* [en línea]. <https://jmeter.apache.org/usermanual/index.html/>. [Consulta: 8 de junio de 2018].

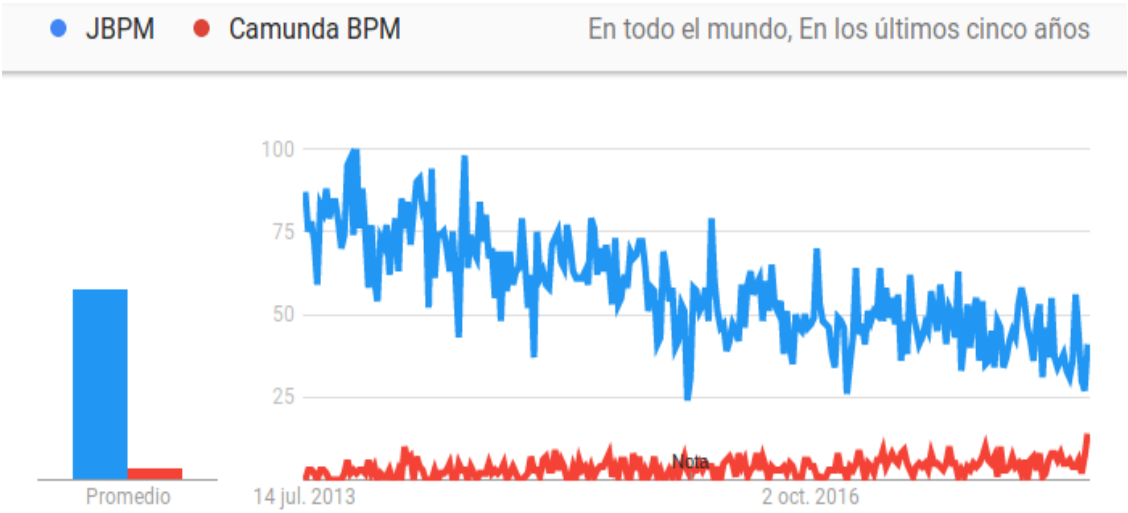
11. *Manual Apache Maven.* [en línea]. <https://maven.apache.org/guides/introduction/introduction-to-profiles.html/>. [Consulta: 10 de junio de 2018].

12. SILVER, Bruce. *Método y estilo BPMN*. 2da ed. Estados Unidos: Cody-Cassidy Press, 2015. 306 p.

13. SIMONE FIORINI, Arun V. Gopalakrishnan. *Mastering jBPM6*. 1a ed. Reino Unido:Packt Publishing Ltd.,2015.297 p.

ANEXOS

Anexo 1**. Tendencias en el mundo en el uso de jBPM y Camunda**



Fuente: Google Trends. trends.google.com. Consulta: julio de 2018.