UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Una propuesta arquitectónica para integrar una herramienta BPMS y un sistema de gestión de reglas de negocio.
Autores: Julián Matías Parra
Director: Patricia Bazán
Asesor profesional: José Martinez Garro
Carrera: Licenciatura en Sistemas

Resumen

En el presente trabajo se busca identificar las bases esenciales en las que se apoya la integración entre un sistema gestor de procesos de negocio y uno de reglas con el fin de englobarlas en una propuesta arquitectónica que sirva de modelo a la hora de implementar la comunicación entre las herramientas. Para lograr el propósito planteado partiremos de la definición básica de los conceptos que involucran a ambos sistemas. Se analizarán distintos casos reales del mercado en los que se ha trabajado con ambas tecnologías en forma conjunta con el fin de observar elementos en común y mecanismos involucrados. Se resumirán las particularidades encontradas bajo el marco de una propuesta metodológica, utilizando dos herramientas de software libre para realizar la validación del modelo, esto implicará la implementación por etapas del proceso de integración y se empleará un caso de estudio real para verificar la viabilidad de la propuesta definida.

Palabras Claves

Proceso - Regla de negocio – BPMS - BRMS
Repositorio – LDAP – ORM - API REST - Web Service

Conclusiones

Se obtiene un modelo arquitectónico para la integración de herramientas BPMS y BRMS, identificando la existencia de aspectos clave que infieren en ambos componentes y que intervienen en el proceso de conexión. La propuesta realizada es validada a través de la implementación de un caso de estudio sobre los productos de código abierto BonitaOS y Drools.

Trabajos Realizados

Planteo de una propuesta arquitectónica para integrar un BRMS con un BPMS.

Desarrollo de la componente de integración que interactúe con los procesos y sus reglas.

Aplicación de la solución a un caso de estudio

Trabajos Futuros

Podemos pensar en incorporar la gestión de roles de usuario para el repositorio central, agregando distintos niveles de responsabilidad al modelo integrador.
Se pueden considerar otros mecanismos de comunicación entre las herramientas como RPC, RMI, Agentes JADE, entre otros.
La interfaz web del módulo adaptador construido puede extenderse abordando un mayor nivel de configuración, permitiendo definir información de interés sobre procesos y/o reglas.

Fecha de la presentación: Marzo 2015

Facultad de Informática
Universidad Nacional de La Plata

Una propuesta arquitectónica para integrar una herramienta BPMS y un sistema de gestión de reglas de negocio

Tesina de grado para Licenciatura en Sistemas

Alumno: Julián Matías Parra
Directora: Mg. Patricia Bazán
Asesor Profesional: Lic. José Nicolás Martínez Garro

Agradecimientos:

En primer lugar, quiero agradecer el apoyo de mi familia durante mis años en la facultad y es a ellos a quienes les dedico especialmente este trabajo, siendo el afecto y la motivación transmitida tanto por mis padres como mi hermano factores claves en la culminación de la carrera. Les agradezco infinitamente la confianza que me depositaron desde un principio, entendiendo también las dificultades presentadas en el camino.

Agradezco a mi directora Patricia Bazán por darme la posibilidad de realizar este trabajo bajo su acompañamiento, por su paciencia, predisposición y tiempo empleado para revisar los capítulos, siendo el aporte de sus conocimientos indispensables para realizar el informe. Agradezco también a José Martinez Garro por estar siempre presente cuando me surgían dudas en la escritura, aportando su experiencia y brindando los consejos necesarios.

Agradezco a mis amigos y compañeros de la facultad, por los años compartidos juntos y por estar cerca para dar una mano cuando la necesitaba.

Índice General

1

1. Introducción

1.1. Contexto

En los últimos años las organizaciones fueron en busca continua de formas efectivas de conectar sus usuarios con la información requerida. Debido al crecimiento de Internet y el paso acelerado que presenta el negocio se genera constantemente una enorme cantidad de información por lo que las organizaciones deben buscar mecanismos más rápidos y eficientes para gestionar tal volumen de datos.
La metodología BPM (*Business Process Management o Gestión de procesos de negocio*) ha permitido a las organizaciones hacer que su información sea accesible y utilizable mediante la automatización de tareas humanas, la integración de diversos sistemas preexistentes basados en diferentes plataformas de IT y la optimización del rendimiento de los procesos mediante la disponibilidad de métricas en herramientas analíticas y tableros de control. Esto puede llevarse a cabo utilizando una herramienta BPMS (*Business Process Management System o Sistema de gestión de procesos de negocio*).
Otra necesidad que surge en las empresas es la de organizar sus reglas de negocio, estas pueden estar explícitamente detalladas en documentos y sistemas o conocerse implícitamente en las personas encargadas de llevar a cabo las operaciones. Debido a la diversidad subyacente en las fuentes de información y ante la necesidad de cambio constante junto a una rápida respuesta a las restricciones del dominio dominantes es que surge la necesidad de centralizar y delegar el manejo de las reglas. Es por esto que surgen los BRMS (Business Rules Management Systems o Sistemas gestores de reglas de negocio) cuya función es centralizar la administración de tales restricciones que impone el dominio, permitiendo a usuarios técnicos y no técnicos poder realizar alteraciones sobre las mismas y que esto impacte en forma directa en el funcionamiento de la organización.

1.2. Motivación

Las organizaciones han encontrado en la tecnología BPM una forma de conectar sus usuarios con la información necesaria, de forma eficiente y en los plazos establecidos.
Como resultado del incremento en el volumen de datos a gestionar y en la necesidad de tomar elecciones rápidas a lo largo de múltiples unidades del negocio, los procesos se han vuelto más complejos por lo que requieren una mayor capacidad para la toma de decisiones y que estas sean lo mas flexibles posibles. La mayoría de las herramientas actuales cuentan con alguna funcionalidad de toma de decisiones, pero estas se limitan a motores rudimentarios que analizan expresiones lógicas para definir los caminos que toman los procesos. Esto no nos provee capacidad alguna de reusabilidad o centralización, además de que cualquier cambio en la definición de las reglas de negocio solo se verá reflejado en nuevas instancias de nuestros procesos.
Los sistemas de gestión de reglas de negocio ayudan a automatizar la ejecución de decisiones, ofreciendo un potente motor donde las reglas son interpretadas. Permiten además que los usuarios puedan crear y editar políticas dentro de sus aplicaciones, en un lenguaje familiar y sin tener que delegar esta tarea en el personal IT (Information Tecnology), utilizando para ello un repositorio de reglas de negocio con capacidades de gestión.

Conectar un BRMS y un BPMS aporta una serie de beneficios ya que las reglas pasan a administrarse fuera de la definición de procesos, permitiendo que cualquier cambio se vea reflejado en ejecución, aportando así total flexibilidad en el mantenimiento. Por otro lado, al contar con un motor especializado en el manejo complejo de reglas podemos dejar de limitarnos a gestionar simples expresiones lógicas para pasar también a separar cómputos pesados de nuestros procesos, haciendo que la carga de procesamiento en los mismos se vea reducida.
En la actualidad son varias las organizaciones que conectan ambas tecnologías previamente mencionadas y para ello utilizan mecanismos de comunicación propios, no existiendo una metodología concreta para realizar tal integración. Un ejemplo de tal situación podemos encontrar en los productos Websphere Process Server y JRules de IBM, que representan claramente la interacción de un motor de procesos con uno de reglas.
La motivación de este trabajo radica en la investigación de las tecnologías BPMS y BRMS en forma conjunta para poder identificar una arquitectura integradora de ambas partes, permitiendo así que se puedan adaptar en forma adecuada y otorgar además un nivel correcto de flexibilidad y eficiencia.

1.3. Objetivo

El objetivo del siguiente trabajo es el de proponer un modelo arquitectónico para integrar un BPMS y un sistema gestor de reglas de negocio. Se buscará indicar cuáles de sus componentes requieren ser unificados y/o adaptados.
El trabajo abarca la construcción de un sistema intermedio que sirva de nexo entre los dos componentes, contemplando la comunicación y el intercambio de datos entre ambos. Se trabajará utilizando Bonita Open Solution como BPMS y Drools como sistema gestor de reglas de negocio.

Los aportes de este trabajo incluyen:

- Realizar una propuesta arquitectónica para integrar un sistema de gestión de reglas de negocio y un BPMS.
- Desarrollar una componente intermedia capaz de dialogar con los dos sistemas principales, siendo la responsable también de gestionar cuestiones referentes al modelo de datos y la vinculación de los procesos y actividades con sus reglas.

RESULTADOS ESPERADOS

- Investigar de qué forma interactúan las herramientas BPMS y los sistemas de gestión de reglas de negocio de manera de poder identificar un modelo arquitectónico que permita obtener una visión integral de ambas tecnologías.
- Desarrollar una aplicación intermedia que gestione tanto aspectos de la integración como de la comunicación, manejo del modelo de datos y la asociación entre procesos y reglas.
- Mantener un repositorio de usuarios común a las dos tecnologías mencionadas.
- Implementar un caso de estudio que permita aplicar en forma clara todo el contenido investigado.

1.4 Organización del documento:

El presente documento se organiza de la siguiente manera:

En el Capitulo 2 se presentan los conceptos principales sobre procesos de negocio y gestión por procesos, abarcando definiciones y detalles sobre el ciclo de vida los mismos. Se explica en que consisten los sistemas gestores de procesos de negocio, uno de los componentes principales a considerar en este trabajo, detallando sus principales funcionalidades y ventajas a la hora de su utilización en el ciclo de vida de los procesos.
En el capitulo 3 se define el concepto de regla de negocio y se explica en que consiste una solución orientada a reglas. Finalmente se da énfasis a los sistemas gestores de reglas de negocio, segundo componente esencial en el proceso de integración propuesto en esta tesina.

2

2. BPM y BPMS

2.1. Business Process Management (BPM)

En este capitulo se hará un repaso conceptual de proceso de negocio y gestión por procesos, detallando el ciclo de vida que cumplen los mismos como así también una posible clasificación dependiendo de diversos criterios. El objetivo es desembocar en la definición de sistema gestor de procesos de negocios, considerando su arquitectura y principales funcionalidades, esto nos proveerá un marco teórico sobre este componente esencial en la búsqueda del modelo de integración propuesto en este trabajo. Finalmente se hará un especial hincapié sobre la participación de las reglas de negocio en una solución orientada a procesos, sirviendo de nexo al siguiente capitulo donde se detallará en profundidad tal concepto.

2.1.1 Definiciones y generalidades:

Según Mathias Weske en [1], un proceso de negocio consiste en un conjunto de actividades que son realizadas en forma coordinada en un entorno organizacional y técnico, cumpliendo una función específica del dominio. Cada proceso de negocio es implementado a lo largo de una única organización pero puede llegar a interactuar con otros procesos implementados en distintas organizaciones.
La gestión de procesos de negocio incluye conceptos, métodos y técnicas que incluyen el diseño, administración, configuración y análisis de procesos, partiendo de la restricción de ejecución entre los mismos y la representación explicita de sus actividades. Una vez que los procesos son definidos, estos pueden ser sujetos a análisis y mejora.
Un modelo de proceso de negocio consiste en un conjunto de actividades modelo y restricciones de ejecución entre las mismas. Una instancia de proceso representa un caso concreto en el dominio operacional de una compañía, esta consiste en instancias de actividades. Los modelos de procesos son los elementos principales a la hora de la puesta en funcionamiento, que puede ser realizada a través de la ejecución de reglas y políticas organizacionales o utilizando un sistema de gestión de procesos de negocio.

2.1.2 Ciclo de vida de los procesos de negocio:

El ciclo de vida de los procesos de negocio se compone de fases cíclicas que poseen una dependencia lógica, comenzando con la de **diseño y análisis** en la que se realizan entrevistas sobre los procesos de negocio organizacionales y del ambiente técnico. En base a los datos recolectados, los procesos de negocio son identificados, revisados, validados y representados por modelos explícitos que son expresados en notaciones gráficas que facilitan la comunicación sobre estos procesos de tal forma que los diferentes involucrados puedan comunicarse en forma eficiente para así poder redefinirlos y mejorarlos.
Técnicas de modelado de procesos así como también de validación, simulación y verificación son utilizadas durante esta etapa. El modelado de procesos es la fase central durante el diseño del proceso. Mediante la implementación de entrevistas y la mejora de las actividades, la descripción informal es transformada utilizando una notación de proceso de negocio particular.

Una vez que el diseño inicial de un proceso de negocio es desarrollado necesita ser validado, un instrumento de utilidad para tal tarea es la reunión de las personas involucradas para que puedan discutir sobre el mismo, estos verificarán si todas las instancias del proceso se reflejan en el modelo de proceso de negocio.
La simulación puede ser utilizada como soporte a la validación, también permite a los involucrados hacer un recorrido a lo largo del proceso paso a paso verificando que el proceso actual expone el comportamiento deseado. La mayoría de los sistemas gestores de procesos de negocio proveen un entorno de simulación que puede ser utilizado en este paso.
Una vez que el modelo de proceso de negocio es diseñado y validado se pasa a la fase de **configuración**. Existen diferentes formas de realizar tal etapa, se puede implementar mediante un conjunto de políticas y procedimientos que los empleados de la empresa necesitan cumplir o puede ser ejecutada mediante un sistema gestor de procesos de negocio. Eligiendo como alternativa esta última, se deberá elegir una plataforma de implementación. El modelo de proceso de negocio se verá incrementado por información técnica que facilita su representación por el sistema gestor de procesos de negocio, que requiere ser configurado acorde al ambiente organizacional de la empresa y al proceso de negocio cuya representación debe controlar. Esta configuración incluye interacciones de los empleados con el sistema así como también la integración de los sistemas existentes con el mismo.
Una vez cumplida la fase de configuración las instancias de procesos de negocio pasan por la etapa de **representación**, es decir, pasan a ser ejecutadas. Iniciar una instancia normalmente se produce por un evento definido como puede ser el recibir una orden por un cliente.
El BPMS controla activamente la ejecución de las instancias de los procesos de negocio conforme a como están definidas en el modelo, para esto se requiere una correcta orquestación.
Un componente de monitorización de un sistema gestor de procesos de negocio visualiza el estado de las instancias. La monitorización es un mecanismo importante para proveer información acertada del estado de nuestros procesos, tal información es de utilidad por ejemplo para responder a pedidos del cliente que necesita conocer el estado actual de una instancia en curso.
Durante la fase de representación se recolectan datos relevantes de la ejecución de las instancias, normalmente en alguna forma de archivo de *log*, estos archivos consisten en conjuntos ordenados de entradas que indican eventos que ocurrieron durante la ejecución de las instancias. El inicio y fin de las actividades produce información que es almacenada en *logs* de ejecución.
La fase de **evaluación** utiliza información disponible para evaluar y mejorar los modelos de procesos de negocio y sus implementaciones. Los *logs* de ejecución son analizados utilizando monitorización de actividades del negocio y técnicas de *process mining*. Estas técnicas apuntan a identificar la calidad de los modelos de proceso de negocio y la adecuación del entorno de ejecución. Por ejemplo una actividad de monitoreo de proceso de negocio puede identificar que una cierta actividad toma demasiado tiempo debido a la escasez de recursos requeridos para llevarlos a cabo. El objetivo es obtener un seguimiento de los procesos, incluyendo información sobre los actores que intervienen, los tiempos involucrados y además otorgando la posibilidad de descubrir nuevos procesos, controles e información. Dado que la información es útil también para la simulación de procesos de negocio, estos pasos están estrechamente relacionados [1].

2.1.3 Clasificación procesos de negocio:

Los procesos de negocio se pueden clasificar a partir de distintos enfoques, según M.Weske[1] podemos plantear una manera de hacerlo en cinco niveles distintos:

Según el nivel de granularidad

Desde este punto de vista, los procesos de negocios pueden calificarse como organizacionales u operacionales. Hablamos de procesos organizacionales cuando describen funciones de alto nivel y

apuntan a objetivos globales. En cambio los procesos pueden calificarse como operacionales cuando presentan un mayor nivel de detalle y suelen representar funciones de bajo nivel del negocio.

Según el alcance corporativo

Este aspecto permite clasificar a los procesos de negocios según se circunscriban a la organización propia, o la trasciendan hacia otras organizaciones. Se identifican procesos intraorganizacionales e interorganizacionales. Los procesos intraorganizacionales no interactúan con procesos implementados por otras entidades, mientras que los interorganizacionales requieren establecer vínculos con partes pertenecientes a otras organizaciones. Esta clasificación hace claro énfasis en la orquestación de los procesos.

Según el grado de automatización

El grado de automatización de un proceso de negocio permitiría clasificarlos en totalmente automatizados, parcialmente automatizados o manuales, indicando el grado de interacción humana que requiere la puesta en marcha del proceso.

Según el grado de repetición

Los procesos se pueden clasificar según el grado de repetición de los mismos.

Cuando el grado de repetición es alto, el esfuerzo realizado en la modelización y promulgación esta justificado ya que se presentarán múltiples instancias con el mismo modelo.

Los procesos con bajo nivel de repetición requieren mucha participación de actores por lo que se cuestiona la necesidad de automatización de los mismos.

Según el grado de estructuración

Un proceso de negocio estructurado es el que pauta las actividades a realizar y las restricciones de ejecución de una única manera. En la fase de diseño se toman las decisiones esenciales que acatará el proceso. Los procesos estructurados no permiten saltear actividades no requeridas o ejecutar concurrentemente actividades definidas como secuenciales en el diseño del modelo.

Al trabajar con procesos estructurados surge la necesidad de utilizar alguna herramienta de workflow.

2.2 Business Process Management System (BPMS)

Las organizaciones pueden alcanzar beneficios adicionales si utilizan algún software para coordinar las actividades involucradas en los procesos de negocio. Con tal fin se emplean los sistemas de gestión de procesos de negocio (BPMS, Business Process Management Systems), estos son sistemas genéricos que emplean representaciones explícitas de procesos para coordinar la producción de los mismos.

Aplicación de un sistema gestor de procesos de negocio a lo largo del ciclo de vida de nuestros procesos:

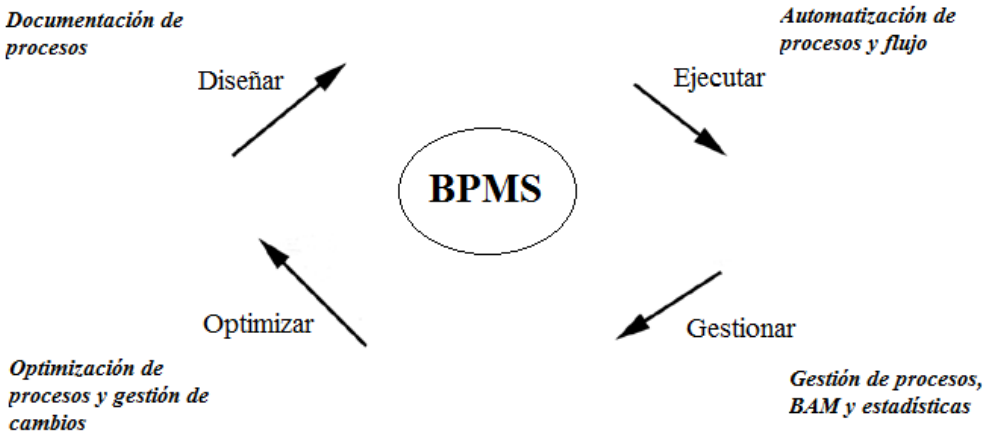

Figura 2.1 *Relacionando BPMS con el ciclo de vida de los procesos*

Es importe conocer que papel emplea un sistema gestor de procesos de negocio en el ciclo de vida de la gestión por procesos, es por esto que podemos vincular las distintas etapas del mismo con funcionalidades especificas del BPMS tal y como se muestra en la Figura 2.1. En la etapa de diseño y análisis el núcleo es el modelado de procesos, algo que los sistemas gestores de procesos de negocios tratan de cubrir en forma completa, podemos definir los datos con los que se trabajará, la organización de las tareas junto con sus participantes, definir reglas especificas extraídas del análisis del dominio, crear formularios para permitir la interacción de los actores, etc. Todo acorde a los requerimientos organizacionales. El modelo de datos puede ser validado utilizando algún mecanismo de simulación provisto por la herramienta. Esto nos permitiría obtener detalles analíticos de eficiencia y funcionamiento del sistema a través de datos de prueba emulados.
La etapa de configuración va a comprender la elección o no de un BPMS para llevar a cabo el proceso de negocio.
En cuanto a la fase de representación, utilizar un BPMS nos permitirá ejecutar nuestros procesos definidos sobre el motor de procesos, para tal fin se contará con alguna interfaz de monitorización que visualice las instancias y se puedan ejecutar tareas según el usuario correspondiente y además gestionar los casos activos, aplicaciones, etc. Los sistemas gestores de procesos de negocio cuentan también con la posibilidad de conectar algún servicio de *process minning* que permita utilizar la información contenida de los procesos y generar *logs* o cualquier tipo de información útil.
En la etapa de evaluación normalmente se cuenta con alguna herramienta de *process minning* que permita obtener datos relevantes que sirvan para un proceso de mejora continua de los mismos.

2.2.1 Arquitectura de los sistemas gestores de procesos de negocio:

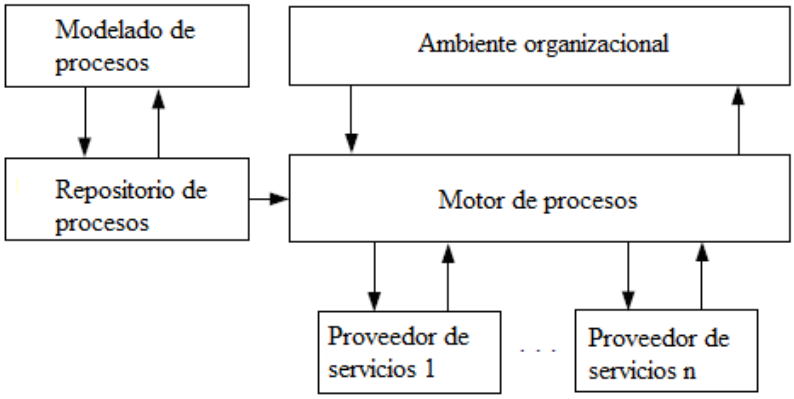

Figura 2.2.Modelo arquitectónico de un BPMS

Según el modelo arquitectónico expuesto en la Figura 2.2, un BPMS consiste en: el entorno de proceso de negocio, el subsistema de modelado de procesos, el repositorio, el motor de procesos y un conjunto de proveedores de servicios. Los roles de estos componentes de la arquitectura se caracterizan de la siguiente manera:

El subsistema de **modelado de procesos** de negocio es utilizado para crear modelos de procesos de negocio, conteniendo información sobre las actividades, sus operaciones y la estructura del proceso. Este subsistema esta representado por las herramientas de modelado.
El **entorno** realiza la instanciación y representación de las instancias de los procesos basándose en los modelos.
El **repositorio** mantiene los modelos de proceso creados.

El **motor de procesos** es el responsable de instanciar y controlar la ejecución de los procesos de negocio. Es el elemento central del sistema gestor, es llamado por el entorno de procesos y utiliza modelos para instanciar y controlar la representación de las instancias.
Los **proveedores de servicios** implementan funcionalidades que realizan las actividades de los procesos de negocio. En el modelo arquitectónico, los proveedores representan una entidad abstracta que incluye también a los usuarios que realizan actividades particulares en los procesos [1].

2.2.2. Componentes y funcionalidades principales de un BPMS

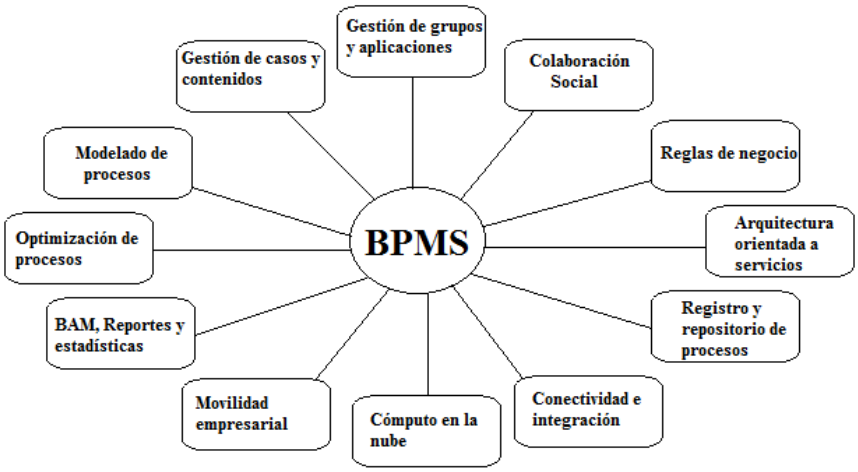

Figura 2.3. *Diagrama de componentes principales de un BPMS*

Los sistemas gestores de procesos de negocio proveen una serie de funcionalidades detalladas en la Figura 2.3. A continuación se provee una descripción de las mismas, haciendo especial atención en la interacción con reglas de negocio [2].

Modelado de procesos:
Los BPMS actuales nos proveen la posibilidad de utilizar una herramienta de modelado para poder plasmar nuestros procesos de negocio en un diagrama. Para esto facilita una completa gama de componentes que nos permiten alcanzar el máximo nivel de detalle en nuestra solución, basándose en el estándar BPMN. Podemos describir tareas de distinto tipo, siguiendo el flujo que dispongamos a lo largo del diagrama, agregar compuertas y reglas de decisión de toma de flujos, excepciones y eventos de tiempo. Existe una gestión de los usuarios de nuestros procesos para que podamos asociarlos con sus responsabilidades específicas.
Gestión del contenido:
La mayoría de los sistemas gestores de procesos de negocio proveen algún mecanismo de gestión de contenidos, ya sean documentos de cualquier tipo. Se otorgan facilidades de repositorio, gestión y control de versionado. Además utilizando interfaces internas se podrá generar documentación de variado tipo.
Gestión de la organización:
Un BPMS permite gestionar la estructura organizacional que tendrá participación sobre los procesos de negocio. Para esto podremos crear, usuarios y asignarlos a distintos grupos y roles.
Arquitectura orientada a servicios (SOA):
Existe la posibilidad de comunicar aplicaciones mediante el uso de servicios web.
Registro de procesos y repositorio:
Los modelos de proceso pueden ser publicados como web services, permitiendo la reutilización de componentes de los procesos utilizando el estándar WDSL.
Conectividad e integración:
Los BPMS actuales nos permiten integrar nuestros procesos con servicios externos como pueden ser los datos empresariales mediante una base de datos u aplicaciones. Nos permite conectar a

directorios corporativos, gestores documentales, motores de reglas de negocio, servicios de mail, mensajería y afines.

Cloud Computing:
Existen mecanismos de almacenamiento en la nube, exponiendo los procesos como servicios.
Movilidad organizacional:
Es posible crear aplicaciones móviles nativas para nuestros procesos para una gama amplia de dispositivos, con la posibilidad de ejecutar tareas desde interfaces adaptables.
BAM (Business Activity monitoring), reportes y análisis:
En estos sistemas encontraremos métricas en tiempo real para todos los procesos y tareas. Se incluye:

- Monitoreo de actividades del negocio para capturar las métricas.
- Reportes gráficos para mostrar diferentes estadísticas de nuestras instancias.
- Provee además integración con herramientas de Business Intelligence para poder así incluir datos de performance de nuestros procesos junto con datos operacionales propios de la metodología BI.

Optimización de procesos:
Nos permite a través de herramientas de seguimiento poder identificar problemas de rendimiento en tiempo real, poder predecir situaciones basándose en rendimientos pasados y tendencias. Podremos exportar reportes hacia alguna herramienta de terceros encargada de su análisis.
Reglas de negocio:
Los sistemas gestores de procesos de negocio actuales proveen algún mecanismo de implementación de reglas de negocio para nuestros procesos. Normalmente se incluye la posibilidad de definir tablas de decisión para la toma de flujos y en algunos casos se proveen motores rudimentarios que permiten la ejecución de determinadas operaciones a lo largo de las instancias.
Haremos especial hincapié en este último punto, en una metodología orientada a procesos las restricciones del dominio pueden acarrear un papel importante, en el caso que el dominio se vea influido por una gran cantidad de las mismas y estas presenten cambios constantes, se necesitará una respuesta adecuada por parte del sistema gestor de procesos de negocio encargado de orquestar las acciones. Si bien dijimos anteriormente que tales sistemas cuentan con alguna funcionalidad que les permite incorporar reglas a la solución, estas no son las adecuadas para abordar gran cantidad de reglas y de elevada complejidad dado que proveen funciones básicas. Ahora bien, sabemos que una gran cantidad de BPMSs soportan la conexión con otros sistemas, por lo que una forma de solucionar esta carencia seria acudiendo a sistemas externos que se especialicen en el tratamiento de restricciones como es el caso de los sistemas gestores de reglas de negocio, uno de los pilares de nuestra propuesta integradora. Es por esto que en el próximo capítulo nos abocaremos a la descripción de los sistemas gestores de reglas de negocio o BRMS, para conocer que herramientas los constituyen y como interactúan con el exterior, esto nos ayudara en la búsqueda de una forma estándar de vincular las tecnologías que enfatiza esta trabajo. Antes será necesario abordar más detalladamente los conceptos de regla de negocio y solución orientada a reglas para así entender el marco teórico por el que surgen los motores previamente citados.

3

3. Reglas de negocio en las aplicaciones y BRMS

3.1. Soluciones orientadas a reglas de negocio

En este capítulo se hará hincapié en el concepto de regla de negocio. Teniendo en cuenta que las organizaciones se ven cada vez más abordadas por dichas reglas, y ante la necesidad de gestionarlas, es que surgen las soluciones orientadas a reglas de negocio. En consecuencia, se describe sintéticamente tal metodología de desarrollo, incluyendo un repaso de un modelo particular de la misma. Esto nos llevara a contar con las bases necesarias para descubrir los sistemas gestores de reglas de negocio, segundo componente esencial en la integración propuesta y cuya arquitectura y funcionalidades primarias abordaremos al final de este capitulo.

3.1.1 Definiciones y generalidades:

The Object Management Group (OMG) en [3] define una regla como "una proposición que proclama una obligación necesaria" y una regla de negocio como una regla que esta bajo la jurisdicción del dominio del negocio. *The Business Rules Group* [4] produjo una serie de documentos sobre el enfoque a reglas de negocio y las considera desde dos perspectivas, desde el negocio y desde la visión de sistema de información.
Desde la **perspectiva del negocio** una regla es un indicio de que hay una cierta obligación que involucra una conducta, una acción, práctica o procedimiento sobre una actividad o ámbito particular. Existen dos características importantes en una regla de negocio, por un lado debe existir una motivación explicita para la misma y por otro debe indicar claramente que consecuencias sucederán si la regla no es respetada.
Desde la **perspectiva del sistema de información** una regla es una sentencia que define o restringe algún aspecto del dominio. La intención que se busca es apegarse a la estructura, controlar o influir en el dominio de negocio.
Esta distinción entre las dos perspectivas se debe tener en cuenta por el hecho que los procesos de negocio normalmente involucran actores humanos y sistemas de información, siendo ambos guiados por reglas explicitas.
Motivación:
Las organizaciones necesitan conocer que reglas de negocio están usando y si las están empleando en forma consistente, se requiere describir las reglas de negocio que están embebidas en sus sistemas de información de forma tal que los actores involucrados (*stakeholders*) puedan entenderlas, y también se necesita una forma de asegurar trazabilidad entre aquellas descripciones de reglas y sus implementaciones.
Las organizaciones necesitan un paradigma ágil de desarrollo que permita reaccionar a un ambiente cambiante en forma eficiente.
El enfoque a reglas de negocio es definido por Barbara Von Halle en [6] como "una manera formal de gestionar y automatizar las reglas de negocio de una organización para que estas puedan desarrollarse y evolucionar de la forma que los líderes pretendían".

3.1.2 Componentes de una solución orientada a reglas

Para poder comprender las aplicaciones que cuentan con un enfoque de reglas de negocio debemos considerar tres componentes principales:

1) Una metodología de gestión de reglas que permita recolectar, almacenar, validar, implementar y mejorar las mismas.
2) Uno o varios lenguajes de expresión en distintos niveles del ciclo de vida y para diferentes audiencias (personal del dominio, IT, etc.).
3) Un conjunto de herramientas para gestionar y ejecutar reglas, un gestor de reglas de negocio (BRMS, Business Rules Management System).

Como queda representado en la Figura 3.1 las partes se relacionan ya que BRMS soporta la metodología en varios niveles a través de un repositorio compartido de reglas, la capacidad de manejo de procesos/workflows, gestión de permisos en los accesos, etc. Las funcionalidades de gestión de un BRMS soportan la creación y modificación de reglas expresadas en lenguajes particulares. La automatización de reglas en un sistema gestor de reglas de negocio permite la ejecución de las mismas en uno o varios lenguajes.

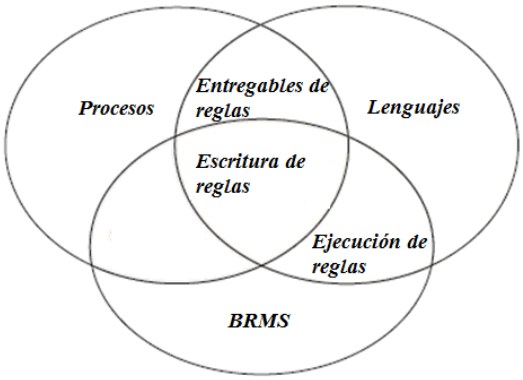

Figura 3.1.Componentes de una solución orientada a reglas

Las aplicaciones desarrolladas con el enfoque a reglas de negocio difieren de las aplicaciones convencionales básicamente en cuatro aspectos: el código mismo, la puesta en marcha, el comportamiento en tiempo de ejecución y mantenimiento [5].
Un buen diseño de una aplicación enfocada a reglas debería exhibir pocas diferencias a nivel de **código** con respecto a las aplicaciones tradicionales. La única diferencia radica en la forma en que se delegan responsabilidades relacionadas con una regla particular ya que para el caso de las aplicaciones orientadas a reglas se delega al BRMS.
Si nos enfocamos en el **despliegue de la aplicación**, una orientada a reglas de negocio difiere en que la lógica de la aplicación pasa a dividirse en dos partes, por un lado las reglas de negocio son gestionadas y ejecutadas por el BRMS y por el otro la infraestructura que es responsable del resto de las operaciones.
En cuanto al **comportamiento en tiempo de ejecución**, no deberíamos contemplar diferencias en este enfoque, salvo que arquitectónicamente se necesite de un servicio para invocar las reglas de negocio.
Es en el **mantenimiento** donde mas se pueden encontrar diferencias entre una aplicación orientada a reglas de una convencional ya que uno de los factores clave de motivación es la necesidad de

agilidad, de tal manera que las aplicaciones puedan evolucionar tan rápido como el dominio lo necesite. Varios factores hacen el mantenimiento más sencillo y rápido:

- Entendimiento del dominio: Las reglas de negocio son expresadas en lenguajes que los usuarios del negocio pueden entender, permitiéndoles tanto especificar las reglas ellos mismos como validarlas.
- Despliegue separado: Debido a que las reglas de negocio son implementadas en forma separada del código base podemos llegar a tener un mantenimiento aislado del convencional.
- División durante la ejecución. Las reglas son ejecutadas por el BRMS mediante peticiones desde la aplicación principal, esto implica que podemos desplegar nuevas reglas sin necesidad de suspender la aplicación.

Los sistemas de gestión de reglas de negocio introducen un alto nivel de flexibilidad en la arquitectura IT, permitiendo a los desarrolladores cambiar rápido y fácilmente el comportamiento de una aplicación y las decisiones que produce. Para nivelar la agilidad de los componentes BRMS, el proceso de desarrollo de la aplicación también debe ser ágil, siendo que lo desarrolladores, arquitectos y gestores del proyecto trabajan en forma iterativa e incremental.
Metodologías ágiles como eXtreme Programming, SCRUM y más recientemente OpenUp, proveen una base ideal desde la cual podemos empezar a encontrar las particularidades de desarrollar aplicaciones usando reglas de negocio. Estos aspectos contemplan la manera de descubrir reglas, la verificación de si todas pueden ser implementadas por un BRMS, la posibilidad de integración con productos BPM o inserción en una arquitectura orientada a servicios, como se representa y gestiona la información usada por las reglas, y la manera de gestionar el ciclo de vida de una restricción desde los requerimientos hasta la prueba y posterior producción.
Teniendo en cuenta que tales aspectos requieren ser ubicados en el contexto de un paradigma se desarrolló la metodología ágil de desarrollo en reglas de negocio (ABRD, *Agile Business Rule Development*) en el año 2003. La misma incluye una descripción de todos los diferentes actores BRMS, las actividades involucradas, las tareas a producir y las mejores prácticas a seguir.

3.1.3 Metodología ABRD

La metodología ágil de desarrollo de reglas de negocio es un proceso de desarrollo de software incremental e iterativo que considera los nuevos conceptos requeridos para desplegar componentes BRMS, BRE, BPEL y BPM en las aplicaciones [5].
Los principales objetivos son:

- Mantener reglas separadas como componentes administrables usando actividades de descubrimiento, análisis y autoria y sus productos de trabajo relacionados.
- Poder hacer seguimiento de las restricciones durante el ciclo de vida completo.
- Unir reglas al contexto y motivación del dominio.
- Poder describir reglas utilizando términos acordes al dominio y un lenguaje de alto nivel.
- Preparar la implementación del conjunto de reglas y desarrollar servicios de decisión en un ambiente SOA.
- Validar la calidad del conjunto de reglas a emplear utilizando un modelo de prueba con integración continua.

Las actividades propias de ABRD pertenecen a cinco categorías donde cada una de estas actividades es ejecutada múltiples veces durante la ejecución del proceso.

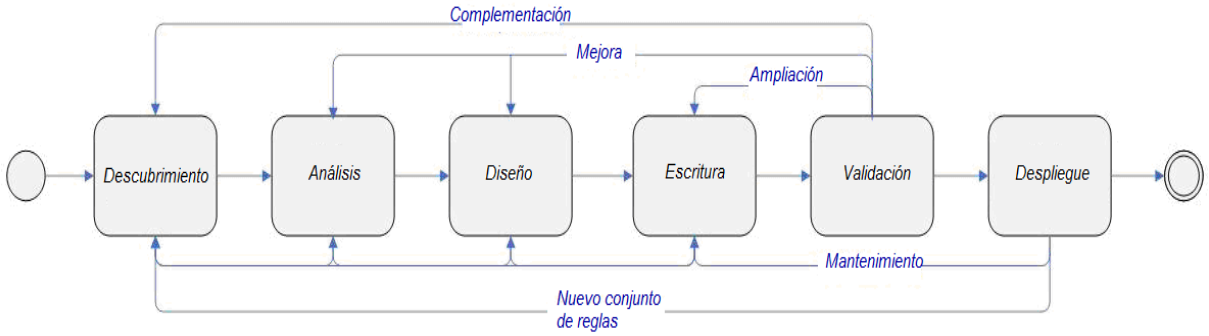

Figura 3.2.Tareas de la metodología ABRD

En el diagrama de procesos de la Figura 3.2 cada una de las tareas itera con la anterior del flujo, estos 'loops', o ciclos, son repetidos a medida que se van requiriendo, con las iteraciones siendo gradualmente trasladadas de izquierda a derecha a lo largo del proceso hasta que el conjunto de reglas sea desplegado. Este método de desarrollo de reglas refleja la madurez de un BRMS, un modelo de objetos del dominio estable y maduro es un prerrequisito para el desarrollo de reglas, por lo tanto debemos multiplicar los ciclos a lo largo de la etapa temprana del flujo del proceso para así poder desarrollar modelos tempranos. A medida que los ciclos se completan, el conjunto de reglas crecerá gradualmente hasta que alcanza un estado que refleje los requerimientos impuestos por el negocio.

La metodología ABRD provee un enfoque orientado a ciclos. El primer ciclo, como muestra la Figura 3.3, es el de **recolección** y su objetivo es el de entender las entidades del negocio y documentar la cantidad de reglas suficientes como para comenzar con la implementación. El equipo del proyecto realiza actividades de modelado del negocio donde se apunta a describir los procesos de negocio y las decisiones aplicadas sobre el ámbito de la aplicación. Este paso también ayuda a identificar y evaluar potenciales patrones de reglas.

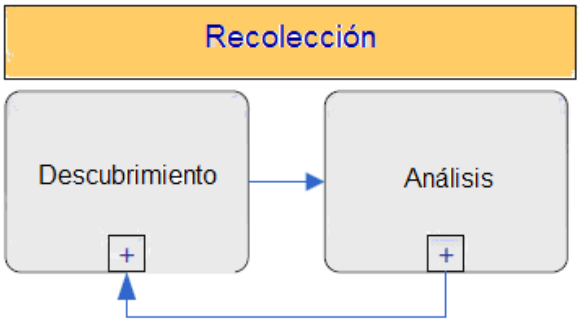

Figura 3.3.Ciclo de Recolección

Para comenzar, el equipo de desarrollo divide el día en dos partes, empleando la mañana en una sesión de descubrimiento de reglas, esta dura entre dos y tres horas y para esto se emplea cualquier fuente posible de información. El equipo encuentra restricciones en la descripción de los procesos de negocio, entrevistas con expertos de la materia, especificaciones de casos de uso y cualquier otro recurso disponible. El resto del día se emplea realizando análisis y documentación de los resultados. Dependiendo del número y la complejidad de las reglas, el equipo itera en estos dos pasos entre dos y cinco días.

Uno de los documentos producidos durante esta fase de modelado es la tabla de decisión que describe los puntos del proceso (tareas, actividades, transiciones) donde hay muchas decisiones de negocio involucradas (condiciones y acciones). Estos puntos de decisión reprendan potenciales candidatos a pasar a constituir un conjunto de reglas.

El ciclo de **prototipado** reflejado en la Figura 3.4 comprende la preparación de la estructura del proyecto y la decisión de cómo las reglas serán organizadas en un conjunto. Una vez que un cierto nivel de descubrimiento es realizado, el equipo de desarrollo puede empezar a implementar la estructura del conjunto de negocio y comenzar a escribir las reglas mientras las actividades de descubrimiento y análisis continúan.
Comenzar a escribir reglas utilizando las herramientas disponibles (normalmente un entorno de desarrollo integrado) lo antes posible permite encontrar problemas de análisis o diseño tempranamente. Las reglas pueden aparentar bien en papel pero los verdaderos problemas surgen durante la implementación y el testeo, cualquiera de estos inconvenientes encontrados durante el prototipado son comunicados al equipo de negocio durante la mañana siguiente a las tareas de descubrimiento. Este ciclo de realimentación provee un mecanismo eficiente para generar un conjunto de reglas práctico, adecuado y relevante al dominio.
Para asegurar que el equipo de desarrollo comprende las reglas pueden trabajar con la tabla de decisión producida durante el ciclo de recolección. Como un punto de partida para el diseño inicial cada punto de decisión en la tabla es mapeado a un conjunto de reglas. El arquitecto necesita considerar el modelo de datos, el flujo de ejecución de la regla, como los errores y excepciones serán reportados, parámetros de entrada y salida usados por el motor de reglas para cada conjunto de reglas candidato y como los resultados de decisión serán reportados nuevamente al cliente.

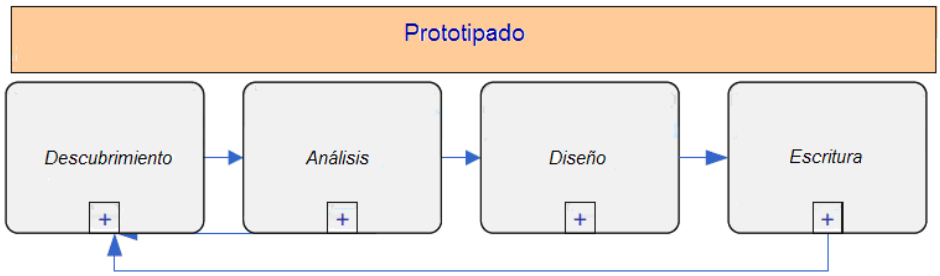

Figura 3.4.Ciclo de prototipado

El objetivo de la etapa de **construcción** (Figura 3.5) es el de implementar un conjunto de casos de prueba con datos realísticos, escribir las reglas a ser testeadas para probarlas en su respectivo contexto.
Este ciclo de tres a cuatro semanas de duración incluye en forma diaria actividades de escritura, que se pueden ver como pequeños pasos que involucran implementación de casos de prueba, escribiendo y ejecutando reglas, y haciendo algún tipo de validación con los miembros del equipo. Las iteraciones de cada día incluyen:

- Iteración sobre la escritura y validación para desarrollar casos de prueba y reglas.
- Iteración sobre análisis, diseño, escritura y validación para escribir reglas ejecutables, completar el análisis, hacer alguna prueba y resolver problemas. Esta es una iteración de mejora, el diseño es tratado para mejorar el modelo de datos y la estructura del conjunto de reglas (variables locales, cambios en el flujo, etc.).
- Iteración dos veces en un día de descubrimiento, análisis, escritura y validación. El relevamiento será usado para completar el alcance del conjunto de reglas y para hacer énfasis en los problemas identificados durante la implementación.

Para el final de este ciclo, el modelo de datos utilizado por las reglas debe estar al menos 90% completo, la estructura del proyecto debe ser finalizada y un conjunto de reglas debe ser desplegado para probar de qué manera se puede comenzar a probar la aplicación con el servicio de decisión. El

conjunto de reglas debe estar al menos un 40 o 60% completo, pero los usuarios y los autores pueden elaborarlo y completarlo en un ciclo posterior.
Si el conjunto de reglas es demasiado largo para estar al 40% completo al final de tres semanas, se puede ejecutar el mismo varias veces. Sin embargo, se recomienda mantener este ciclo durante tres semanas para poder otorgar una concreta construcción que permita asegurar calidad o validación antes de embarcar en otro ciclo.

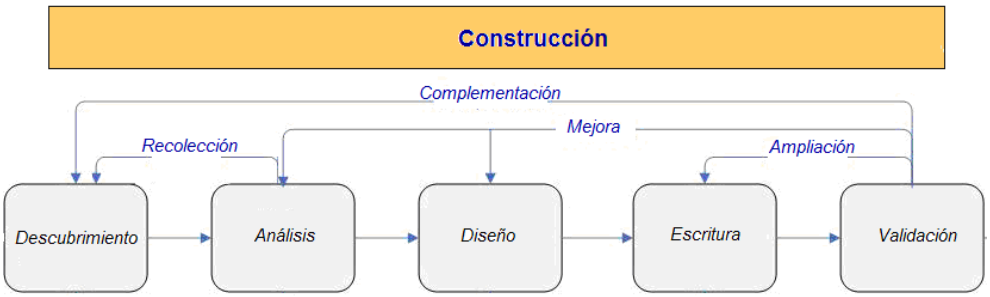

Figura 3.5.Ciclo de Construcción

El objetivo de este ciclo es el de desplegar las reglas bajo construcción en un servidor de ejecución para probarlas en un escenario pautado.
La integración del servicio de decisión y del dominio del modelo de datos es una tarea importante. Los datos provenientes del conjunto de datos son enviados al motor de ejecución para así poder desplegar las reglas. Durante las fases previas, el equipo de desarrollo desarrollará un conjunto de escenarios de prueba con datos realísticos que infieren la ejecución de reglas, tales escenarios son ejecutados durante la fase de integración para hacer de soporte a las pruebas de principio a fin.
El último ciclo es el de **mejora** (Figura 3.6) y puede ser visto como un paso de maduración donde el objetivo es completar y mantener el conjunto de reglas. Se incluye escritura, validación y despliegue pero debe incluso requerir algunas actividades de descubrimiento cara a cara con expertos en la materia para afianzar conceptos y resolver problemas.
Los actores responsables para completar el conjunto de reglas pueden ser diferentes de aquellos involucrados en los ciclos iniciales. Los miembros del equipo pueden estar más orientados al negocio y son los responsables de las políticas del negocio. Una vez que la infraestructura es implementada por el equipo de desarrollo, puede completar el conjunto de reglas. Las mejoras serán incluso necesarias para el modelo de objetos o el modelo físico de datos ya sea agregando nuevos hechos, atributos o entidades.
El propósito de este ciclo es el de mejorar la calidad y la completitud del conjunto de reglas.

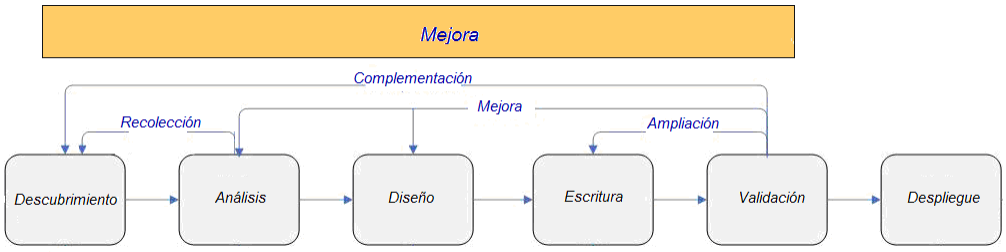

Figura 3.6. Ciclo de mejora.

Las prácticas óptimas de gestión de reglas de negocio deben ajustarse a los objetivos de la organización, esto nos lleva a considerar el modelo RMM (*Rule Maturity Model*).

3.1.4 Modelo de madurez en reglas de negocio (RMM)

El modelo de madurez en reglas (RMM), según Barbara Von Halle en [6], es esencialmente un modelo simple y práctico por el cual una organización alinea sus objetivos de negocio y las prácticas óptimas de gestión de reglas de negocio para alcanzar tales objetivos. Provee una ruta personalizable para cada organización o proyecto.
Como otros modelos de madurez, RMM tiene seis niveles desde 0 hasta 5, tal y como están representados en la Figura 3.7. En el nivel 0 el personal percibe que las reglas de negocio tienen un valor, en el 5 las organizaciones utilizan reglas de negocio en forma proactiva y predictiva al cambio así como también obtienen cierta ventaja competitiva. Por lo tanto cada nivel RMM representa una alineación entre los objetivos organizaciones y las prácticas de gestión de reglas, soportadas por roles, técnicas y requerimientos de software.
Cada nivel también representa un cambio mayor en la cultura de una organización y su habilidad para alcanzar objetivos más altos.

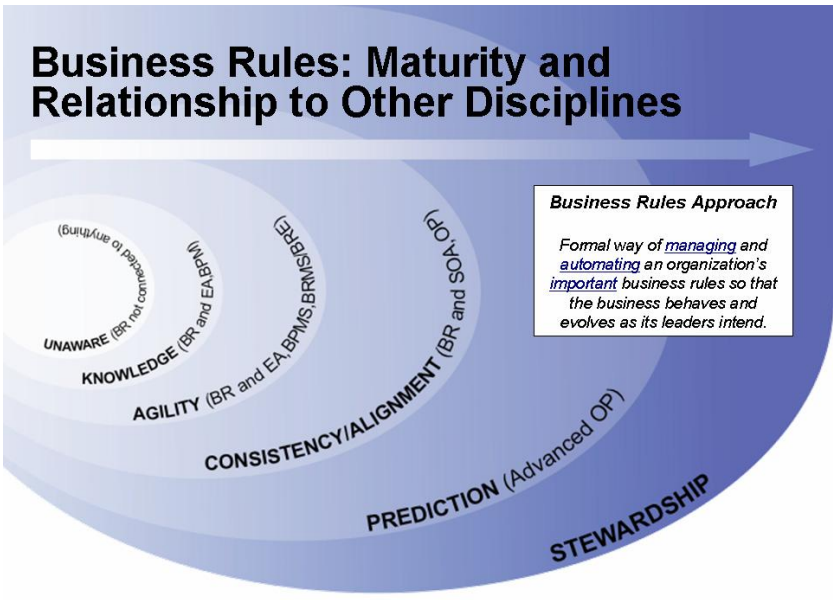

Figura 3.7.RMM. Modelo de madurez en reglas

Nivel 0: La organización desconoce la importancia de usar reglas de negocio, las mismas están ocultas en detalles sobre procesos, políticas y procedimientos o en diseños de sistemas, siendo vulnerable a la pérdida. La organización sobrevive con cambios en las reglas que son difíciles, consumen tiempos e incrementan costos.
Nivel 1: La organización quiere obtener conocimiento de sus reglas, para ello separa las reglas de negocio de otro tipo de componentes o tipos de requerimientos. Para lograr la separación la organización en nivel de madurez 1 captura las reglas en una forma simple ya que son recolectadas de documentos, extensiones simples o una base de datos relacional que tiene espacio para metadatos sobre las mismas.
Nivel 2: La organización apunta a la agilidad en sus reglas de negocio, se busca mayor rigor en la recolección y en la gestión de las mismas. Una organización en un nivel 2 de madurez no solo separa las reglas de otros hechos del negocio sino que lo hace con un proceso de escritura bien definido. Este proceso comienza con la escritura de reglas o la modificación de las mismas, analizándolas, probándolas y poniéndolas en producción (si son automatizables). Una organización con un nivel 2 de madurez esta bien posicionada para tomar ventaja de un BRMS porque las reglas son expresadas

en forma rigurosa, acercándose a la necesidad de la automatización provista por estos gestores de reglas. Se almacenan las reglas de negocio en un repositorio sofisticado que es utilizando tanto por el usuario del negocio como por personal IT. Se definen roles para el tratamiento de reglas.
Nivel 3: Una organización que apunta a un nivel 3 de madurez busca consistencia sobre las reglas definidas y quiere lograr una alineación de las mismas a los objetivos actuales que son cambiantes. Tal organización identifica los beneficios de estandarizar o compartir reglas e incluso servicios de reglas automatizados a lo largo de los proyectos. Estas organizaciones establecen un centro de excelencia de reglas de negocio que provee los estándares en las metodologías basadas en las mismas. Pueden existir varios BRMs.
Nivel 4: Una organización con nivel 4 ve las reglas de negocio como predicciones para éxito a futuro. Personal del dominio o analistas realizan hipótesis sobre futuros eventos a los que desean responder en una forma precalculada. Se realizan una serie de pruebas y simulaciones comparando las predicciones y se tienen preparados conjuntos de reglas en anticipo a posibles oportunidades. Estas personas utilizan reglas de negocio para reaccionar a tales eventos y predecir los impactos a tales cambios.
Nivel 5: Una organización apuntando a nivel 5 incorpora una total administración de las reglas de negocio para redefinirse y reinventarse como sea necesario.
La diferencia con respecto al nivel 4 radica en que el nivel 5 apunta a futuros a largo plazo, la organización define varios futuros anticipables.

En el marco de una metodología de gestión de reglas como ABRD, al tomar la decisión de automatizar las reglas pensamos inmediatamente en utilizar un sistema gestor de reglas de negocio o BRMS. Es por esto que en el próximo apartado abordaremos en profundidad tales sistemas, conociendo sus funcionalidades principales, las ventajas corporativas que proveen y sus componentes arquitectónicos.

3.2 Business Rules Management System (BRMS)

La gestión de reglas de negocios es una práctica de implementación de sistemas basada en el enfoque a reglas. Una alternativa muy difundida y empleada en el actualidad es la de usar un sistema de gestión de reglas de negocio.
Los BRMS tienen las siguientes características y responsabilidades:

- Almacenan y mantienen un repositorio de reglas de negocio que representan las políticas y procedimientos de una empresa.
- Mantienen las reglas en forma separada de las aplicaciones.
- Proveen integración con las aplicaciones empresariales de tal forma que las reglas puedan ser usadas en cualquier toma de decisión usando datos comunes al dominio.
- Permiten componer conjuntos de reglas y realizar inferencias sobre los mismos.
- Permiten a los analistas del dominio e incluso a los usuarios crear, entender y mantener las reglas y políticas de la empresa con el menor conocimiento necesario.
- Automatizan y facilitan los procesos de negocio.
- Proveen aplicaciones inteligentes que interactúan con los usuarios a través de interfaces entendibles.

La idea de que las reglas sean almacenadas en un repositorio es algo muy importante, si queremos gestionarlas parece no haber otra alternativa que contenerlas en alguna especie de base de datos centralizada. Concentrar las reglas en una capa separada tanto de las aplicaciones y de las varias bases de datos que puedan existir en una organización real nos da ventajas a la hora del mantenimiento además de proveer reusabilidad y legibilidad.

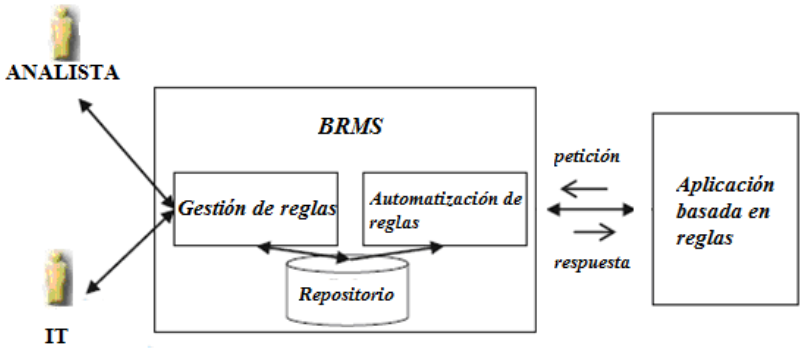

Figura 3.8.Componentes principales de un BRMS

3.2.1 Motivación y ventajas del uso de un sistema gestor de reglas de negocio

Existen diversos motivos por los cuales una organización puede requerir el uso de un BRMS. Las prácticas de desarrollo actual inhiben la entrega rápida de nuevas soluciones e incluso cualquier cambio por modesto que resulte puede conllevar demasiado tiempo. Acelerar la presión competitiva significa que las políticas y las reglas que gobiernan procesos automatizables tienen que adaptarse a un cambio rápido. Esto se puede alcanzar con una nueva forma de desarrollo, basada en la necesidad de ofrecer personalización y la necesidad de aplicar mejoras en los procesos de negocio en forma rápida a un gran número de grupos de consumidores. Personalizar servicios, contenidos y estilos de interacción basados en los tipos de proceso y en las características del usuario, esto puede añadir un valor considerable a los procesos de negocio de una organización pese a la complejidad. En algunas industrias, como por ejemplo las relacionadas con finanzas las reglas que rigen y regulan van a verse modificadas fuera del control de la organización por lo que mantenerlas separadas del código de aplicación y el hacerlas sencillas de modificar es esencial. Las reglas de negocio y los procesos pueden ser compartidos por muchas aplicaciones a lo largo de la organización usando múltiples canales como voz, web y aplicaciones intermedias, permitiendo prácticas consistentes. Usar BRMSs debería reducir los costos de desarrollo y acortar notablemente los ciclos de desarrollo y mantenimiento [5].
Tradicionalmente las aplicaciones que utilizan un BRMS incluyen aquellas relacionadas con:

Automatización de procedimientos como:
- Procesamiento de pedidos
- Gestión de servicios al cliente
- Aprobación de créditos y gestión de límites
- Resolución de problemas
- Salarios

Toma de decisión en:
- Elección de beneficios
- Promociones de ventas
- Estrategias de créditos
- Estrategias de mercado

Planificación de:
- Publicidades
- Diseños de productos
- Presupuestos

Diagnóstico y detección de:

- Condiciones médicas
- Fraudes
- Datos válidos e inválidos

Clasificación de:

- Clientes
- Productos y servicios
- Riesgos

El costo del mantenimiento en los sistemas modernos es relevante, se estima que un 90% de los costos totales provienen del mantenimiento de sistemas exigentes más que del desarrollo. Esta es una de las razones por la que el desarrollo orientado a objetos es tan atractivo actualmente ya que al cambiar la estructura de datos o alguna función no se propagan cambios a otros objetos. Sin embargo este beneficio no se aplica en el caso que existan cambios a las reglas de negocio, si es que están esparcidas por la aplicación, y las definiciones de interfaces porque los cambios se van a propagar y el mantenimiento seria muy costoso. Para enfrentar esto necesitamos separar la definición de las políticas de la implementación y el código. Esto nos facilita un sistema gestor de reglas de negocio. Preferentemente, las reglas son subdivididas en módulos que son encapsulados en objetos individuales, que son visibles a todos los objetos que son registrados y se interesan en ellos. Se encapsulan políticas globales u organizacionales. Se requiere un control de versión para este conjunto de reglas, esto implica que un BRMS almacenará conjunto de reglas en forma centralizada en un repositorio.
Los beneficios de adoptar un sistema de gestión de reglas de negocio se resumirían en:

- Desarrollo y mantenimiento más rápido, lo que particularmente es relevante en arquitecturas orientadas a servicios donde el mantenimiento de las reglas es apuntado fuera del amplio contexto IT a mantener.
- Auditoría mejor organizada y entendible.
- Lógica del negocio con mayor nivel de reutilización.
- Mayor consistencia a lo largo de la empresa.
- Mejor alineamiento y entendimiento entre personal del dominio y personal IT.

Hay que considerar que con un BRMS por si solo no alcanza para lograr los beneficios, estos deben ser implementados lado a lado junto con ingeniería orientada al negocio y las mejores prácticas de madurez con respecto a reglas.

3.2.2 Componentes y aspectos técnicos de un BRMS:

Los sistemas de gestión de reglas de negocio están ligados (tanto intelectual como comercialmente) con los sistemas expertos de los años 80. Para entenderlos es necesario conocer un poco de sus orígenes.
Los sistemas basados en reglas o “expertos” son sistemas que dan indicios o toman decisiones en un área definida o cerca del nivel de un experto humano. Hay dos tipos de estos sistemas, por un lado los que toman decisiones y por otro los que actúan como soporte de decisión. Mas importante aun, los sistemas basados en reglas son definidos por lo que hacen, su arquitectura. La parte más importante de la arquitectura es que el conocimiento es almacenado en forma separada del código que implica el conocimiento del problema en cuestión. Esto se aplica en la misma forma para los BRMS. Algunos sistemas expertos tempranos lidiaron con hechos, datos, procedimientos y reglas en la base de conocimiento mientras que en los BRMSs actuales usualmente se mantienen una clara diferenciación entre reglas de negocio y datos del negocio.

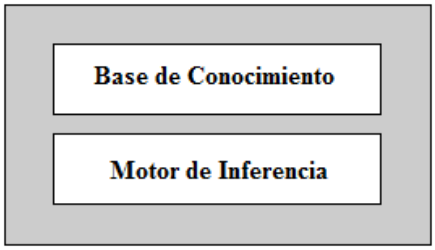

Figura 3.9. Arquitectura de un servicio de reglas de negocio

El repositorio de partes de conocimiento en un sistema de gestión de reglas de negocio es llamado base de reglas o base de conocimiento y el mecanismo que aplica tal información a los datos es denominado motor de reglas o motor de inferencia.

Actualmente se acepta la idea de que los sistemas de gestión de reglas de negocio poseen cuatro componentes. En primer lugar, el ambiente de tratamiento de símbolos y valores que puede ser pensado como los lenguajes de programación y el ambiente de soporte, editores, procesadores de punto flotante, estructuras de datos, compiladores, etc. Esto se vería reflejado en el área gris de la Figura 3.9. En segundo lugar tenemos la estructura de la base de conocimiento que incluye métodos de representación y acceso y finalmente debe haber alguna técnica para aplicar el conocimiento de una forma razonable al problema de turno. Este tercer elemento es el motor de inferencia, que encadena las reglas en conjunto para alcanzar conclusiones válidas. El cuarto elemento es el repositorio en donde las reglas se almacenan y desde donde son accedidas, versionadas, compartidas y gestionadas. La Figura 3.10 muestra al servicio de reglas en un contexto arquitectónico mas amplio, haciendo énfasis en la separación del repositorio y en la posible presencia de una o varias bases de datos.

La base de conocimiento y el motor de inferencia se separan uno del otro para facilitar el mantenimiento. Después de todo en la mayoría de los casos las reglas y políticas van a verse modificadas en el transcurso de la ejecución del sistema y no se va a requerir reescribir el motor de inferencia cada vez que una nueva regla se adhiere.

La base de conocimiento usualmente contiene información de diferentes tipos, sobre objetos, procedimientos y relaciones. Los objetos mencionados se suelen almacenar por medio de un modelo de objetos, esquema XML, modelo de datos o red semántica. Algunos procedimientos del negocio pueden ser representados con reglas [7].

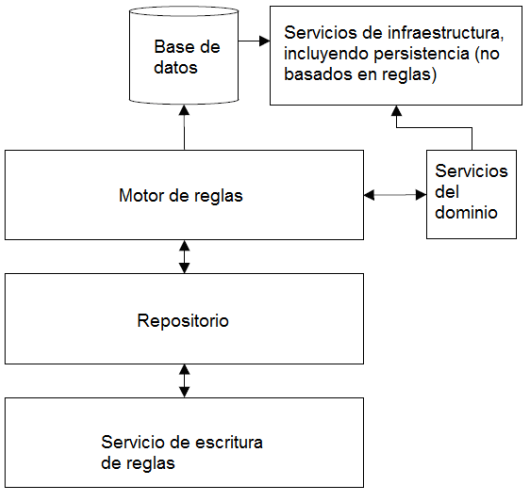

Figura 3.10.Componentes y servicios básicos provistos por un BRMS

Arquitectura de un motor de reglas

Siendo el componente esencial para la ejecución de reglas por parte de un BRMS haremos una breve descripción de la arquitectura que presentan los motores de reglas, considerando las partes con las que mantiene una interacción constante, estas son ilustradas en la Figura 3.11.

Figura 3.11.Arquitectura de un motor de reglas

Normalmente un evaluador de reglas contiene cientos o miles de las mismas y es probable que en un momento dado se activen más de una. La lista de reglas que son potencialmente ejecutables se almacenan en lo que se denomina **agenda**.

Al conjunto de reglas que tienen posibilidad de ser ejecutadas se les denomina grupo conflictivo de reglas. Al proceso de ordenarlas para ser ejecutadas se le denomina resolución de conflictos; el resultado de la resolución de conflictos es una lista ordenada de actividades que es la agenda propiamente dicha.

La **memoria de trabajo** (*working memory* o *fact base*), contiene toda la información con la cual un motor de inferencia trabaja. Por lo general se almacenan tanto las premisas como las conclusiones de las reglas, constituye la base de conocimiento en tiempo de ejecución.

El diseñador de reglas se encarga de decidir que información se almacena en la memoria de trabajo y en forma explícita deberá cargar en el contexto de ejecución los hechos relevantes sobre los cuales se aplicará la base de reglas existente.

Un hecho es la unidad de información más pequeña con la que se puede trabajar en la memoria de trabajo. Generalmente se ofrece un conjunto de operaciones para interactuar como por ejemplo:

Assert o Insert: agrega hechos a la memoria de trabajo
Retract: elimina un hecho de la memoria de trabajo.
Modify: modifica un hecho al cambiar ciertas condiciones
Clear: elimina todos los elementos
Deffacts: define el contenido inicial de la memoria de trabajo
Facts: despliega el contenido de la memoria de trabajo.
Reset: inicializa la memoria de trabajo.
Watch: imprime diagnóstico cuando se dan modificaciones sobre los hechos como por ejemplo cuando hay cambios en la memoria de trabajo.

La unidad de ejecución del motor es la regla de negocio, los BRMS actuales definen un lenguaje propio para su escritura y proveen herramientas que ayudan a realizar tal tarea .El estándar para la definición de reglas es de la forma “IF A THEN X”. Diferente a las sentencias if/then encontradas en los lenguajes convencionales, los lenguajes de reglas son declarativos o no procedurales, es decir que el orden en que las reglas son escritas no es importante. Estas reglas trabajan sobre el conocimiento sobre entidades u objetos.

En todos los productos BRMS, las reglas son representadas como sentencias, usualmente conteniendo las palabras IF y THEN. Según Morgan en [8] el mejor estilo apunta a remover la ambigüedad, haciendo las relaciones explícitas, evitando terminología difusa y removiendo palabras de más. Su estilo es llamativamente cercano al lenguaje natural, prefiriendo formas del tipo:

1) Un préstamo puede ser aprobado IF
el estado del cliente es alto
y el préstamo es menor a 2000
salvo que el cliente tenga un bajo rating

en vez de:

2) IF el estado del consumidor es alto y el préstamo es menor a 2000
Y el cliente no tiene un rating bajo
THEN aprobar el préstamo
IF el estado del cliente el alto y el préstamo es menor a 2000
Y el cliente tiene un bajo rating
THEN no aprobar el préstamo.

La mayoría de los BRMS comerciales soportan el segundo estilo de escritura de reglas, solo unos pocos como HaleyAuthority, proveen un soporte natural completo para el primer caso. Algunos productos proveen una forma de "verbalizacion" para hacer las reglas más legibles, por ejemplo utilizando plantillas.
Las **plantillas** son patrones de diseño para reglas. En muchas circunstancias, una regla puede ser aplicable a varios datos. En tales casos las plantillas permiten la creación de reglas con huecos vacíos que deberán ser llenados mas adelante. Se pueden utilizar para crear varias reglas con una estructura similar, donde solo varían los valores a insertar en los espacios, ahorran tiempo cuando se escribe y ayudan a forzar un estilo estándar y legible.
Un buen BRMS ofrecerá facilidades para verificar la sintaxis de las reglas en tiempo real, a medida que las reglas vayan ingresando. Con lenguajes de reglas estructurados, es útil si el validador remarca palabras claves, variables y valores usando diferentes colores. Deberían existir lazos más claros entre los objetos del modelo y las reglas.
Alguna parte del conocimiento es estrictamente procedural. Un BRMS adecuado ofrecerá la habilidad al conjunto de reglas de invocar procedimientos y a los procedimientos de invocar reglas para ejecutar y devolver valores.
El poder incorporar **flujos** permite al diseñador especificar que los módulos de conocimiento o tareas deben ser desplegados siguiendo un orden.
En algunos productos, existe una representación alternativa para definir reglas a la tradicional if/then. Los **árboles de decisión** representan gráficamente las reglas, con una estructura de árbol. Esto puede ser útil para la depuración o comunicación entre usuarios y desarrolladores o analistas, pero no es usualmente la forma en que los usuarios del dominio visualizan el conocimiento.
Las **tablas de decisión** representan el mismo conocimiento y reglas de un árbol de decisión en un formato tabular.
El principal problema con las tablas de decisión es que son demasiado extensas cuando hay un gran número de condiciones en la base de reglas. Se dificulta la lectura y el entendimiento. Es mejor usar algún sistema de *data mining* para extraer las restricciones de las tablas de decisión y almacenarlas en un BRMS.

Una vez definidos los conceptos de regla de negocio y BRMS, sumado al conocimiento adquirido sobre procesos de negocio y BPMS obtenidos en el capitulo anterior, estamos en condiciones de adentrarnos en los mecanismos en que los componentes (gestor de procesos y gestor de reglas) pueden interactuar entre sí para poder acercarnos a la definición de un modelo integrador. Para tal propósito nos valdremos del estudio de herramientas del mercado existentes y de alto renombre.

4

4. Conectando BPMS y BRMS

En este capitulo nos adentraremos en la conexión entre las herramientas BPMS y BRMS, definiendo cuales son los fundamentos que nos llevan a querer unirlas y estudiando distintos casos reales en los que se ha realizado una exitosa vinculación. La meta es poder llegar a alcanzar el objetivo final de este trabajo que es el de definir una propuesta arquitectónica para integrarlas, que sirva de modelo a la hora de decidirse por conectar las herramientas y que pueda orientar su realización de una forma práctica y sencilla.

4.1. Motivación

La mayoría de las herramientas BPMS actuales cuentan con alguna funcionalidad para identificar e instrumentar las restricciones del dominio. Estas se limitan a motores rudimentarios que analizan expresiones lógicas para definir los caminos que toman los procesos. Supongamos que durante la construcción de nuestro diseño de procesos queremos marcar una condición de flujo, seguramente el BPMS que estemos utilizando nos proveerá la posibilidad de generar tablas de decisión en las que podremos indicar los caminos a tomar dependiendo de ciertas condiciones del dominio. Podremos encontrar también algún pequeño repositorio interno de restricciones. Esto no nos provee capacidad alguna de reusabilidad o centralización, además de que cualquier cambio en la definición de las reglas de negocio solo se verá reflejado en nuevas instancias de nuestros procesos.
Una opción más sencilla seria definir y mantener un conjunto de reglas en un BRMS donde las mismas sean almacenadas en un repositorio y compartidas por muchas aplicaciones y procesos a lo largo de la organización. Pensemos en que las restricciones del dominio no se limitan a chequeos básicos de flujo sino que muchas veces se deben realizar cálculos complejos que requieren de mucha información organizacional. Podemos implementar la lógica de decisión utilizando un proceso y embebiéndola en sus tareas pero hacerlo en un motor de reglas de negocio hace que las restricciones del dominio obtengan mayor visibilidad para los analistas del dominio, que esta sea más consistente a los largo de la empresa y mas fácil de cambiar cuando se requiere. Utilizar un BRMS nos permitirá controlar cualquier acción que el proceso requiera relacionada con la lógica del negocio, como puede ser el cálculo de un descuento, o la aprobación de un pedido de compra.
Las reglas de nuestros procesos son parte del modelo ejecutable, por lo tanto cualquier cambio en las mismas requerirá que se despliegue una nueva versión del proceso en cuestión, lo cual añade complejidad. Incorporar un BRMS hace que modificar las reglas sea sencillo ya que estas se mantienen en forma separada de las aplicaciones que las invocan, por lo que pueden ser modificadas versionando nuevamente el proceso o no. Los cambios toman efecto inmediatamente, incluso para las instancias en ejecución.
Varias suites BPM proveen su propio motor de reglas pero ninguno cuenta con un repositorio central y con una verdadera gestión de reglas. Al extraer la lógica de decisión del BPMS y gestionándola en el BRMS, podemos aplicar reglas en forma consistente a lo largo de múltiples procesos, plataformas BPMS e incluso aplicaciones no BPM.

Conectar un sistema de gestión de procesos de negocio con un sistemas gestor de reglas implica antes que nada conocer la función de cada uno, tal y como fue detallado en el capitulo 2, por lo que el objetivo principal de este capitulo es el de mostrar que rol representa cada herramienta cuando las vemos como un conjunto y además el de poder identificar que elementos son relevantes a la hora de vincularlos.

4.2 Integración BPMS+BRMS en la actualidad

En la actualidad son muchas las herramientas BPMS y BRMS que proveen mecanismos de comunicación con sistemas externos. En el caso de los sistemas gestores de procesos de negocios son muchas las suites que facilitan conectores programados para un producto particular, la mayoría *open source*. En el caso de los sistemas gestores de reglas de negocio suelen proveer una API de comunicación que les permite recibir órdenes externas, ya sea para la ejecución de reglas, administración de las mismas en el repositorio, entre otras operaciones. Podemos citar, a modo de ejemplo, dos productos, por un lado la herramienta BPMS BonitaOS (Bonita Open Solution) desarrollada por BonitaSoft y al sistema gestor de reglas de negocio Drools desarrollado por Red Hat JBoss. BonitaOS provee una serie de conectores nativos para que los procesos puedan interactuar con aplicaciones externas, para la gestión de reglas nos provee una API de comunicación con el motor de Drools, pudiendo así ejecutar reglas complejas. Por otro lado, Drools facilita una serie de interfaces tanto para la ejecución de reglas mediante su motor Expert como para la gestión del repositorio Guvnor pudiendo agregar, eliminar y modificar reglas bajo usuarios específicos.
Otro caso de interés que podemos abordar es el de los productos privados empaquetados como IBM ILOG y su motor de procesos WebSphere que se integra al BRMS ILOG JRules.
Al fin de conocer los aspectos relevantes a tener en cuenta cuando se piensa en la unión de sistemas gestores de procesos y de reglas estudiaremos en detalle la conexión entre los mismos para los dos casos previamente citados:

4.2.1 Conexión Drools y BonitaOS:

BonitaOS provee conectores nativos para poder interactuar con el gestor Drools, estos conectores están escritos en lenguaje java y básicamente permiten conectarse a la API del motor Expert de Drools para invocar la ejecución de una determinada regla para así poder obtener resultados que cooperen en la toma de decisiones del proceso en curso.

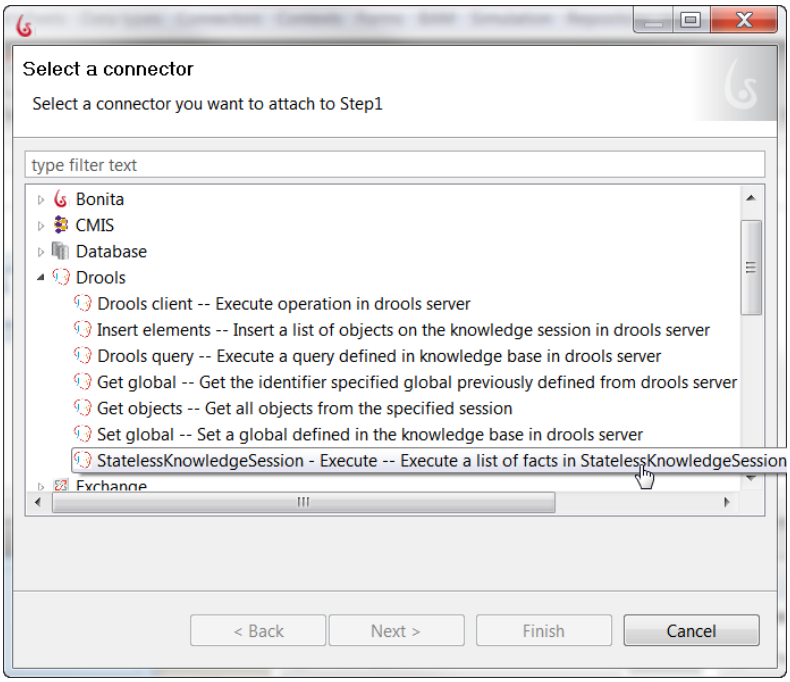

Figura 4.1 Conectores de Drools disponibles en BonitaOS

En la Figura 4.1 se detalla la lista de conectores disponibles, de los que haremos un breve resumen a continuación:

Ejecutar operación en un servidor Drools
Permite ejecutar una operación en un servidor Drools remoto, para tal fin indicaremos la url del servidor junto con parámetros de autenticación, tal como se muestra en la Figura 4.2. En base a una petición específica, como puede ser la ejecución de una determinada regla, se obtiene un resultado aplicable a cualquier variable de nuestro proceso.

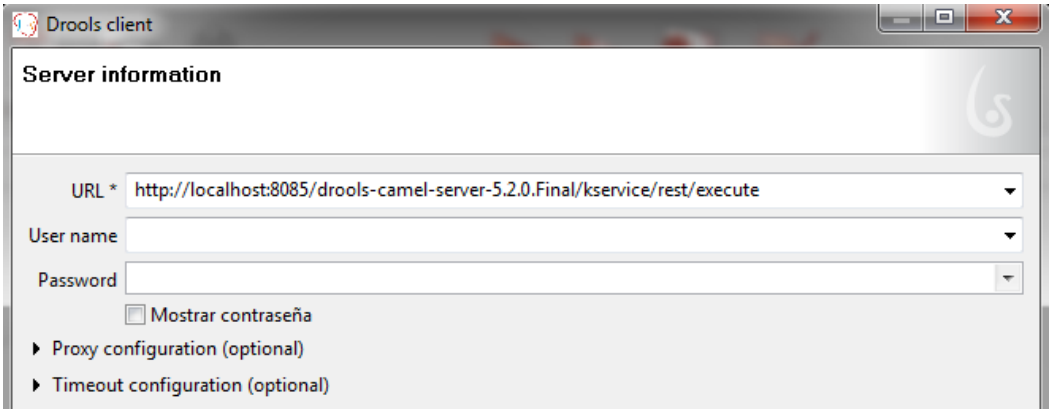

Figura 4.2 Parámetros necesarios para ejecutar una operación en Drools remoto

En la Figura 4.3 se detalla la implementación del conector, este opera creando un cliente HTTP que realiza una petición (*request*) a la url que recibe como parámetro y recibe los resultados como un string (*response*) en la respuesta.

```
protected void executeConnector()
  throws Exception
{
  String request = getRequest();
  if (LOGGER.isLoggable(Level.FINE)) {
    LOGGER.fine("request = " + request);
  }
  HttpClient httpClient = getHttpClient();
  PostMethod postMethod = new PostMethod(this.apiUrl);
  try
  {
    if (request == null) {
      throw new Exception("Request can not be empty!");
    }
    postMethod.setRequestEntity(new StringRequestEntity(request, "text/plain", "UTF-8"));
    httpClient.executeMethod(postMethod);
    this.status = String.valueOf(postMethod.getStatusCode());
    this.response = postMethod.getResponseBodyAsString();
    if (LOGGER.isLoggable(Level.FINE))
    {
      LOGGER.fine("status = " + this.status);
      LOGGER.fine("response = " + this.response);
    }
  }
  catch (Exception e)
  {
    if (LOGGER.isLoggable(Level.WARNING)) {
      LOGGER.warning(e.getMessage());
    }
    throw e;
  }
  finally
  {
    if (postMethod != null) {
      postMethod.releaseConnection();
    }
```

Figura 4.3 Implementación del conector de ejecución

Insertar una lista de objetos
Permite insertar un conjunto de objetos en la sesión de conocimiento (*Knowledge Session*) de un servidor Drools, esta es la encargada de almacenar tales objetos hasta que son analizados por el motor, para tal objetivo se indican los parámetros de conexión y la lista de elementos a insertar, es de utilidad para cuando deseamos que más entidades sean objeto de evaluación por parte de reglas de negocio.
Ejecutar una consulta
Nos permite indicar el momento en el que deseamos ejecutar una consulta definida (*Query*) en un servidor Drools.
Obtener o setear un indentificador global
Podemos obtener o modificar un identificador global (*Global*) definido en un servidor Drools, indicando los parámetros de acceso al mismo.
Obtener objetos
Permite recuperar objetos almacenados en la sesión de conocimiento de un servidor de Drools.
Ejecutar una sesión de conocimiento
Podemos ejecutar una sesión de conocimiento en Drools para una regla o conjunto de las mismas cuando deseemos, esto retornará los cambios en una lista como resultado del intercambio de datos. Bonita nos solicitará como parámetro de operación la ubicación del archivo DRL (Drools Rule Language) que contiene la definición de las operaciones que comprenden la regla a ejecutar. También se requerirá, tal como se muestra en la Figura 4.4, una lista de objetos extraídos del modelo de datos BPM que podrán ser procesados por el motor de reglas y a los que se le aplicarán los resultados de la evaluación de las restricciones correspondientes.

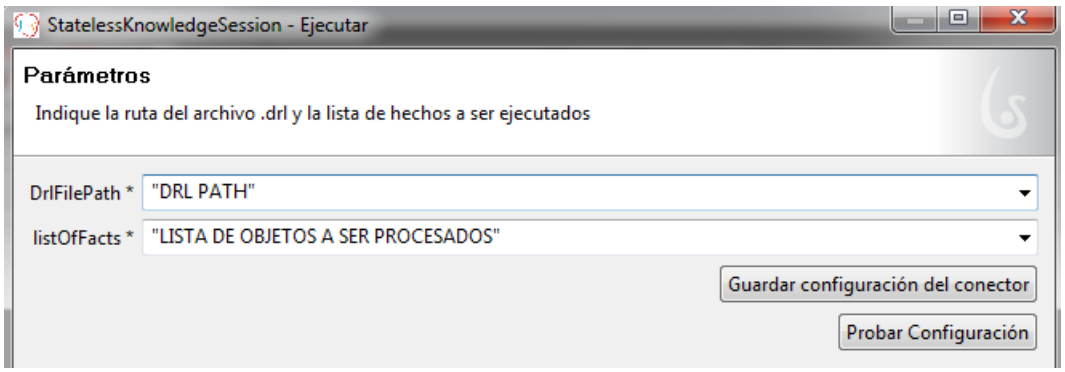

Figura 4.4 Parámetros para la ejecución de una sesión de conocimiento

Al finalizar la definición de los parámetros del conector y ejecutar la instancia de proceso llegará un punto en donde se producirá la ejecución de la regla especifica en el motor de ejecución.
BonitaOS trae en forma nativa, dentro de su paquete de librerías, la implementación del motor de reglas Expert por lo que no es necesario iniciar una instancia separada del mismo, igualmente se debe considerar que los conectores son totalmente editables por lo que es posible modificar el comportamiento como uno desee. En la Figura 4.5 se muestra la implementación del conector en BonitaOS, vemos que en base a la url indicada se carga la definición de la regla escrita en lenguaje DRL y se aplica a los objetos ingresados en la lista como *listOfFacts*.

```
protected void executeConnector()
  throws Exception
{
  LOGGER.info("Input parameters: drlFilePath:" + this.drlFilePath + " listOfFacts:" + this.listOfFacts);
  KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();


  kbuilder.add(ResourceFactory.newFileResource(this.drlFilePath), ResourceType.DRL);
  if (kbuilder.hasErrors())
  {
    System.out.println(kbuilder.getErrors().toString());
    throw new RuntimeException("Unable to compile \"" + this.drlFilePath + "\".");
  }
  Collection<KnowledgePackage> pkgs = kbuilder.getKnowledgePackages();


  KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
  kbase.addKnowledgePackages(pkgs);

  KnowledgeRuntimeLogger logger = null;

  StatelessKnowledgeSession ksession = kbase.newStatelessKnowledgeSession();

  ksession.addEventListener(new DebugAgendaEventListener());
  ksession.addEventListener(new DebugWorkingMemoryEventListener());


  logger = KnowledgeRuntimeLoggerFactory.newFileLogger(ksession, "drools");


  ksession.execute(CommandFactory.newInsertElements(this.listOfFacts));
```

Figura 4.5 Implementación del conector de ejecución Drools en BonitaOS

4.2.2 Aspectos a considerar en la comunicación BonitaOS + Drools:

A fin de alcanzar el objetivo de este trabajo, que es el de proponer una arquitectura de integración, debemos hacer un análisis de los aspectos fundamentales en la comunicación de las dos herramientas previamente abordadas, basándonos en el estudio previamente realizado.

Modo de acceso

El uso de conectores de BonitaOS para realizar la interacción con Drools requiere, en la mayoría de las ocasiones, que uno deba autenticarse utilizando un nombre de usuario y contraseña. No existe relación alguna entre los usuarios definidos en el BPMS con los del sistema gestor de reglas de negocio, pueden estar replicados en cada caso pero el mantenimiento de los mismos deberá hacerse siempre dos veces, una por cada herramienta. Por un lado el BPMS (en este caso BonitaOS) cuenta con su gestión de usuarios particular, almacenados en una base de datos y que permite asignarlos roles y grupos, tal como queda en evidencia en la Figura 4.6.

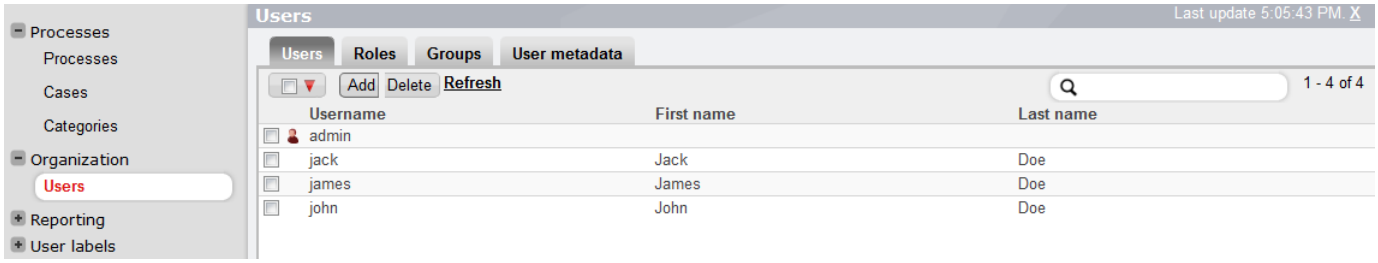

Figura 4.6 Modulo de gestión de usuarios en BonitaOS

En el caso del sistema gestor de reglas de negocios (Drools) se realiza una gestión similar, agregando además la posibilidad de gestionar permisos específicos del motor sobre paquetes y/o categorías de reglas, tal como se muestra en la Figura 4.7. Se utiliza una base de datos interna, con la posibilidad de adaptarlo a otros tipos de bases [9].

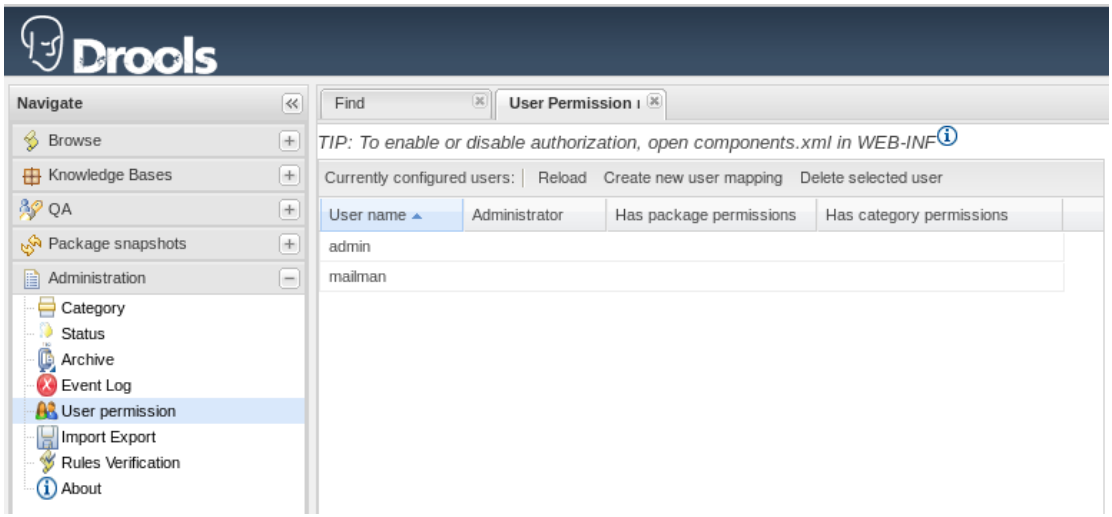

Figura 4.7 Modulo de gestión de usuarios en Drools

Modelo de datos

Es importante destacar como afecta al modelo de datos de nuestras aplicaciones la conexión de las herramientas, como observamos en los conectores de Bonita se le indica que datos debe manipular el gestor de reglas y este retorna resultados analizados y modificados por el motor. Esta integración de datos se realiza a nivel de variables definidas en nuestro diseño del modelo de proceso, Bonita OS permite que definamos las variables que creamos necesarias para el curso de nuestras instancias de procesos de negocio, tales como las relacionadas a los flujos o a las que sirvan de utilidad en las distintas tareas. La Figura 4.9 visualiza un ejemplo de definición de variables.

En Drools las reglas son analizadas en base a un modelo de datos escrito en JAVA compuesto por clases, este modelo normalmente suele responder a los requisitos del motor, por lo que es acotado. A modo de ejemplo la Figura 4.10 muestra la descripción de una clase Empleado cuya definición de atributos y operaciones es utilizada después a la hora de definir una regla concreta en lenguaje DRL.

Ahora bien, debemos tener en cuenta que BonitaOS permite utilizar conectores nativos para leer, escribir y eliminar datos de las bases empresariales, en el caso de Drools se podría hacer lo mismo aplicando algún mecanismo de mapeo objeto relacional. Por lo tanto nos encontramos con dos posibles opciones de implementación, por un lado cada herramienta puede gestionar un modelo de datos propio, siempre basado en el organizacional, y realizar los ajustes necesarios para que puedan interactuar entre si o por otro lado tenemos la posibilidad de acceder en forma directa al modelo de la organización por parte de las dos herramientas, realizando por ese vínculo la comunicación de datos.

Figura 4.9. Variables de proceso en BonitaOS

```
package model;

import java.io.Serializable;

public class Empleado implements Serializable {


    private String nombre;
    private Integer promedioConocimientos;
    private String cargo;
    private BigDecimal salario;

    getters y setters...


}

import model.Empleado;
import java.math.BigDecimal;

rule "Programador"
    when
        empleado : Empleado(promedioConocimientos >= 8, promedioConocimientos <= 10)
    then
        System.out.println("Programador: " + empleado.getNombre());
        empleado.setCargo("Programador");
        empleado.setSalario(BigDecimal.valueOf(1000));
end
```

Figura 4.10. Ejemplo de utilización de clases en reglas definidas sobre Drools

Comunicación

La comunicación para el caso estudiado se realiza mediante la utilización de una API, la misma esta provista por Drools y permite mediante una petición *HTTP REST* ejecutar una operación particular. Esto permite que las dos herramientas puedan convivir en ambientes distintos y se comuniquen remotamente utilizando la interfaz mencionada. En el caso de que compartan un entorno, es decir la comunicación se realiza localmente, la tarea será muy sencilla dado que ambas herramientas están escritas en Java y esto facilitará enormemente la flexibilidad en el intercambio.

Detallados ciertos aspectos de interés encontrados en la comunicación entre BonitaOS y Drools procederemos a detallar el siguiente caso de estudio que retrata el vínculo entre los productos ILOG Websphere y JRules.

4.2.3 Conexión ILOG Websphere y JRules:

Dentro de la implementación BPM actual de IBM, Websphere process Server es la plataforma de ejecución de procesos de negocios para la empresa que proporciona una infraestructura de procesos de negocios poderosa, ampliamente extensible y compatible con los estándares de la industria. Process Server se basa en la plataforma WebSphere Application Server y brinda la capacidad *Enterprise Service Bus* (ESB), la cual habilita la arquitectura orientada a servicios (SOA) en la

empresa. WebSphere Integration Developer es la herramienta de desarrollo y montaje del proceso de negocios que luego pueden implementarse en Process Server para ejecutarse directamente.
Considerando que Integration Developer y Process Server ya incorporan un editor y un motor de reglas de negocios, estos solamente pueden usarse para implementar reglas o tablas de decisión simples con una interacción de usuarios de negocios limitada. En la mayoría de los casos este motor de reglas embebido no logra cumplir con requisitos complejos por lo que se necesita un sistema de gestión de reglas de negocios dedicado y poderoso que soporte todo el ciclo de vida de modelado, ejecución y gestión de reglas de negocios, integrable fácilmente con productos BPM.
ILOG JRules es un sistema de gestión de reglas de negocios que permite tanto a usuarios del dominio y técnicos poder escribir y mantener la lógica de decisión de las aplicaciones.
ILOG JRules brinda una forma sistemática de modelar, implementar e implantar reglas de negocios y soporta la colaboración de manera ordenada y eficiente. Este producto incluye herramientas personalizadas en base a las habilidades y el conocimiento de usuarios individuales que permiten que los administradores de políticas, los analistas de negocios y los desarrolladores obtengan el soporte que necesitan para aprovechar al máximo sus BRMS [10].
Posee tres componentes principales, retratados en la Figura 4.11:

- Rule Team Server: un repositorio que permite separar las reglas del código de las aplicaciones, con el mismo la lógica de decisión puede ser gestionada, haciendo más sencillo entender y actualizar la lógica de negocio. El repositorio utiliza lógica consolidada desde diversas aplicaciones y fuentes de información de tal forma que pueda ser compartida y reutilizada a lo largo de la organización.
- Rule Studio: herramientas (eclipse, web) que permiten a los expertos del negocio definir y gestionar lógica de decisión que fue previamente alojada en el código. Comprende
- Rule Execution Server: un motor de reglas que permite a los sistemas de producción acceder y ejecutar lógica de decisión gestionada en el BRMS.

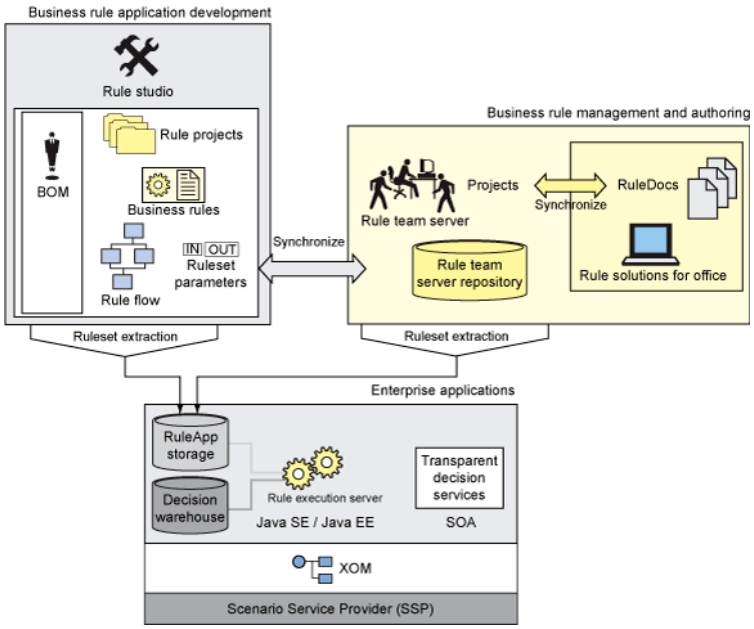

Figura 4.11. Arquitectura de ILOG JRules

Para que una aplicación logre comunicarse con el servidor de ejecución de reglas, ésta debe contar con ciertas credenciales de seguridad. Para lograr esto desde Webpshere Process Server es necesario en una primera instancia habilitar la seguridad global (*global security*), luego se deberá crear una serie de grupos que interactuaran con JRules:

- Administrators (Administradores)
- Deployers (Encargados de desplegar reglas)
- Monitors (Encargados de monitorear el servidor)

Se deberá crear un usuario para cada uno de estos grupos, tal proceso es ilustrado en la Figura 4.12. Finalmente se mapearan los roles de seguridad definidos por el servidor de ejecución con los grupos de usuarios correspondientes.

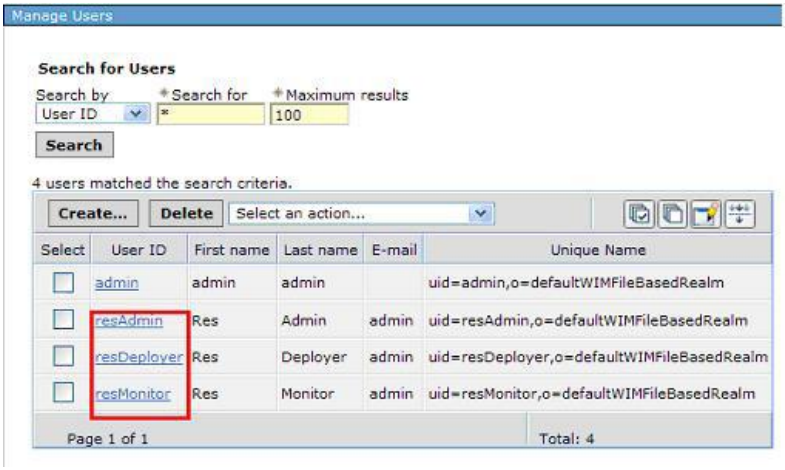

Figura 4.12.Definición de usuarios de JRules en WebSphere Process Server

La integración entre JRules y WebSphere se realiza tal y como lo describe la Figura 4.13. En vez de tener la responsabilidad de tomar una decisión, el servidor de procesos se comunica con el gestor de reglas mediante una tecnología provista (*SCA, Web Services, session bean, message bean, POJO*). Se envían parámetros que provienen del modelo de datos organizacional junto con la identificación del conjunto de reglas a ejecutar, el gestor recibe tal información, la analiza internamente mediante la utilización del motor y retorna un resultado.

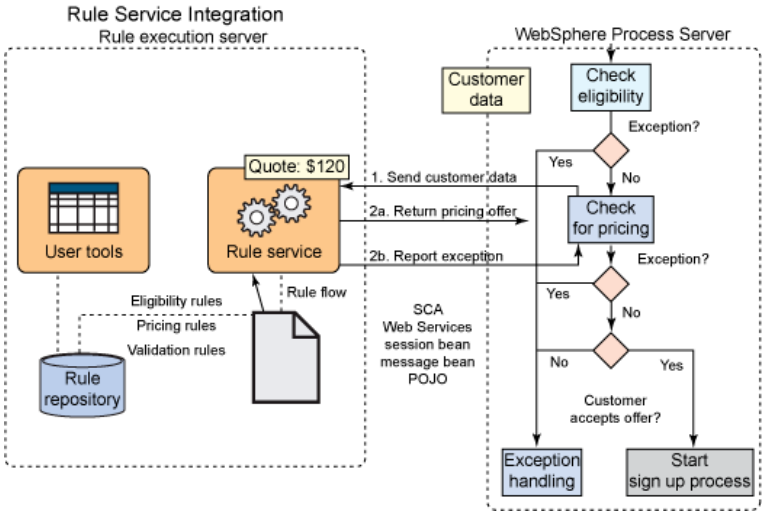

Figura 4.13. Ejemplo de integración de ILOG JRules con WebSphere Process Server

Basándose en *XOM*, modelo de objetos de ejecución definido en la aplicación de reglas, la comunicación entre los componentes se puede realizar a través de diferentes métodos, como POJO (*Plain Old Java Object*), EJB (*Entreprise Java Bean*) (remoto o local), JMS (*Java Message Service*), entre otros. Al mismo tiempo, JRules también proporciona un servicio de decisiones transparente para que el cliente, en este caso Process Server, pueda acceder al conjunto de reglas a través de un servicio web usando *SOAP/HTTP* [10].

Invocación de servicio de reglas usando una sesión POJO
La invocación *SCA POJO*, que consiste en el manejo de objetos planos, es muy eficiente porque no requiere de serialización o desearialización de parámetros de entrada y respuesta. Sin embargo, este método soporta únicamente la invocación local por lo que el servidor de ejecución JRules deberá estar ubicado en la misma JVM que Process Server.
Invocación de reglas usando una sesión EJB
Puede acceder a los servicios de reglas implementados en ILOG Jrules a través de una invocación de EJB, ya sea local o remota.
Invocación de reglas usando un servicio web
Puede acceder al servicio de reglas desde dentro de ProcessServer a traves de un servicio web. JRules proporciona los dos tipos de servicios de decisiones transparentes:

- **Servicio de decisiones transparentes alojado**: Se trata básicamente de un conjunto de reglas implementado como servidor Web. Se instala e integra en el mismo servidor de aplicaciones donde se aloja Rule Execution Server.
- **Servicio de decisiones transparentes monitoreado**: Es generado por Rule Studio y reside en el mismo servidor de aplicaciones donde se encuentra Rule Execution Server, pero no se integra con este sino que solo accede a el para ejecutar reglas.

En el caso del servicio de decisiones transparentes monitoreado, los únicos servidores de aplicaciones soportados son JBoss y Tomcat. Por lo tanto, para la integración con Process Server se usara únicamente el servicio de decisiones transparentes alojado.

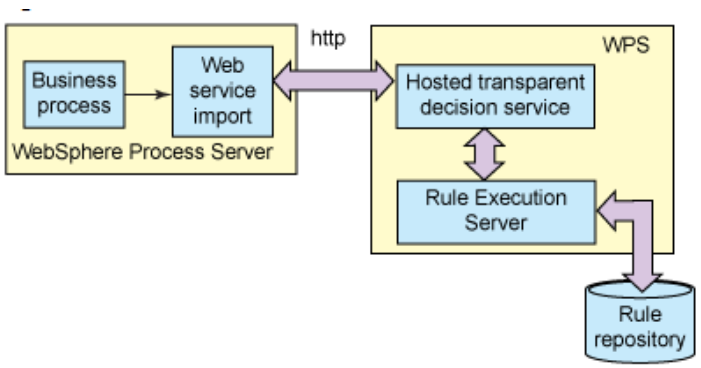

Figura 4.14.Invocación de JRules utilizando un servicio web

Invocación de reglas usando JMS
Un proceso de negocios puede invocar el conjunto de reglas que se esta ejecutando en JRules asincrónicamente usando los estándares de mensajera de *Java Message Service* (JMS).

4.2.4 Aspectos a considerar en la comunicación WebSphere Process Server + ILOG JRules:

La integración de estas herramientas se realiza en el mismo marco empresarial (IBM) que produce una conexión más fuerte, esto facilita la comunicación entre componentes. Haremos un resumen de los puntos relevantes encontrados:

Modo de Acceso:
Como detallamos anteriormente JRules requiere de un proceso de autenticación para permitir su acceso por parte de aplicaciones externas, en este caso WebSphere nos permite utilizar usuarios definidos en el mismo servidor de procesos que mediante un rol particular ligado a la gestión de reglas puede solicitar el permiso para interactuar.
Intercambio de datos:
WebSphere realiza peticiones al motor de reglas enviando parámetros particulares y espera un resultado como puede ser un mensaje de éxito o fallo.
Comunicación
Como quedo detallado el intercambio de información se puede dar mediante la utilización de diversos mecanismos provistos por el entorno.

4.2.5 Comparación de casos de estudio

Podemos realizar una comparación de los casos estudiados, por un lado estamos hablando de productos con licencias diferentes, lo que permite un estudio mas amplio en el caso del software libre. La gestión de usuarios en la conexión entre Bonita y Drools implica que se indique en cada conector los datos de autenticación ya que cada herramienta cuenta con su propio control de usuarios. En cambio, para el caso de WebsPhere Process Server y JRules podemos tener usuarios definidos en el motor de procesos que sirvan de autenticación directa en el gestor de reglas.
Al referirnos a los datos intercambiados notamos que para el caso de Process Server y JRules la comunicación se realiza con parámetros simples, mientras que en el otro caso Bonita y Drools realizan una exposición superior del modelo de datos con particularidades en cada caso, tengamos en cuenta que los productos no son de la misma empresa por lo que tienen configuraciones diversas.
En cuanto al último aspecto, el de comunicación, notamos una variedad de alternativas en los productos IBM que el resto no cuenta.

Una vez definidos los casos de estudio y siendo estos comparados, podemos encaminarnos al desarrollo de una propuesta arquitectónica que contemple los aspectos considerados esenciales en un proceso de comunicación entre un sistema gestor de procesos y uno de reglas.

5

5. Una propuesta arquitectónica de integración

El objetivo de este capitulo es definir una propuesta arquitectónica integral de las herramientas BPMS y BRMS, cumpliendo así el propósito de este trabajo. Para llegar a la meta establecida nos valdremos del estudio realizado previamente, tanto de los conceptos generales que rodean a ambas tecnologías como así también de los distintos casos de estudio en los que ambas se vieron involucradas en forma conjunta. Iniciaremos explicando los motivos que nos llevan a tener una metodología de integración hasta llegar al modelo arquitectónico en cuestión.

5.1. Motivación

En el capitulo anterior se expusieron dos casos concretos en donde se realiza un intercambio entre una herramienta gestora de procesos y otra de reglas de negocio. En el caso en que una organización desee implementarlas en forma conjunta tendrá que conocer que aspectos son relevantes a la hora de realizar tal tarea, lo que siempre tiene un impacto en el tiempo de implementación de tal arquitectura. Seria de utilidad conocer de antemano las bases de datos que comparten ambas herramientas para hacer más sencilla y eficiente su implementación en conjunto, se necesita de un modelo a seguir que marque las bases fundamentales a tener en cuenta en tal proceso. Es por eso que dado el estudio realizado sobre herramientas del mercado consolidadas es que podemos constituir una propuesta arquitectónica que identifique una metodología de integración.
A continuación haremos un resumen de las características particulares encontradas al momento de querer integrar, lo que nos llevará a poder definir un esquema a seguir.

5.2 Seguridad en el acceso

Un aspecto fundamental en cualquier sistema de información es del de control de acceso a los datos, para esto se implementan políticas de seguridad organizando el acceso por usuarios, grupos y roles.
Las herramientas BPMS proveen mecanismos de administración de usuarios que interactuarán en los distintos procesos, permitiendo otorgar diferentes roles a los mismos y vincularlos así con permisos específicos para ejecutar una tarea. Por otro lado, los sistemas de gestión de reglas de negocio también cuentan con su propio módulo encargado del manejo de usuarios; estos tendrán permisos particulares, ya sean de creación edición o eliminación de reglas, otros solo podrán visualizarlas. Como estudiamos en el capitulo anterior, conectar BPMS y BRMS implica que se deba respetar el acceso a cada componente, implicando costos de mantenimiento específicos dependiendo el caso, por ejemplo en la comunicación entre BonitaOS y Drools observamos que es necesario indicar un usuario de acceso al repositorio para ejecutar una regla, no teniendo en cuenta el usuario que opera en Bonita. Podemos pensar en unir la gestión de usuarios utilizando un repositorio en común, tal como lo ilustra la Figura 5.1, ya que nos facilitará la administración y mantenimiento del conjunto de

usuarios del dominio para el caso en que se agreguen nuevos o dejen de existir otros. El administrador del sistema integrado será el encargado de gestionar tal repositorio.

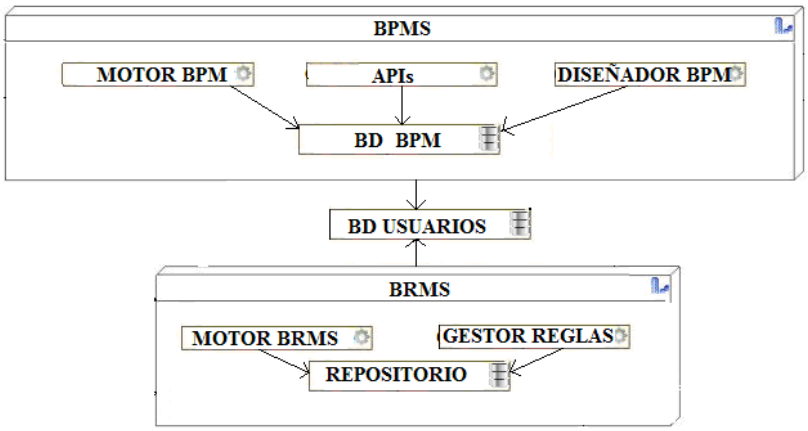

Figura 5.1. Centralización del repositorio de usuarios

5.2.1. Formas de gestionar el acceso centralizado:

Podemos pensar en una serie de alternativas que nos permitan centralizar los usuarios de las herramientas BPMS y BRMS, por un lado la información que deseamos almacenar es acotada ya que nos centraremos en nombres de usuario, contraseña, grupos y roles sin que esto quite el agregado de información adicional.

La primera alternativa es la de almacenar la información de autorización en una **base de datos centralizada**, para esto deberíamos poder vincular los componentes con la misma. Esto requeriría la configuración de las suites para trabajar con controladores que puedan conectarse a la base, además de realizar consultas a la misma. Otra posible alternativa es la de usar un **servicio de directorios**, una opción muy empleada para almacenamiento de información de usuarios y cuyo concepto pasaremos a explicar a continuación.

Servicio de directorios:

Un directorio es una base de datos optimizada para lectura, navegación y búsqueda. Los directorios tienden a contener información descriptiva basada en atributos y tienen capacidades de filtrado muy sofisticada. Los directorios generalmente no soportan transacciones complicadas ni esquemas de vuelta atrás como los que se encuentran en los sistemas de bases de datos diseñados para manejar grandes y complejos volúmenes de actualizaciones. Las actualizaciones de los directorios son normalmente cambios simples, siempre y cuando estén permitidos. Los directorios están optimizados para dar una respuesta rápida a grandes volúmenes de búsquedas.

Hay muchas formas diferentes de proveer un servicio de directorio. Diferentes métodos permiten almacenar distintos tipos de datos, tener distintos requisitos sobre como la información ha de ser referenciada, consultada y actualizada, como es protegida de los accesos no autorizados, etc. Algunos servicios de directorio son locales, es decir, proveen el servicio a un contexto restringido. Otros servicios son globales y proveen servicio a un contexto mucho más amplio (como por ejemplo, Internet). Los servicios globales normalmente son distribuidos, esto significa que los datos están repartidos a lo largo de distintos equipos, los cuales cooperan para otorgar el servicio. El servicio DNS (*Domain Name System*) es un ejemplo de un sistema de directorio globalmente distribuido [11].

Componentes de un directorio:
Un directorio contiene una colección de objetos organizados en una estructura de árbol. El modelo de nombres LDAP *(Lightweight Directory Access Protocol*) define como las entradas son identificadas y organizadas. Las entradas son organizadas en una estructura en forma de árbol denominada "*Directory Information Tree*" (DIT) basándose en su nombre de distinción (DN, *Distinguished Name*). Un DN es un nombre único que identifica en forma única una entrada particular. Los DNs están constituidos por una secuencia de nombres distinguidos relativos (RDNs., *Relative Distinguished Name*). Cada RDN en un DN corresponde a una rama del DIT abarcando desde la raíz del mismo hasta la entrada de directorio. Un DN esta compuesto por una secuencia de RDNs separados por comas, como puede ser cn=thomas, ou= itso, o=ibm. La Figura 5.2 ilustra un ejemplo de la organización de un directorio [12].

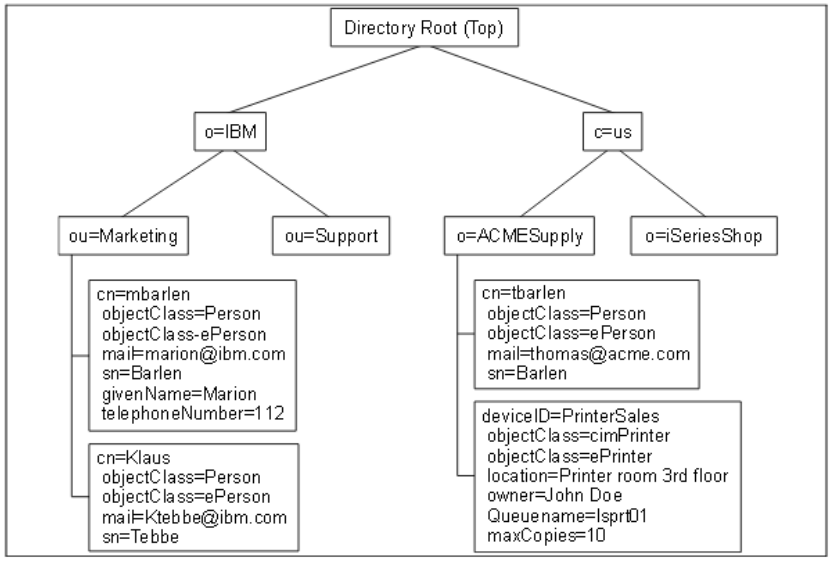

Figura 5.2. Ejemplo árbol DIT

Directorio versus Base de datos:
Un directorio es una base de datos especializada que tiene características que la apartan de las bases de datos relacionales de propósito general. Una característica particular de los directorios es que poseen muchos mas lecturas que actualizaciones. Cientos de personas buscarán un número de teléfono particular, o miles de clientes de impresión buscarán las características de una impresora particular, pero el número de teléfono o las características de las impresoras raramente cambien.
Al poder soportar gran cantidad de lecturas, los directorios están optimizados para tal propósito. El acceso para escritura debe ser limitado a administradores de sistemas o al dueño de cada parte de la información. Una base de datos de propósito general, en cambio, necesita realizar soporte a las aplicaciones como reservas de aerolíneas, aplicaciones bancarias, que tienen un gran caudal de actualizaciones, por lo que esta optimizada para tales operaciones. Considerando que los directorios están diseñados para almacenar información relativamente estática no terminan siendo apropiados para almacenar información que cambia rápidamente.
Otra diferencia es que la mayoría de las implementaciones de directorios aun no soportan transacciones.
Las bases de datos de propósito general permiten almacenar una gran variedad de tipos de datos mientras que los directorios pueden resultar limitados en tal aspecto, ya que manejan tipos simples. Por ejemplo, un directorio especializado para información de contacto puede ser limitado a almacenar solo información personal como nombres, direcciones y números de teléfono. Si un

directorio es extensible puede ser configurado para almacenar una variedad de tipos de información haciéndolo mas útil a una variedad de programas.
Otra diferencia importante entre un directorio y una base de datos de propósito general es la forma en que se accede a la información. La mayoría de las bases de datos soportan el lenguaje de consulta SQL que permite actualizaciones complejas y funciones de consulta al costo del tamaño del programa y de la complejidad de la aplicación. Los directorios como los LDAP, en cambio, usan un protocolo simplificado y optimizado que puede ser usado en pequeñas y relativamente simples aplicaciones [13].

Ventajas del uso de directorios:
Un directorio por si solo almacena la información necesaria por una aplicación particular y no es accesible por otras aplicaciones, probablemente solo almacene un tipo específico de información, no tienen capacidades de búsqueda general, no soportan replicación ni un esquema de particiones y probablemente no tengan un conjunto de administración completo. En tal ambiente, cada aplicación crea y gestiona su propio directorio lo que rápidamente complica la administración. La misma dirección de mail almacenada por una aplicación de calendario puede también ser almacenada en una encargada de los mails y por otra que notifica a los operadores del sistema sobre problemas técnicos. Mantener múltiples copias de la información actualizada y sincronizada es difícil, especialmente cuando diferentes interfaces de usuarios e incluso distintos administradores de sistemas están involucrados. Lo que se necesita es centralizar. Si los desarrolladores pueden asegurar de la existencia de un servicio de directorios, entonces los directorios particulares de cada aplicación no serían necesarios. Sin embargo, se deben tener en cuenta todos los problemas mencionados, debe estar basado en un estándar soportado por varios proveedores en distintas plataformas y debe ser accesible a través de una API estándar. Ya que la mayoría de los usuarios y las aplicaciones van a acceder y depender del directorio común, este debe ser robusto, seguro y escalable. Cuando tal infraestructura de directorios se encuentra activa, los desarrolladores pueden dedicar su tiempo en las aplicaciones y no necesitan pensar en los directorios ya que esto pasa a ser responsabilidad del servicio, por lo que se gana tiempo considerable. De la misma manera en que en la actualidad se confía en la infraestructura de comunicaciones de TCP/IP y RPC para liberar cuestiones de comunicación de bajo nivel, es posible delegar la labor de los directorios a un servidor, particularmente LDAP es el protocolo a ser utilizado para acceder esta infraestructura común. Como HTTP y FTP, LDAP se ha convertido en una parte indispensable del conjunto de protocolos de Internet. Almacenar los datos en un directorio y compartirlos a lo largo de las aplicaciones nos ahorra tiempo y dinero haciendo que el esfuerzo en administración y recursos del sistema se vea reducido. El numero de aplicaciones que soportan los directorios LDAP esta en constante incremento.

A modo de conclusión podemos decir que la centralización del repositorio de usuarios permite reducir costos de mantenimiento además de proveer un enfoque integro de los participantes de una solución BPMS+BRMS. Existen dos alternativas principales para el almacenamiento de usuarios como pueden ser las bases de datos relacionales o los directorios. En el primer caso se debe tener en cuenta que una base de datos relacional normalmente soporta operaciones extras que no tienen utilidad alguna para el caso deseado, las herramientas deberán poder conectarse utilizando controladores específicos al tipo de base de datos empleada. En el caso de utilizar un servicio de directorios estos están optimizados para operaciones de lectura, para el manejo de usuarios puede resultar la mejor opción dado que las actualizaciones (es decir, la escritura) se realiza en pocas ocasiones y las mismas son espaciadas en tiempo. Se debe tener en cuenta que los sistemas deberán poder utilizar algún protocolo de acceso como LDAP para poder interactuar con el directorio, siendo muchas las herramientas que en la actualidad lo implementan.

5.3. Modelo de datos

La ejecución de reglas de negocio mediante el uso de un BRMS impacta directamente en el modelo de datos de nuestras aplicaciones. Al implementar un proceso sobre un BPMS y teniendo este reglas asociadas, nos vemos ante la necesidad de sincronizar las herramientas para administrar datos en común.

Los BRMS suelen utilizar un modelo de datos liviano, es decir, utilizan solamente los datos del dominio que sufren algún impacto por la ejecución de las reglas. Los BPMS, en cambio, suelen manejar gran cantidad de información organizacional. Podemos plantear dos metodologías:

1. Dos modelos de datos: Gestionar ambos modelos en forma separada y sincronizar la información en común (ida y vuelta). Para esto necesitaremos algún módulo encargado de unirlos.
2. Un único modelo de datos: el foco estará siempre sobre el modelo de datos organizacional (que utiliza el BPMS) y por lo tanto el sistema gestor de reglas de negocio deberá interactuar directamente con el mismo, sin perjuicio de que exista algún componente intermedio que se encargue de lidiar con cuestiones especificas de implementación para cada caso [9][10].

La metodología BPM trabaja sobre la información de la organización para dirigir las actividades de los distintos procesos. Al trabajar con sistemas gestores de procesos de negocio una fase muy importante en la implementación de nuestro modelo de proceso es el diseño del modelo de datos. Para esto las herramientas nos permiten definir los datos que se verán afectados o tendrán relevancia durante la ejecución del proceso, sin embargo la fuente de datos no solo puede provenir de la definición interna del proceso sino que también estos sistemas nos permiten el acceso y modificación de datos provenientes de diversos repositorios, bases de datos, y demás fuentes de información organizacional que pueda existir con anterioridad al montaje de nuestros procesos.

Las herramientas gestoras de reglas de negocio se apoyan en información organizacional para poder definir y ejecutar las reglas contenidas en su repositorio. Se pueden plantear dos formas de gestión de su modelo de datos, por un lado se puede crear un modelo de datos acotado que sea útil a los fines del motor de reglas y permita conocer cuales son los datos necesarios para la ejecución de reglas y por el otro esta la posibilidad de contemplar toda la información organizacional, conocerla en su totalidad y poder interactuar en forma directa con ella.

5.3.1. Primer Caso: Dos modelos de datos

Podemos gestionar dos modelos de datos, por un lado el organizacional que es el importante, cuyos datos nos sirven a la hora de tomar los caminos de los flujos en nuestros procesos, aplicar los resultados de la ejecución de los mismos, realizar minería de datos, etc. y por el otro un modelo acotado que solo conocerá el sistema gestor de reglas de negocio y le servirá solamente a sus propios fines, tal enfoque se describe en la Figura 5.3. Pensar en utilizar esta metodología nos lleva a pensar directamente en las distintas formas de implementarlas y cuales son sus ventajas y desventajas.

Una vez definido el modelo de datos organizacional (puede ser antes o durante la implementación de una gestión por procesos) al considerar integrar un sistema gestor de reglas de negocio debemos pensar que datos son relevantes para le definición de reglas y su posterior ejecución. Este proceso nos consumiría cierto tiempo ya que conlleva conocer que reglas se van a implementar para poder así obtener el conjunto específico de datos a gestionar. Una vez definidos los datos con los que se va a trabajar debemos considerar de que manera se deberá sincronizar la información proveniente de los sistemas externos (en nuestro caso particularmente el BPMS) para que pueda ser recibida, interpretada y utilizada en la ejecución de reglas para así poder devolver resultados. Al tener dos modelos de datos diferentes es necesario algún modulo intermedio que los mantenga sincronizados, seguramente la tarea de este componente será la de convertir datos de entrada estructurados de una forma en otra tanto en la interacción BPMS – BRMS como viceversa.

Este enfoque tiene como ventaja para el sistema gestor de reglas que le permite trabajar con un modelo mas pequeño y esto reduce costos en ejecución, no es necesario mover grandes cargas de

información organizacional en un dominio que no lo requiere, la funcionalidad del motor de reglas es la de ejecutar las mismas según las definiciones provistas y nada mas. Como desventajas podemos decir que conlleva un gran costo el tener que definir nuevas estructuras de datos o modificar las existentes cada vez que se define una nueva regla, además el modulo intermedio deberá conocer cuales han sido los cambios para modificar su funcionamiento y seguir sincronizando la información. Funcionalmente puede resultar muy engorroso tener que programar el componente sincronizador ya que se deben tener en cuenta todos los mapeos necesarios y chequear continuamente los cambios a realizar. Si intentamos integrar componentes escritos en distintos lenguajes, que utilizan modelos de datos de diverso tipo como orientado a objetos, estructurado, con un lenguaje de marcado, entre otros, se puede complicar bastante el diseño y ejecución de este sistema.

Figura 5.3. Enfoque "Dos modelos"

5.3.2. Segundo caso: Un modelo de datos

Los sistemas gestores de procesos de negocio utilizan información organizacional para ejecutar los procesos, siendo esta información definida en el sistema de gestión mismo o extraído desde repositorios o bases de datos preexistentes. Los sistemas gestores de reglas de negocio trabajan con datos organizacionales para ejecutar sus reglas. Podemos pensar en tener un solo modelo de datos organizacional en el que ambos sistemas trabajen, tal como lo ilustra la Figura 5.4. Como ventaja podemos citar que este mecanismo no necesita mantenimiento constante ya que las operaciones se apoyan sobre la misma definición de datos por lo que no es necesaria ninguna interpretación diferente. Para tener un enfoque integrador de las tecnologías BPMS y BRMS podemos decir que este enfoque trata de integrar los datos utilizando una única fuente lo que permite una visión mas en conjunto de las herramientas trabajando sobre los mismos datos. Esto no quita que exista algún mecanismo intermedio que permita adaptar herramientas construidas en distintos lenguajes u con distintas maneras de trabajar con los datos, pero lo importante es que no necesitemos mantener modelos distintos constantemente sino que se trabaja sobre la definición de uno solo. Como desventaja podemos decir que el sistema gestor de reglas de negocio pasa a conocer mas información de la que necesita, esto es un detalle mas a nivel conceptual que funcional ya que no afectaría a la eficiencia de nuestras aplicaciones.

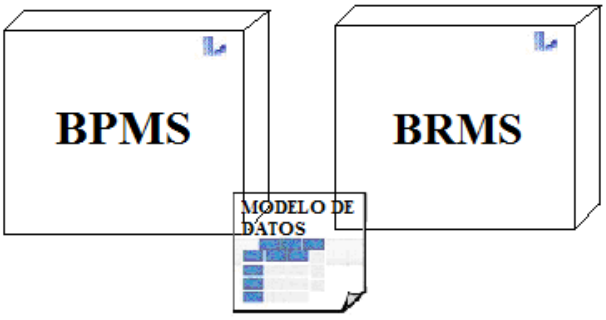

Figura 5.4. Enfoque "Un modelo"

Podemos decidir entonces que metodología implementar, la que mejor se ajuste a nuestro caso, teniendo en cuanto las ventajas y desventajas de cada una.

5.4. Comunicación

Como estudiamos en el capitulo anterior existen diversos mecanismos de comunicación posibles entre un BPMS y un BRMS, vimos en el caso de BonitaOS que mediante un conector nativo se accede vía REST a la API de Drools para ejecutar una operación y este retorna un resultado. En el caso de WebSphere Process Server y JRules la comunicación se puede realizar de varias maneras utilizando tecnologías diversas que son implementadas comúnmente por aplicaciones pertenecientes a IBM como puede ser POJO, JMS, WebServices, entre otras. Ahora bien, considerando tal variedad de opciones estamos sujetos a las posibilidades de comunicación que tengan las herramientas a integrar, debemos tener en cuenta que no siempre vamos a tener interfaces REST o SOAP disponibles del lado del gestor de reglas ya que cada uno elige una opción particular. Es por esto que consideramos apropiado contar con un componente intermedio que logre abstraer aspectos de comunicación entre BPMS y BRMS. Este módulo tendrá el deber de proveer una interfaz al gestor de procesos para que envíe ordenes y por el otro deberá ser capaz de enviar tales comandos hacia el gestor de reglas utilizando la interfaz que el provea, esto incluye el retorno de resultados posibles[9][10].

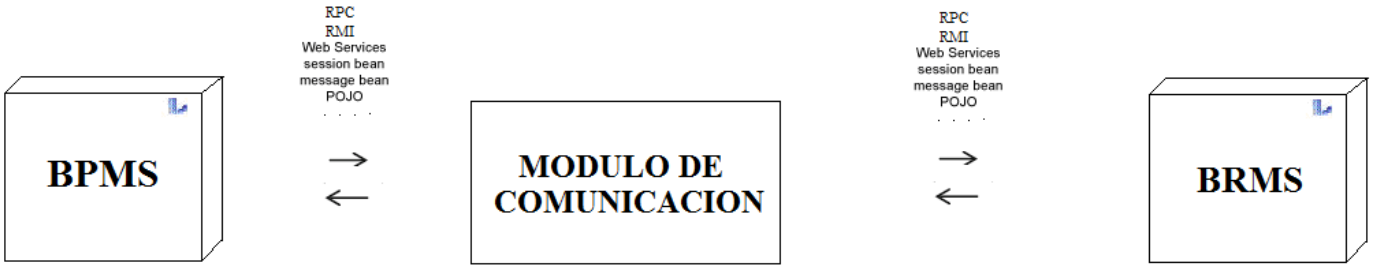

Figura 5.5. Comunicación con modulo intermedio

La Figura 5.5 muestra el concepto aplicado, el módulo de comunicación utiliza la tecnología que corresponda y se elija, según se trate de interactuar con el gestor de procesos o de reglas. Existe un abanico de posibilidades para seleccionar como por ejemplo RPC (*Remote Procedure Call*), RMI (*Remote Method Invocation*), WebServices (REST,SOAP), ESB (*Enterprise Service Beans*), objetos POJO, entre otros.

5.5. Asociación Tarea – Regla

La interacción entre un BPMS y un BRMS no se requiere en forma constante sino que son momentos específicos en donde se realiza basándose principalmente en la ocurrencia de algún evento particular. Los modelos de procesos de negocio describen las diversas tareas que ejecutan los usuarios o el sistema y es durante el desarrollo de esas tareas que se produce algún evento que puede requerir la intervención de un sistema gestor de reglas. Es durante la fase de diseño donde se indica en que tareas y en que momento (inicio, finalización) se procede a invocar al BRMS. La interacción siempre se realiza mediante el pedido de ejecución de una o varias reglas de negocio, lo que puede requerir el pasaje de parámetros hacia el gestor de reglas. Este mecanismo se mostró claramente en funcionamiento cuando se estudio el caso de integración BonitaOS y Drools, vimos como se implementaba el conector en una tarea especifica y que requería en algunos casos el pasaje de datos a ser procesados. Ahora bien, para tener un enfoque integrador y proponer un modelo a seguir debemos indicar que en toda relación BPMS y BRMS existirán vínculos entre tareas y reglas con parámetros de por medio, es de interés considerar la posibilidad de gestionar en forma separada tales vínculos por parte de un usuario encargado a la gestión de reglas, descartando gran parte de su

responsabilidad a los usuarios dedicados al diseño del modelo de proceso. Es por esto que proponemos delegar la tarea de relación tarea - regla a un módulo aparte, que se pueda desarrollar por ejemplo mediante una aplicación web.

5.6. Creación de un modulo de integración

Basándonos en los puntos de vinculación detallados anteriormente notamos que existe funcionalidad que puede separarse en módulos específicos como es el caso de la comunicación y la gestión de asociación entre tareas y reglas. Otro aspecto a tener en cuenta es del modelo de datos, si se sigue la metodología de dos modelos se requerirá alguna adaptación, en el caso de un solo modelo, no se garantiza que dadas cuestiones tecnológicas se deba acudir a métodos de mapeo de la información como por ejemplo el objeto-relacional. Es por esto que al encontrarnos con una variedad de funciones relacionadas a la integración podemos pensar en tener un módulo que se encargue de realizar todas las tareas previamente mencionadas.

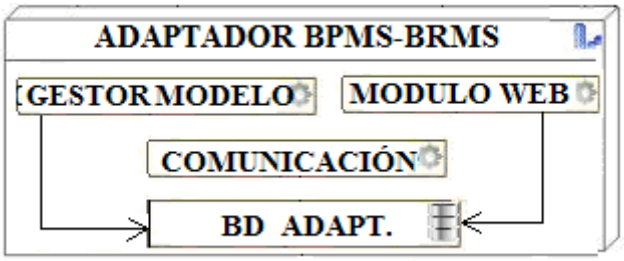

Figura 5.6. Adaptador BPMS-BRMS

Como vemos en la Figura 5.6. llamamos “Adaptador BPMS-BRMS” al módulo de integración, que tiene un conjunto de funcionalidades:

- El **gestor del modelo de datos** es el encargado de lidiar con cuestiones ligadas al modelo de datos organizacional y su relación con los sistemas BPMS y BRMS para poder lograr consistencia en la información
- El **gestor de comunicación** es el encargado de establecer el medio de comunicación entre el sistema gestor de procesos y el de reglas para poder intercambiar órdenes y datos, debe tratar con temas de implementación de los métodos de conexión determinando las tecnologías adecuadas y empleándolas para lograr los resultados esperados.
- El **modulo web** representa en funcionalidad todo lo que detallamos en la sección 5.5 sobre asociación tarea y regla, es decir que es el encargado de proveer alguna interfaz accesible al usuario gestor de reglas para que pueda vincularlas con las tareas correspondientes e indicar alguna particularidad como puede ser un parámetro o información adicional que se requiera.

La componente adaptadora deberá contar con una base de datos local para poder almacenar información relativa al modulo web sobre tareas y reglas asociadas, también podrá ser de utilidad para la gestión del modelo de datos en el caso que requiera almacenar cuestiones vinculadas al mapeo de información o utilización de parámetros.

5.7. Propuesta arquitectónica

Estudiados los aspectos de integración y habiendo propuesto una serie de conceptos a tener en cuenta en cualquier implementación BPMS+BRMS, podemos hacer un resumen de los mismos bajo la representación de una arquitectura que englobe toda la funcionalidad abordada.

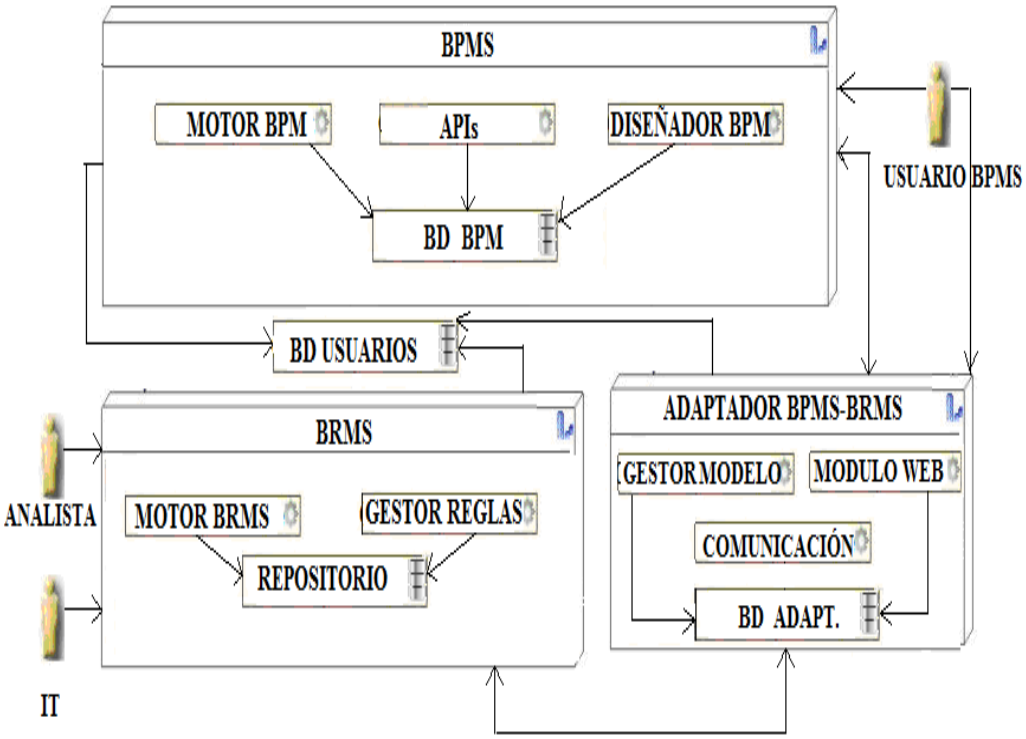

Figura 5.7. Arquitectura BPMS+BRMS propuesta

La Figura 5.7 representa la arquitectura propuesta y engloba todos los aspectos detallados previamente. Denominaremos "Usuario BPMS" a quien interactúa particularmente con el motor de procesos iniciando instancias, ejecutando tareas, entre otras. La arquitectura incorpora además al grupo "IT" que conoce detalles de implementación de la herramienta BRMS y a los usuarios "Analistas" del dominio que no poseen conocimientos técnicos pero que igualmente pueden gestionar las reglas utilizando alguna interfaz que les provea un lenguaje natural. Observamos como se pretende vincular BPMS y BRMS de una forma conveniente utilizando el adaptador intermedio. Un detalle a tener en cuenta es que el adaptador también utiliza el repositorio centralizado para gestionar usuarios ya que debe existir un mecanismo propio de restricción en el acceso a la gestión web que implementa tal componente.

5.7.1. Ventajas de contar con una propuesta arquitectónica

Definida finalmente la arquitectura integradora debemos hacer un resumen de las ventajas que encontramos a la hora de implementarla. En primer lugar, la centralización de la información relativa a usuarios nos permite pensar en una única organización de los mismos y no tener que lidiar con el

mantenimiento asociado a tener varios repositorios por separado. En cuanto al tratamiento del modelo de datos organizacional podemos decir que contamos con dos metodologías marcadas de implementación que pueden llevarse a cabo o no según las tecnologías BPMS y BRMS seleccionadas. La comunicación siempre es una cuestión que consume tiempo en cualquier integración de herramientas informáticas, sean de cualquier índole, es por eso que el delegar tal tarea a una componente intermedia permite flexibilidad dado que no siempre contamos con la posibilidad de que tanto el BPMS elegido como el BRMS puedan comunicarse directamente entre si. En el ultimo aspecto tratado, que es el de la asociación entre tareas y reglas descubrimos un mecanismo interesante para separar tal funcionalidad y delegarla a un usuario encargado de la gestión de reglas. Consideramos que tener una arquitectura de implementación resulta útil acortando tiempos de pruebas y estudio de casos. El contar con un marco metodológico ordena y clarifica la manera en que se pueden vincular las dos herramientas pilares de este trabajo.

6

6. Integrando BonitaOS y Drools

Una vez propuesta y definida la metodología de integración se procederá a implementarla sobre herramientas concretas del mercado, es por eso que seleccionamos al sistema gestor de procesos de negocio BonitaOS y al gestor de reglas Drools para realizar tal demostración. El objetivo de este capítulo es el de mostrar paso a paso el plan de integración de las herramientas elegidas basándonos en el modelo arquitectónico propuesto y en la metodología asociada al mismo.

6.1. Motivación

En el capítulo anterior se expuso la metodología de integración propuesta en este trabajo, por lo que ante la definición teórica de la misma se requiere llevarla al ámbito práctico, verificando su posibilidad de implementación y sirviendo de ejemplo concreto para otro tipo de herramientas que se deseen integrar. Se eligen dos opciones pertenecientes al ámbito del software libre, BonitaOS y Drools, al no ser productos empaquetados podemos conocer su funcionamiento interno, conllevando a una descripción mas clara y detallada del proceso de integración, pudiendo observar las interfaces que se proveen en cada lado de la misma.

6.2 Herramientas elegidas

Al contar con un amplio rango de posibilidades en la elección de herramientas BPMS y BRMS nos declinamos por el subconjunto de las versiones libres, como mencionamos anteriormente el contar con documentación y la posibilidad de observar como realizan sus funcionalidades nos sirve de gran ayuda a la hora de integrar y clarificar que aspecto se aplico en cada caso. A continuación se hará un breve resumen de las características de los sistemas elegidos, haciendo hincapié en sus componentes principales que nos servirán de base de estudio a la hora de realizar el proceso de integración.

6.2.1. Bonita Open Solution

BonitaOS es una suite de herramientas de código abierto ligada a la construcción de soluciones orientadas a procesos. Uno de sus objetivos es facilitar una solución fácil e intuitiva que permita minimizar el costo de implementación, proporcionando soluciones de BPM flexibles y potentes para las organizaciones. Posee tres componentes principales, reflejados en la arquitectura de la Figura 6.1.

BonitaStudio combina una serie de funcionalidades, este utiliza la notación BPMN para diseñar los modelos, brindando un nivel de estandarización que facilita el poder compartir nuestros procesos entre distintos sistemas. Bonita provee el uso de tablas de decisión para tomar distintos flujos, guarda y administra versiones provisionales del diseño mientras se modela un proceso, nos permite además administrar la organización del personal de la empresa, para ello nos facilita la creación de actores como así también la posibilidad de asignarles roles y grupos determinados. La suite facilita

conectores que sirven como nexo con otras aplicaciones. Además de los conectores que vienen por defecto podemos también crear nuevos. Para cada proceso y tarea disponible podemos editar la plantilla que se mostrará en el portal de Bonita. BonitaOS facilita a través de sus conectores la posibilidad de comunicarse con sistemas externos, por defecto cuenta con una serie de APIs nativas que nos permiten interactuar con diversos servicios líderes [14].

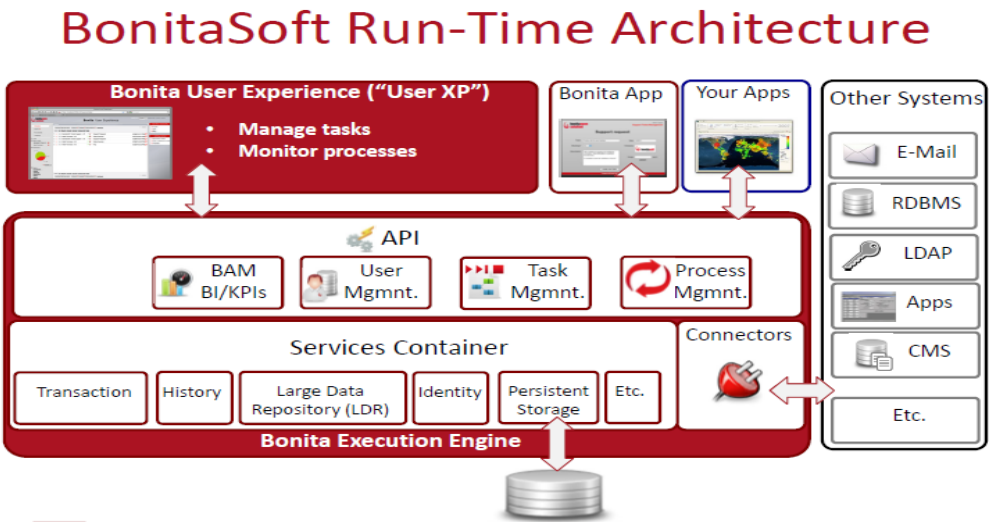

Figura 6.1.Arquitectura BonitaOS

Bonita User XP es un entorno web que permite una completa ejecución y gestión de nuestros procesos por parte de los distintos usuarios. A través de una interfaz amigable permite iniciar nuevas instancias y gestionar las activas (según el rol), además cada usuario tendrá su bandeja de entrada, como ejemplifica la Figura 6.2, donde se mostrarán todas las tareas que puede ejecutar el mismo. El usuario administrador tiene los privilegios para gestionar la organización del sistema (usuarios, grupos, roles, perfiles) y las instancias y tareas en curso. Podemos instalar nuevas definiciones de procesos, desactivar las activas, gestionar la categoría a la que pertenecen, entre otras operaciones. Para un caso particular podemos ver los detalles de su ejecución incluso las variables definidas junto a sus valores, nos permite editarlos en tiempo de ejecución. Se nos permite eliminar los casos activos, registrándolos como "archivados". Además se clasifican las tareas según su tiempo estimado de finalización, considerando si están en riesgo o si se han excedido.

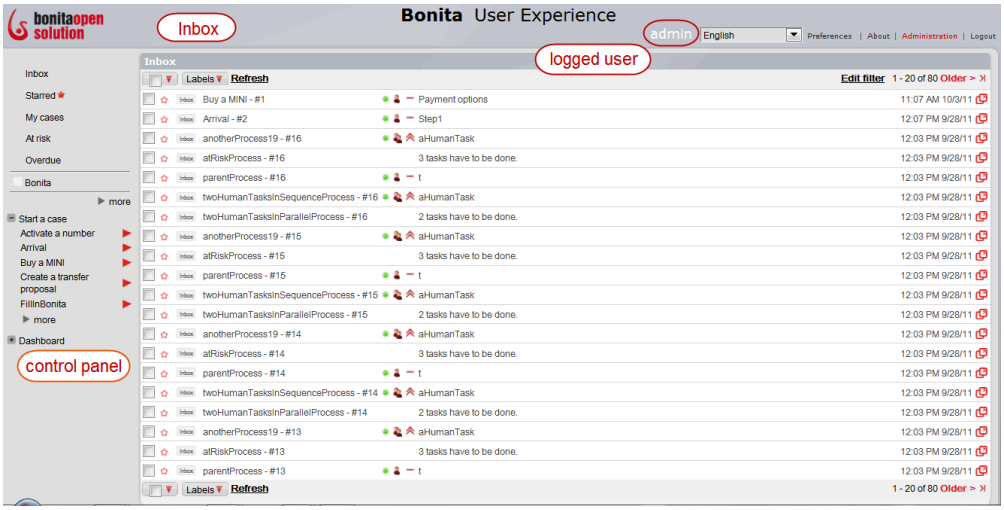

Figura 6.2.Bandeja de tareas en Bonita User XP

El motor de ejecución de BonitaOS, **Bonita Engine** representa la base de procesamiento que corre en segundo plano y conecta Bonita Studio, los formularios creados, Bonita User XP y aplicaciones web customizadas.
BonitaOS confía en su motor para crear, acceder y procesar datos. El motor también gestiona y ejecuta los procesos creados en BonitaStudio, Sin el motor de ejecución no se podrían crear formularios ni tampoco conexiones internas o externas a la API, servicios web o base de datos. Se cuenta con una API escrita en Java que nos permite interactuar en forma directa con el engine.

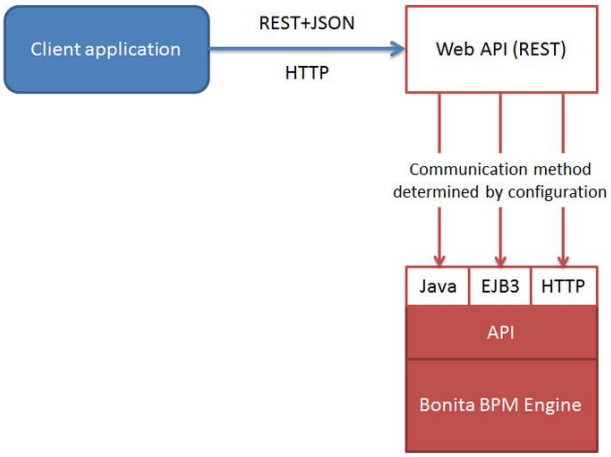

Figura 6.3. Bonita API REST

BonitaOS provee una API de acceso vía REST que permite a clientes desarrollados en lenguajes distintos a Java poder interactuar con el motor de procesos, tal como lo ilustra la Figura 6.3[15].

6.2.2. Drools

Drools es una plataforma de integración de lógica del negocio, escrita en Java. Es un proyecto open source con el soporte de JBoss y Red Hat. Opera bajo la licencia Apache Version 2.0. La plataforma consiste en cuatro módulos principales:

- Drools Expert: El motor de reglas.
- Drools Fusion: Modulo de procesamiento de eventos complejos (CEP, Complex Even Processing)
- Drools Flow: Workflow, combina reglas y procesos en forma conjunta.
- Drools Guvnor: Un sistema gestor de reglas de negocio (BRMS).
- Drools Solver: Modulo opcional, es un algoritmo de búsqueda creado encima del motor de reglas para solucionar problemas de planificación.

A fines prácticos, nos enfocaremos en los componentes Expert y Guvnor que consideramos esenciales a la hora de implementar reglas y nos servirán en el marco de este trabajo. **Drools Expert** puede ser considerado el núcleo del proyecto Drools y es utilizado para especificar, mantener y ejecutar las reglas de negocio [16]. **Guvnor** es un gestor de reglas de negocio, que permite a las personas gestionar las reglas en un entorno multi usuario, permitiendo el cambio en una forma controlada, con interfaces amigables al usuario.

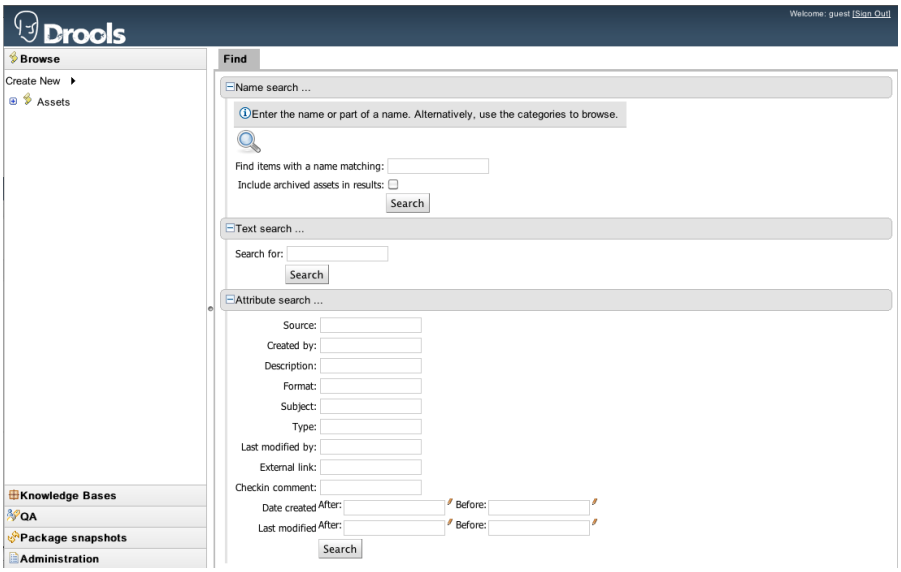

Figura 6.4.Modulo Web para la gestión del repositorio Guvnor

Guvnor es el nombre de los componentes web y de red que gestionan las reglas con Drools. Esto combinado con el motor de reglas y otras herramientas forma el gestor de reglas de negocio, se puede utilizar por varias necesidades como el requerir gestionar versiones, desplegar reglas, acceder por medio de múltiples usuarios de diversos niveles, el no contar con una infraestructura existente para gestionar las reglas, o contar con demasiadas reglas de negocio. El repositorio puede ser utilizado por si solo, o con una herramienta IDE, también puede ser tomado como parte de la aplicación o puede ser un repositorio central de reglas [9]. La Figura 6.4 ilustra la interfaz que se provee por el sistema para la gestión de reglas.
Los roles principales de personas que utilizarían Guvnor pueden ser: Analistas del dominio, expertos en reglas, desarrolladores, administradores.

Las funcionalidades provistas son las siguientes:

Múltiples tipos de editores de reglas (GUI, texto) incluyendo:
- Editor guiado de reglas
- Plantillas de reglas
- Tablas de decisión

Almacenar múltiples reglas en forma conjunta en un paquete
Lenguajes específicos de soporte de dominio
Soporte a eventos de procesamiento complejo
Control de versiones
Pruebas sobre reglas
Validación y verificación de reglas
Categorización
Construcción y despliegue
REST API para manipular reglas
Integración WEBDAV

Partiendo de la noción de que todo BRMS cuenta con un lenguaje de reglas, Drools no es la excepción ya que incorpora un lenguaje nativo para la escritura de las mismas, este formato es liviano en cuanto a puntuación y soporta lenguaje natural y del dominio a través de extensiones que le permiten adaptarse al ambiente del problema. La escritura se realiza en archivos de extensión .drl

donde se pueden definir múltiples reglas, consultas y funciones así como también declaraciones de recursos importados, globales y atributos que son asignados y utilizados.
La estructura básica de un archivo de reglas se puede definir de la siguiente manera:

package “nombre_del_paquete”
imports
globals
functions
queries
rules

La sección de reglas (rules) es la de mayor relevancia y cuenta con la siguiente estructura:

rule “nombre de la regla”
    “atributos”
    when
        “lista de condiciones a analizar”
    then
        “código a ejecutar si se cumplen las condiciones”
end

Ejemplos:

Considerando una empresa donde se desea determinar el cargo de sus empleados y su salario se determinan tres restricciones:

“Si el promedio de conocimiento este entre 8 y 10 el cargo del empleado es el de programador y su salario de $1000”

“Si el promedio de conocimiento este entre 4 y 4 el cargo del empleado es el de líder de proyecto y su salario de $2000”

“Si el promedio de conocimiento este entre 0 y 3 el cargo del gerente es el de programador y su salario de $3000”

El código DRL que representa las reglas descritas se puede escribir de la siguiente forma:

```
rule "Programador"
    when
        empleado : Empleado(promedioConocimientos >= 8, promedioConocimientos
<= 10)
    then
        System.out.println("Programador: " + empleado.getNombre());
        empleado.setCargo("Programador");
        empleado.setSalario(BigDecimal.valueOf(1000));
end

rule "Lider de Proyecto"
    when
        empleado : Empleado(promedioConocimientos >= 4, promedioConocimientos
<= 7)
    then
```

```
        System.out.println("Lider de Proyecto: " + empleado.getNombre());
        empleado.setCargo("Lider de Proyecto");
        empleado.setSalario(BigDecimal.valueOf(2000));
end

rule "Gerente"
    when
        empleado : Empleado(promedioConocimientos >= 0, promedioConocimientos
<= 3)
    then
        System.out.println("Gerente: " + empleado.getNombre());
        empleado.setCargo("Gerente");
        empleado.setSalario(BigDecimal.valueOf(3000));
end
```

Una vez detalladas las generalidades de las herramientas seleccionadas procederemos a iniciar el proceso de integración, para esto nos apoyaremos en las bases establecidas en el capítulo anterior.

6.3 Integrando la gestión de usuarios

Uno de los aspectos a considerar en el proceso de integración es el de la seguridad en el acceso, en este sentido explicaremos el control de usuarios que emplean BonitaOS y Drools. Para el caso del sistema gestor de procesos encontramos que la administración de usuarios se realiza en forma nativa utilizando la gestión de archivos propia del gestor y que posteriormente podemos modificar mediante el módulo User XP, en forma adicional se nos permite cambiar la fuente de usuarios pudiendo implementarla en una base de datos o en un servidor LDAP. Drools trabaja en forma similar, el componente Guvnor otorga la facilidad de administrar usuarios pero también podemos elegir hacerlo desde una base de datos o a través de un servicio LDAP. En el capitulo anterior se mencionaron las ventajas que conlleva realizar una gestión por usuarios utilizando un servicio LDAP por lo que nos decantaremos por esta opción para iniciar el proceso de integración. La idea no es contar con dos repositorios sino que intentaremos unificar el acceso teniendo un gestor centralizado, por lo que los usuarios creados podrán ser utilizados en ambos sistemas.

Al elegir utilizar un servidor LDAP para realizar la integración de usuarios nos vemos obligados a elegir un producto particular que opere bajo este estándar, siguiendo con la línea del software libre y basándonos en la documentación disponible y en su renombre es que nos decantamos por el producto OpenLDAP, que es una implementación libre y de código abierto del protocolo Lightweight Directory Access Protocol desarrollada por el proyecto OpenLDAP. Está liberada bajo su propia licencia OpenLDAP Public License.

Para realizar las pruebas pertinentes se optó por instalar el servidor OpenLDAP bajo un sistema operativo Linux con distribución Centos. Se crea una estructura, visualizada en la Figura 6.5, “ParraJulian” hace referencia a la organización mientras que “repCentral” es la raíz del árbol que comprende los usuarios y roles del sistema. En la siguiente imagen se visualiza, a través del visor LDAP Explorer, la estructura de nuestro repositorio, con dos usuarios a modo de ejemplo.

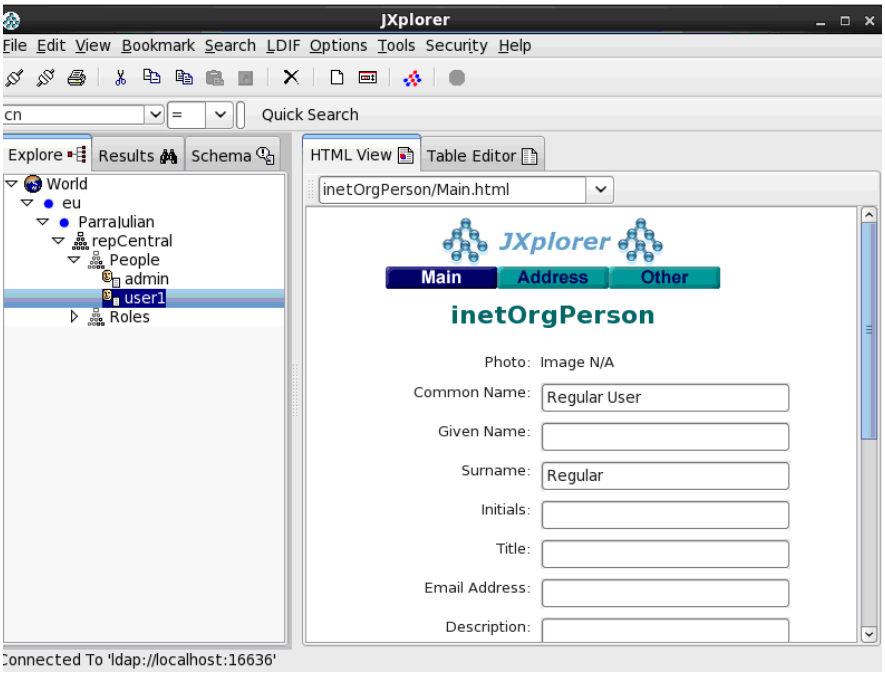

Figura 6.5.Vista del servidor OpenLDAP creado

6.3.1. Conexión de BonitaOS con OpenLDAP:

Se procede a la configuración de BonitaOS para que su acceso se realice a través de los usuarios definidos en el repositorio. Hay que tener en cuenta que los siguientes pasos se realizan sobre el componente User XP alojado en un servidor tomcat, denominaremos "bonitaTomcatDir" a la ruta raíz del servidor mencionado.

En primer lugar hay que abrir el archivo "bonitaTomcatDir"/external/security/jaas-standard.cfg y agregar el siguiente apartado de código [17]:

```
BonitaAuth {
 com.sun.security.auth.module.LdapLoginModule REQUIRED
 userProvider="ldap://host:port/ou=people,dc=example,dc=com"
 authIdentity="uid={NOMBREUSUARIO},ou=people,dc=example,dc=com"
 userFilter="(&(uid={ NOMBREUSUARIO }))"
 useSSL=false
 debug=true;
};
BonitaStore {
 org.ow2.bonita.identity.auth.LocalStorageLoginModule required;
 };
```

Continuamos abriendo el archivo "bonitaTomcatDir"/bonita/ onfig/conf/bonita-server.xml y comentar la siguiente porción de código.

```
<!-PRI comment <authentication-service name='authentication-service'
class='org.ow2.bonita.services.impl.DbAuthentication'>
 <arg><string value='bonita-session:core' /></arg>
 </authentication-service> →
```

Agregamos las siguientes líneas:

```
 <!-Pri added →
 <authentication-service name='authentication-service'
class='com.sun.security.auth.SimpleLdapAuth'>
 <arg><string value='bonita-session:core' /></arg>
 </authentication-service>
 <!-Pri added upto this →
```

El tercer paso es crear una clase Java que sea la encargada de la configuración LDAP, para esto creamos un archivo con nombre "SimpleLdapAuth.Java" con el contenido que se detalla a continuación (a modo de ejemplo):

```
import org.ow2.bonita.facade.exception.UserNotFoundException;
import org.ow2.bonita.services.AuthenticationService;
public class SimpleLdapAuth implements AuthenticationService {

    private String persistenceServiceName;

    public SimpleLdapAuth(String persistenceServiceName) {
        super();
        this.setPersistenceServiceName(persistenceServiceName);
    }

    public  boolean checkUserCredentials(String username, String
password) {
        return true;
    }

    public  boolean checkUserCredentialsWithPasswordHash(String arg0,
String arg1) {
        // TODO Auto-generated method stub
        return false;
    }

    public void setPersistenceServiceName(String
persistenceServiceName) {
        this.persistenceServiceName = persistenceServiceName;
    }

    public String getPersistenceServiceName() {
        return persistenceServiceName;
    }
}
```

Compilamos el archivo creado utilizando el siguiente comando:

```
 Javac –cp ~/bonitaTomcatDir/lib/bonita/bonita-server-5.10.jar
SimpleLdapAuth.Java
```

Esto resultará en un archivo de clase SimpleLdapAuth.class utilizando el jar del servidor de bonita, finalmente creamos un jar del mismo.

```
Jar cvf SimpleLdapAuth.jar SimpleLdapAuth.class
```

Debemos ubicar el jar obtenido en el directorio lib de nuestra distribución, creando la siguiente jerarquía de carpetas:

com/sun/security/auth

Ubicamos entonces el archivo SimpleLdapAuth.class file en la carpeta auth y agregamos el siguiente extracto en el archivo bonitaTomcatDir /conf/logging.properties

```
org.apache.catalina.realm.level = ALL
org.apache.catalina.realm.useParentHandlers = true
org.apache.catalina.authenticator.level = ALL
org.apache.catalina.authenticator.useParentHandlers = true
catalina.org.apache.juli.FileHandler.bufferSize = -1
org.ow2.level = ALL
org.ow2.handlers.useParentHandlers = true
```

6.3.2. Conexión de Drools Guvnor con OpenLDAP:

En el caso de Drools, el componente Guvnor provee como parte de su abanico de funcionalidades la posibilidad de gestionar los usuarios de la aplicación, podemos modificar tal aspecto para que los usuarios se autentiquen contra el servidor OpenLdap montado. Para lograr tal fin debemos editar el archivo $JBOSS_SERVER/conf/login-config.xml (refiriéndonos como $JBOSS_SERVER a la ruta del servidor JBOSS montado con la aplicación Guvnor) reemplazando la configuración de login anterior y con el código detallado a continuación [18]:

```
<application-policy name="guvnor">
<authentication>
<login-module code="org.jboss.security.auth.spi.LdapExtLoginModule" flag="required" >
<module-option name="java.naming.provider.url">ldap://localhost:16636</module-option>

<module-option name="bindDN">cn=DirManager,dc=ParraJulian,dc=xx</module-option>
<module-option name="bindCredential">xxxxxx</module-option>
<module-option name="baseCtxDN">ou=People,o=guvnor,dc=ParraJulian,dc=xx</module-option>
<module-option name="baseFilter">(uid={0})</module-option>
<module-option name="rolesCtxDN">ou=Roles,o=guvnor,dc=ParraJulian,dc=xx</module-option>
<module-option name="roleFilter">(member={1})</module-option>
<module-option name="roleAttributeID">cn</module-option>
<module-option name="roleRecursion">-1</module-option>
<module-option name="searchScope">ONELEVEL_SCOPE</module-option>
</login-module>
</authentication>
</application-policy>
```

Como parámetros relevantes observamos la url del servidor ldap y los nombres de dominio base que forman la estructura de árbol del servidor. Al reiniciar el servidor Jboss que contiene la aplicación Guvnor podremos realizar el login con los usuarios creados.

Definidas las formas de conexión de BonitaOS y Drools con el servidor LDAP podemos adentrarnos en el siguiente aspecto clave, el modelo de datos.

6.4 Integración a nivel del modelo de datos

El segundo aspecto a considerar en el proceso de integración es el referente al modelo de datos organizacional, que en este caso al intentar unir las herramientas BPMS y BRMS se debe tener en cuenta que ambas trabajen en forma adecuada con el mismo. En el capítulo anterior se contemplaron dos posibilidades a la hora de considerar tal aspecto, por un lado podríamos tener dos modelos

diferenciados, el utilizado por el gestor de procesos, normalmente apoyado en la base de datos organizacional y por el otro un modelo acotado que sirve a la funcionalidad del BRMS, el segundo enfoque sugerido es el de unificar, para esto el gestor de reglas pasaría a conocer la estructura de datos manejada por el BPMS y trabajaría directamente con ella. Podemos elegir entonces una de estas dos posibilidades para el proceso de integración entre Bonita y Drools. Estudiando ambas herramientas encontramos que es posible aplicar ambos enfoques ya que Drools puede manejar un modelo de datos separado, con una metodología orientada a objetos típica del lenguaje Java en la que se utilizan clases y objetos para llevar a cabo las operaciones y también esta la posibilidad de importar modelos externos, en el caso que estos sean orientados a objetos la importación seria muy sencilla y no traería problemas, para otro tipo de enfoque como el relacional existe la posibilidad de incorporar un mapeador que permita trabajar con objetos del lado de Drools pero que estos siempre hagan referencia a diversas tablas almacenadas en la base de datos. Ante la posibilidad de implementar ambas alternativas nos decantaremos por la que considera un único modelo de datos, ya que consideramos que pese a tener el trabajo extra de definir las cuestiones relativas al mapeador, nos ahorraremos tiempo a futuro ya que el mantenimiento se haría solo cuando cambie el modelo organizacional, si optáramos por la opción de dos modelos separados tendríamos el costo adicional de tener que construir un adaptador intermedio que se tenga que mantener constantemente ante el ingreso de nuevas reglas en nuestro sistema. Ante la elección de unificar y teniendo cuenta las herramientas seleccionadas nos vemos obligados a recurrir a un trabajo de mapeo utilizando un producto que nos sirva de soporte a tal tarea. BonitaOS provee conectores nativos que permiten trabajar directamente con una base de datos relacional, teniendo en cuenta este aspecto creemos que podemos recurrir al producto Hibernate para realizar el mapeo, siendo este un producto muy utilizado en la práctica y que se adapta correctamente en aplicaciones Java como Drools[21]. Hibernate es un producto ORM (*Object Relational Mapping*) que permite a las aplicaciones lograr persistencia en la información, es decir que en general permite a las aplicaciones orientadas a objetos extender el tiempo de vida de tales entidades, para esto se apoya en bases de datos relacionales(RDBMS, *Relational Database Management Systems*). Es Open Source, pertenece a la gama del software libre siguiendo la línea de las herramientas que fueron seleccionadas en este trabajo. Los mapeadores como Hibernate surgen en la búsqueda de reducir las desventajas inherentes a trabajar con objetos y tablas relacionales, ya que existen consecuencias en distintos aspectos:

A nivel de granularidad ya que podemos tener un modelo de objetos con mas clases que el número de tablas en la base de datos.
El concepto de herencia es algo inherente en los lenguajes orientados a objetos mientras que no existe tal concepto en los sistemas gestores de bases de datos relacionales.
Las asociaciones son representadas como referencias unidireccionales en los lenguajes orientados a objetos mientras que en un RDBMS se utiliza la noción de claves foráneas. Si se necesitan relaciones bidireccionales en Java se debe definir la asociación dos veces. Asimismo no se puede determinar la multiplicidad de un a relación mirando el modelo de objetos.
La forma en que se acceden a los datos en Java es totalmente diferente de cómo se hace en una base de datos relacional. En Java, se puede navegar de una asociación a otra caminando sobre la red de objetos, esto no es una forma eficiente en una base de datos relacional ya que siempre se intenta minimizar el numero de consultas SQL y por lo tanto se deben usar varios JOINs y filtros antes de que se puede ir sobre la red de objetos [19].

Ante el panorama definido lo que necesitamos ahora es aplicar Hibernate en Drools para poder trabajar con las entidades definidas en nuestro modelo de datos, en el caso de BonitaOS como ya mencionamos este trabajará directamente con los datos a través de sus conectores nativos.
La instalación de Hibernate se realiza descargando las librerías desde la pagina oficial, normalmente el proceso de mapeo se realiza teniendo un modelo de objetos organizacional que luego utilizando el

mapeador de Hibernate es plasmado en una base de datos relacional. En nuestro caso el proceso es el inverso, considerando que el modelo de datos ya existe con anterioridad en un base relacional, suponiendo que el sistema BPMS es el primero que se implanta para dar curso a los procesos de negocio y luego este da lugar a la posible incorporación de un producto BRMS que se tendrá que adaptar a tal manejo de datos. Para nuestro caso utilizamos la funcionalidad de ingeniería inversa que provee Hibernate[20], esta permite que teniendo un modelo plasmado en la base, incluso con datos, podamos importarlo en forma de clases de manera automática, una vez que tenemos este modelo de clases ya podemos utilizarlo para realizar todo tipo de operaciones que impacten en forma directa en la base relacional. Para realizar la ingeniería inversa se requieren una serie de pasos, primero hay que definir el archivo de configuración de Hibernate que contiene los datos de acceso a la base de datos relacional, a continuación se detalla como ejemplo el acceso a una base de datos MySQL.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.bytecode.use_reflection_optimizer">false</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">auditoria</property>
        <property name="hibernate.connection.url">jdbc:mysql://192.168.1.101:3306/rulesadapter</property>
        <property name="hibernate.connection.username">auditoria</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.search.autoregister_listeners">false</property>
    </session-factory>
</hibernate-configuration>
```

Para el segundo paso es necesario definir una configuración de consola, para esto debemos indicarle a Hibernate la ubicación del archivo de configuración, además podrá ser necesario indicar el *classpath* a utilizar para trabajar con la base de datos, es decir la ubicación del archivo .jar que sirve de controlador. La Figura 6.7 refleja la configuración mencionada.

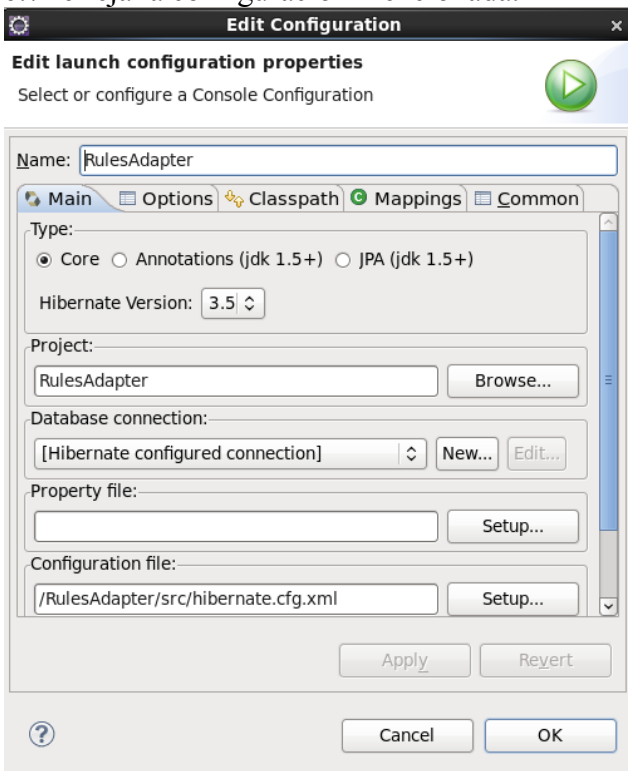

Figura 6.7.Vista de la configuración de consola Hibernate utilizando Eclipse

Una vez definida la configuración de consola, se debe crear el archivo de ingeniería inversa, a través de la configuración de consola definida se indican cuales son las tablas que serán importadas como objetos, tal como se observa en la Figura 6.8.

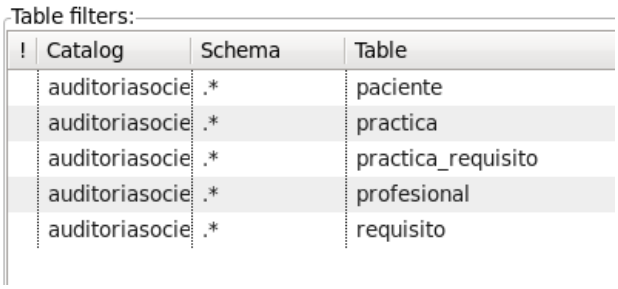

Figura 6.8.Tablas seleccionadas en el proceso de ingeniera inversa

Teniendo la definición del archivo, podemos ejecutar la operación de ingeniería inversa que genera las clases correspondientes y que en forma opcional nos permite además obtener clases de acceso a los objetos denominadas clases DAO(Data Access Object) que permiten realizar operaciones de búsqueda, modificación, agregación y eliminación de entidades. Se crean operaciones de búsqueda *find*, que permiten obtener objetos mapeados desde la base de datos, la operación *persist* permite, como su nombre lo indica, persistir los objetos creados o modificados en la base relacional, estos métodos entre otros como puede ser el de borrado se crean automáticamente para cada entidad, otorgándonos también la posibilidad de agregar los propios ya sean para realizar diversas consultas u operaciones complejas de todo tipo. A continuación, a modo de ejemplo, podemos observar una clase resultante del proceso de ingeniería inversa, la clase Regla hace referencia a la tabla con mismo nombre.

```
package misClases;

// Generated Feb 17, 2015 12:26:49 PM by Hibernate Tools 3.4.0.CR1

/**
 * Regla generated by hbm2Java
 */
public class Regla implements Java.io.Serializable {

        private Integer idRegla;
        private Tarea tarea;
        private String nombre;
        private String descripcion;
        private String path;
        private String variableParametro;

        public Regla() {
        }

        public Regla(Tarea tarea, String variableParametro) {
                this.tarea = tarea;
                this.variableParametro = variableParametro;
        }

        public Regla(Tarea tarea, String nombre, String descripcion, String path,
                        String variableParametro) {
                this.tarea = tarea;
                this.nombre = nombre;
                this.descripcion = descripcion;
                this.path = path;
                this.variableParametro = variableParametro;
        }

        .....getters y setters
```

Podemos también observar, a modo de ejemplo, código que hacen referencia a las clases DAO, en este caso mostrando la operación de recuperación por Id de una entidad y por el otro el método de persistencia para guardar nuevas entidades o realizar modificaciones.

```
public Regla findById(Java.lang.Integer id) {
		log.debug("getting Regla instance with id: " + id);
		try {
			Regla instance = (Regla) sessionFactory.getCurrentSession().get(
					"misClases.Regla", id);
			if (instance == null) {
				log.debug("get successful, no instance found");
			} else {
				log.debug("get successful, instance found");
			}
			return instance;
		} catch (RuntimeException re) {
			log.error("get failed", re);
			throw re;
		}
}

public void persist(Regla transientInstance) {
		log.debug("persisting Regla instance");
		try {
			startOperation();
			sessionFactory.getCurrentSession().persist(transientInstance);
			session.getTransaction().commit();
			log.debug("persist successful");
		} catch (RuntimeException re) {
			log.error("persist failed", re);
			throw re;
		}
}
```

6.5 Mecanismos de comunicación provistos por BonitaOS y Drools

Para seguir con la línea de la metodología propuesta, otro de los aspectos claves a considerar en la integración es el de la comunicación, considerando que en la misma habrá un modelo intermedio encargado de gestionarla debemos ocuparnos en identificar los mecanismos que proveen tanto BonitaOS como Drools para comunicarse con aplicaciones externas y terminar por definir cuales elegiremos finalmente.

6.5.1. Comunicación en BonitaOS:

Desde el punto de vista del sistema BPMS debemos considerar la comunicación en dos sentidos, por un lado hacia el modulo intermedio que luego se comunicará con Drools y por otro desde el módulo hacia el gestor de procesos para requerir información que se precise, ante esta situación analizaremos las dos posibilidades encontrando la tecnología adecuada en cada caso. Considerando la primer posibilidad, en la que BonitaOS debe enviar peticiones para ejecutar una regla de negocio, sabemos que la herramienta cuenta con múltiples conectores nativos, estos permiten comunicar las instancias de nuestro procesos con entidades externas en tiempo de ejecución, ya sea una base de datos o una aplicación de gestión documental o servicio de mail, basta con proveer los parámetros necesarios para realizar la conexión. Una de las tecnologías posibles para realizar la comunicación con una nueva aplicación es el uso de servicios web, Bonita provee un conector que hace de cliente de servicios y permite consumirlos. En la Figura 6.9 se visualiza la configuración necesaria para consumir el servicio web, en nuestro caso requerir la ejecución de una regla.

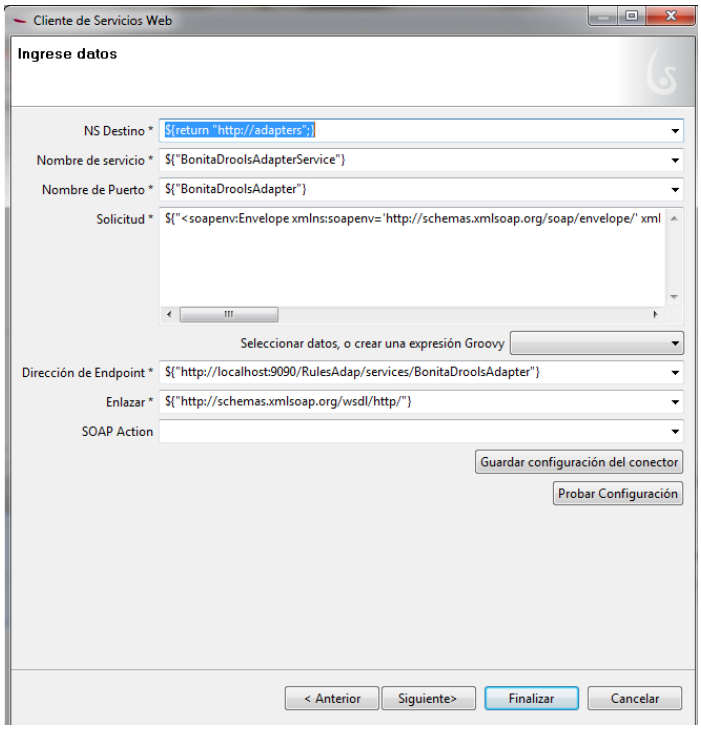

Figura 6.9.Configuración del cliente de servicios web en BonitaOS

La petición de ejecución de las reglas definidas para una tarea requiere que se le envíe al modulo de comunicación dos parámetros, por un lado el nombre del proceso y por otro el nombre de la tarea, estos dos valores identifican en forma unívoca a la definición de una tarea de proceso, en la Figura 6.10 observamos la solicitud SOAP que utiliza el conector para consumir el servicio web, con los parámetros correspondientes.

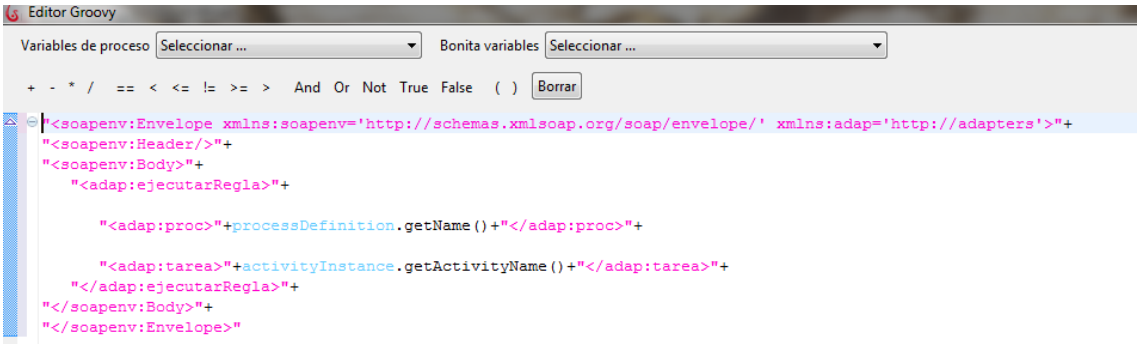

Figura 6.10.Peticion SOAP con los parámetros proceso y tarea

Consideremos ahora el otro caso, en el que la aplicación intermedia mediante su módulo de comunicación debe consultar datos del gestor de procesos, contamos con dos posibles alternativas ya que BonitaOS provee dos métodos de acceso a su API que permiten recuperar información sobre los procesos instalados y las instancias en curso, por un lado la API Java es un conjunto de librerías importables para trabajar con el motor a través de objetos y métodos propios del lenguaje, por otro lado provee de una API REST que permite a través de peticiones HTTP especificas y parametrizables recuperar datos o enviar información desde y hacia el gestor de procesos. La API REST puede ser utilizada en lenguajes distintos a Java, lo que es una gran ventaja a la hora de independizarnos de la plataforma. Elegiremos entonces utilizar esta ultima opción ya que puede

servir de modelo a la hora de integrar. En este tipo de acceso la aplicación cliente deberá autenticarse utilizando el mecanismo HTTP BASIC e indicar la operación que requiera con los parámetros pertinentes. En la Figura 6.11 se muestra el resultado de realizar una petición vía REST al motor de BonitaOS utilizando una herramienta visual.

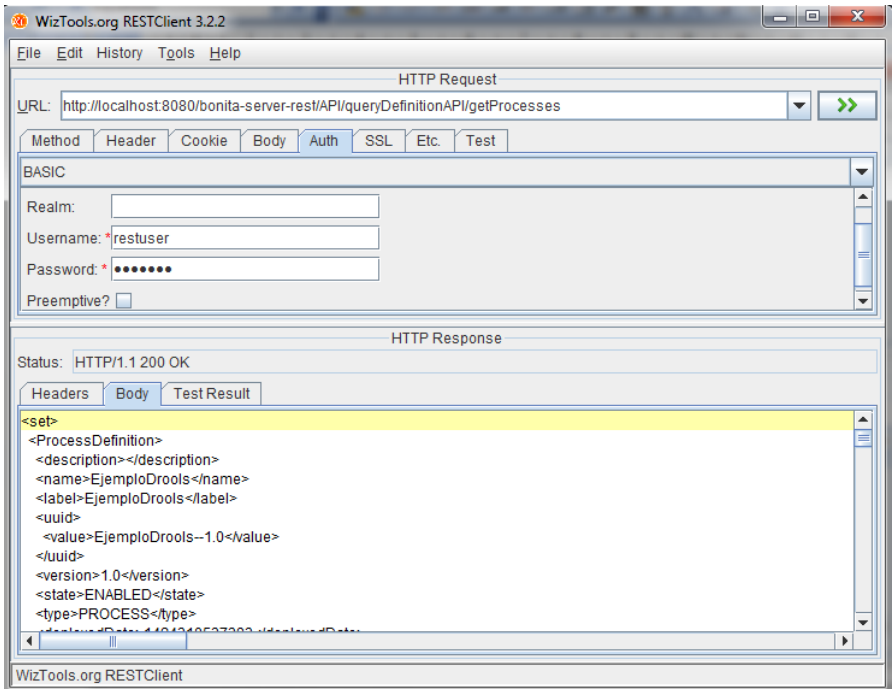

Figura 6.11.Ejemplo de solicitud REST utilizando una herramienta grafica

La petición denominada *getProcesses* se define dentro del grupo de peticiones en tiempo de ejecución queryDefinitionAPI, básicamente su funcionalidad es la de recuperar el conjunto de procesos definidos en el servidor, como vemos en el apartado de resultado se listan en formato xml la definición de cada uno de estos procesos. Si implementamos esta petición en una aplicación particular utilizando un lenguaje de programación, se requerirán las libreras que permitirán hacer peticiones HTTP y luego se deberá procesar el resultado xml en la forma que se crea conveniente.

6.5.2. Comunicación en Drools:

Desde el lado de Drools también es necesario identificar dos formas de comunicación, para obtener información del gestor y para el envío de peticiones hacia el mismo. En el primer caso si requiriéramos obtener información del servidor de reglas tenemos la posibilidad de de acceder utilizando el modulo WebDav que se provee, este nos permite obtener información acerca del contenido del repositorio, como pueden ser paquetes, categorías y reglas almacenadas, el módulo WebDav se accede vía HTTP como una petición REST, en la Figura 6.12 se ilustra una consulta sobre un paquete particular del repositorio y el resultado es el conjunto de reglas almacenadas.

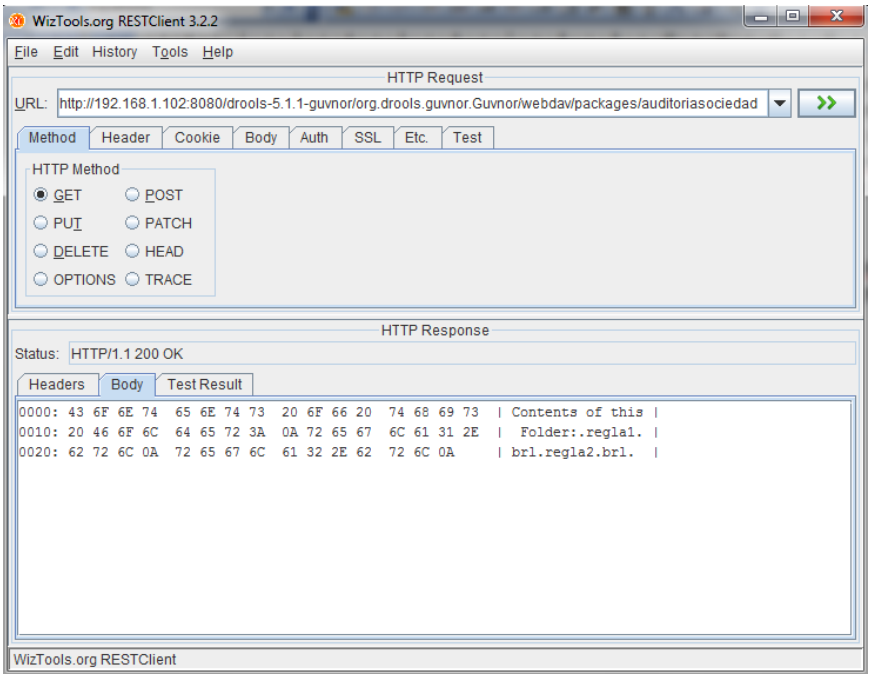

Figura 6.12. Ejemplo de solicitud REST interactuando con el modulo WebDav

Será responsabilidad entonces de la aplicación que desee comunicarse con el módulo de realizar la petición utilizando las librerías http y procesar el resultado, tal como el caso de BonitaOS.
La necesidad que podemos enfrentar con mayor frecuencia es la de enviar peticiones al gestor para que se ejecute una regla de negocio, en este caso nos apoyamos en el motor Expert de Drools que con las librerías propias permite interactuar con el repositorio Guvnor. El motor esta escrito en lenguaje Java, a continuación exponemos el código ejemplo que permite ejecutar las reglas contenidas en un paquete de Guvnor, para esto es necesario indicar los parámetros de autenticación y el recurso (en este caso el paquete "políticas")

```
StatefulKnowledgeSession kSession;
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
UrlResource rF= (UrlResource) ResourceFactory.newUrlResource( "http://192.168.1.102:9090/"
        + "drools-5.1.1-guvnor/org.drools.guvnor.Guvnor/webdav/packages/politicas" );
rF.setBasicAuthentication("enabled");
rF.setUsername("admin");
rF.setPassword("admin");

kbuilder.add( rF ,
              ResourceType.DRL);

//crear la base de conocimiento
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();

// agregar el paquete a la base
kbase.addKnowledgePackages( kbuilder.getKnowledgePackages() );

//crear sesion
kSession = kbase.newStatefulKnowledgeSession();
//ejecutar las reglas
kSession.fireAllRules();
```

6.6 Construcción de una aplicación intermedia

Una vez definidos los aspectos de seguridad, acceso a la información y comunicación nos enfrentamos a la etapa de construcción de una aplicación intermedia que sirva de adaptador entre las

herramientas BPMS y BRMS. Este adaptador se encargará de lidiar con aspectos de comunicación entre las mismas, es decir servirá de modulo de conexión, también hará las intervenciones necesarias para que el impacto de la ejecución de reglas en los datos sea la adecuada según el modelo elegido, en nuestro caso el modelo organizacional, y también proveerá de una interfaz web propia que permita gestionar la asociación de los procesos y sus tareas con las reglas almacenadas en el repositorio Guvnor. Se decide construir la aplicación utilizando el lenguaje Java, dado el conocimiento propio del lenguaje y por las posibilidades que provee el mismo de incorporar infinidad de librerías que permitan interactuar con las distintas entidades utilizando las tecnologías mencionadas previamente. Para la creación del modulo web se utiliza el Framework Java Server Faces que permite crear interfaces sencillas utilizando Beans que sirven de controlador a las peticiones. En primer lugar debemos considerar que esta aplicación deberá gestionar datos internos relativos a las distintas asociaciones que se generen, la idea es que desde Bonita se realice la petición de ejecución y el adaptador en base a los parámetros recibidos (proceso y tarea) busque en sus definiciones internas si existe alguna asociación con alguna tarea y en caso afirmativo solicita la ejecución de la misma por parte de Drools. Considerando que se debe persistir información consideramos necesario crear una pequeña base de datos para almacenar los datos. El modelo de datos consta de tres tablas, proceso, tarea y regla tal como se visualiza en la Figura 6.13.

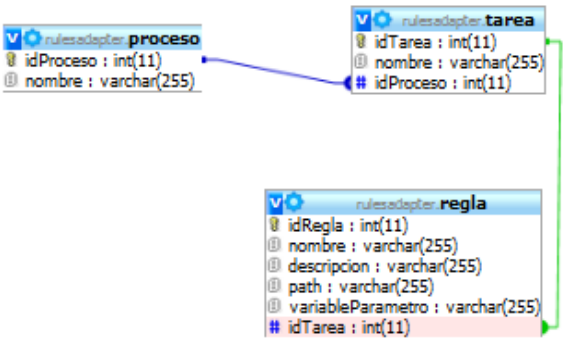

Figura 6.13.Modelo de datos del adaptador

El objetivo es almacenar información básica de cada entidad y el vínculo entre las mismas. Se crea entonces una base MySQL y para mayor flexibilidad se utiliza Hibernate con el método de ingeniería inversa para mapear las tablas con Clases, es decir trabajaremos con las clases Proceso, Tarea, regla, creando además los respectivos objetos DAO para realizar las operaciones. Una vez definido el modelo de datos del adaptador estamos en condiciones de diseñar una aplicación web que permita gestionar la información propia, tal modulo exigirá la autenticación de usuarios, como la idea es centralizar todo lo relativo a los mismos haremos que el sistema se autentique contra los usuarios almacenados en el servidor LDAP creado con anterioridad. Para lograr esto último requerimos de las librerías disponibles para Java que permiten el acceso a tales servidores. El código necesario se visualiza a continuación:

```
String username = this.getUsername();
String password = this.getPassword();
String base = "ou=People,o=repCentral,dc=ParraJulian,dc=eu";
String dn = "uid=" + username + "," + base;
String ldapURL = "ldap://192.168.1.102:16636";

// Configurar ambiente para la autenticacion

Hashtable<String, String> environment =
    new Hashtable<String, String>();
environment.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
environment.put(Context.PROVIDER_URL, ldapURL);
environment.put(Context.SECURITY_AUTHENTICATION, "simple");
environment.put(Context.SECURITY_PRINCIPAL, dn);
environment.put(Context.SECURITY_CREDENTIALS, password);

try
{
    DirContext authContext =
        new InitialDirContext(environment);

    return "ok";

}
catch (AuthenticationException ex)
{

    return "invalido";
```

Como observamos se requiere indicar el nombre de usuario y contraseña, la url del servidor LDAP y las nombres de dominio base correspondientes. Con este esquema estamos en condiciones de generar la primera pantalla de nuestra interfaz web, en la Figura 6.14 se visualiza la misma solicitando los datos correspondientes.

Figura 6.14. Pantalla de login propia del modulo web

Una vez realizado el login es necesario seleccionar el proceso y la tarea a la que queremos gestionar en cuestión de reglas, en cada caso nos apoyaremos en el uso de la API REST de Bonita [22] para solicitar la información sobre procesos y tareas. En el primer caso la petición se debe hacer como el siguiente ejemplo, como vemos se utiliza la clase *HttpClient* para realizar la petición http, se indican los parámetros de autenticación, la url correspondiente al método *getProcesses*, que retorna todos los procesos definidos en el motor de procesos y se obtiene una respuesta xml de la que se realiza el respectivo procesamiento para obtener así una lista de nombres de procesos.

```
CredentialsProvider provider = new BasicCredentialsProvider();
UsernamePasswordCredentials credentials = new UsernamePasswordCredentials("restuser", "restbpm");
provider.setCredentials(AuthScope.ANY, credentials);


HttpClient client = HttpClientBuilder.create().setDefaultCredentialsProvider(provider).build();
HttpPost postRequest = new HttpPost(
    "http://localhost:8080/bonita-server-rest/API/queryDefinitionAPI/getProcesses");
//postRequest.addHeader("accept", "application/json");

List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
urlParameters.add(new BasicNameValuePair("options", "user:restuser"));


postRequest.setEntity(new UrlEncodedFormEntity(urlParameters));
HttpResponse response = client.execute(postRequest);
ResponseHandler<String> handler = new BasicResponseHandler();
String body = handler.handleResponse(response);
 try {
        DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        InputSource is = new InputSource();
        is.setCharacterStream(new StringReader(body));

        Document doc = db.parse(is);
        NodeList nodes = doc.getElementsByTagName("ProcessDefinition");

        // iterate the employees
        for (int i = 0; i < nodes.getLength(); i++) {
           Element element = (Element) nodes.item(i);

           NodeList name = element.getElementsByTagName("uuid");
           Element ln = (Element) name.item(0);

           NodeList nd= ln.getElementsByTagName("value");
```

La lista obtenida se refleja en un formulario web que permite seleccionar un proceso y proseguir a la lista de tareas. La Figura 6.15 muestra la interfaz mencionada.

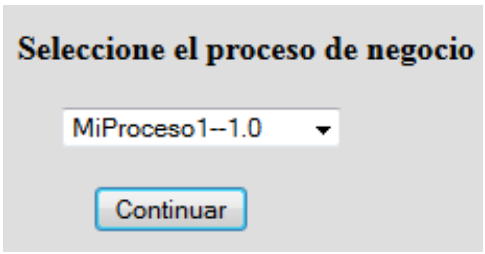

Figura 6.15.Elección del proceso a gestionar

La obtención de la lista de tareas del proceso seleccionado se realiza de la misma forma que el paso anterior con la salvedad que la url se ve modificada adecuándose a la petición pertinente, la misma pasa a armarse de la siguiente manera:

```
HttpPost postRequest = new HttpPost(
                                "http://localhost:8080/bonita-server-
        rest/API/queryDefinitionAPI/getProcessActivities/"+proceso);
```

La variable *proceso* hace referencia al proceso seleccionado, esta es un parámetro más en la petición que retorna la lista de tareas correspondientes, tal como se visualiza en la Figura 6.16.

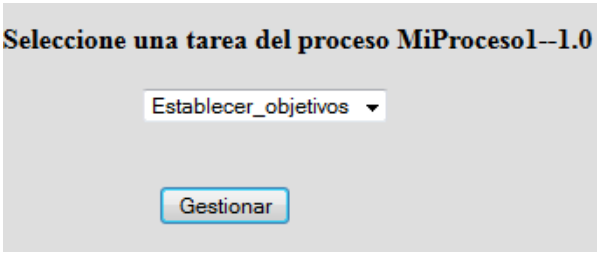

Figura 6.16.Eleccion de la tarea a gestionar

Al seleccionar la tarea deseada, se despliega un formulario que muestra las reglas existentes en el adaptador, tal como lo ilustra la Figura 6.17, con la posibilidad de removerlas si se lo cree necesario. También se nos da la posibilidad de agregar una nueva asociación indicando el nombre de la regla, una descripción alusiva, el nombre de la regla en Drools(se usa una petición http hacia el modulo WebDav para obtener el listado) y por último se puede agregar además el nombre de un parámetro en Bonita que pueda llegar a contener información de interés para la regla, la idea de contar con este campo radica en el hecho que muchas reglas en algún momento pueden requerir leer alguna variable propia del proceso en curso para poder ejecutarse.

Figura 6.17.Formulario para asociar nuevas reglas a la tarea

Al agregar la regla se crea una nueva tupla en la base de datos y el sistema refleja el cambio. En este grupo de funcionalidades mencionadas consiste básicamente el modulo web, nos resta ahora detallar como se produce la ejecución de una regla, en primer lugar es el motor de procesos el que a través de su conector definido en alguna tarea solicita consumir un servicio web, este servicio debe ser

provisto por nuestra aplicación intermedia, para esto creamos una implementación utilizando el motor axis2. Se diseña una operación que permita ejecutar reglas recibiendo como parámetros el proceso y la tarea correspondientes desde el gestor de procesos, en base a estos parámetros se realiza la búsqueda en la base de datos local para así determinar que reglas deben ser ejecutadas, finalmente se realiza la ejecución correspondiente utilizando la api del motor de Drools (Expert), se deberá indicar como atributo global la variable bonita que servirá como parámetro. La figura 6.18 muestra la implementación del método en Java que permite realizar la ejecución antes mencionada.

```
private ComunicacionREST comRest= new ComunicacionREST();
    @Override
    public boolean ejecutarReglas(String proceso, String tarea) {
        // TODO Auto-generated method stub
        Set<Regla> reglasProcesar = new HashSet<Regla>();
        ProcesoDAO daoProc= new ProcesoDAO();
        Proceso proc=daoProc.findByName(proceso);
        Iterator it = proc.getTareas().iterator();
        while (it.hasNext()) {
            Tarea tar = (Tarea) it.next();
            if(tar.getNombre().equals(tarea)){
                reglasProcesar= tar.getReglas();
                break;
            }
        }
        Iterator itReglas= reglasProcesar.iterator();
        while (itReglas.hasNext()) {
            Regla regla = (Regla) itReglas.next();
            try {
                this.testDroolsWithGuvnor(proceso,regla.getPath(),regla.getVariableParametro());
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
```

Figura 6.18.Método "ejecutarReglas"

Para realizar la operación es indispensable apoyarse en algún mecanismo que permita pasar parámetros al motor de reglas para que los considere a la hora de ejecutarlas, una de las posibles alternativas es la de utilizar atributos globales, como vemos en la Figura 6.19 realizamos el pasaje de parámetros tanto de la sesión Hibernate, con la que se realizaran consultas desde dentro de la regla, como de la variable proveniente desde Bonita para que pueda ser consultada. También podemos observar el uso de una variable global de salida que se utiliza para obtener el resultado de la ejecución de las reglas asignando el valor a una variable de proceso con el resultado de la ejecución, por defecto la variable *resultadoReglas*. Para asignar el valor del resultado a la variable debemos apoyarnos nuevamente en la API REST de BonitaOS, en esta oportunidad la petición será de la forma:

```
HttpPost postRequest = new HttpPost(
                    "http://localhost:8080/bonita-server-
rest/API/queryRuntimeAPI/setProcessInstanceVariable/"+proceso);

List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
urlParameters.add(new BasicNameValuePair("options", "user:restuser"));
urlParameters.add(new BasicNameValuePair("variableId", variableId));
urlParameters.add(new BasicNameValuePair("variableValue", variableValue));
```

proceso representa el proceso cuya variable se desea asignar, *variableId* es el identificador de la variable y *variableValue* contiene el valor a asignar.

```
public void testDroolsWithGuvnor(String proceso,String paqueteRegla,String variable) throws Exception {
        KnowledgeBase knowledgeBase = createKnowledgeBase(paqueteRegla);
        StatefulKnowledgeSession session = knowledgeBase.newStatefulKnowledgeSession();
        try {
            Session sesHibernate = SuperDAO.getSessionFactoryAuditoria().openSession();
            session.setGlobal("$hibernateSession", sesHibernate);

            session.setGlobal("$parametro", comRest.getVariableProceso(proceso, variable));

            session.fireAllRules();

            comRest.setVariableProceso(proceso,"resultadoReglas",session.getGlobal("$salida").toString());

        }
        finally {
            session.dispose();
        }
    }
        private static KnowledgeBase createKnowledgeBase(String paqueteRegla) {
        KnowledgeAgentConfiguration kaconf = KnowledgeAgentFactory.newKnowledgeAgentConfiguration();
        kaconf.setProperty( "drools.agent.scanDirectories", "false" );
        KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test agent",kaconf);

        kagent.applyChangeSet( ResourceFactory.newClassPathResource(paqueteRegla+".xml"));

        return kagent.getKnowledgeBase();
    }
```

Figura 6.19.Carga de reglas desde Guvnor y ejecución

Una vez realizadas todas las etapas del proceso de integración (seguridad, comunicación, mapeo del modelo de datos y construcción del componente adaptador) nos adentraremos a detallar un ejemplo que sirva como caso de estudio y permita visualizar en forma mas clara la funcionalidad del modelo y su viabilidad en un ambiente real.

6.7 Validación a través de un caso de estudio

El caso de estudio se basa en las prácticas odontológicas donde normalmente existe un proceso de atención de las mismas a través de la cobertura provista por las obras sociales. Los pacientes suelen tener que presentar documentación específica para poder ser atendidos dependiendo de la práctica que realicen o su asociación a algún convenio (como puede ser el de embarazo o discapacidad). Los pacientes no son los únicos involucrados en este proceso, existe una persona encargada de recepcionar la documentación, pudiendo ser el mismo profesional u otro responsable, por otro lado la documentación presentada se deberá evaluar posteriormente por el sector de auditoría de la sociedad odontológica local, en este punto se realizan las verificaciones según el paciente comprobando que se entrego lo necesario para poder realizar el pago de la práctica al profesional correspondiente. Teniendo en cuenta los pasos mencionados, y considerando que existen reglas implícitas en cuanto a la presentación de cierta documentación, es que definimos un proceso de negocio de atención odontológica visualizado en la figura 6.20. Los actores involucrados son por un lado la secretaría (o profesional), encargada de cargar la información del paciente y la documentación que él presenta, y por el otro la auditoría de la sociedad odontológica que debe revisar lo presentado y efectuar un análisis aprobatorio o no sobre el mismo. En el caso que el análisis sea satisfactorio se le informará a la obra social para que realice el pago correspondiente y en caso de rechazo se le informará al profesional sobre la situación.

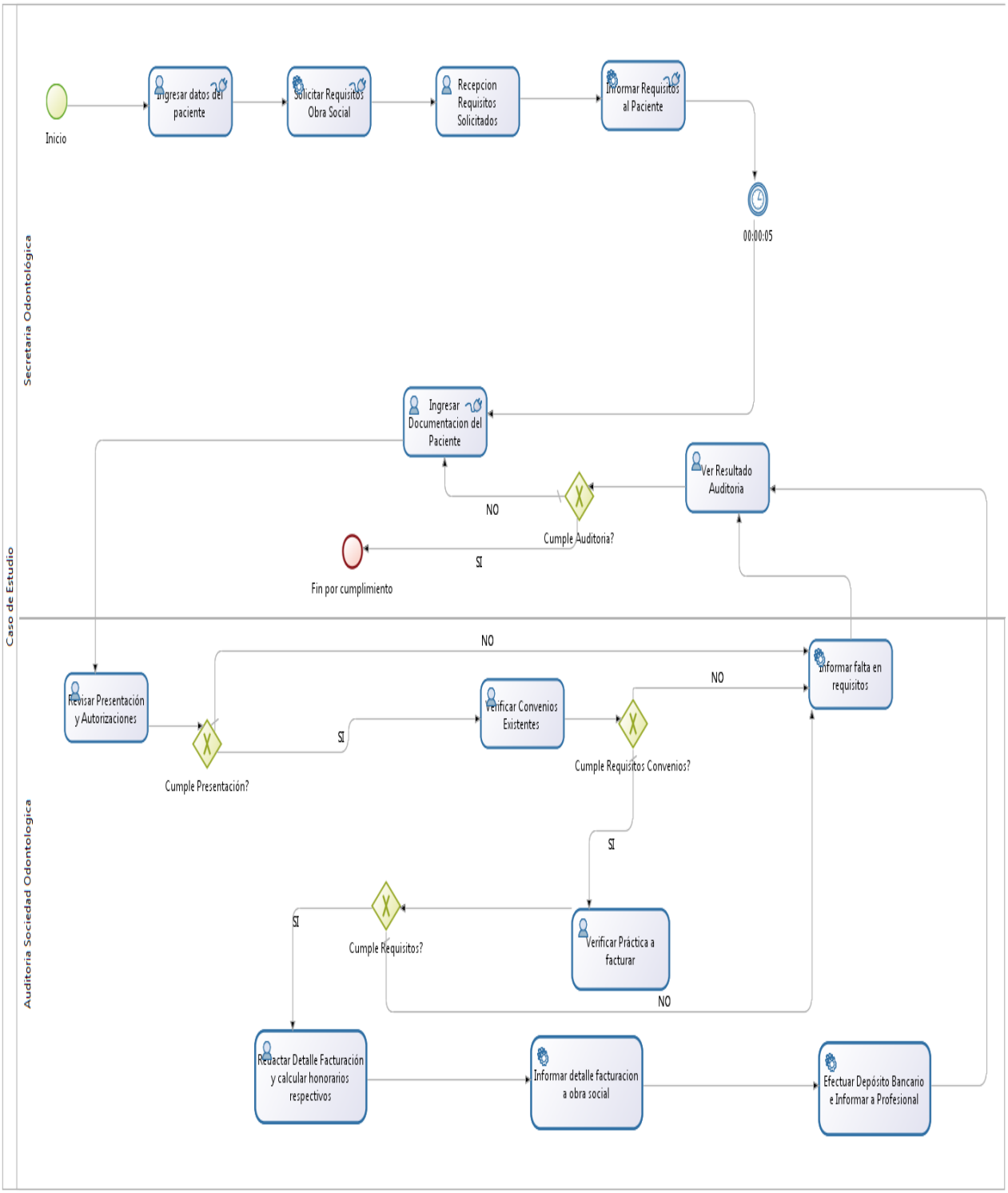

Figura 6.20.Proceso de atención odontológica

A continuación haremos un resumen de las tareas propias del proceso, observando principalmente aquellas en las que será necesario acudir a un gestor de reglas para determinar ciertas decisiones.

Actores:
Secretaría Odontológica, Auditoría Sociedad Odontológica

Tareas:
Ingresar Datos del Paciente:
Actor: Secretaría Odont.
Descripción:
Se ingresan los datos básicos del paciente cuya práctica se pretende facturar.

Solicitar Requisitos Obra Social:
Actor: Sistema
Descripción:
Mediante el consumo de un web service provisto por la Sociedad Odontológica, se solicita el listado de documentación requerida por el paciente dependiendo de su obra social.

Recepción Requisitos Solicitados:
Actor: Secretaría Odont.
Descripción:
Se muestra en pantalla la lista de requisitos solicitados.

Informar Requisitos al Paciente:
Actor: Sistema
Descripción:
Se le informa vía mail al paciente los requisitos que deberá presentar cuando concurra a la cita con el Odontólogo.

Evento de espera: Espera documentación solicitada
Descripción:
Simula el tiempo de espera desde el envío del mail con los requisitos al paciente hasta la presentación de los mismos.

Ingresar Documentación del paciente
Actor: Secretaría Odont.
Descripción:
Se ingresa los datos requeridos junto con la documentación necesaria dependiendo de los requisitos aplicados al paciente en particular.

Revisar Presentación y Autorizaciones
Actor: Sociedad Odontológica
Descripción:
Se solicita que se indiquen que requisitos básicos de presentación se cumplen. **Adicionalmente se efectúa un análisis completo por parte del motor de reglas.**

Verificar Convenios Existentes
Actor: Auditoría Sociedad Odontológica
Descripción:
Se muestra en pantalla un formulario para ingresar el convenio de la cobertura social del paciente, si posee, además marcara que documentación fue presentada. **Al indicar el procesamiento de los datos el sistema llamará al gestor de reglas de negocio para procesar los requisitos necesarios y así detectará si son validos.**

Verificar Práctica a facturar
Actor: Auditoría Sociedad Odontológica

Descripción:
Se muestra en pantalla un formulario de ingreso de datos sobre la práctica específica del paciente a facturar, **se permite procesar los mismos por medio de las reglas establecidas en el motor de reglas de negocio.**

Redactar Detalle Facturación y calcular honorarios respectivos
Actor: Auditoría Sociedad Odontológica
Descripción:
Se muestra en pantalla un formulario de ingreso de datos para poder detallar la facturación correspondiente a la obra social, también se contemplará el cálculo de los honorarios propios de la Sociedad Odontológica.

Informar Detalle Facturación a Obra Social
Actor: Sistema
Descripción:
Se envía vía mail un detalle de la facturación pretendida con destino a la obra social del paciente.

Efectuar Depósito Bancario e Informar al Profesional
Actor: Sistema
Descripción:
Se realiza el depósito bancario y se informa vía mail al profesional.

Ver Resultado Auditoría
Actor: Secretaría Odontológica
Descripción:
Se visualiza en pantalla cual ha sido el resultado del proceso de facturación para la práctica ingresada.

El modelo de datos del sistema esta representado en la figura 6.21, utilizando ingeniería inversa provista por Hibernate podemos obtener el modelo de clases asociado para poder utilizar desde Drools

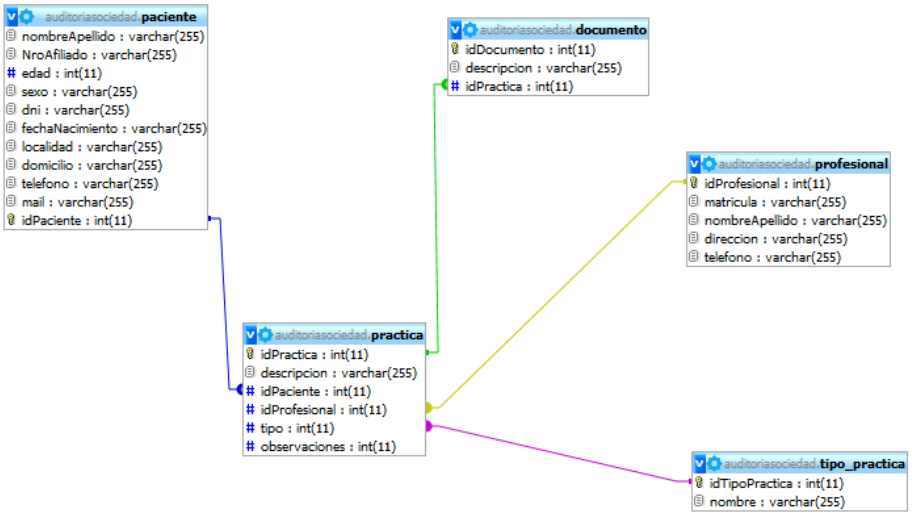

Figura 6.21.Modelo de datos del sistema de atención odontológica

Una vez definidas las tareas del proceso y el modelo de datos, nos dispondremos a detallar como se realiza la ejecución de una instancia del mismo en BonitaOS, haciendo hincapié en los pasos que presentan interés en este trabajo que son los relacionados a la ejecución de reglas de negocio. La primera tarea a ejecutar, por parte del grupo *secretaría* es la de carga de datos del paciente, se solicitan los datos básicos tal como lo ilustra la figura 6.22.Una vez ingresada la información requerida se utiliza un servicio web para solicitar la documentación requerida para el paciente en curso, al recibir los datos se envían vía mail al afiliado. Hay un período de espera hasta que el paciente se presenta el día de la práctica con los documentos pertinentes, en este instante se completa el formulario de verificación de ingreso de documentación (figura 6.23), marcando los campos que correspondan(es decir, se marcan los ítems que fueron presentados).

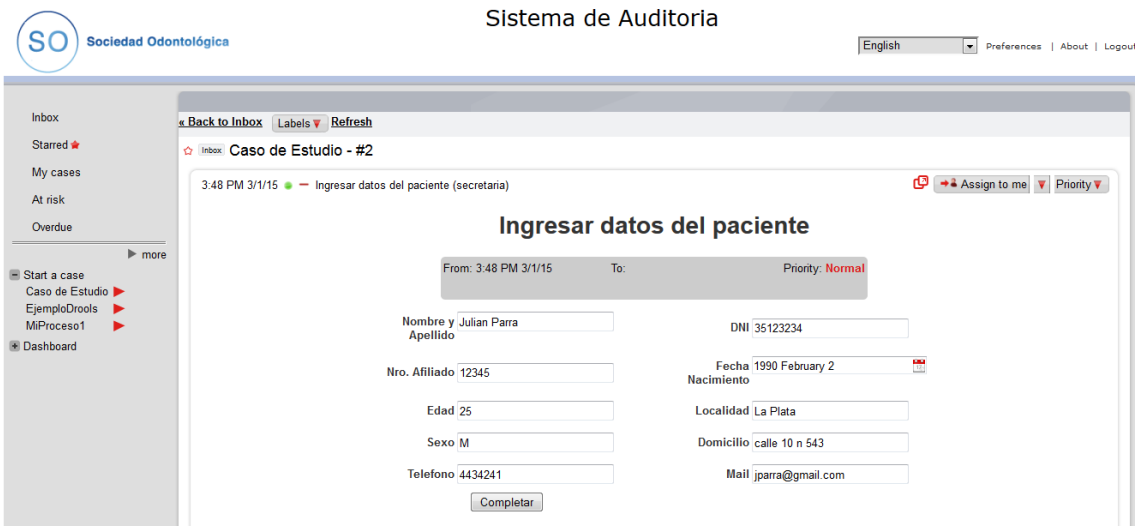

Figura 6.22.Formulario de ingreso de datos

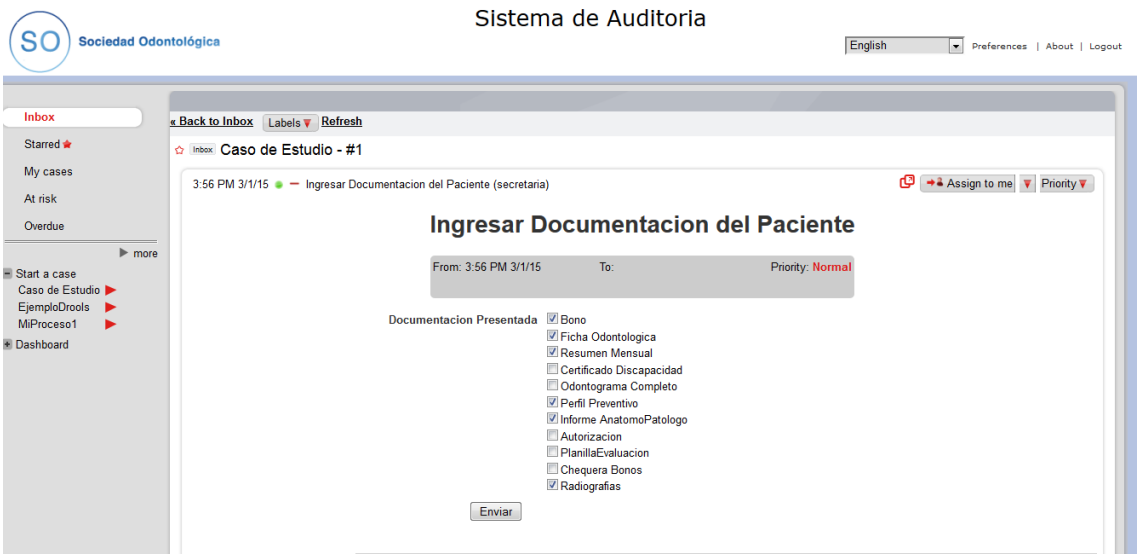

Figura 6.23.Formulario de verificación de documentación entrante

Al procesar los datos previamente mencionados la responsabilidad sobre la instancia se asigna al grupo de auditoría de la sociedad odontológica, es a partir de ahora donde las reglas propias del dominio pasan a tomar relevancia. La persona encargada de la auditoría (pudiendo ser mas de una) deberá completar una serie de tareas de revisión de la documentación propia de la práctica en curso, las tareas Revisar presentación y autorizaciones, Verificar Convenios Existentes y Verificar práctica a facturar presentan al usuario un formulario con ítems a marcar, estos ítems representan los documentos ingresados, al ordenar el procesamiento de los datos (clickeando en el botón procesar) se solicitará la ejecución de las reglas del dominio que verificarán los requisitos presentados y en base a ciertos criterios relativos a la practica y al paciente se validará o rechazará la operación. La figura 6.24 muestra la tarea Verificar Convenios Existentes, se solicita el ingreso del convenio correspondiente y la lista de requisitos presentados, la validación de la documentación en base al convenio se realiza al clickear en procesar.

Figura 6.24.Formulario de verificación de convenios

Al solicitar el procesamiento de la información, se ejecuta el conector correspondiente en BonitaOS que consume el servicio web implementado para la ejecución de reglas, funcionalidad que fue detallada en los apartados de comunicación (6.5) y creación del adaptador (6.6) correspondiente. No debemos olvidar que en forma previa a este paso se debe realizar la asignación entre tarea y reglas desde la interfaz web provista por el adaptador. En la figura 6.25 observamos un ejemplo, para la tarea *Revisar Presentación y Autorizaciones* en la que se ingresa el nombre y descripción correspondientes junto con el *path* en Drools, este *path* hace referencia a un paquete de reglas almacenado en el repositorio Guvnor.

Figura 6.25.Formulario de asignación Tarea-Regla en adaptador

Contando con la asociación definida y habiendo ordenado el procesamiento de los datos se consume entonces el servicio web y se realiza la ejecución de las reglas pertinentes, para el caso de los convenios tendremos una serie de reglas definidas en el repositorio, por ejemplo las figuras 6.26 y 6.26 representan la evaluación de un convenio de discapacidad, la primera informando aprobación y la segunda rechazo, como observamos se utiliza Hibernate para obtener el objeto *Práctica* correspondiente a la práctica en curso, para tal fin se utiliza el parámetro global que sirve como id de búsqueda. Se evalúa que se cuente con la ficha y el certificado de discapacidad como requisitos presentados. Como observamos en cada caso, al cumplirse la condición se ejecuta una acción (dentro del apartado "then"), la misma consiste en asignar la variable global *salida* con el resultado correspondiente, siendo en un caso afirmativo y en el otro negativo.

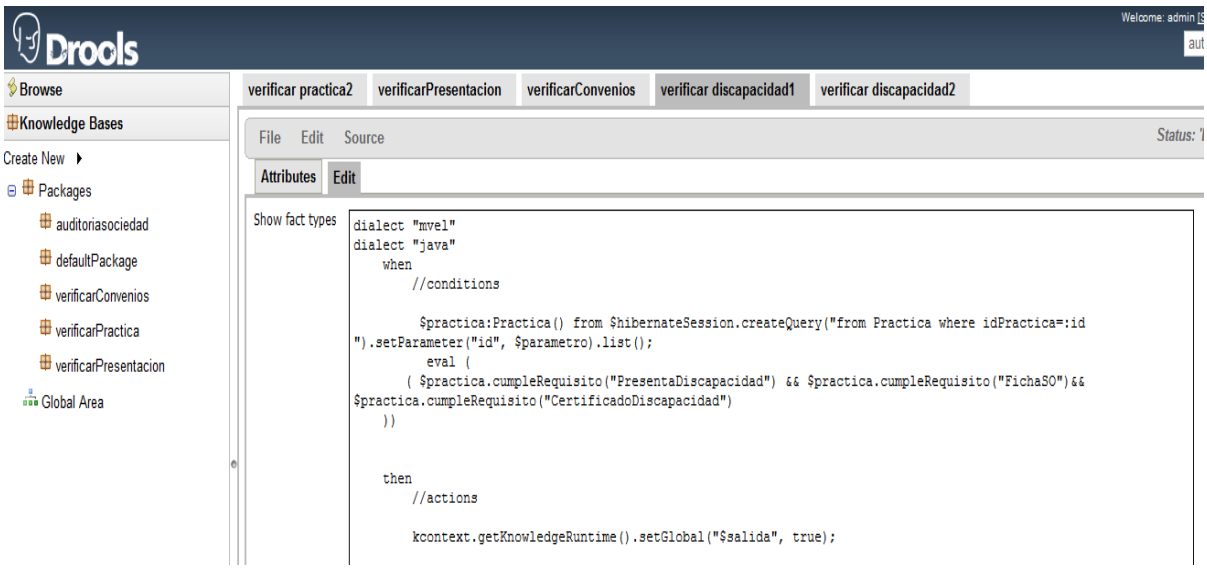

Figura 6.25.Ejemplo de regla de verificación de convenios en Drools Guvnor (Caso positivo)

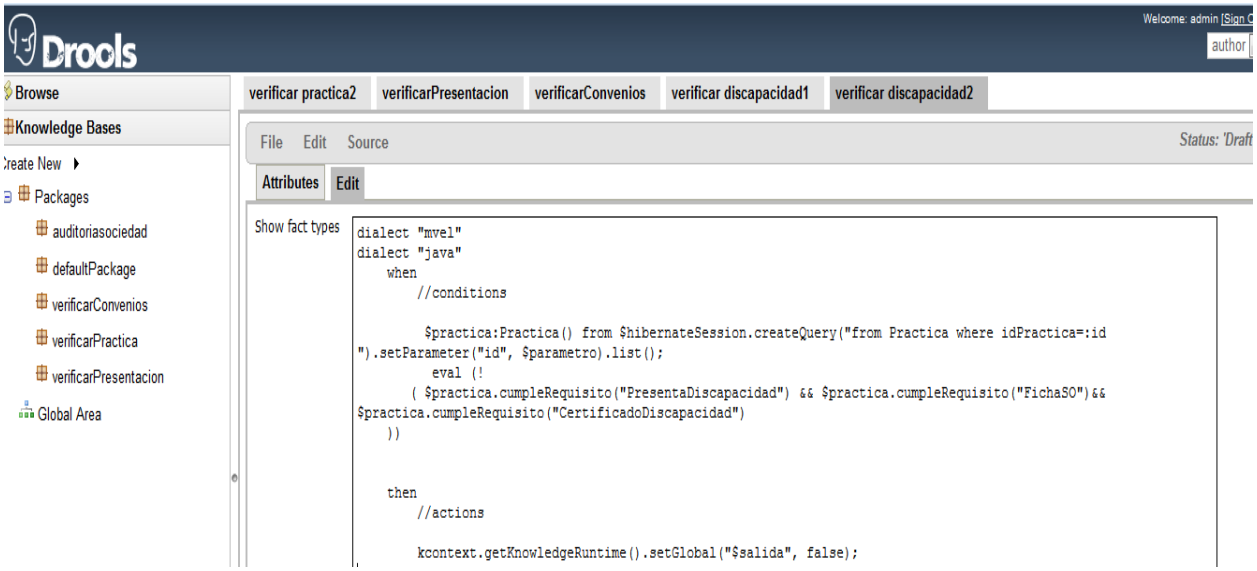

Figura 6.26.Ejemplo de regla de verificación de convenios en Drools Guvnor (Caso negativo)

Al obtener el resultado en la variable *resultadoReglas* se decide por continuar o no con el proceso, en caso de aprobación se prosigue con la siguiente tarea de verificación y en caso de rechazo se le informará al profesional utilizando la tarea correspondiente. Al pasar por los tres pasos de aprobación satisfactoriamente se realiza entonces el cálculo de los honorarios respectivos efectuando el depósito bancario e informando al profesional de la acreditación, para tal fin se utilizan los conectores nativos de Bonita para el envío de mails. El proceso termina entonces cuando se cumplen los requisitos necesarios y se efectúan los avisos de pago correspondiente.

7

7. Conclusiones y trabajos futuros

Los sistemas BPMS y BRMS tienden a utilizarse cada vez más en forma conjunta, existen empresas de renombre en el mercado que utilizan el poder de ambas herramientas de manera integrada. Existiendo aplicaciones organizacionales ya implementadas con los componentes mencionados nos encontramos que la conexión en cada caso se realiza mediante reglas propias de los productos involucrados, como en el caso de Websphere Process Server y JRules de IBM. En muchas oportunidades del ámbito práctico es necesario utilizar herramientas que no pertenezcan a una misma suite empresarial y que requieren ser de código libre. Ante la búsqueda de tal objetivo es que observamos la falta de conocimiento de las bases esenciales que hacen a la comunicación entre un sistema gestor de procesos y uno de reglas. El objetivo del trabajo presentado es el de hacer de soporte a tal necesidad, para eso se estudiaron distintos casos reales de integración de sistemas como Bonita y Drools y los productos de la suite IBM mencionados anteriormente.

Una vez revisados los ejemplos de comunicación comenzamos a definir una propuesta a nivel arquitectónico que reuniera las condiciones que consideramos esenciales para el proceso de integración. Descubrimos que el aspecto de seguridad en el acceso de los usuarios se consideraba en ambos productos y ante tal hecho tomamos la decisión de centralizar la gestión de los mismos ganando así reusabilidad y mantenibilidad.

Otro de los puntos clave identificado fue el relativo al modelo de datos, este aspecto, que en lo personal representó la mayor dificultad del trabajo dado el grado de importancia que conlleva la información organizacional, fue abordado definiendo dos enfoques diferentes de implementación, teniendo en cuenta o no la posibilidad de utilizar un modelo separado acotado empleado únicamente por el gestor de reglas. Se identificó el nivel de importancia que puede tener un sistema mapeador de entidades como Hibernate para adaptar el modelo de información con herramientas que trabajan con objetos, esto sirve de ejemplo a la hora de lidiar con la diversidad en las tecnologías actuales que suelen trabajar en su mayoría de manera orientada a objetos mientras que las bases de datos suelen ser relacionales.

Los mecanismos de comunicación entre las herramientas constituyó otra de las fuentes de estudio, contando con abundante documentación sobre interfaces provistas por distintos productos, facilitando enormemente la definición de un módulo intermedio que se encargue de tal aspecto utilizando las tecnologías existentes como pueden ser las librerías REST y los servicios web, de las cuales se detectó el alto grado de adaptabilidad que proveen a la hora de conectar sistemas diferentes, incluso fuera del ámbito de este trabajo.

La identificación de los tres aspectos mencionados (acceso por usuarios, gestión del modelo de datos y comunicación) conllevaron a pensar en la idea de tener un componente intermedio que conociendo al sistema gestor de procesos y reglas sirva de nexo entro los dos y trate las particularidades relativas a tal integración por su cuenta, liberando a las herramientas de tal responsabilidad. La creación de un componente adaptador intermedio constituyó el producto con mayor riqueza de este trabajo, dada la

cantidad y variedad de funcionalidad que engloba y que sirve de claro modelo de arranque a la hora de querer comenzar a conectar otros sistemas del mercado.
La necesidad de contar con una interfaz web para realizar la configuración de las asociaciones entre tareas del proceso y reglas de negocio implica un valor agregado a la responsabilidad de los usuarios encargados de la administración del sistema. BonitaOS y Drools sirvieron efectivamente en la puesta en marcha del modelo arquitectónico definido, la flexibilidad que otorgaron tales herramientas de código abierto nos facilitó la adaptación correcta del procedimiento.
La identificación de un caso de estudio real como la atención de prácticas odontológicas nos indica que la propuesta arquitectónica es viable y representa una metodología documentada y ordenada que facilita la implementación en conjunto.
En cuanto a los trabajos futuros, o de la forma en que podría extenderse la propuesta arquitectónica planteada, podemos enunciar:

1. Incorporación de la gestión de roles de usuario para el repositorio central dado que estos no fueron considerados a la hora de unificar el acceso para no apartarnos del aspecto de mayor interés que lo constituían los usuarios por sí mismos, la implementación de esta nueva funcionalidad no resulta trivial ya que se deben tener en cuenta los roles tanto del sistema gestor de procesos como el de reglas, observar cuáles se comparten y cuáles no e identificar algún mecanismo propio de cada herramienta que permita identificar los roles almacenados y proporcionar la funcionalidad que corresponda en cada caso.

2. Estudiar la posibilidad de tener que incorporar otros métodos de comunicación entre las herramientas, según las interfaces que estas provean. Ejemplos de tecnologías a considerar pueden ser RPC, RMI, Agentes JADE, entre otros.

3. Ampliar el módulo de interfaz web que se diseñó durante la etapa de construcción del adaptador. Podría ser necesario definir cuál es la variable de salida en Bonita para la obtención del resultado, recordemos que por defecto tal operación se realiza sobre la variable resultadoReglas. También se podrían agregar más variables de entrada e incluso podemos pensar en diseñar un componente que se encargue de mantener los datos del adaptador, encontrando referencias a procesos o reglas que dejaron de ser válidas por algún motivo.

Referencias Bibliográficas

[1] Weske, M. – *Business Process Management: Concepts, Languages*, Architectures
Springer-Verlag Berlin Heidelberg 2007

[2] Appian Paper, *Components of an Intelligent BPM Suite*

[3]OMG, *Semantics of Business Vocabulary and Business Rules*, SBVR v1.0, 2008

[4]BRG, *The Business Rules Group*. , Diciembre 2013, Disponible en:
http://www.businessrulesgroup.org/defnbrg.shtml

[5] Boyer, Mili. – *Agile Business Rule Development*
Springer-Verlag Berlin Heidelberg 2011.

[6] von Halle, Goldberg. – *The Business Rule Revolution*
Springer-Verlag Berlin Heidelberg 2011.

[7] Ian Graham. – *Business Rules Management and Service Oriented Architecture: A Pattern Language.*
Wiley, 2006.

[8]Morgan, A. *Business Rules and Information Systems: Aligning IT with Business Goals*, Boston MA: Addison-Wesley, 2002

[9] The JBoss Drools team*, Guvnor User Guide.*

[10] *Overview of ILOG JRules and WebSphere Process Server integration*
IBM, 2010

[11] *¿Que es un servicio de directorios?*, Diciembre 2013, Disponible en http://olsacupy.berlios.de

[12] Steven Tuttle, Ami Ehlenberger, Ramakrishna Gorthi, Jay Leiserson, Richard Macbeth, Nathan Owen, Sunil Ranahandola, Michael Storrs, Chunhui Yang
Understanding LDAP Design and Implementation
IBM, 2004

[13] Pradas Rafael, RedIris. – *Introducción al Servicio de Directorio*

[14]BonitaSoft, *Diseño de procesos*, Agosto 2014, Disponible en
http://documentation.bonitasoft.com/5x/bos-59/process-design

[15] BonitaSoft, *List of APIS*, Agosto 2014, Disponible en:
http://documentation.bonitasoft.com/5x/list-apis

[16] JBoss Drools Team, *Drools Expert User Guide, Version 5.5.0 Final*

[17] Kijanowski, Red Hat, *JBoss Drools how-to: Tuning Guvnor*, Agosto 2014 , Disponible en:
http://magazine.redhat.com/2008/08/14/jboss-drools-how-to-tuning-guvnor-part-2/

[18] Priyanka Kapoor , *How to configure LDAP with Bonita User XP?* , Agosto 2014 , Disponible en: https://priyankacool10.wordpress.com/2012/07/25/how-to-configure-ldap-with-bonita-user-xp/

[19] *What is Object/Relational Mapping?* , Agosto 2014 , Disponible en: http://hibernate.org/orm/what-is-an-orm/

[20] *Cómo generar clases POJO y mapeos de XML con Hibernate a partir de bases de datos, Agosto 2014,* Disponible en*:* http://es.wikihow.com/generar-clases-POJO-y-mapeos-de-XML-con-Hibernate-a-partir-de-bases-de-datos

[21] Kijanowski, Red Hat, *JBoss Drools meets Hibernate*, Agosto 2014, Disponible en: http://magazine.redhat.com/2008/08/14/jboss-drools-how-to-tuning-guvnor-part-2/

[22]BonitaSoft, *ResourceAPI*, Agosto 2014, Disponible en: http://documentation.bonitasoft.com/javadoc/rest/5.10/API/index.html