Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Uruguay

Portal Genérico para la Gestión de Procesos de Negocio

Belén Remedi, Diego Rodriguez, Alejandro Guggeri

Proyecto de Grado
Ingeniería en Computación
Universidad de la República

Montevideo, Uruguay, 12 de diciembre de 2018

Supervisores: Dra. Ing. Andrea Delgado
Dr. Ing. Daniel Calegari

Resumen

Un Sistema de Gestión de Procesos de Negocio (BPMS) es una tecnología que permite soportar el ciclo de vida de un proceso de negocio, desde su modelado hasta su ejecución y evaluación. Los BPMS suelen disponer de un portal web que permite la interacción entre usuarios con sus tareas, conectados con un motor de procesos que ejecuta los procesos. Los elementos manejados en el portal web, se corresponden con un modelo de datos subyacente que define conceptos asociados a los modelos de procesos y sus actividades.

Los BPMS han ido evolucionando para lograr una mayor independencia entre los portales web y el motor de estos sistemas. Es por esto que la mayoría provee de una API de tipo REST/SOAP para acceder a las funcionalidades desde cualquier punto de acceso.

Si bien cada herramienta implementa las funcionalidades de distinta forma y basadas en un modelo de datos diferente, ambos elementos se basan en conceptos similares. De la misma forma, la navegabilidad y pantallas definidas en el portal de usuario web se comportan en general en forma similar. Dadas estas características, parece viable disponer de un portal web genérico que permita interactuar con cualquier motor de procesos.

Dada la existencia de un trabajo previo que estudió esto y propuso un modelo de datos y API genéricos a partir del análisis de tres BPMS, el objetivo de este trabajo es profundizar en el análisis y extenderlo.

Se analizaron seis BPMS, incluyendo las posibilidades que ofrecen de interacción con sus motores y las herramientas para esto. Se profundizó el análisis de los portales web y de las APIs de cada uno de los sistemas y a partir de esto se represento para cada uno de ellos un modelo de datos que abarque los conceptos manejados, categorizándolos como de Ejecución, Evaluación o Administración.

Una vez realizado esto, se unificaron los conceptos identificados para extender y adaptar el modelo de datos genérico del trabajo previo.

Luego, también a partir de la definición de API genérica existente, se comparó cada uno de los métodos brindados por esta con los identificados en el análisis anterior de cada uno de los BPMS.

Finalmente, se creó un prototipo que implementa parte de esta API genérica y se adaptó el portal web existente, para dar soporte a esta nueva implementación, integrando tres de los BPMS analizados.

Palabras clave: Sistema de Gestión de Procesos de Negocio, modelo de datos, API.

Contenido

1 Introducción

Las organizaciones necesitan contar con mecanismos que les permitan su mejora contínua. El paradigma de Business Process Management (BPM, [Wes07]) tiene como objetivo lograr esto, a través de la gestión de procesos de Negocio. Un proceso de negocio es un conjunto de actividades ejecutadas de manera coordinada en un ambiente técnico y organizacional para lograr un objetivo de negocio [Wes07].

Un Sistema de Gestión de Procesos de Negocio (BPMS, [Wes07]) es un software que permite modelar, ejecutar y evaluar procesos de negocio a partir de un conjunto de módulos que componen la herramienta como ser el modelador de procesos (Business Process Model and Notation (BPMN, [OMG13]) u otros), motor de workflow, motor de reglas de negocio y herramientas de Business Intelligence.

Los BPMS suelen disponer de un portal web que permite a los usuarios ejecutar procesos de negocio, tomar tareas pendientes, realizar las tareas pendientes que les fueron asignadas o reasignar estas tareas, entre otras. Para permitir que el portal web soporte estas funcionalidades, los BPMS proveen generalmente una API de tipo REST/SOAP con operaciones asociadas al motor de procesos.

Existe una gran cantidad de herramientas de BPMS [Wes07] y todas ellas presentan similitudes en cuanto a los conceptos que manejan en relación a sus portales y APIs de acceso y la forma en que la navegabilidad y pantallas definidas en estos portales se comportan. Dado esto, lograr independencia con una plataforma particular, siendo el proceso de negocio el activo principal de las organizaciones, reduciría los costos asociados, por ejemplo, en cuanto al uso de la interfaz gráfica que proveen estos portales. Además, es común que las organizaciones accedan desde distintas aplicaciones, a través de su propia interfaz gráfica, al motor de ejecución de un BPMS [Wes07].

En el trabajo previo [DCA16] se realizó un análisis del modelo de datos, API del motor de procesos y portal de usuario web provistos por tres BPMS [Wes07]: Activiti [Act18],

Bonita [Bon18] y Bizagi [Biz18]. A partir de este estudio, se definió un modelo de datos, una API y un portal genérico que permite la ejecución de procesos de negocio de manera independiente de la plataforma. El portal web genérico realiza invocaciones a la API genérica definida, desacoplando la interface web del motor específico de ejecución de procesos, los cuales pueden ser intercambiados mediante la implementación de adaptadores específicos hacia la API de cada uno.

Este proyecto tiene como objetivo ampliar este trabajo previo, fortaleciendo la definición del modelo de datos y API genéricos a partir del análisis de otros BPMS [Wes07], como los son jBPM [JBP18a], Camunda [Cam18a] y Flowable [Flo18], además de realizar análisis comparativo para los BPMS ya contemplados en el trabajo previo e implementar adaptadores para demostrar la factibilidad de la propuesta general.

Objetivos

El objetivo general de este proyecto es desarrollar un portal genérico para la ejecución de procesos de negocio, a partir de la definición de un modelo de datos y API genérica que pueda ser adaptada a implementaciones concretas en diferentes BPMS.

Los objetivos específicos de este proyecto son:

- Relevar propuestas existentes sobre unificacion de conceptos de BPM y comparar con la propuesta existente en un trabajo previo.
- Evaluar portales y APIs de servicios para el acceso al motor de procesos de una selección de BPMS [Wes07]: Activiti, Bizagui, Bonita, Flowable, Camunda y jBPM.
- Extender y adaptar el modelo de datos y API genéricos existentes de acuerdo a los resultados obtenidos en el relevamiento y evaluación anteriores.
- Generar automáticamente el portal web genérico en una tecnología web actual, a partir de su definición en IFML, utilizando la API genérica ampliada.
- Desarrollar un caso de estudio que permita mostrar la ejecución de un proceso de negocio utilizando el portal genérico y su conexión con tres motores a través del API genérica.

Resultados esperados

Los resultados esperados de este proyecto son:

- Evaluación de una selección de BPMS en relación a sus portales y APIs de servicios para el acceso al motor de procesos.
- Extensión y adaptación del modelo de datos y API genéricos según los resultados de una evaluación comparativa del trabajo previo y selección de BPMS.
- Implementación de una aplicación web, prototipo del portal genérico de BPMS.

- Implementación de un caso de estudio de uso del portal y su conexión con los BPMS Camunda, Bonita y jBPM.

El resto del documento está estructurado de la siguiente manera. En el capítulo 2 se define el marco de la tesis, introduciendo los conceptos de Proceso de Negocio, BPM y BPMS. También se introducen algunos trabajos previos relacionados. En el capítulo 3 se analizan los portales web y el acceso programático a los motores de seis BPMS, como son Bonita, Camunda, jBPM y el resto, ya estudiados en la tesis de maestría: Activiti, Bonita y Bizagui. En el capítulo 4, se define la arquitectura de la solución para el portal genérico, el modelo de datos para el mismo y la API genérica. En los capítulos 5 y 6 presentamos el prototipo realizado y la integración con tres BPMS. Por último en el capítulo 7, se abordan las conclusiones y trabajo a futuro.

2 Estado del Arte

En este capítulo introduciremos los conceptos más importantes en relación a la gestión de procesos de negocio y a los sistemas de gestión de procesos de negocio. Se presentan además propuestas existentes sobre unificación de conceptos de BPM y un trabajo previo [DCA16] que fue la base de este trabajo.

2.1 Gestión de Procesos de Negocio

En toda organización se busca identificar los procesos necesarios que ayudan en las distintas operaciones.

A continuación se citará una definición de los mismos:

> *"Proceso de negocio es el conjunto de actividades ejecutadas de manera coordinada en un ambiente técnico y organizacional. Estas actividades realizan en conjunto un objetivo de negocio. Cada proceso de negocio es creado por una organización, pero puede interactuar con procesos de negocio llevados a cabo por otras organizaciones". [Wes07]*

La gestión de estos como tal, es una disciplina que se enfoca en mejorar dichos procesos que hacen de las operaciones de una organización. Para esto se basan en un ciclo de mejora continua que se compone principalmente de cuatro etapas, tal como se muestra en la FIGURA 2.1:

- **Análisis y Diseño**, etapa en la cual se diagraman los mismos utilizando una notación (por ejemplo el estándar Business Process Model and Notation (BPMN, [OMG13])) para poder visualizar y discutir como funcionan. Ésta etapa permite unificar la forma de comunicación entre los diferentes interesados, haciendo más eficiente de esta.

- **Configuración**, etapa en la cual se realiza la implementación del modelo realizado en la etapa anterior, considerando las políticas particulares de cada organización.
- **Ejecución**, etapa donde se automatizan los procesos con un motor de ejecución. Se realiza de forma paralela a la ejecución real de los dichos procesos. Toda información generada en esta etapa son fundamentales para evaluar los mismos.
- **Evaluación**, etapa en la cual se monitorizan las actividades y se relaciona la información de los procesos con la estrategia empresarial para conocer si se esta alineado con los objetivos de la misma, y de esta forma tomar las decisiones pertinentes. Algunas técnicas utilizadas para evaluar la ejecución de los procesos son técnicas de monitoreo de actividades y ”process mining”.

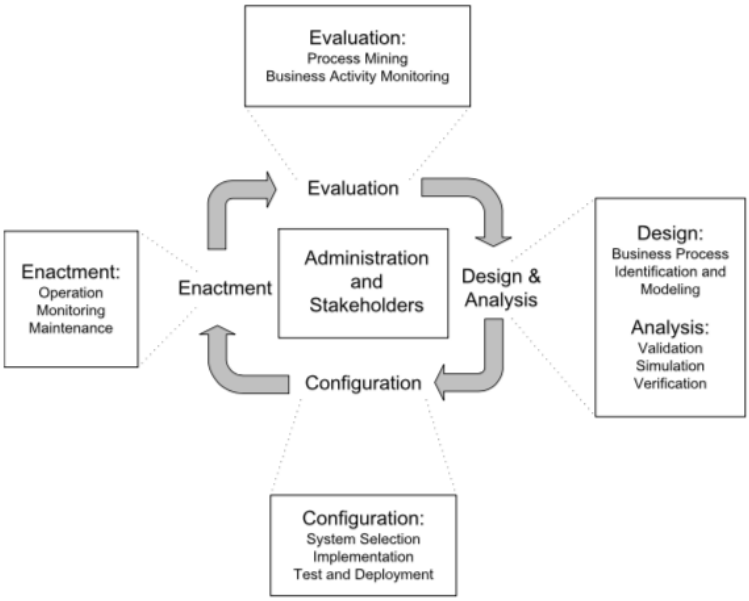


Figure 2.1: Ciclo de vida de un Proceso de Negocio [Wes07]

2.2 Sistemas de Gestión de Procesos de Negocio

> "*Un sistema de gestión de procesos de negocio (BPMS), es un software genérico manejado por representaciones explícitas de procesos, utilizado para coordinar la ejecución de procesos de negocio." [Wes07]*

En otras palabras, un BPMS [Wes07] es software que da soporte a la ejecución de procesos

de negocio facilitando la realización de tareas en todas las etapas del ciclo de vida de un proceso, de forma que las organizaciones puedan adoptar e implantar dichos procesos.

En la FIGURA 2.2 se muestra un esquema general de un BPMS.

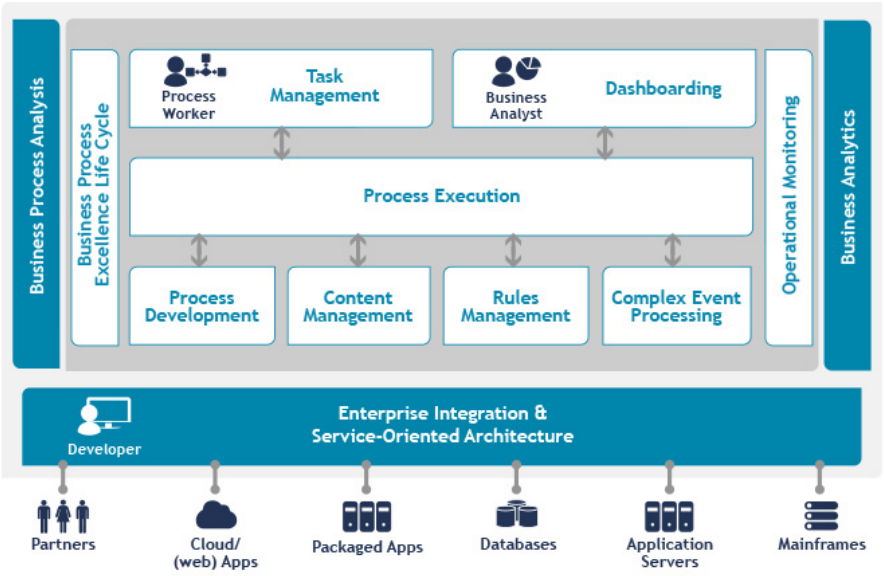


Figure 2.2: Abstracción BPMS [Cro]

Los componentes que integran un BPMS se caracterizan de la siguiente forma: [AG]:

- **Process execution:** Es donde se da la ejecución de un proceso de negocio con la ayuda de un motor de procesos que actúa como un agente centralizado para controlar las orquestaciones de procesos.
- **Process Development:** Se modela un proceso de negocio especificando las actividades, con sus relaciones, que se realizan dentro de una organización. Además, la configuración correcta del flujo de trabajo debe establecerse y también debe reflejar la lógica del proceso.
- **Business rules management:** Se definen y administran las reglas de negocio. Se pueden modificar sin necesidad de hacer un nuevo desarrollo.
- **Dashboarding:** Monitoreo y seguimiento de los procesos de negocio. Esto se controla según los roles del usuario.
- **Task and team Managment:** Se mejora la interacción entre los procesos y las per-

sonas que lo usan.

El siguiente análisis pronfundiza los conceptos anteriores para cada uno de los Business Process Management System (BPMS, [Wes07]) mencionados en los objetivos.

2.2.1 Activiti

Activi [Act18] es un sistema para gestión de procesos de negocio, desarrollado en lenguaje Java y es de código abierto. La suite de aplicaciones de este se compone de: Activiti Engine, que es el motor de gestión de procesos; Activiti Rest, la API que permite el acceso programático al motor de procesos y Activiti UI application, Activiti Admin app UI y Activiti Designer que se detallan a continuación.

Activiti UI application

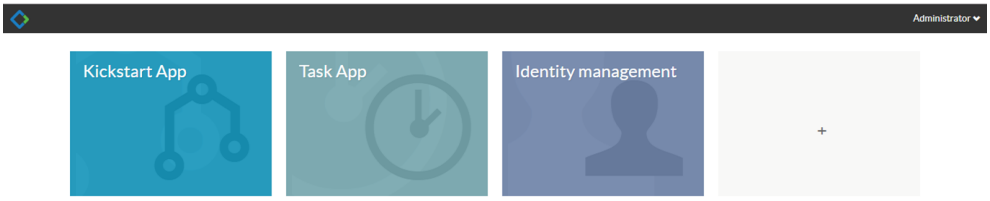


Figure 2.3: Activiti 6 app UI home

Es un entorno web compuesto por tres aplicaciones:

- Kickstart App: incluye un editor BPMN [OMG13] 2 para diseñar modelos de proceso, un editor de formularios para diseñar visualmente formularios, un editor de tablas de decisiones para crear tablas de decisiones de DMN(Decision Model Notation) y un editor de aplicaciones para crear y publicar aplicaciones de procesos que agrupan todos los modelos en un solo paquete. Esta aplicación se relaciona con el componente Process Development presentado en el esquema general de un BPMS [Wes07] en la FIGURA. 2.2.

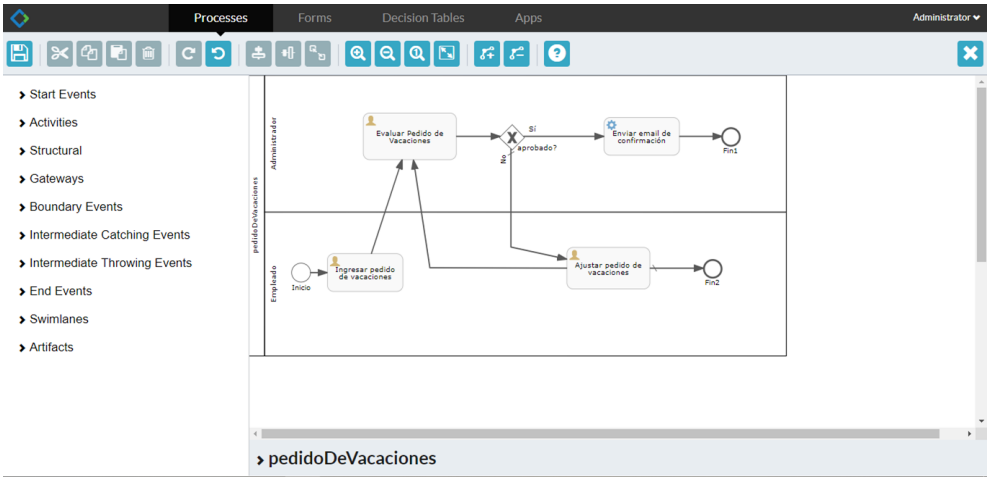


Figure 2.4: Activiti 6 kickStart App- vista del diseñador de procesos

- Task App: Permite iniciar nuevos procesos y tareas, y acceder a tareas asignadas desde cualquier aplicación de proceso. Esta aplicación se relaciona con el componente Task Management presentado en el esquema general de un BPMS [Wes07] en la FIGURA 2.2.

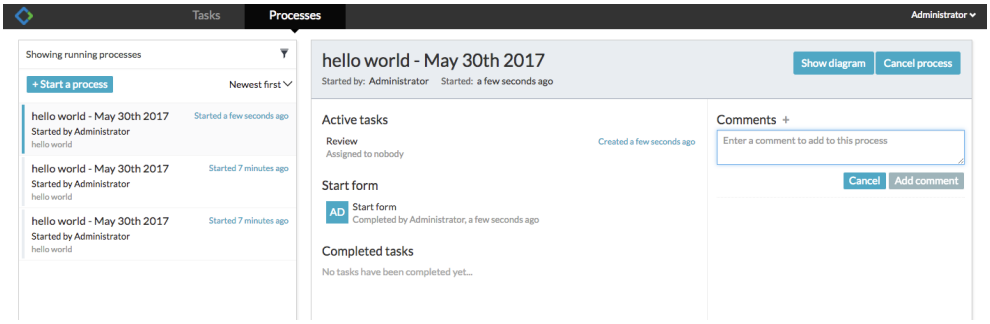


Figure 2.5: Activiti 6 Task App- vista de lista de procesos

- Identity Management App: les da a los administradores la capacidad de crear y administrar usuarios y grupos. Esta aplicación se relaciona con el componente Content Management presentado en el esquema general de un BPMS [Wes07] en la FIGURA. 2.2.

Figure 2.6: Activiti 6 Identity management app- vista de lista de usuarios

Activiti Admin App UI

Es una consola de administración para que los administradores supervisen las tareas en ejecución, las instancias de proceso y jobs. Los administradores pueden realizar diversas acciones, como asignar / reclamar / delegar tareas, finalizar / eliminar instancias de proceso, actualizar / agregar / eliminar variables, ejecutar / eliminar jobs, entre otras. Esta aplicación se relaciona con el componente Operational Monitoring presentado en el esquema general de un BPMS [Wes07] en la FIGURA. 2.2.

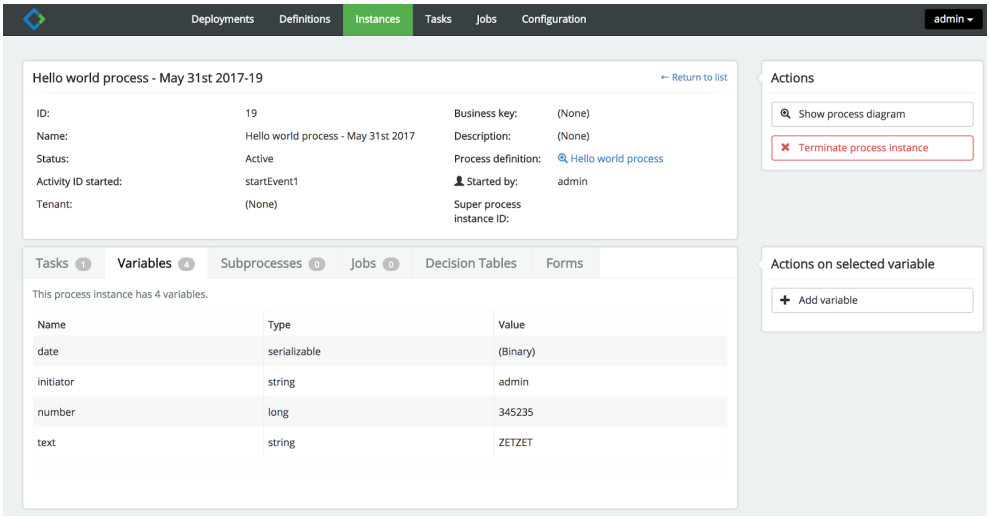


Figure 2.7: Activiti 6 consola Admin- vista detallada de la instancia en ejecución.

Form Designer & Engine

El Diseñador de formularios es un entorno web para diseñar rápidamente formularios sin necesidad de habilidades técnicas especiales. Permite a los usuarios empresariales formar parte de la fase de diseño. El motor de formularios centraliza la lógica de formularios que se introdujo con este Diseñador. Los formularios se pueden implementar con definiciones

de proceso BPMN [OMG13] y también por separado. Pueden ser referenciados tanto en un evento de inicio como en una tarea de usuario.

DMN & Engine

El DMN Engine and Designer permite crear tablas de decisiones y obtener un resultado basado en los valores de entrada y las reglas definidas en la tabla de decisiones. Estas tablas de decisión pueden invocarse desde una tarea de regla de decisión en una definición BPMN [OMG13], pero también independientemente de una instancia de proceso. Esto se relaciona con el componente Rules Management presentado en el esquema general de un BPMS [Wes07] en la FIGURA. 2.2.

Activiti Designer

Activiti viene con un plugin de Eclipse, Activiti Eclipse Designer, que se puede usar para modelar gráficamente, testear y desplegar procesos BPMN [OMG13] 2.0.

2.2.2 Flowable

Flowable [Flo18] es un motor de ejecución de procesos de negocio escrito en Java. El mismo permite desplegar modelos según el estándar BPMN [OMG13] 2.0 (estándar xml para la definición de procesos de negocio). Luego de haber realizado el despliegue, el motor brinda la posibilidad de crear instancias de procesos cuya definición fue realizada en el modelo, tanto como ejecutar consultas sobre los mismos, acceder al histórico o a los procesos activos y cualquier información relativa a los mismos.

Flowable es muy flexible al momento de querer integrar este con una aplicación ya existente. El motor puede ser embebido dentro de la misma como una librería y acceder directamente a él mismo desde la aplicación. Otra posibilidad que brinda este proyecto es la de acceder mediante la API REST. La misma expone una serie de métodos los cuales pueden ser accedidos mediante HTTP. El análisis de estos métodos expuestos se podrá ver más adelante en este trabajo.

Por otro lado también existen un conjunto de aplicaciones que simplifican el uso y el desarrollo de procesos de negocios, las misma son Flowable Modeler, Flowable Admin, Flowable IDM y Flowable Task. A continuación se hará una breve explicación de el uso de cada una de estas aplicaciones y se expondrá alguna imagen a modo de referencia.

Flowable IDM:

Es una aplicación que permite administrar los privilegio de los usuarios. Es decir los usuarios que tienen privilegios administrativos van a poder modificar los privilegios a otros usuarios, crear nuevos usuarios y nuevos grupos dentro de este portal. Las funcionalidades brindadas por esta aplicación se relacionan con el módulo Content Mangement presentado en la FIGURA 2.2. En la FIGURA 2.8 se visualiza el mismo.

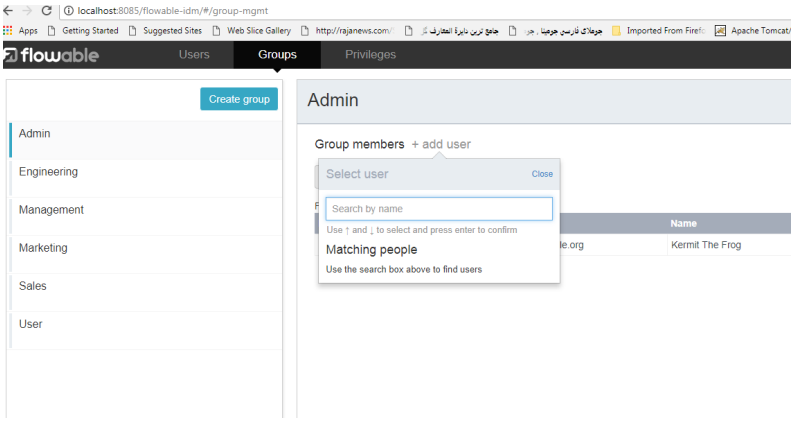


Figure 2.8: Flowable IDM

Flowable Modeler:

Es una aplicación que permite a los usuarios que tienen privilegios de "modeler" a diseñar procesos y formularios. Aquí es donde se puede diseñar un modelo. En la FIGURA 2.9 se puede visualizar la ventana de esta aplicación. Las funcionalidades brindadas por ésta se relacionan con el módulo Process Development presentado en la FIGURA 2.2

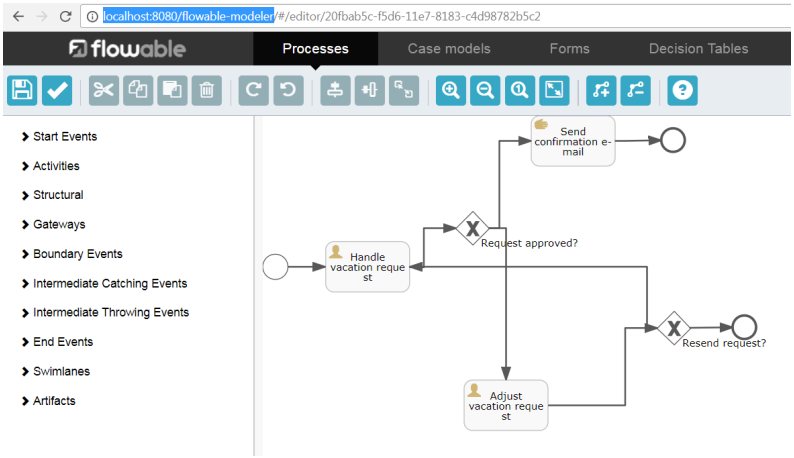


Figure 2.9: Flowable Modeler

Flowable Task:

Es una aplicación para el manejo de tareas en tiempo de ejecución. Provee la funcionalidad

de empezar una instancia de un proceso, editar un formulario de una tarea, completar tareas y consultar tareas e instancias de procesos. Todo el manejo de la ejecución se ve en este portal, como se muestran en la FIGURA 2.10. Estas funcionalidades se relacionan con los módulos Process Execution y Task Management presentados en la FIGURA 2.2.

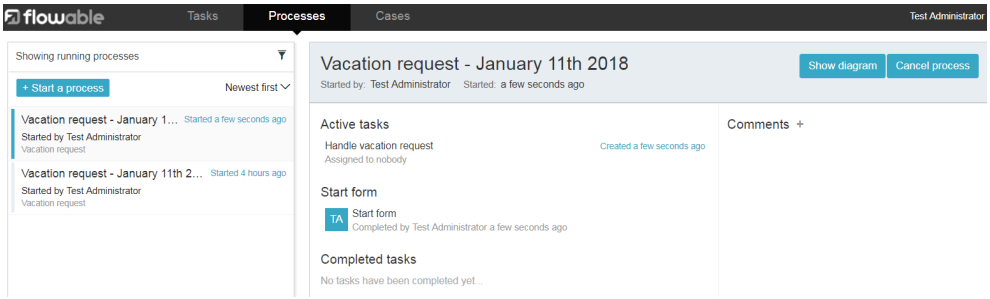


Figure 2.10: iniciar un proceso

Flowable Admin:

Es una aplicación de administración que permite a usuarios con este rol realizar consultas al motor de ejecución. Brinda la información referente de los procesos de instancia, tareas, trabajos y todo lo que respecta. Todo la información que se puede ver o editar es considerada importante para un administrador. La FIGURA 2.11 muestra esta vista. Todas estas funcionalidades se encuentran dentro del módulo Context Mangemant de la FIGURA 2.2.

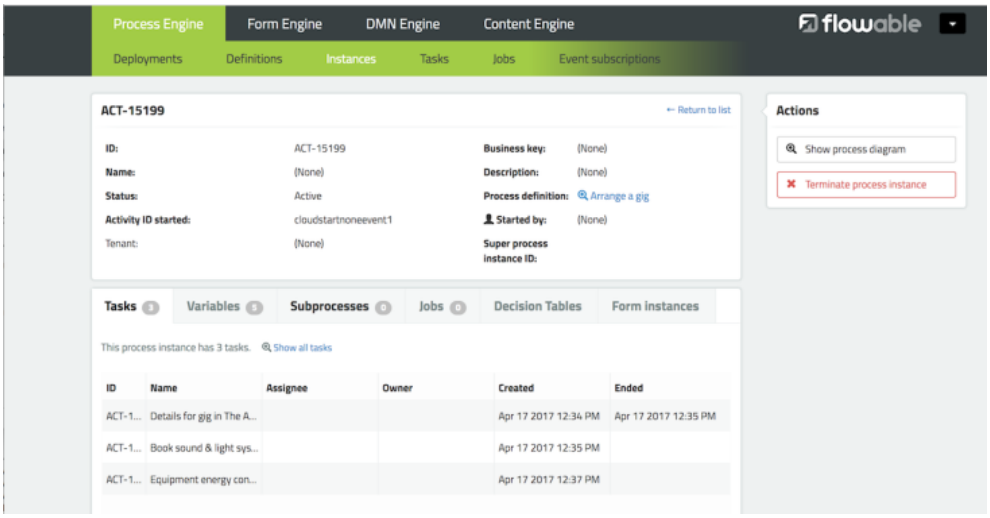


Figure 2.11: Flowable Admin Portal

2.2.3 Bizagi

Bizagi [Biz18] es una plataforma de ejecución de modelos de negocio. Esta plataforma busca adaptarse de manera ágil y automatizada a los diseños ya implementados. La suite de Bizagi se compone en principio por tres herramientas: el modelador de procesos, Bizagi Studio y el Servidor BPM de Bizagi. Estas herramientas están disponibles en la web de Bizagi para descargarlas y usarlas localmente, como también existe la posibilidad de usarlas en la nube. Bizagi dispone de una plataforma Cloud Service en donde se puede acceder a la mismas y establecer un desarrollo colaborativo. A continuación se hará una breve explicación de qué consiste cada uno de estos componentes.

Bizagi Modeler:

Bizagi Modeler es una herramienta que permite modelar y documentar procesos de negocio basados en el estándar BPMN [OMG13]. Las funcionalidades brindadas por esta aplicación se relacionan con el módulo Process Development presentado en la FIGURA 2.2. Permite publicar documentación en Word, PDF, SharePoint o Wiki. Por otra parte, los procesos pueden ser importados y exportados usando Visio, XML, entre otras herramientas. Dentro de esta aplicación, cada archivo se denomina modelo y puede contener uno o más diagramas. Un modelo puede referirse a una organización completa, un departamento o un proceso específico dependiendo de sus necesidades. Los diagramas son posicionados como pestañas. De la mano con lo mencionado anteriormente, existe la posibilidad de guardar estos documentos localmente o en la nube, de esta última forma la herramienta facilita el trabajo colaborativo. En la FIGURA 2.12se muestra una captura de pantalla del diseño de un modelo BPMN [OMG13] hecho con Bizagi modeler.

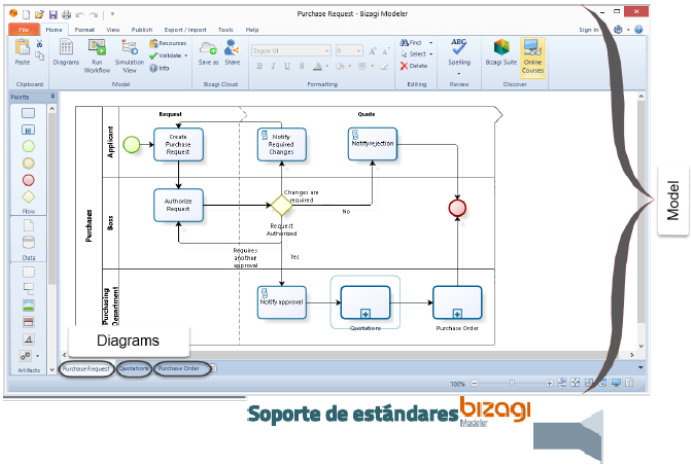


Figure 2.12: Bizagui Modeler

Bizagi Studio: Es el módulo de construcción. En Bizagi Studio el usuario puede definir el flujo de un proceso de negocio, reglas de negocio e interfaz de usuario para la ejecución del mismo. Las funcionalidades brindadas por esta aplicación se relacionan con el módulo Context Mangement presentado en la FIGURA 2.2. Los modelos se guardan en una base de datos y son utilizados posteriormente en la ejecución por Bizagi BPM Server. En la FIGURA 2.13 se muestra la interfaz de Bizagi studio con la vista de experto. Simplemente a modo informativo.

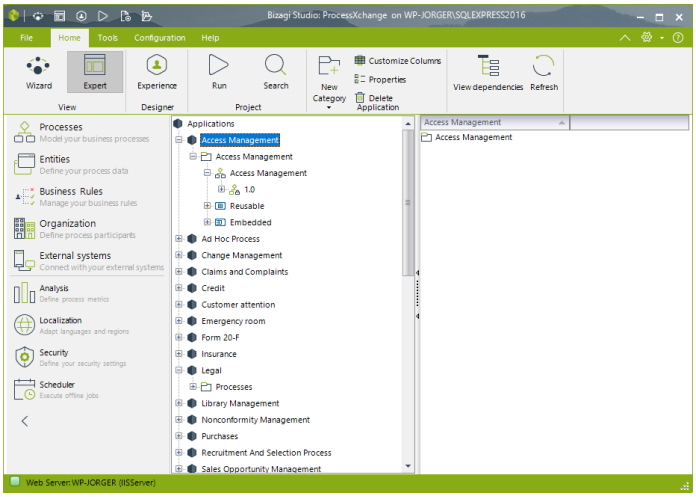


Figure 2.13: Bizagui Studio

Bizagi BPM Server: Se encarga de la ejecución y control del modelo BPMN [OMG13]. Las funcionalidades brindadas por esta aplicación se relacionan con los módulos Process Execuston y Task Managment presentados en la FIGURA 2.2.Bizagi BPM Server ejecuta un Portal de Trabajo para los usuarios finales en un PC o cualquier dispositivo móvil. En la FIGURA 2.14 se ejemplifica como puede verse la ejecución de un modelo de BPMN [OMG13] desde bizagi server.

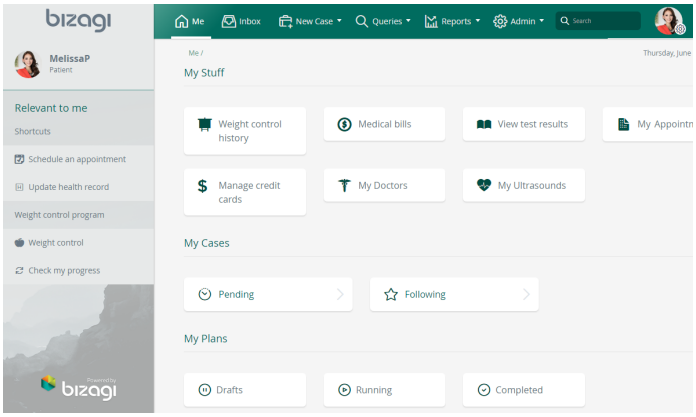


Figure 2.14: Bizagui Server

2.2.4 Bonita

Bonita BPM [Bon18] es una plataforma de desarrollo de aplicaciones de código abierto compuesta por dos principales partes: el ambiente de desarrollo, Bonita Studio y el ambiente de ejecución, Bonita Platform.

Bonita Studio:

Es el ambiente gráfico para la creación de procesos, aplicaciones, modelos y vistas de usuario (páginas y formularios). Contiene tres principales herramientas de diseño: una de ellas llamada Whiteboard que permite crear diagramas de flujo para un proceso, definir detalles de cada paso, transición, puntos de decisión y cualquier elemento de proceso; el menú de desarrollo, que permite crear modelo de datos y el elemento UI designer, que es utilizado para crear páginas y formularios.

Bonita Studio contiene una Bonita Platform (Tomcat, UI Designer, Bonita Portal, Bonita Engine, y una base de datos h2), adecuada para testear una aplicación que está en desarrollo. Cuando se ejecuta un proceso, se despliega automáticamente en la plataforma de desarrollo. A partir de todo esto se puede deducir que Bonita Studio está estrictamente relacionado con el componente Process Development presentado en el esquema general de un BPMS [Wes07] en la FIGURA 2.2

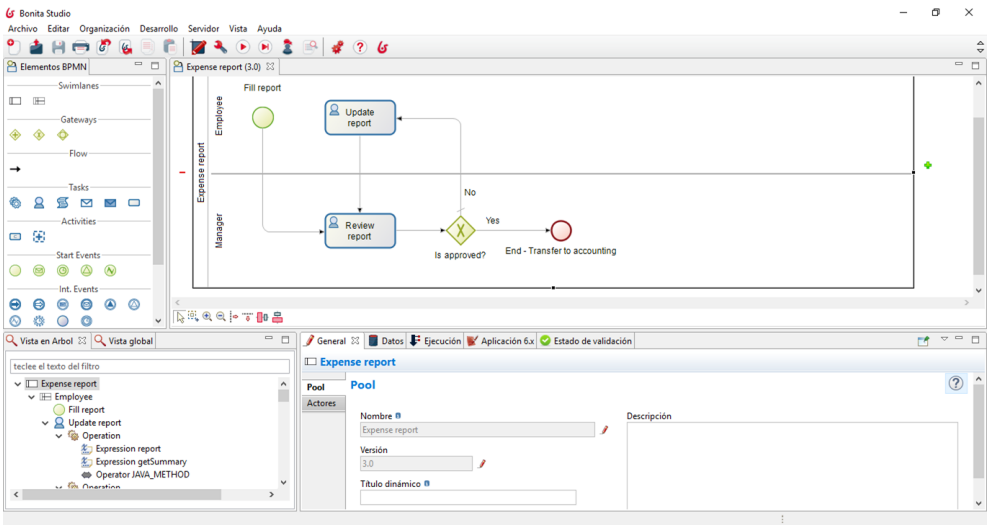


Figure 2.15: BonitaStudio

Bonita Platform:

Está compuesta a su vez por Bonita Engine, el motor de ejecución de Bonita y Bonita Portal, que es la parte visible a los usuarios de proceso. También el portal de Bonita es usado por el administrador "tenant" para desplegar, instalar y administrar procesos y para construir aplicaciones.

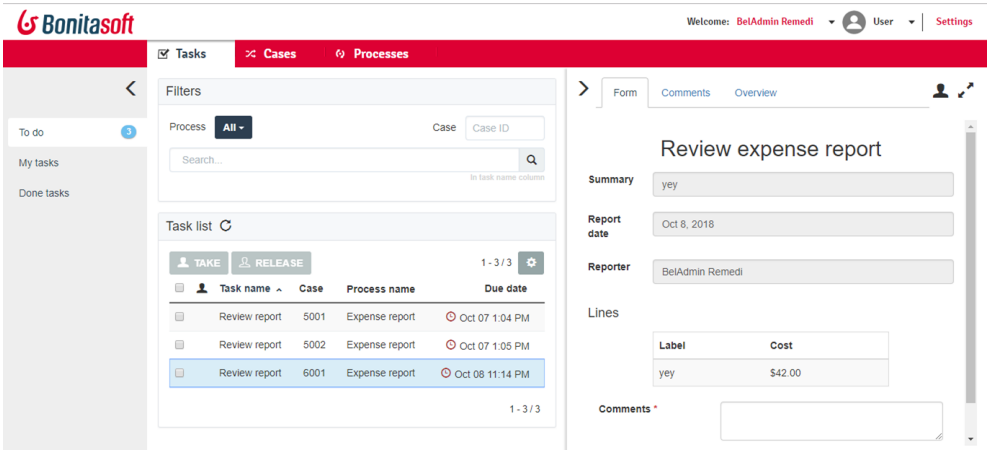


Figure 2.16: Portal Web Bonita

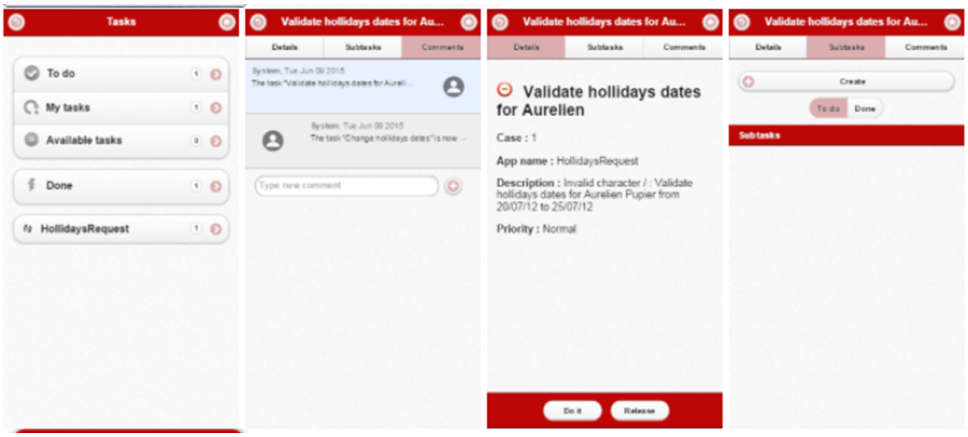


Figure 2.17: Portal de Bonita en su versión Mobile

Haciendo un paralelismo con los componentes principales de un BPMS [Wes07] que se muestran en la FIGURA 2.2, Bonita engine es el componente relacionado a Process execution y por otro lado, Bonita Portal incluye los componentes Task Management, Content Management y Operational Monitoring.

2.2.5 Camunda

Camunda [Cam18a] es una plataforma para la gestión de los procesos de negocios, de código abierto, desarrollada en Java. Los principales componentes manejados en la misma se visualizan en la FIGURA 2.18, así como los potenciales usuarios que utilizan cada uno de estos. Posteriormente se expondrán los mismos.

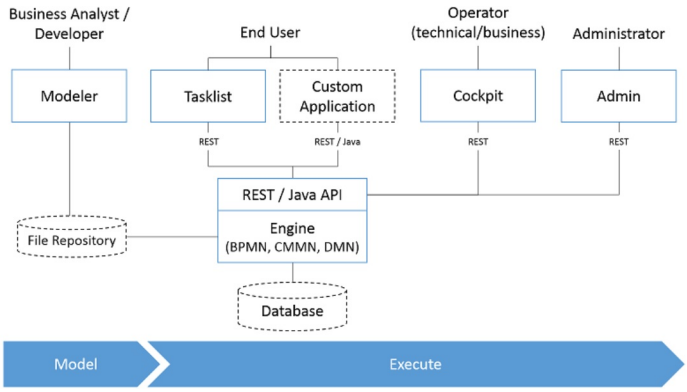


Figure 2.18: Principales componentes en la Plataforma Camunda

Modeler:

Es una herramienta de escritorio de modelado que provee Camunda[Cam18a] para realizar modelos BPMN [OMG13] así como diagramas de Case Management Model and Notation (CMMN,[OMGa]) o tablas Decision Model Notation (DMN,[OMGb]). Brinda las funcionalidades de los modulos Process Development, Rules Management presentados en la FIGURA 2.2. En la *Figura* 2.19 se ilustra dicha herramienta.

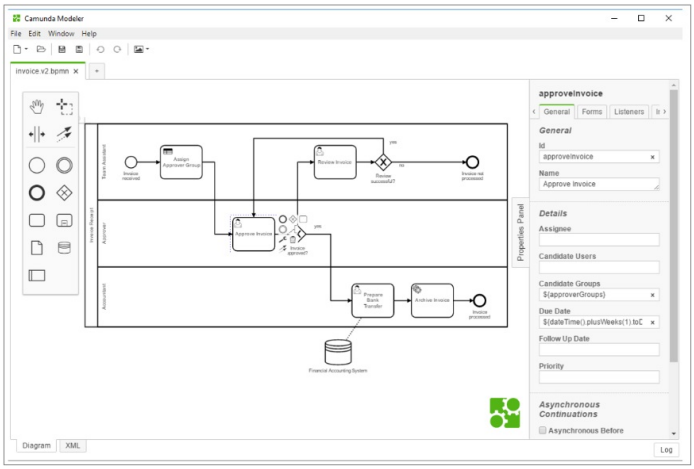


Figure 2.19: Herramienta de Modelado en Camunda

Tasklist:

Es una aplicación web que permite gestionar las tareas de los usuarios. En ella se puede iniciar procesos, crear filtros sobre las tareas existentes, tomar o trabajar sobre algunas de ellas comentando o actualizando información de las mismas, entre otras. Brinda funcionalidades asociadas a los modulos de Process Execution y Task Management presentados en la FIGURA 2.2. En la FIGURA 2.20 se presenta dicha aplicación.

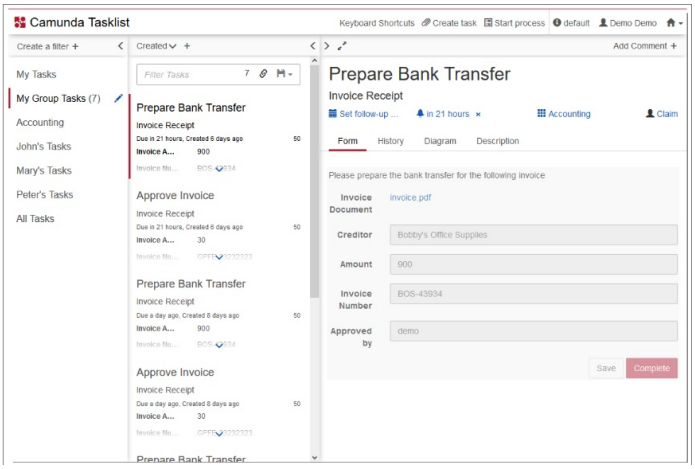


Figure 2.20: Aplicación para gestión de tareas en Camunda

Cockpit:

Es una aplicación web para monitorizar y ver las operaciones en los procesos. Permite definir distintos dashboards que se crean convenientes para el negocio, de forma de visualizar y poder sacar distintas conclusiones sobre los mismos. Brinda las funcionalidades asociadas a los modulos de Dashboarding y Operational Monitoring presentados en la FIGURA 2.2. En la FIGURA 2.21 se ilustra dicha aplicación con algunos dashboards a modo de ejemplo.

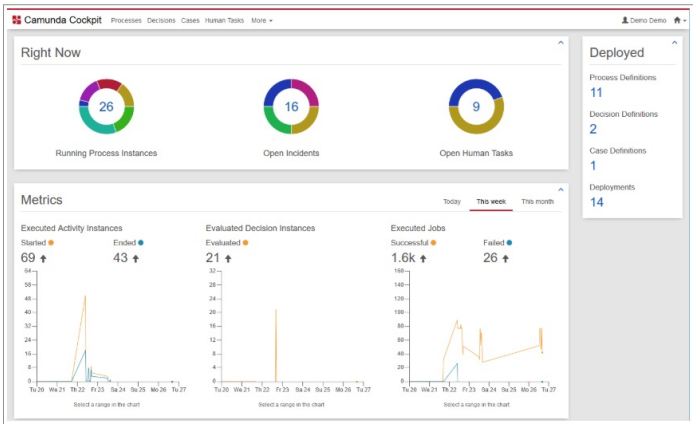


Figure 2.21: Aplicación de monitoreo en Camunda

2.2.6 JBPM

JBPM [JBP18a] es una suite de gestión de procesos de negocios, de código abierto desarrollada en Java por RedHat [Red18]. En ella se puede modelar, desplegar, ejecutar y monitorizar procesos y casos de negocio a lo largo de su ciclo de vida. El poder ejecutar los procesos de negocio hace que la brecha existente entre los usuarios del negocio y los desarrolladores sea mínima. Para esto brinda una web de trabajo, a la cual se le denomina Portal, en donde se podrá realizar y/o administrar todo con gran facilidad utilizando conceptos del dominio especifico.

Para hacer posible todo esto, dicha suite se estructura como en la FIGURA 2.22.

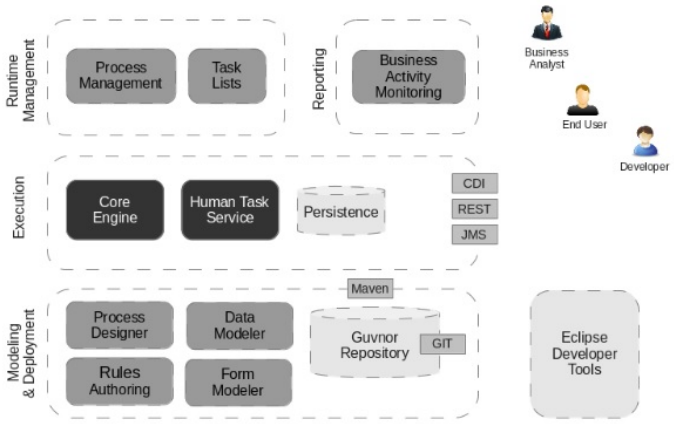


Figure 2.22: Arquitectura Suite JBPM [JBP18a]

Como se puede apreciar, por encima del Motor central, existen varias herramientas y/o aplicaciones que hacen posible el soporte de los procesos de negocio a lo largo de todo su ciclo de vida dentro de la misma. A continuación haremos mención y explicaremos algunas de estas:

Diseñador de procesos:

Dicha aplicación web permite modelar los procesos de negocio en un entorno web, utilizando un editor gráfico para visualizar y editar los mismos. Brinda la funcionalidad asociada al módulo de Process Development presentado en la FIGURA 2.2. A continuación en la FIGURA 2.23 se ilustra la aplicación.

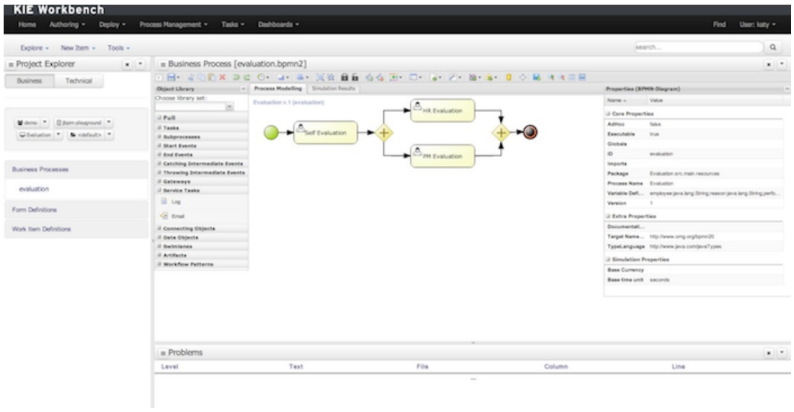


Figure 2.23: Diseñador de procesos en JBPM [JBP18a]

Gestión de instancias de procesos y tareas:

La presente aplicación es la que brinda la posibilidad de gestionar las instancias de los procesos y las tareas de los usuarios. Respecto a los procesos se permite iniciar nuevas instancias, listar todas las existentes así como visualizar el estado de alguno en particular, entre otras. En lo que refiere a las tareas se permite obtener todas las tareas actuales para el usuario actual, así como completar y/o actualizar una de estas. Se vinculan dichas funcionalidades brindadas por esta aplicación con los módulos Task Management y Process Execution presentados en la FIGURA 2.2. Dicha aplicación se presenta en la FIGURA 2.24.

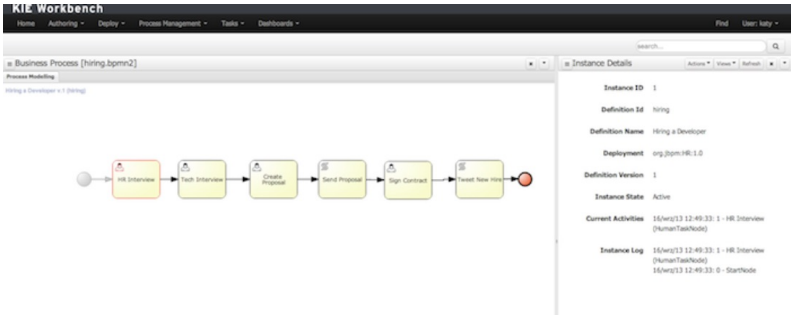


Figure 2.24: Gestión de instancias de procesos y tareas en JBPM [JBP18a]

Seguimiento de actividades:

Se dispone de una herramienta para desarrollar distintos dashboards e informes con información importante sobre nuestro negocio. De esta forma nos es posible realizar un monitoreo de nuestras actividades y obtener reportes de una forma sencilla. Brinda funcionalidades asociadas a lo modulos Dashboarding y Operational Monitoring presentados en la FIGURA 2.2. A continuación en la FIGURA 2.25 se presenta una serie de dashboards de ejemplo.

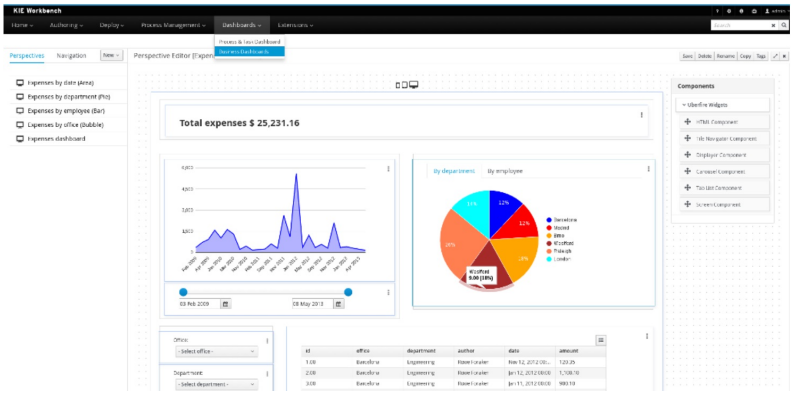


Figure 2.25: Herramienta para seguimiento de actividades en JBPM [JBP18a]

2.3 Trabajos relacionados

A continuación se resumen diversos trabajos relacionados con la definición de un modelo de datos para los distintos BPMS, en función de presentar el estado actual del tema.

2.3.1 Towards a generic BPMS user portal definition for the execution of business processes

El trabajo [DCA16] consistió en realizar un modelo genérico dando soporte a una API genérica y un portal genérico que contemple los recursos manejados por los motores Activiti [Act18], Bonita [Bon18] y Bizagi [Biz18], centrándose principalmente en la ejecución. Dentro de esta área se consideraron las funcionalidades básicas que un usuario final realizaría al momento de utilizar el mismo.

En la FIGURA 2.26 se presenta el modelo de dominio para el portal genérico presentado en el mismo.

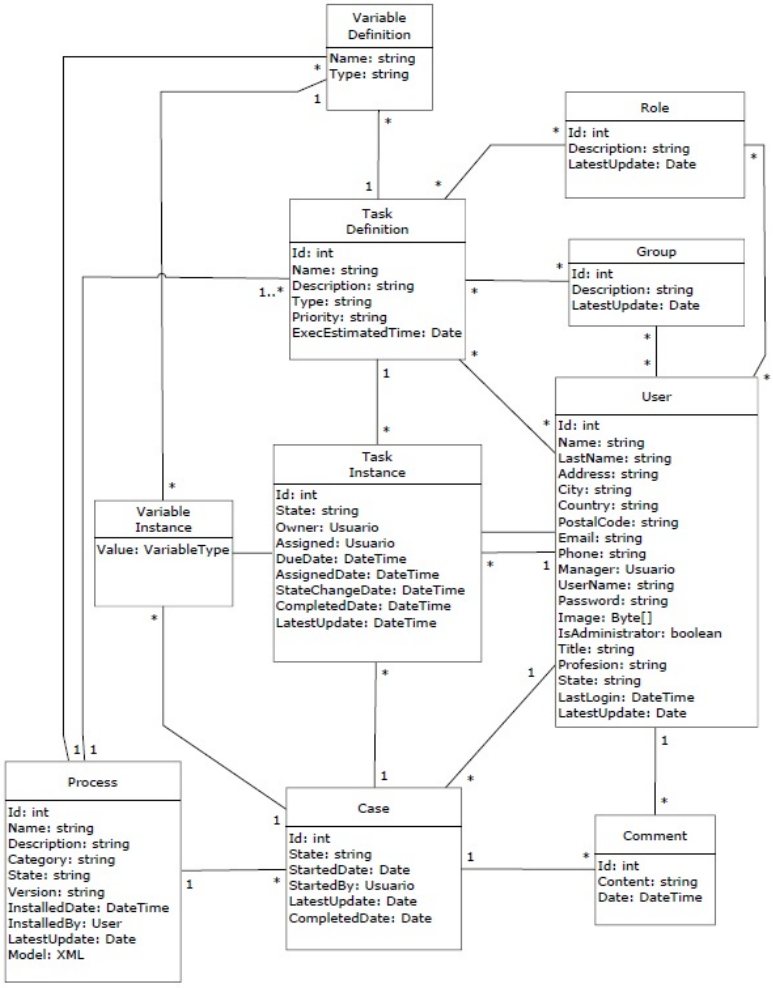


Figure 2.26: Modelo Genérico presentado [DCA16]

Dicho modelo fue el resultado de la unificación de concepto de los tres motores contemplados. A modo de ejemplo, los conceptos Process y Case representan la definición de un proceso y su instancia respectiva. De igual forma Task definition y Task instance; Variable definition y Variable instance. También se visualizan los conceptos de User, Role y Group los cuales permiten gestionar lo que en una organización se conoce como autorizaciones de los usuarios sobre las actividades y tareas. Esta la contemplamos en el rol que un usuario tenga. Por ultimo incluyo el concepto de Comment, el cual se permite realizar sobre una instancia de proceso (Caso).

El enfoque tomado al momento de realizar la implementación tanto de la arquitectura como del software en sí fue el de manejar un motor a la vez, es decir, se debían hacer las configuraciones necesarias para la utilización de un motor en concreto. En caso de optar por cambiar de motor se debía realizar una nueva instanciación con las configuraciones adaptadas y el adaptador correspondiente.

Respecto al portal genérico, manejó WebML como lenguaje de modelado, el cual es una extensión del lenguaje IFML, permitiéndole representar las interfaces web de acuerdo a componentes e interacciones. De esta forma planteó un desarrollo dirigido por modelos. Como herramienta utilizó WebRatio, el cual es un IDE que soporta el modelado de aplicaciones web. En la FIGURA 2.28 se muestra un diagrama del modelo de la interfaz de usuario del portal genérico proporcionado.

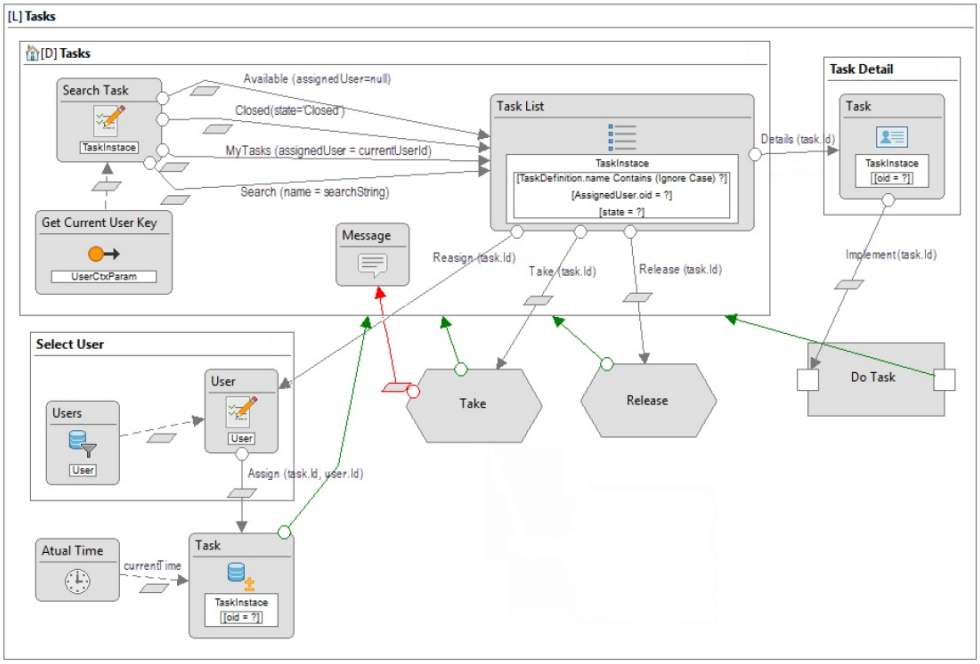


Figure 2.27: Diagrama IFML página de Tareas [DCA16]

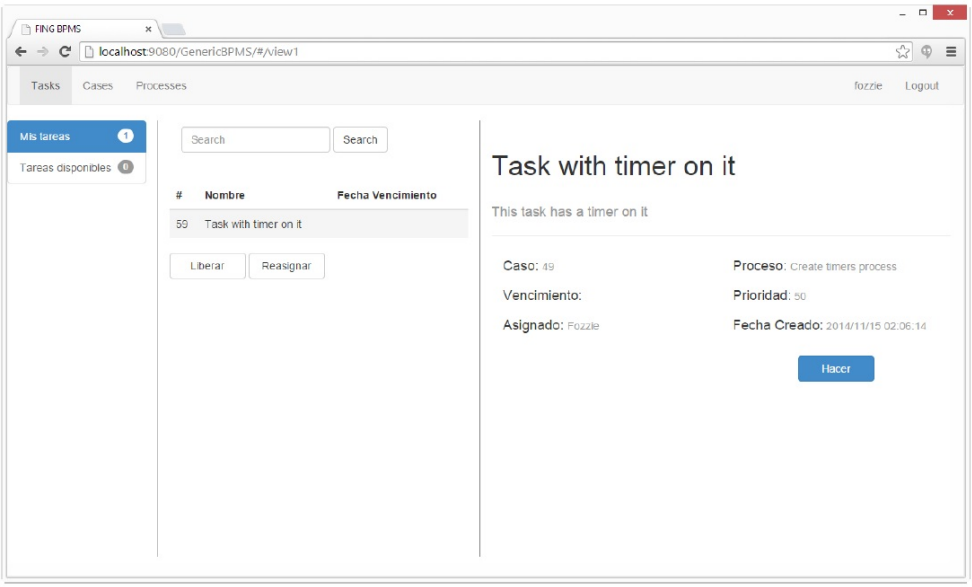


Figure 2.28: Pagina de Tareas - Portal genérico [DCA16]

2.3.2 AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects

En este trabajo [KR06] los autores definen un meta-modelo AMFIBIA, que formaliza principales aspectos y conceptos de un modelo de procesos de negocio.
Previo a esto, categorizan los aspectos básicos de un proceso de negocio en tres principales:

- Aspecto de Control: describe el orden en el cual las diferentes actividades son ejecutadas.
- Aspecto de Organización: describe la estructura de la organización y, en particular, los recursos y agentes, y de qué manera están involucrados en el proceso de negocio.
- Aspecto de información: describe la información que está involucrada en un proceso de negocio, cómo se representa y cómo se propaga entre diferentes actividades.

Los conceptos de actividad, tarea, proceso, y casos aparecen en todos los aspectos y pertenecen al núcleo del modelado de procesos de negocios, es por esto que cada uno de los aspectos se muestra disjunto pero intersectándose con este núcleo común.

Como se observa en la FIGURA 2.29, los conceptos también se distinguen entre aquellos de instancia y de modelo.

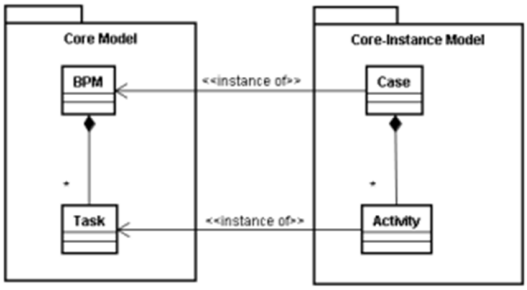


Figure 2.29: Núcleo del modelo AMFIBIA [KR06]

También se define la relación entre un elemento en el núcleo y su correspondiente en un aspecto, tal como se ve en la FIGURA 2.30.

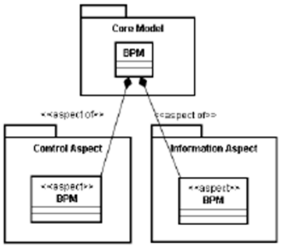


Figure 2.30: Relación entre elementos del núcleo y elementos de un aspecto [KR06]

En este trabajo, si bien el foco está en el modelado de procesos, los conceptos de modelo se distinguen de los de instancia, con una correspondencia directa entre ellos. Por ejemplo, la correspondencia que se hace entre Process-Case. Estos llamados "conceptos de instancia" están relacionados con una ejecución particular y es aquí donde se encuentra relación con el trabajo previo [DCA16] que también considera conceptos de ejecución, teniendo en cuenta más conceptos de este estilo.

2.3.3 Workflow Management Facility Specification

En este artículo [OMG00] se definen los flujos de trabajo y los manejadores de estos flujos. Se describe como principal característica, que estos manejadores automatizan los procesos en los que se envuelve una combinación entre actividades computarizadas y humanas. Particularmente esas en los que existe interacción con aplicaciones y herramientas de tecnología de la información.

Se plantea la inexistencia de un framewok que permita a diferentes administradores de flujos de trabajo y a aplicaciones que utilicen los mismos, trabajen en conjunto. Se refiere a este problema como "resultando en islas incompatibles de automatización de procesos". Como solución se plantea la facilidad que lleva el nombre del artículo aprovechando las características comunes que tienen estos sistemas y que tienen la potencialidad de alcanzar un nivel de interoperabilidad bajo el uso de estándares.

El framework se basa en la abstracción realizada en el trabajo [Son97], en donde se identifican las distintas áreas funcionales llevadas a cabo por este. A partir del mismo se obtiene el siguiente dominio, FIGURA 2.31, para la especificación de las interfaces para el control de ejecución del flujo de trabajo, el monitoreo y la interoperabilidad entre ellos.

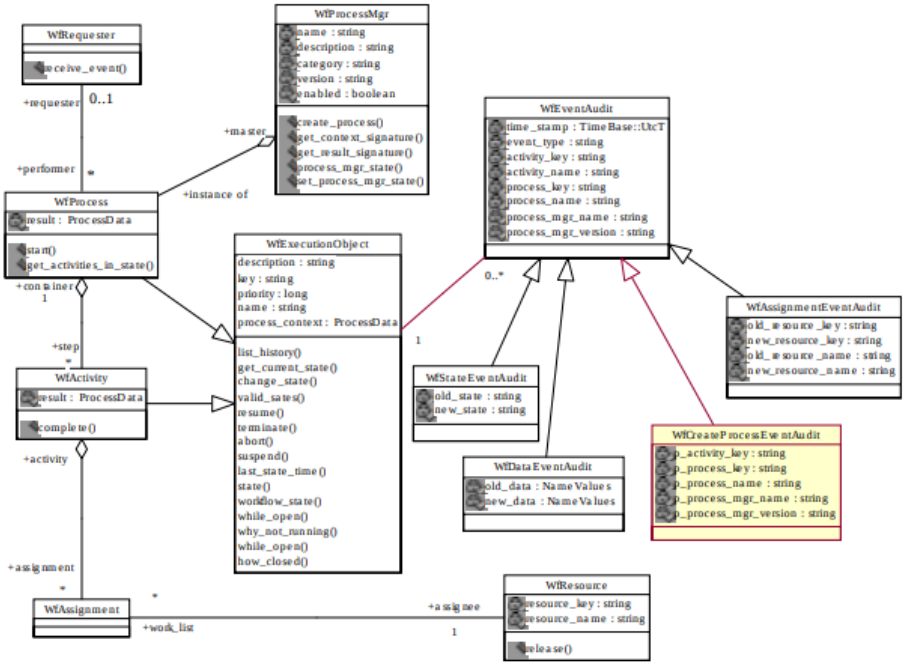


Figure 2-1 Joint Workflow Management Facility Model

Figure 2.31: Workflow Reference Model Facility- Componentes e interfaces [OMG00]

En este trabajo se definen un conjunto de características que tienen todos los BPMS [Wes07]. Este enfoque se relaciona con el trabajo [DCA16] aunque en el último, el nivel de abstracción que se logra es distinto. Se profundiza a nivel conceptual, es decir se logra unificar conceptos entre distintos BPMS [Wes07] pero sin perder especificidad entre cada uno. En [OMG00]

se agrupa también la información de los BPMS [Wes07] según la definición de procesos, en el trabajo [DCA16] si bien se modelan los conceptos, no se tratan las funcionalidades al respecto.

2.3.4 WfMC, The Workflow Reference Model. Workflow Handbook

En este trabajo [Son97] se presenta un "Modelo de Referencia" común para los sistemas de gestión de flujo de trabajo (WFMS por sus siglas en inglés). Explica que estos sistemas son aquellos que proporcionan automatización procedural de un proceso de negocio a través de la gestión de una secuencia de actividades de trabajo y la invocación de recursos humanos y/o de IT asociados a los distintos pasos de estas actividades.

En FIGURA 2.32 se ilustra los principales componentes e interfaces dentro de la arquitectura de flujo de trabajo.

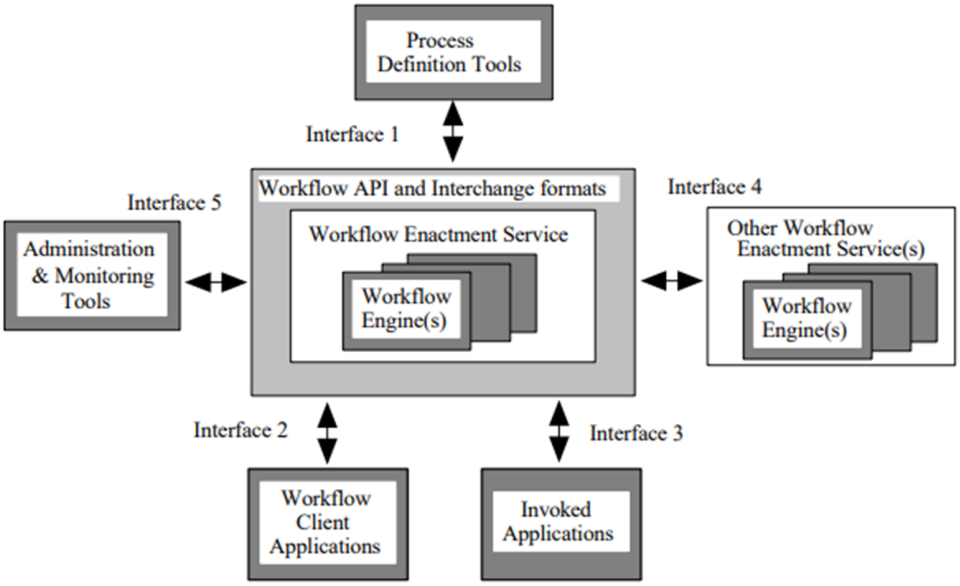


Figure 2.32: Workflow Reference Model- Componentes e interfaces [Son97]

Breve descripción de las áreas identificadas:

- Process Definition: Especificaciones para los datos de definición de proceso y su intercambio con el entorno de ejecución de flujo de trabajo.
- Workflow Interoperability: interfaces que dan soporte a la interoperabilidad entre difer-

entes sistemas de flujo de trabajo.

- Invoked Applications: interfaces que dan soporte a la interacción con una variedad de tipos de aplicación IT.
- Workflow Client Applications: interfaces que soportan la interacción con funciones de escritorio de la interfaz de usuario.
- Administration and Monitoring: interfaces para proporcionar monitoreo y funciones métricas para facilitar la administración de entornos compuestos de aplicaciones de flujo de trabajo.

Si bien estás áreas son clave para cualquier motor de procesos, el enfoque que se le da es con un nivel de abstracción más alto que el que se le da en el trabajo previo [DCA16]

2.3.5 Separating Execution and Data Management: A Key to Business-Process- as-a-Service (BPaaS)

En el presente trabajo [SY14] se plantea la problemática necesidad de adaptar los BPMS [Wes07] ante los cambios en las resoluciones y políticas del mercado, de requerimientos, de su entorno así como de su competencia. Esto implica asumir costos importantes en cada organización.

El principio fundamental para BPaaS es la independencia entre la gestión de la información y la gestión de la ejecución. Esto implica que el motor de ejecución no debe manejar ningún datos, así como el que gestiona los datos no debe interferir en las decisiones que tome el motor de ejecución.

Para ejecutar los procesos de negocio menciona la necesidad de manejar cinco tipos de información, estos son: información del negocio, modelos de los procesos, estado de la ejecución y histórico, la correlación entre las instancias de los procesos y los recursos y sus estados.

Para evitar el desarrollo de un nuevo BPMS [Wes07], con el fin de brindar BPaaS, para proporcionar la independencia necesaria se definen unos artefactos auto-guiados los cuales son un empaquetado que contiene toda la información necesaria para que este ejecute, como ser el modelo y la información en si. A su vez se definió un framework el cual utiliza dichos artefactos por un lado, y por otro los motores de ejecución empaquetándolos como servicios sin estado para soporte a esto. Dicho trabajo fue una extensión de otros anteriores.

En comparación con el trabajo [DCA16], aquí se manejan conceptos con un nivel de abstracción mas alto. Por otro lado también se manejan los modelos de BPM y una correlación entre las instancias.

2.3.6 Workflow-based Process Controlling: Foundation, Design, and Implementation of Workflow-driven Process Information Systems

En este libro [Mue04] se analizan las funcionalidades en lo que respecta a la administración y el monitoreo de los BPMS [Wes07]. El objetivo del mismo es lograr un meta modelo que permita auditar la información de un BPMS [Wes07]. En contraposición, el modelo presentado en [DCA16] cubre todos los conceptos necesarios para ejecutar un Proceso en un BPMS [Wes07].

En principio se realizó el análisis y comparación de cuatro modelos de auditoría, tres modelos correspondientes a los siguientes sistemas de workflow, IBM MQSeries Workflow, Staffware 2000, y Carnot Process Engine y el modelo de referencia WfMC FIGURA 2.32.

Dentro de este análisis se obtiene que el único modelo donde se encuentra representado el concepto de usuario es en el de Carnot. Vale aclarar que si bien en los otros modelos no está representado, el texto menciona que se pueden mapear los identificadores de los objetos con información que se encuentra en el modelo organizacional. Esto difiere con lo presentado en el trabajo [DCA16].

El texto concluye el análisis de los cuatro modelos presentados con un meta modelo FIGURA 2.33 de referencia para el contenido de auditoría. En dicho meta modelo se encuentra el concepto de evento, y de originador de evento, que puede ser a su vez discriminado en “Workflow Engine”, el cual puede causar eventos de sistema y “Workflow Participant”, que son quienes manipulan activamente los objetos presentados por el "Workflow Engine". En contraposición, en el modelo de del trabajo [OMG00] solamente se encuentra el concepto el usuario que puede identificarse con el concepto “Workflow Participant” y los eventos originados por este tipo de usuario. Nuevamente, esto se debe al enfoque que se le da a ambos modelos.

Otro aspecto relevante en el meta modelo de referencia, es que se encuentra el concepto de Estado modelado con una estructura genérica en donde pueden ser estados simples o complejos, es decir compuestos por otros estados. Un estado además se relaciona con un tipo de evento y un tipo de evento se puede relacionar con muchos estados.

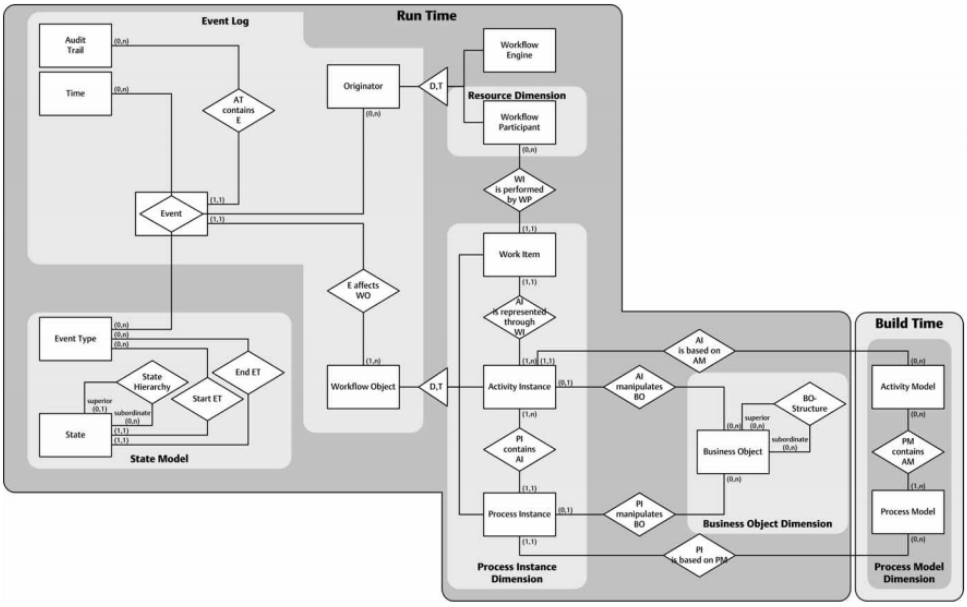


Figure 2.33: Workflow zur Muehlen [Mue04]

3 Análisis de BPMS

En este capítulo se analizarán los BPMS [Wes07] Activiti [Act18] en su versión 6, Bonita [Bon18] en su versión 7, Flowable [Flo18] en su versión 6.4, Bizagi [Biz18] en su versión 11, Camunda [Cam18a] en su versión 7.9 y JBPM [JBP18a] en su versión 7.7, para lograr el objetivo general de este proyecto, que es desarrollar un portal genérico y una API genérica que brinda soporte a este para la ejecución de procesos de negocio, a partir de la definición de un modelo de datos. Extendiendo, de esta forma, el trabajo [DCA16] en relación a dos aspectos, los portales y los servicios que estos proveen.

Se mostrarán cada una de las funcionalidades a las que un usuario final puede acceder en el portal web, dejando de lado otros aspectos de interés como son el modelado, la gestión y los dashboards. Continuando con esta idea, el análisis de las APIs contemplará las mismas funcionalidades.

Como resultado se brindará una descripción de los servicios y un modelo de datos que abstrae esta información clasificando los conceptos según Administración, Ejecución e Histórico.

3.1 Activiti

Análisis del portal web:

Las aplicaciones web llamadas IdentityApp y TaskApp permiten a los usuarios interactuar con el motor de ejecución de Activiti [Act18]. Identity App permite cambiar configuraciones de perfil, además, a los usuarios administradores administrar grupos y usuarios. TaskApp provee acceso a la lista de tareas para que un usuario pueda trabajar en cualquiera asignada desde cualquier proceso. También se pueden iniciar procesos y tareas.

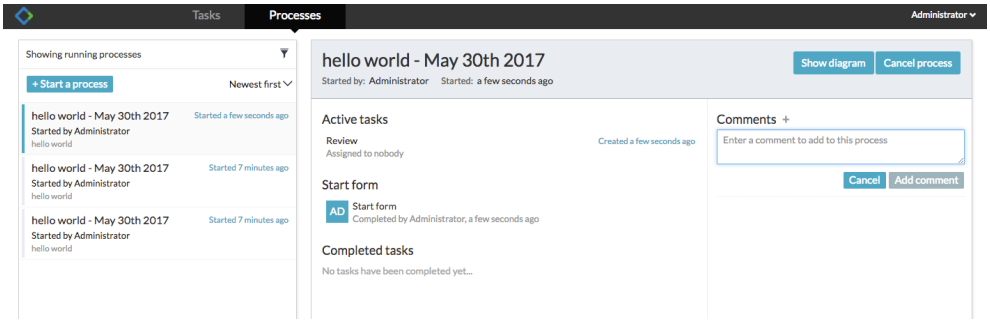


Figure 3.1: Activiti 6 Task App- vista de lista de procesos [Act18] [Flo18]

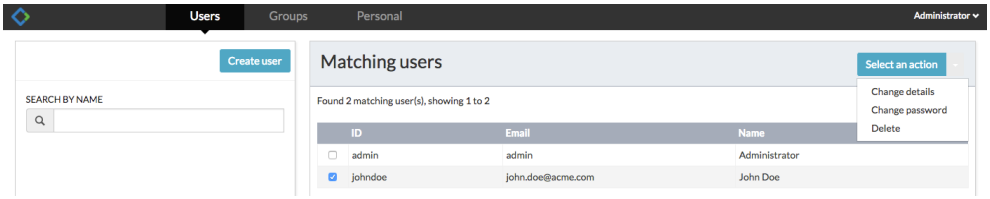


Figure 3.2: Activiti 6 Identity management app- vista de lista de usuarios [Act18]

API expuesta

Activiti [Act18] incluye una API REST en el motor de procesos. Todos los recursos REST requieren un usuario válido para ser autenticado por defecto. Se usa autenticación de acceso HTTP básica, por lo que siempre se debe incluir un **Authorization Basic** al realizar solicitudes o incluir el nombre de usuario y la contraseña en la url de solicitud (por ejemplo, http: // nombre de usuario: contraseña @ localhost).

Los recursos que maneja la API agrupados por sub API son:

- **Repository**
  - deployments: componentes utilizados para la instalación de procesos de negocio, que luego serán ejecutados. Pueden incluir procesos BPMN [OMG13] 2.0, formularios para la ejecución de tareas, reglas, etc.
  - process-definitions
  - models
  - process-instances
- **Runtime**

  - executions: representa el concepto de Token en BPMN [OMG13] 2.0. Es básicamente un puntero a dónde la instancia de proceso se encuentra actualmente.
  - tasks
  - signals: "Señales" que pueden enviarse al motor.
- **History**
  - historic-process-instances
  - historic-variable-instances
  - historic-task-instances
  - historic-activity-instances
  - historic-detail
- **Form**
  - form-data
- **Management**
  - tables
  - properties
  - engine
  - jobs
- **Identity**
  - users
  - groups

En la tabla 3.1 se encuentra la descripción de los métodos de la API REST de Activiti.

Método	**Operación**
GET	Obtener un recurso o una colección de recursos.
POST	Crear un recurso. También es usado para ejecutar consultas sobre los recursos que tienen una estructura de solicitud demasiado compleja para agregar en la URL de una solicitud GET.

PUT	Actualizar propiedades de un recurso existente. También es utilizado para invocar operaciones sobre un recurso existente.
DELETE	Borrar un recurso existente.

Table 3.1: Métodos HTTP y correspondientes operaciones en Activiti

Se pueden especificar parámetros como parte de una URL. Por ejemplo: `http://host/actviti-rest/service/repository/deployments/{deploymentId}`

Parámetros agregados en la URL como query-string pueden ser de tipo String, Integer, Long, Boolean, Date. Por ejemplo: `http://host/activiti-rest/service/deployments?name=Deployment`

Los parámetros de paginación y orden se pueden agregar como query-string en la URL. Por ejemplo: `http://host/activiti-rest/service/deployments?sort=name`.
Se pueden especificar los siguientes parámetros:

- sort: nombre de una clave de búsqueda.
- order: orden de los elementos(asc, desc).
- start: parámetro para la paginación del resultado. Por defecto comienza en 0.
- size: parámetro para la paginación del resultado. Por defecto el tamaño es 10.

En la FIGURA 3.3 se puede ver el modelo de datos con los conceptos identificados en Activiti.

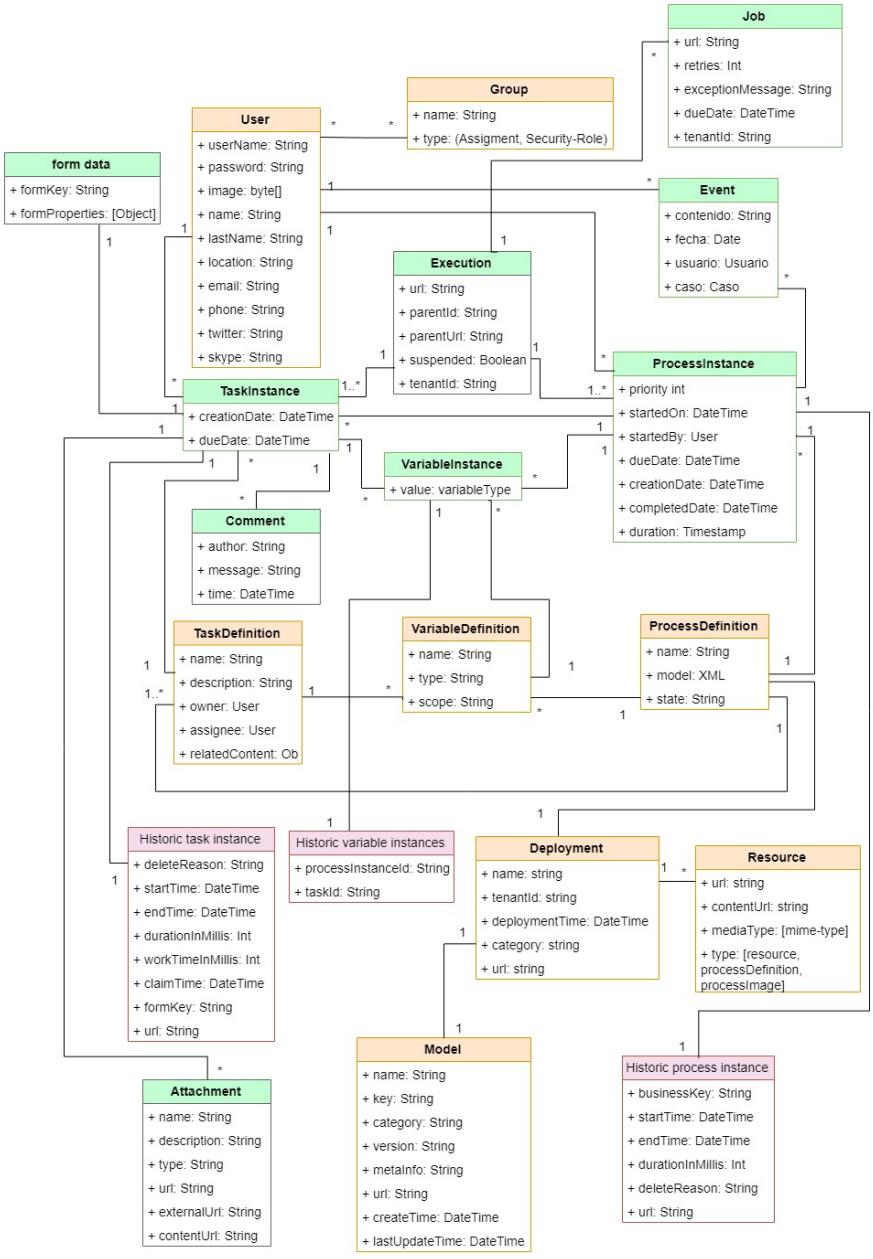


Figure 3.3: Modelo de datos de Activiti

3.2 Flowable

Análisis del portal web:

Este motor de Flowable [Flo18] es muy parecido, estructuralmente hablando, al de activiti. Este proyecto es libre y de código abierto. Como se menciono en el estudio realizado en el estado del arte el mismo tiene varios componentes. En esta sección se mostrará brevemente las operaciones principales que puede realizar un usuario de Flowable en el componente flowable-task que es el que se usa para la ejecución de un modelo BPMN [OMG13].

En la FIGURA 3.4 se muestra el home del portal web con la pestaña de procesos seleccionada. Aquí se ven los distintos despliegues de modelos que el motor tiene en ejecución. También se puede iniciar los mismos entre otras opciones que se visualizan.

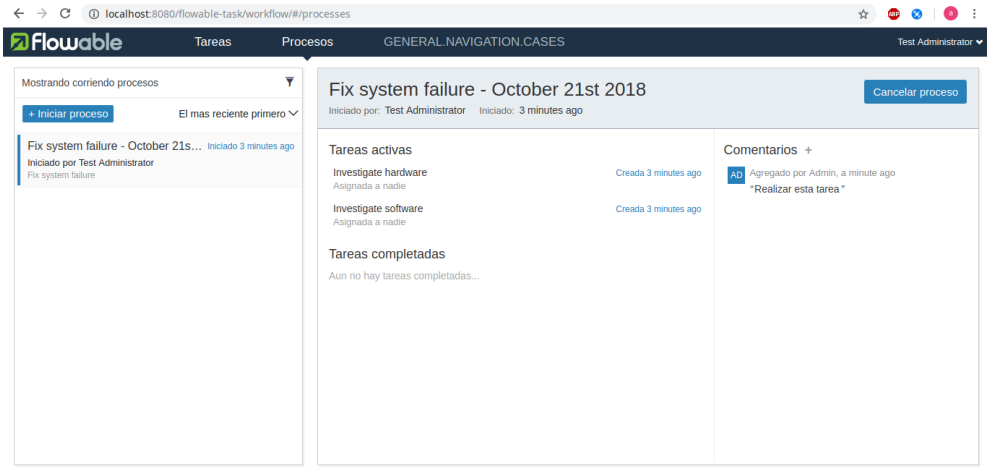


Figure 3.4: Flowable Procesos [Flo18]

En la FIGURA 3.5 se visualiza la pestaña de tareas de un usuario, a la izquierda se muestran todas las tareas disponibles en el sistema, y a la derecha información contextualizada a la tarea seleccionada. En este lugar es donde se realiza una tarea. Se puede asignar a otro usuario y realizar las operaciones pertinentes.

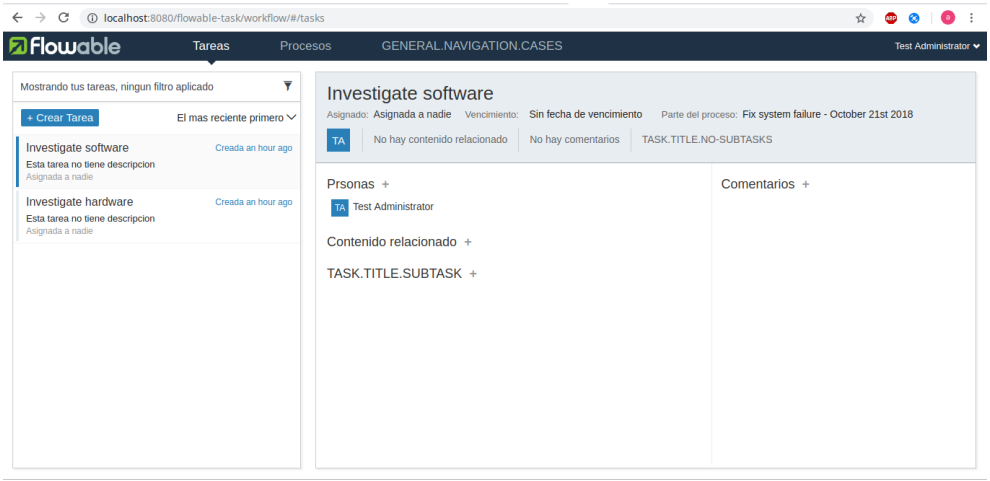


Figure 3.5: Flowable Tareas [Flo18]

API expuesta:

Al realizar el estudio de la misma, se llegó a un modelo de dominio compuesto por los conceptos que se pudieron recabar. Tras haber obtenido el modelo, se comparó con los existentes y se concluyó que el modelo de datos que tenía era exactamente el mismo al que se recabó de Activiti. Por su parte, los servicios REST que exponen ambos motores también son idénticos. Se investigó y se obtuvo el resultado de que el proyecto Flowable es un un fork del proyecto Activiti. Se cambió el código y se agregaron algunas funcionalidades dentro del motor, pero estructuralmente y a nivel del análisis que se hizo no agrega información al modelo de Activiti. Se decidió entonces, usar dicho dominio como referencia.

3.3 Bizagi

Análisis del portal web:

Se relevó las principales características del portal web que brinda Bizagi [Biz18]. Se destaca en este cliente web, que la interfaz de usuario es amigable y sencilla de usar. Por otra parte existe un material muy basto en la página del producto, en donde se tomaron las imágenes que se muestran a continuación y se explica claramente cada una de las herramientas que el portal contiene.

Una vez ingresado, al usuario se le presenta la FIGURA 3.6:

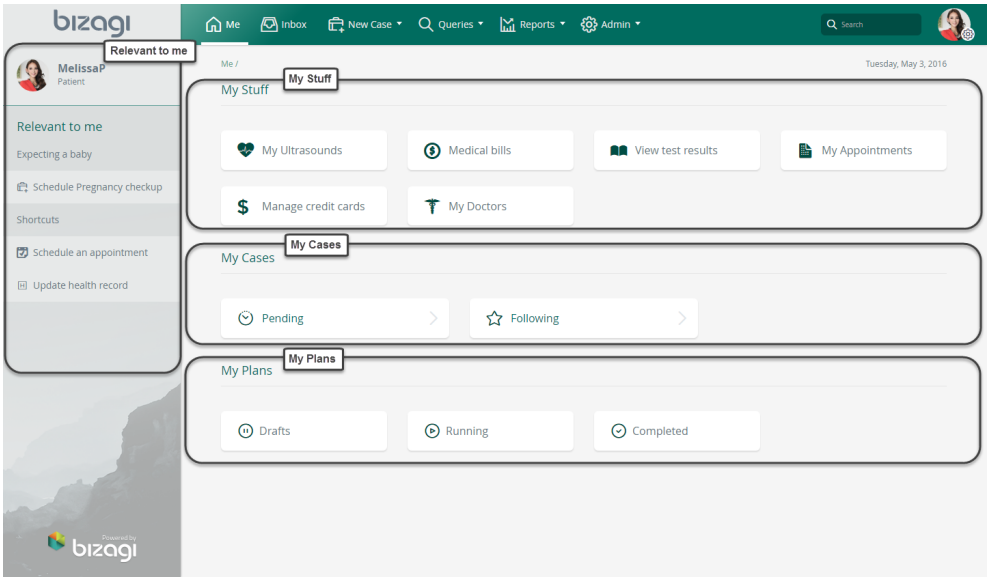


Figure 3.6: Bizagi Home del portal [Biz18]

Esta información se puede dividir en cuatro secciones:

- My stuff: contiene toda la información que pertenece a cada usuario y es relevante para su trabajo.
- My cases: lista todos los casos abiertos y cerrados que son relevantes para el usuario que ha iniciado sesión.
- My plans: funcionalidad que permite crear planes dinámicamente para ayudar al usuario a controlar una serie de actividades sobre la marcha, decidiendo las actividades que necesitan ser realizadas, el orden en el cual son ejecutadas (secuencial o paralelo), estableciendo fechas de vencimiento y la persona responsable para cada una de ellas.
- Relevant to me: muestra atajos a los procesos frecuentemente lanzados o un conjunto de procesos sugeridos por Bizagi de acuerdo al contexto.

Luego, existe el menú de Inbox. El mismo está diseñado para presentar todas las actividades asignadas y facilitar el acceso a las tareas pendientes. Este está compuesto de diferentes áreas que le ayudarán a un usuario realizar, administrar y controlar sus procesos. En esta área es en donde se realizan las tareas asignadas, como se muestran en la FIGURA 3.7.

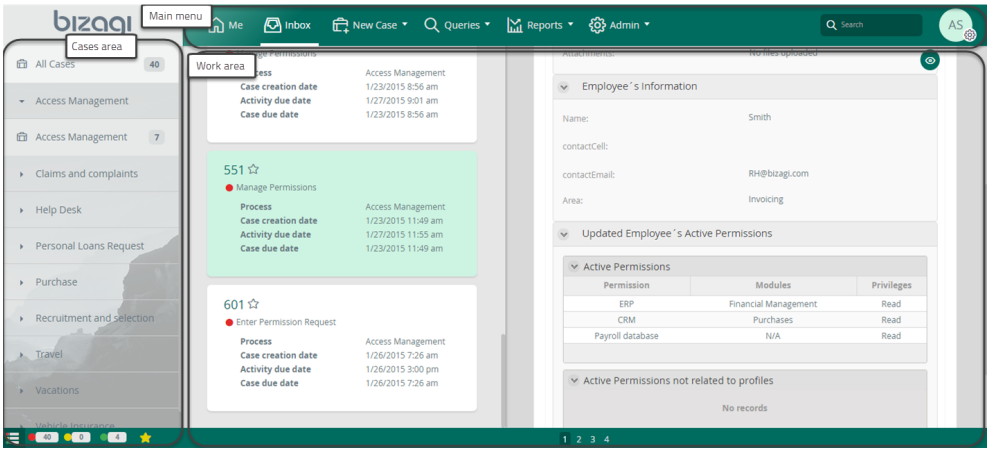


Figure 3.7: Bizagi Menu inbox [Biz18]

Finalmente, se cree interesante presentar que este motor tiene un dashboard de análisis, donde se presentan distintos datos, tanto de análisis sobre las tareas realizadas como los resultados de las mismas. Esta información se considera muy interesante, y en la FIGURA 3.8 se puede contemplar.

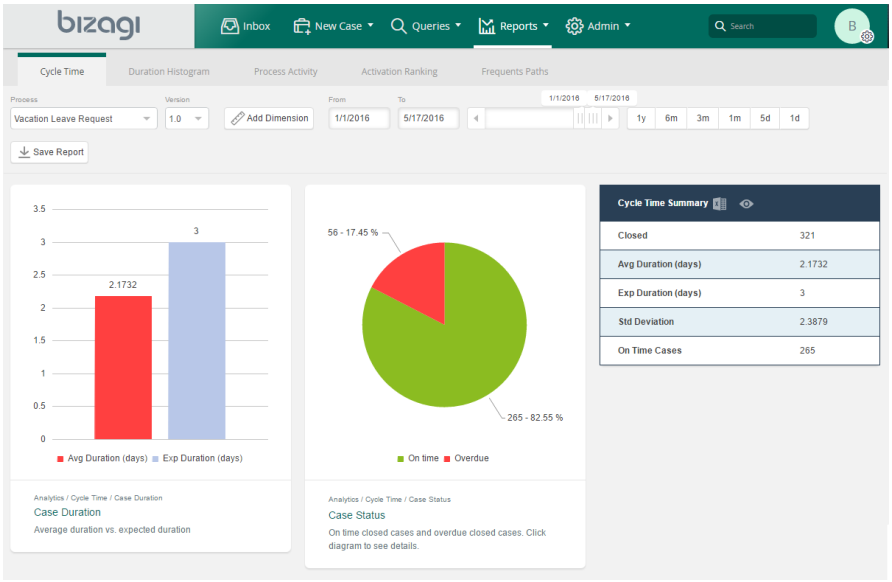


Figure 3.8: Bizagi Dashboard[Biz18]

API expuesta:

En bizagi se agrupan los distintos servicios SOAP según tres grupos: motor BPMN [OMG13] (Workflow), motor de acceso a datos (entity manager) y motor de consultas (query). El primero es el encargado de manejar los flujos de proceso. El segundo, en cambio, brinda acceso al modelo de datos de cada proceso. Y el último, provee acceso a los reportes de Bizagi. Cabe destacar que los servicios expuestos se comunican mediante XML y cada funcionalidad puede ser invocada enviando el archivo o un string que tenga el contenido del mismo. Para hacer más simple el desarrollo, a continuación se describe y sin perder generalidad, únicamente los métodos que se les envía el XML. Los que se invocan con un string se llaman igual pero se les concatena un "AsString" al final de la firma del método.

Los métodos se agrupan según se muestran en la tabla 3.2:

Grupo	**URL**
WorkflowEngineSOA (Motor ejecucion)	`http://[nombre_servidor] /[proyecto]/WebServices/ WorkflowEngineSOA.asmx?wsdl`
EntityManagerSOA (Motor de datos)	`http://[nombre_servidor] /[proyecto]/WebServices/ EntityManagerSOA.asmx?wsdl`
QuerySOA (Motor de consulta)	`http://[nombre_servidor] /[proyecto]/WebServices/ QuerySOA.asmx?wsdl`

Table 3.2: Definición de la API de Bizagi

Se irán describiendo cada unos de los servicios que tienen esta clasificación y a partir de estos obtener su modelo de dominio.

Métodos asociados al motor de ejecución:

1. Abortar Casos: abortCases. Cancela uno o más casos dado el número de creación. Aporta al modelo de dominio: case, user, domain, comment y process

2. Asignar actividades: assignActivity. Asigna una actividad al usuario. Aporta al modelo de dominio: task.

3. Crear Casos: createCases. Crea casos para un proceso. Aporta al modelo de dominio: entity, organization.

4. Establecer evento: setEvent. Lanza un evento intermedio que esté disponible. Aporta al modelo de dominio: event, activity.

5. Guardar Actividad: saveActivity. Este método es utilizado en las actividades manuales para simular el clic en el botón guardar. Este método utiliza conceptos ya definidos.

6. Obtener Actividades: getActivities. Se utiliza para obtener las actividades no completadas. Este método utiliza conceptos ya definidos.

7. Obtener Actividades Cerradas: getClosedActivies. Obtiene todas las actividades que han sido completadas. Este método utiliza conceptos ya definidos.

8. Obtener Eventos: getEvents. Retorna todos los eventos intermedios de un proceso. Este método utiliza conceptos ya definidos.

9. Otorgar acceso a Casos: grantCaseAccess. Puede asignar usuarios a casos. Este método utiliza conceptos ya definidos.

10. Realizar Actividad: performActivity. Se utiliza para que una actividad manual se complete. Aporta al modelo de dominio: varible.

11. Revocar acceso a casos: revokeCaseAccess. Puede restringir el acceso a un conjunto de casos específicos Este método utiliza conceptos ya definidos

Métodos asociados al motor de datos:

1. Guardar Entidad: saveEntity. Se utiliza para guardar una entidad. Este método utiliza conceptos ya definidos.

2. Obtener Datos del Caso Utilizando Esquema: getCaseDataUsingSchema. Obtiene datos del caso. Este método utiliza conceptos ya definidos.

3. Obtener datos del caso utilizando Esquema ligero: getCaseDataUsingSchemaLite. Obtiene datos del caso. Este método utiliza conceptos ya definidos.

4. Obtener Datos del caso Usando XPaths: getCaseDataUsingXPaths. Obtiene los datos utilizando como parámetro la ruta (xpath) al caso deseado. Este método utiliza conceptos ya definidos.

5. Obtener el Esquema de una Entidad: getEntitySchema. Trae el esquema de una entidad con el nombre de la misma como parámetro. Este método utiliza conceptos ya definidos.

6. Obtener Entidades: getEntities. Se obtiene los datos asociados a una entidad. Este método utiliza conceptos ya definidos.

7. Obtener entidades utilizando esquemas: getEntitiesUsingSchema. Obtiene la información de una entidad especificando su esquema. Este método utiliza conceptos ya definidos.

Métodos asociados a el motor de consulta de registros:

1. Consultar casos: queryCases. Realiza consultas sobre los casos. Aporta al modelo de dominio: query.

2. Consultar entidades: queryEntities. Consulta sobre las entidades. Este método utiliza conceptos ya definidos.

A en la FIGURA 3.9 se presenta el modelo de datos con los conceptos recabados.

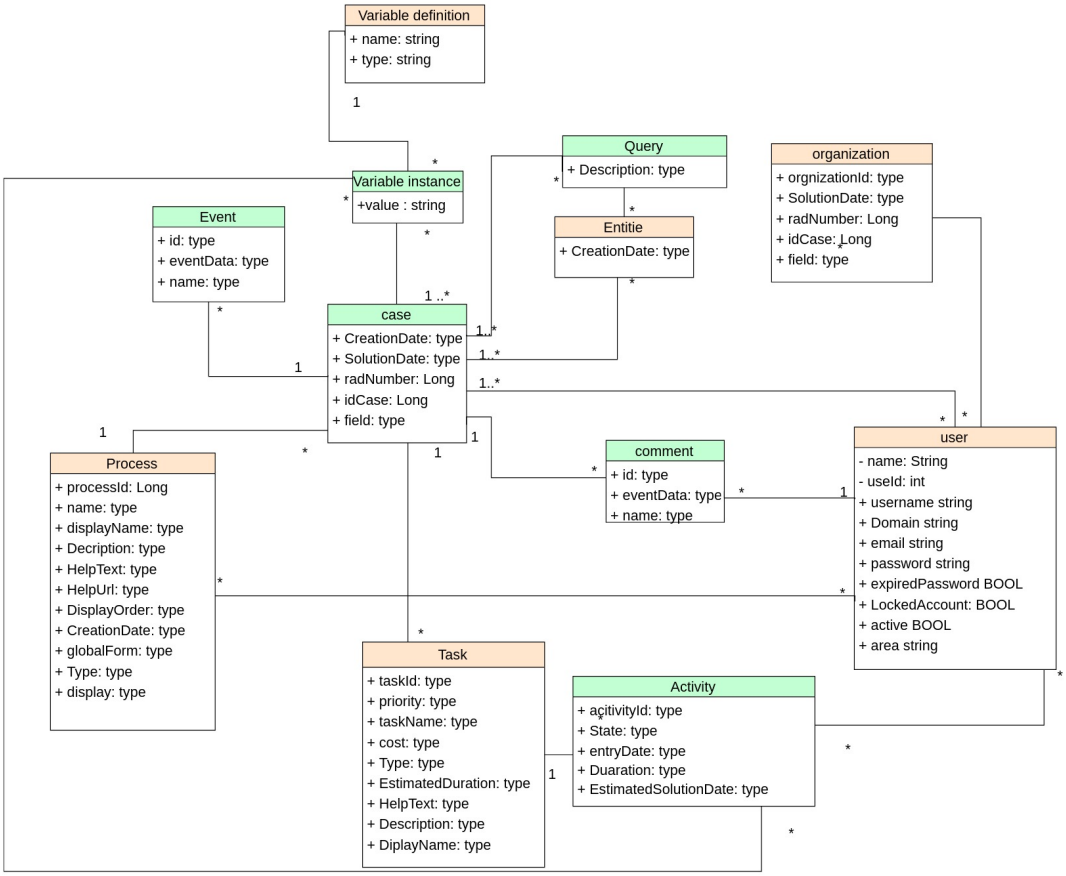


Figure 3.9: Modelo de Dominio de Bizagi

3.4 Bonita

Análisis del portal web:

En el caso de Bonita [Bon18], el portal web es una única aplicación en donde los usuarios pueden ver, realizar y administrar tareas y también es usado por el administrador "tenant" para desplegar, instalar y administrar procesos y para construir aplicaciones.

La cuenta del usuario técnico es la única que existe cuando aún no hay información de la organización.

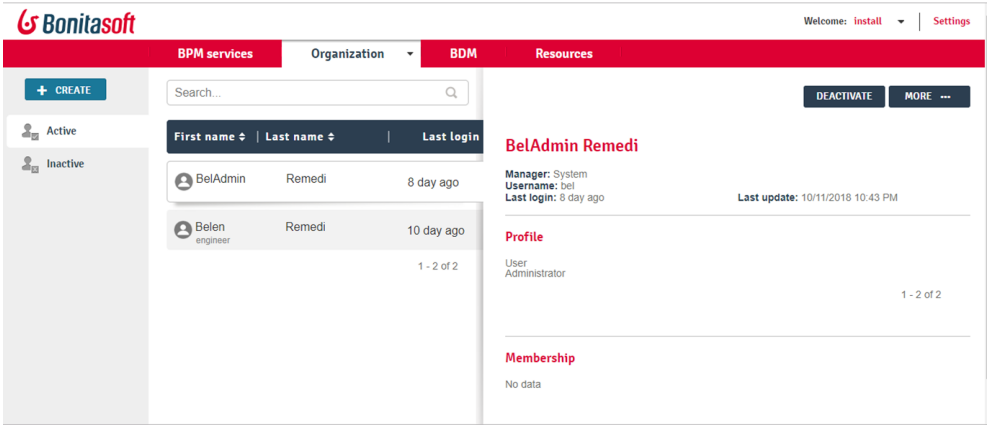


Figure 3.10: Portal Web Bonita- vista de Usuarios para el usuario de la plataforma [Bon18]

El usuario es responsable de realizar las tareas para las que es candidato y también de iniciar nuevos casos de los procesos a los que tiene acceso.

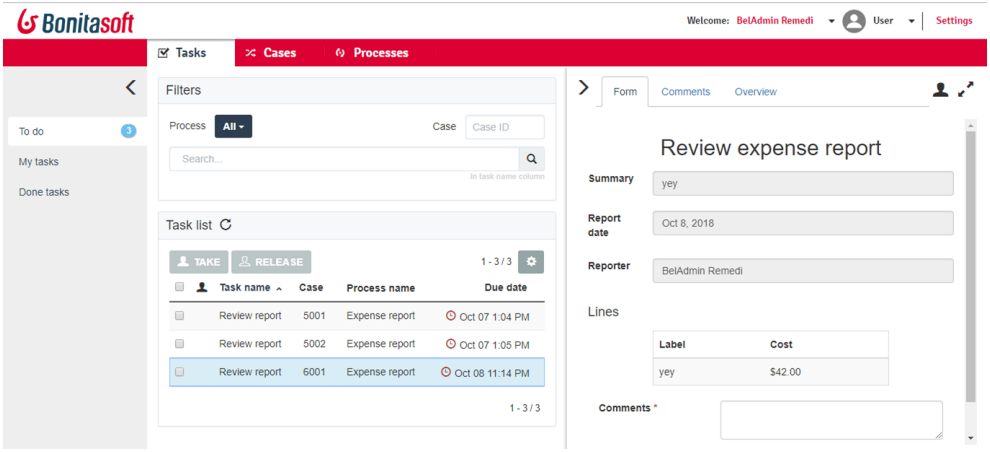


Figure 3.11: Portal Web Bonita [Bon18]

El Administrador es responsable de la administración del Portal a nivel de tenant, y en particular de la administración de los procesos, la organización, los informes, los perfiles personalizados y el Look & Feel.

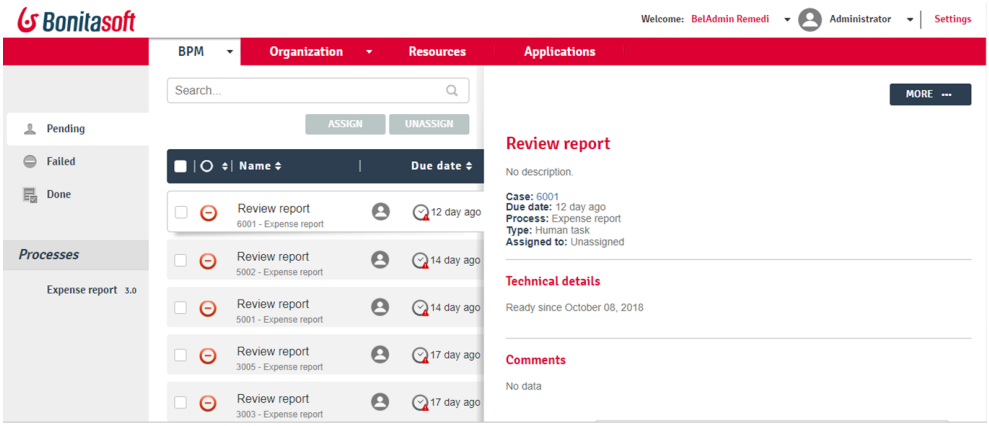


Figure 3.12: Portal Web Bonita- Vista para un administrador [Bon18]

API expuesta:

La API REST de Bonita [Bon18] provee acceso a todos los objetos tales como procesos, tareas, usuarios, etc. para ejecuar operaciones sobre ellos (crear, obtener, actualizar, eliminar).

Hay tres fases de operación para una aplicación que se integra con Bonita a través de la API REST: autenticación, ejecución y cierre de sesión.

Para poder realizar llamadas a la API REST se requiere la autenticación de un usuario registrado en la base de datos del motor. La respuesta a la invocación del servicio de autenticación genera una cookie JSESSIONID que debe ser transferida en cada subsiguiente invocación.

La seguridad contra ataques CSRF está habilitada de forma predeterminada. Esta seguridad se basa en la información de X-Bonita-API-Token que se encuentra en una cookie con el mismo nombre. Todas las subsiguientes llamadas a la API REST que usan los métodos DELETE, POST o PUT HTTP deben contener un encabezado HTTP con esta información.

La API REST de Bonita se compone a su vez de las siguientes sub-APIs:

- bdm API: Permite manejar los datos de negocio. Bonita admite el modelado y la persistencia de objetos con los que las aplicaciones y los procesos interactúan. Dichos datos pueden almacenarse automáticamente en una base de datos dedicada a estos o pueden residir en bases de datos existentes del sistema.
- bpm API: Es la sub API más extensa del grupo, permite administrar elementos del proceso.
- customuserinfo API: Permite administrar definiciones de usuario personalizadas.

- identity API: Se utiliza para administrar información relacionada a la organización tales como usuarios, roles y grupos.
- platform API: maneja conceptos específicos de la plataforma community de Bonita.
- portal API: maneja conceptos relacionados con interfaces de usuario, perfiles que definen qué páginas un usuario puede acceder, menúes que puede ver, entre otros.
- system API: orientado a configuración. Maneja conceptos como sesión, i18ntranslation, tenant.
- application API: Permite crear un entorno funcional aplicativo para que los usuarios interactúen con los procesos y datos empresariales desde un solo lugar.

Otras sub APIs que se encuentran son tenant API y form API.

Acción	**Método**	**URL**
Crear un recurso	POST	`http://.../API/{API_name}/{resource_name}/`
Leer un recurso	GET	`http://.../API/{API_name}/{resource_name}/{id}`
Actualizar un recurso	PUT	`http://.../API/{API_name}/{resource_name}/{id}`
Borrar un recurso	DELETE	`http://.../API/{API_name}/{resource_name}/`
Buscar un recurso	GET	`http://.../API/{API_name}/{resource_name}?p={page}&c={count}&o={order}&s={query}&f={filter_name}={filter_value}&f=...`

Table 3.3: Definición de la API REST de Bonita

Para un método GET que devuelve más de una instancia de un recurso, se pueden especificar los siguientes parámetros:

- p: índice de la página a mostrar.
- c: máximo número de elementos a devolver.
- o: orden de los elementos(asc, desc).
- f: lista de filtros, especificados como attributeName=attributeValue. Para filtrar en más d eun atributo se debe especificar un parámetro f para cada uno.
- s: búsqueda sobre nombre o sobre índice.

A continuación se muestra el modelo de datos con los conceptos que maneja Bonita.

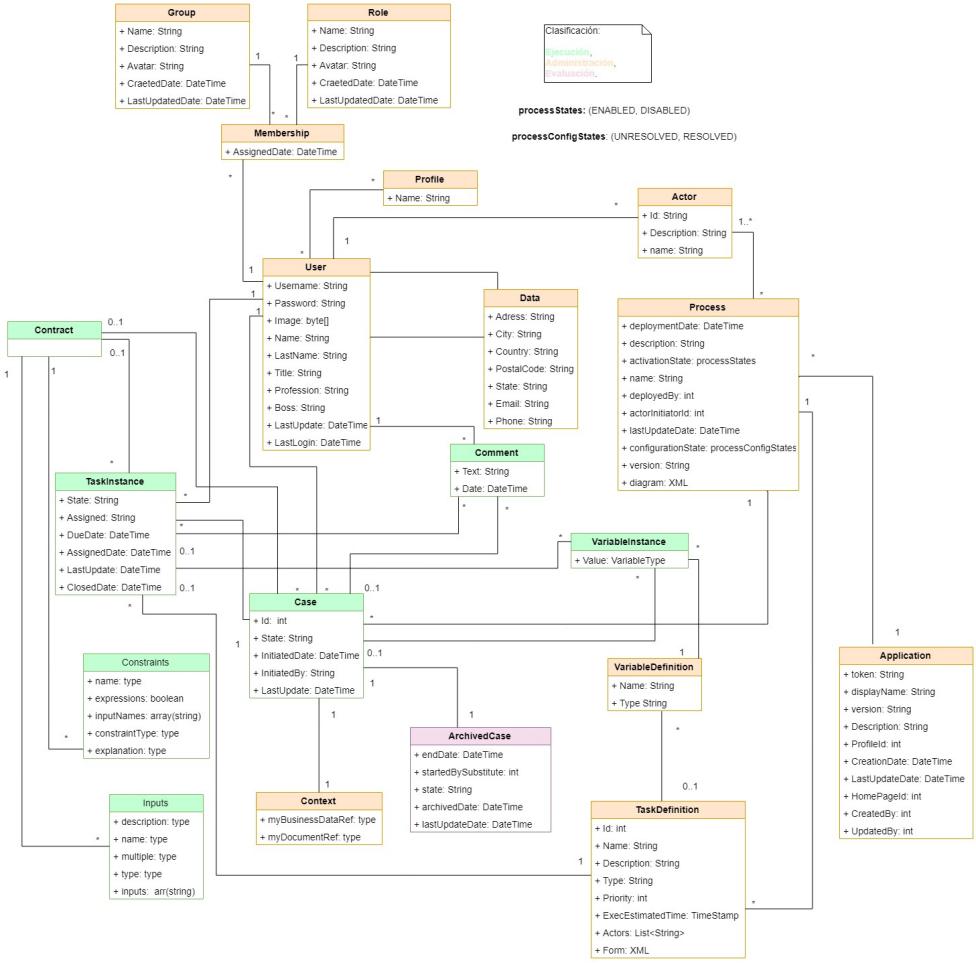


Figure 3.13: Modelo de datos de Bonita

3.5 Camunda

En esta sección abordaremos Camunda [Cam18a], describiendo su portal, el modelo de información que este utiliza así como de su API que permite el acceso programático al motor.

Análisis del Portal Web:

El portal web proporcionado por este BPMS [Wes07] se presenta de una forma simple y compacta, incluyendo al igual que otros BPMS un panel de gestión de tareas y procesos, otra de monitorización y estadísticas, y una parte de administración para gestiones de permisos, entre otras. En la FIGURA 3.14 se puede observar la pantalla que se visualiza al ingresar al portal.

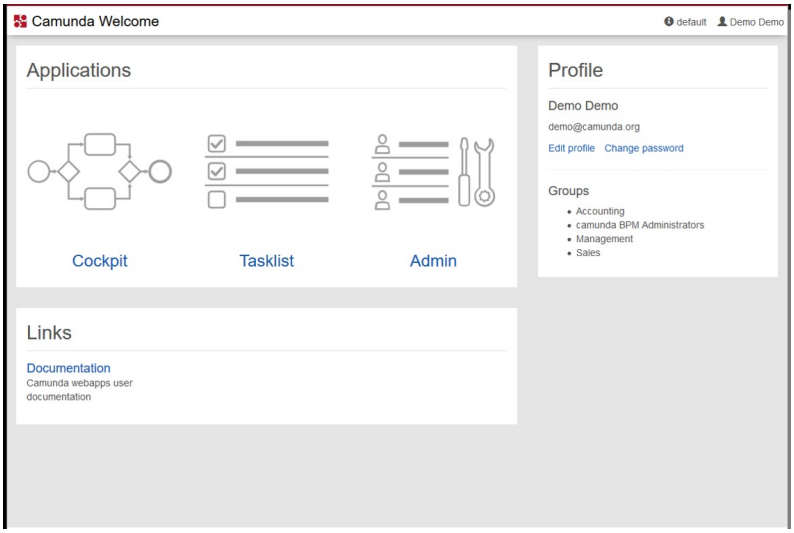


Figure 3.14: Portal Web Camunda [Cam18a]

Dentro de las funcionalidades que encontramos en la web de administración tenemos la gestión de los usuarios, su organización en distintos grupos y las asignaciones de permisos. La gestión de estos últimos se pueden hacer de forma independiente a la definición de los usuarios. Existen distintos niveles de permisos, ya sea para la invocación de servicios del motor central como para servicios rest o para la aplicación web. Toda esta gestión se permite hacerla de forma centralizada en el sitio admin, presentado en la FIGURA 3.15.

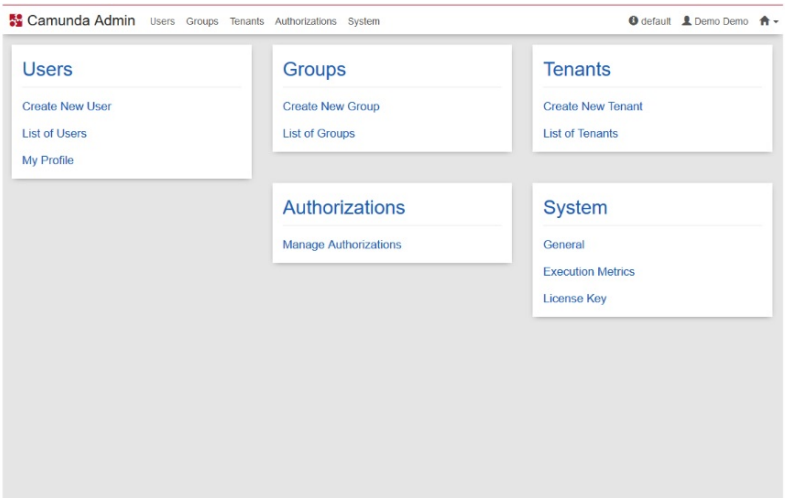


Figure 3.15: Aplicación Admin Camunda [Cam18a]

En la FIGURA 2.20 se presentó la aplicación web para la gestión de las tareas y procesos. En ella es posible iniciar una instancia de proceso a partir de una definición. Brinda la posibilidad de crear filtros, que se pueden utilizar, por ejemplo para listar todas las tareas asignadas al usuario actual. Una vez definido este filtro puede ser compartido para que sea utilizado por otros usuarios. Permite reasignar tareas a otros usuarios, tomar tareas, liberarlas, etc. En el panel de la izquierda, se pueden visualizar las tareas que el usuario tiene pendiente de realizar. Se podrá realizar búsquedas sobre las mismas, así como actualizar su información o realizar comentarios en ellas.

API expuesta:

Provee de una API de servicios java y otra API REST. Esta última fue la utilizada para realizar el análisis y posteriormente en la implementación. Esta API es muy completa, pudiéndose identificar servicios asociados a distintas áreas. Por un lado la parte de definiciones, donde existen sub APIs para definiciones de procesos, tareas, usuarios, variables, despliegues, definiciones de casos, entre otras. Este amplio conjunto de APIs nos permite realizar las operaciones de definición y/o actualización, que se hacen uso por lo general al comienzo. A su vez otra sub API de importancia es la de autenticación, donde se pueden gestionar todas las tareas de permisos a los usuarios existentes en el sistema. Para poder realizar operaciones sobre este, es necesario estar autenticado, por lo cual si no se tienen los permisos pertinentes no será posible realizarlas. Las implementaciones posibles de esta funcionalidad son Basic Authentication o utilizando Cookies.

Por otro lado brinda una API para el histórico. A través de ella se puede tener una trazabilidad de las definiciones así como de las instancias en el correr del tiempo. Esto combinado con la aplicación web que se provee para monitorizar y operar ayudan a optimizar los procesos.

En el CUADRO 3.4 se presentan las URL base para acceder a las principales sub APIs brindadas por Camunda. En [Cam18b] se encuentran todos los servicios expuestos con sus respectivos parámetros y métodos.

«URL BASE»
Authorization
http://SERVER:PORT/camunda/api/engine/engine/default/authorization
Deployment
http://SERVER:PORT/camunda/api/engine/engine/default/deployment
Execution
http://SERVER:PORT/camunda/api/engine/engine/default/execution
Task
http://SERVER:PORT/camunda/api/engine/engine/default/task
History
http://SERVER:PORT/camunda/api/engine/engine/default/history
Identity
http://SERVER:PORT/camunda/api/engine/engine/default/identity
Job
http://SERVER:PORT/camunda/api/engine/engine/default/job
Process Definition
http://SERVER:PORT/camunda/api/engine/engine/default/process-definition
Process Instance
http://SERVER:PORT/camunda/api/engine/engine/default/process-instance
User
http://SERVER:PORT/camunda/api/engine/engine/default/user
Variable Instance
http://SERVER:PORT/camunda/api/engine/engine/default/variable-instance
Group
http://SERVER:PORT/camunda/api/engine/engine/default/group

Table 3.4: URL base de servicios expuestos por Motor de ejecución Camunda

En la FIGURA 3.16 se presentará el modelo realizado luego de haber hecho un análisis y haber identificado los conceptos manejados.

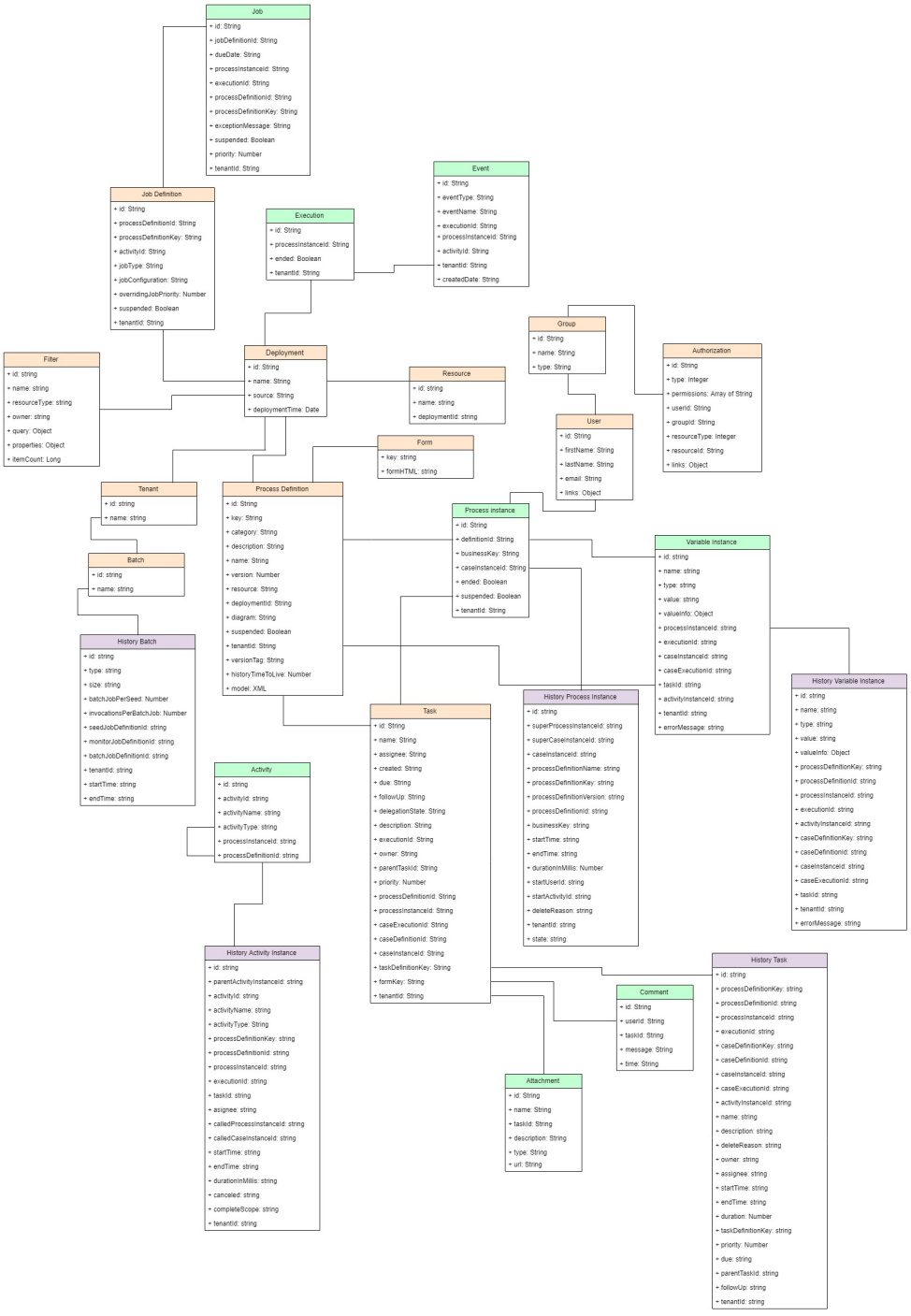


Figure 3.16: Modelo de conceptos Camunda

3.6 jBPM

De igual manera que en los otros BPMS [Wes07], en esta sección profundizaremos sobre jBPM [JBP18a]. Se describirá el portal web proporcionado, el modelo de información que utiliza y la API que permite el acceso programático al motor.

Este motor se puede utilizar de forma embebida, es decir, dentro de alguna aplicación de nuestra organización en la cuál estemos trabajando, o, en modo de servicio, es decir, corriendo propiamente por si solo.

Análisis del Portal Web:

El portal web que nos proporciona JBPM combina todas las herramientas y/o aplicaciones necesarias para manejar los procesos durante todo su ciclo de vida. Nos permite gestionar las autorizaciones, los despliegues, los procesos, las reglas, las tareas, entre otras. También podemos ejecutar procesos y tareas. Nos brinda la posibilidad de crear dashboards relacionados a nuestros procesos, pudiendo así obtener de forma mas visual, información como los tiempos, cantidades estadísticas, etc. Brinda una serie de funcionalidades muy completas, teniendo una buena usabilidad y utilidad para los usuarios finales.

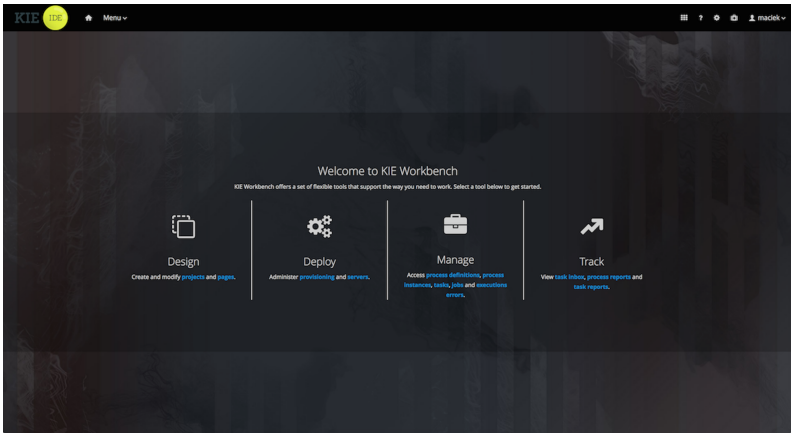


Figure 3.17: Portal Web JBPM [JBP18a]

En la FIGURA 3.17 se puede observar la web de trabajo principal, donde se muestran las cuatro grandes funcionalidades contenidas en ella. Dependiendo de los permisos de cada usuario se brindarán un conjunto de funcionalidades. El usuario que tiene todos los permisos es el administrador, el cual puede realizar las altas, bajas y asignaciones de permisos a nuevos usuarios.

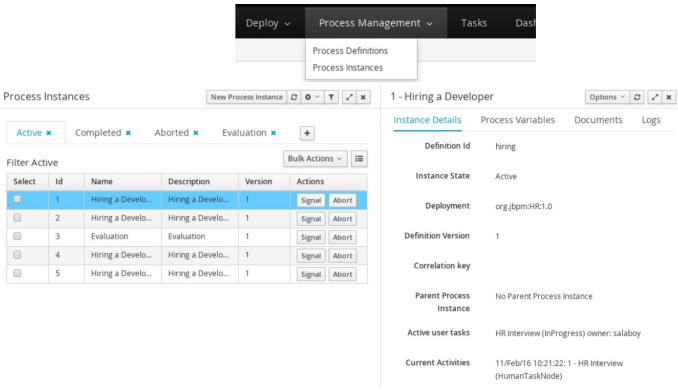


Figure 3.18: Lista de procesos JBPM [JBP18a]

La FIGURA 3.18 muestra el panel donde se listan todos las instancias de procesos, pudiéndose visualizar la información de cada una de estas.

Se presenta en la FIGURA 3.19 el panel donde se muestra la tabla con todas las instancias de las tareas existentes. Para ser mas amigable al usuario permite seleccionar que datos son relevantes y seria interesante mostrar, siendo dicha tabla personalizable de acuerdo a esta selección.

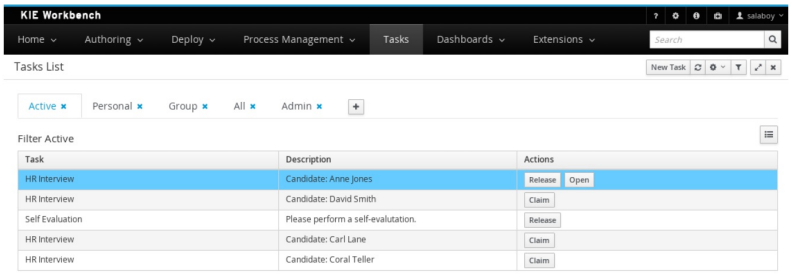


Figure 3.19: Lista de tareas JBPM [JBP18a]

API expuesta:

JBPM [JBP18a] permite el acceso programático al motor de ejecución de procesos de varias formas distintas. Estas son: JMS, SOAP, interfaces EJB y REST. A continuación hablaremos sobre esta última como forma de interactuar con el mismo. Se cuenta con dos APIs REST, una directamente expuesta por el servidor de ejecución "Kie Server" en el cual se ejecutan los procesos y reglas, y otra, expuesta por la consola de procesos de negocio[1], la cual brinda

[1] Conocido como Business Central

servicios para operar sobre cada componente de la web. En esta tesis nos centraremos en la primera.

Considerando la API antes mencionada, pudimos identificar una serie de sub-APIs, de acuerdo a las operaciones expuestas:

- Containers
- Process
- Process definition
- Task Instances
- Query
- Query advance
- Job
- Environment

Cada una de estas sub APIs se pueden acceder utilizando las URL base mostradas en el CUADRO 3.5. En [JBP18b] se encuentran todos los servicios REST para cada una de estas, visualizando los parámetros necesarios en cada caso.

«URL BASE Kie Server»
Containers
http://SERVER:PORT/kie-server/services/rest/server/containers
Process
http://SERVER:PORT/kie-server/services/rest/server/containers/CONTAINER_ID/processes
Process Definition
http://SERVER:PORT/kie-server/services/rest/server/containers/CONTAINER_ID/processes/definitions
Task Instances
http://SERVER:PORT/kie-server/services/rest/server/containers/CONTAINER_ID/tasks/ TASK_ID
Query
http://SERVER:PORT/kie-server/services/rest/server/queries/
Query Advance
http://SERVER:PORT/kie-server/services/rest/server/queries/definitions
Job
http://SERVER:PORT/kie-server/services/rest/server/jobs
Environment
http://SERVER:PORT/business-central/rest/controller

Table 3.5: URL base de servicios expuestos por Motor de ejecución JBPM

La sub API **Containers** nos permite gestionar todos los contenedores del servidor. Estos son los que contienen las definiciones de los procesos. La correspondiente a **Process** provee servicios relacionados a la gestión de las instancias de los procesos. Incluye la creación de una instancia permitiéndole el pasaje de variables de inicialización, la eliminación de una instancia, obtención de información como la actualización de esta, etc. Siguiendo con esto, la sub

API **Process Definition**, la cual permite obtener toda la información de una definición concreta, ya sea sus variables, sus tareas de usuario, sus subprocesos, etc. La de **Task Instances** permite cambiar de estado las tareas, ya sea reasignándolas, solicitándolas, completándolas, iniciándolas, suspendiéndolas, etc. A su vez brinda la posibilidad de realizar comentarios, eliminarlos, también actualizar información asociada a la misma, como ser su nombre, fecha de finalización, descripción, prioridad, etc. Puede existir la necesidad de querer obtener información mas concreta, por ejemplo condicionado a un determinado container al momento de obtener instancias de procesos como de tareas. Para esto se brinda una API denominada **Query** en la que podremos consultar de forma mas general tanto como específica. Entre ellas permite obtener todas las intancias de procesos para un contenedor en particular, todas las instancias de procesos en el servidor, definiciones de procesos para un contenedor, así como en lo general, las tareas que un usuario puede asignarse, obtener eventos o items de trabajo para una tarea en particular, etc. **Query advance** es la API que permite definir nuevas consultas que un usuario quiera realizar, habilitando a un usuario que tenga un conocimiento mas profundo del modelo de datos, satisfacer su requerimiento. Para realizar la gestión de lo que refiere mas al servidor propiamente, brinda una API **Environment** que se encarga del manejo de estos, y la administración de los contenedores en el mismo. Maneja templates para utilizarlos, ya que además del manejo del motor involucra otras reglas avanzadas en las que no se entrará en detalle.

En la FIGURA 3.20 se muestran los conceptos que se manejan en jBPM [JBP18a], los cuales fueron identificados al momento de realizar el análisis. Estos son utilizados en todas sus APIs de acceso programático.

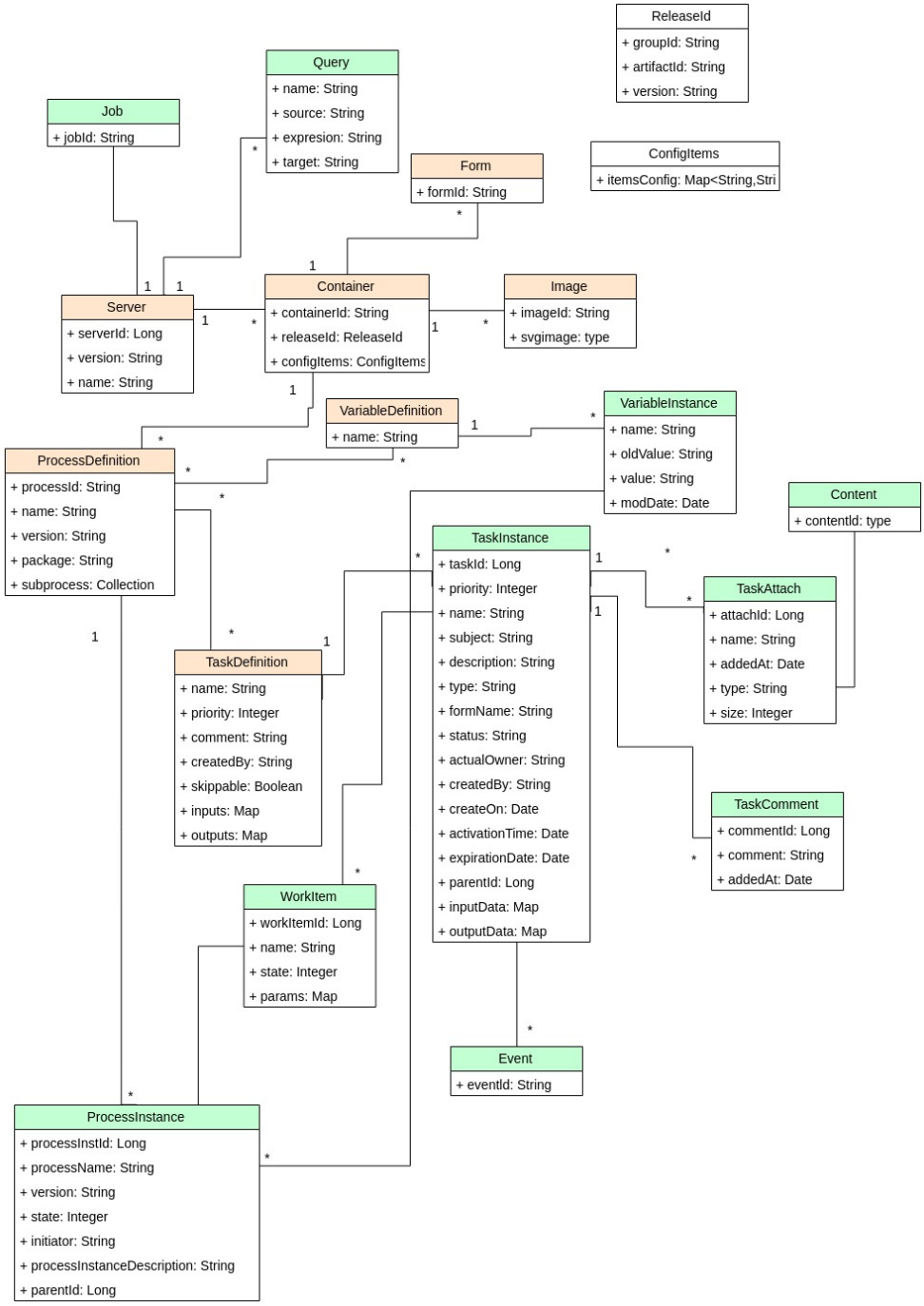


Figure 3.20: Modelo de datos JBPM

En el diagrama se pueden ver los conceptos de Process Definition, Task Definition y Variable Definition, relacionados de forma que un proceso tenga asociado una o mas tareas y una o mas variables, independiente de si estas últimas son usadas en las tareas. A su vez cada uno de dichos conceptos tienen sus correspondientes instancias, donde al instanciarse un proceso se instanciarán sus asociaciones que tengan. Otros conceptos que se pueden visualizar son los de Container, Server, Form y Query. Mediante la utilización de Contenedores, los cuales son quienes contienen las definiciones de sus procesos de negocio, se permite por un lado tener varias aplicaciones que desplieguen sus definiciones en un mismo servidor de forma independiente, y a su vez puedan, en caso de querer, reutilizar y desplegar las mismas definiciones en distintos contenedores. El concepto este tiene similitud a lo que se conoce como despliegue, o mas conocido como deploy en inglés. Por otro lado el concepto de Server permite gestionar varios servidores a la vez con sus configuraciones independientes. Los conceptos de Form y Query se utilizan para predefinir lo que podríamos llamarles templates, que serán utilizados donde correspondan cada uno. Es una característica que se podría ver como un plus frente a lo que sería tener que definir cada vez que se quiera lo mismo.

3.7 Unificación de conceptos

Luego de haber analizado la selección de BPMS [Wes07], se realizó una tabla en la que se unificaron los conceptos identificados en todos ellos,la cual se presenta en CUADRO 3.7, teniendo en cuenta los conceptos que ya fueron unificados en el trabajo [DCA16].

En la columna “Unified model“ se muestran todos los conceptos unificados, en cada una de las otras columnas, se hace referencia al nombre de un motor y se muestran cada uno de los conceptos asociados al mismo.

Como se puede observar en la columna “Unified model“, se dejan los siguientes conceptos: Context, Membership, Profile, Data, Organization y Entity.

En el caso de Entity y Organization, el motor Bizagi provee acceso a los datos de la organización, es decir a las entidades del negocio propiamente. Por simplicidad y homogeneidad con los otros motores, se decidió que los datos de la organización se manejen directamente desde el modelo y no pasando por los métodos expuestos por el motor. Por este motivo no se incluyen dichos conceptos en el modelo genérico.

En el caso del concepto Context perteneciente a Bonita, es similar a los casos anteriores. El contexto es una lista de referencias a los datos de negocio y documentos manipulados por la instancia de un proceso durante su ejecución.Es decir que la forma de hacer públicas estas referencias, es a través de este concepto.

Membership, Profile y Data son todos conceptos que maneja Bonita que aportan mayor información asociada a un usuario, es por esto que si se deseeara incluir esta información se podría hacerlo en el mismo concepto de Usuario. Particularmente, en el caso de Member-

ship,se describe que existe cuando un usuario pertenece a un grupo y a un rol.

Unified model	JBPM	Activiti	Flowable	Bonita	Camunda	Bizagi
Process	Process definition	Process definition	Process definition	Process	Process Definition	Process
Task definition	UserTask Definition	Task Definition	-	Task Definition	Task	Task
Deployment	Container	Deployment	Deployment	Application	Deployment	-
Resource	Image	Resource	Resource	-	Resource	-
Model	-	Model	Model	-	-	-
Variable definition	Variable definition	Variable Definition	Variable	Variable Definition/-Contract	-	variable definition
-	-	-	-	Context	-	-
User	-	User	User	Actor/User	User	User
Group	-	Group	Group	Group	Group	Domain
Role	-	-	-	Role	Authorization	Role
-	-	-	-	Membership	-	-
-	-	-	-	Profile	-	-
-	-	-	-	Data	-	-
Form	Form	Form data	Form data	-	Form	-
Query	Query	-	-	-	filter	Query
Tenant	-	-	-	-	Tenant	-
Variable instance	Variable instance	Variable Instance	Variable	Variable Instance.	Variable instance	Variable instance
Execution	Node instance/-workItem	Execution	Execution	-	Execution	-
Event	Event	Event	Event	-	Event	Event
Comment	Task Comment	Comment	Comment	Comment	Comment	Comment
Attachment	Task Attachment	Attachment	Attachment	-	Attachment	-
Case	Process instance	Process instance	Process Instance	Case	Process Instance	Case
Task instance	Task instance	TaskInstance	Task	Task Instance	Task	Activity
-	-	-	-	-	-	Organization
-	-	-	-	-	-	Entity
Job	Job	Job	Job	-	Job	-
Historic activity instance	-	-	-	-	Historic activity instance	-
History Variable Instance	-	History Variable Instance	-	-	History Variable Instance	-
Historic process instance	-	Historic process instance	-	-	Historic process instance	-
History Task Instance	-	History Task Instance	-	-	History Task Instance	-

Table 3.7: Tabla de Unificación de conceptos

3.8 Comparación de funcionalidades

El portal ofrecerá la opción de que un usuario se autentique y en base a los permisos que tenga definidos, realizar las siguientes funcionalidades:

En cuanto a los procesos podrá, visualizar los activos así como los no activos, pudiendo filtrarlos por nombre, categoría y estado, activar o desactivar uno, ver sus datos, así como asociar y desasociar una categoría al mismo. También se brinda la posibilidad de crear una instancia de proceso, eliminarla, listar todas las instancias de estos, obtener información sobre una, además como agregar comentarios y obtenerlos.

Respecto a las tareas se permite visualizarlas y obtener la información de una de estas, tomar y reasignar una, modificar las variables asociadas y completarla.

Brinda el manejo de los grupos, roles y usuarios, permitiendo agregar, modificar, listar y eliminar. A su vez se puede asignar y quitar roles a los usuarios, así como asignar y quitar usuarios a los grupos.

En el CUADRO 3.8 se mapea cada operación definida en la API genérica del trabajo [DCA16], con las identificadas en las APIs de cada motor. Cabe destacar que para satisfacer una operación de la API genérica podría ser necesario mas de una operación hacia el motor. Por ejemplo, al momento de obtener una tarea se desea tener la información del proceso asociado, en caso que el método del motor no la devuelva, será necesario realizar las operaciones pertinentes para obtenerla.

Operación	Activiti	Bonita	Bizagi	Camunda	JBPM	Flowable
Login(string user, string password): void	-	/bonita/ loginservice	-	-	-	
ChangePassword(string oldPassword, string newPassword): bool	PUT identity/users/{userId}	PUT API/identity/user/{userId}	-	-	-	PUT identity/users/{userId}
GetProcessDefinitions(string name, string category, bool? active):IEnumerable <Process>	GET repository/ processdefinitions	GET /API/bpm/process	-	/processdefinition	/processes/definitions	GET repository/processdefinitions/
GetProcessDefinition(int processId): Process	GET repository/processdefinitions/{proDefId}	GET /API/bpm/process/:procId	-	/processdefinition/{id}	/processes/definitions/{procId}	GET repository/processdefinitions/{proDefId}
SuspendProcess(int processId): void	PUT repository/processdefinitions/{procDefId}	PUT /API/bpm/process/:processId	-	/processdefinition/{id}/suspended	-	PUT repository/processdefinitions/{procDefId}
ActivateProcess(int processId): void	PUT repository/processdefinitions/{proDefId}	PUT /API/bpm/process/:processId	-	/processdefinition/{id}/suspended	-	PUT repository/processdefinitions/{procDefId}

Operación	Activiti	Bonita	Bizagi	Camunda	JBPM	Flowable
AssociateCategory(int processId, string category)	PUT repository/-process-definitions/-{procDefId}	-	-	-	-	PUT repository/-process-definitions/-{procDefId}
RemoveCategory(int processId, string category)	PUT repository/-process-definitions/-{procDefId}	-	-	-	-	PUT repository/-process-definitions/-{procDefId}
GetStartVariables(int processId): IEnumerable-<variable-Instance>	GET form/form-data?process-DefId=procId	GET /API/bpm/-process/-:procId/-contract	-	GET process-definition/-{processId}/-form-variables	GET /{cont_id}/-processes/-definitions/-{procId}/variables	GET form/form-data?process-DefId=procId
startCase(int processId, IEnumerable-<Variable-Instance> casevariables): void	POST runtime/process-instances	POST /API/bpm/-process/:processId/instantiation, POST /API/bpm/-case	createCases	/process-definition/{id}-/start	/cont_id/-processes/-process_id/-instances	POST runtime/process-instances
GetAllCases(string processName, string CreatedBy, string userInvolved, string state): IEnumerable<Case>	GET runtime/process-instances	GET /API/bpm/-case GET /API/bpm/-archivedCase	-	/process-instance	/queries/processes/-instances	GET runtime/process-instances
GetCase(int caseId): Case	GET runtime/process-instances/-{procInsId}	GET /API/bpm/-case/:id	-	/process-instance/{id}	/{contid}/-processes/-instances/-{procinstid}	GET runtime/process-instances/-{procInsId}
DeleteCase(int caseId): void	DELETE runtime/process-instances/-{proInsId}	DELETE /API/bpm/-case/:caseId	abortCases	/process-instance/{id}	/containers/-{contId}/processes/instances/{procInsId}	DELETE runtime/-process-instances/-{procInsId}
GetTasks-(string name, bool? assignee, bool? candidateUser): IEnumerable-<TaskInstance>	GET runtime/-tasks	GET /API/bpm/-humanTask	getActivities, getClosedActivities	/task	/queries/tasks/instances	GET runtime/-tasks
GetTask(string taskId): TaskInstance	GET runtime/tasks/{taskId}	GET /API/bpm/-task/:taskId, GET /API/bpm/-userTask/:userTaskId	getActivities	/task/{id}	/containers/-{contId}/tasks/{taskId}/	GET runtime/tasks/{taskId}
TakeTask(string taskId): bool	POST runtime/tasks/{taskId}	PUT /API/bpm/-userTask/:userTaskId	getActivities	/task/{id}/-claim	/{contId}/tasks/{taskId}-/states/claimed	POST runtime/tasks/{taskId}

Operación	Activiti	Bonita	Bizagi	Camunda	JBPM	Flowable
ReasignTask-(string taskId,string userId): bool	POST runtime/tas-ks/{taskId}	PUT /API/bpm/-userTask/:us-erTaskId	assignActivity	/task/{id}/-delegate	/{contId}/ta-sks/{taskId}/-states/delega-ted	POST runtime/tas-ks/{taskId}
GetTaskVaria-bles(string taskId): IEnumerable-<VariableIns-tance>	GET runtime/tas-ks/{taskId}/-variables?sco-pe={scope}	GET /API/bpm-/userTask/-:taskId/-contract	getActivities	/task/{id}/-variables	/{processId}/-tasks/users/-{taskName}/-inputs	GET runtime/tas-ks/{taskId}-/variables?sc-ope={scope}
ModifyTask-Variables(strin-g taskId, IEnumerable-<VariableIns-tance> taskVa-riables): bool	PUT runtime/tas-ks/{taskId}/-variables/{va-rName}	-	-	/task/{id}/-variables/-{varName}	-	PUT runtime/tas-ks/{taskId}/-variables/{va-rName}
SaveTask(str-ing taskId, IEnumerable-<Variable-Instance> taskvariables): bool	POST runtime/tas-ks/{taskId}	POST /API/bpm/-userTask/-:taskId/-execution	saveActivity	/task/{id}/-resolve	/exited	POST runtime/tas-ks/{taskId}
GetTaskCo-mments(int taskId): IEnumer-able<Comment>	GET runtime/-tasks/{task-Id}/-comments	-	-	/task/{id}/-comment	[GET] /containers/-{contId}/tasks-/{taskId}/co-mments	GET runtime/tas-ks/{taskId}/-comments/
AddComment-ToTask(int taskId, string userId, string comment): void	POST runtime/tas-ks/{taskId}/-comments	-	-	/task/{id}-/comment/-create	[POST] /containers/-{contId}/tas-ks/{taskId}/-comments	POST runtime/tas-ks/{taskId}/-comments/
GetUsers(st-ring like): IEnumer-able<User>	GET identi-ty/users	GET /API/identi-ty/user	-	GET /users	-	GET identi-ty/users
GetGroups(st-ring like): IEnumer-able<Group>	GET identity/-groups	GET /API/identi-ty/group	-	/group	-	GET identity/-groups
GetGroup(str-ing groupId): Group	GET identity/gro-ups/{groupId}	GET /API/identi-ty/group/:gr-oupId	-	/group/{id}	-	GET identity/gro-ups/{groupId}
DeleteGrou-p(string groupId): void	DELETE identity/gro-ups/{groupId}	DELETE /API/identi-ty/group/-:groupId	-	/group/{id}	-	DELETE identity/gro-ups/{groupId}
ModifyGro-up(Group group): void	PUT identity/gro-ups/{groupId}	PUT /API/identi-ty/group/:gr-oupId	-	/group/{id}	-	PUT identity/gro-ups/{groupId}
AddGroup(-Group group): void	POST identi-ty/groups	POST /API/identi-ty/group	-	/group/create	-	POST identi-ty/groups
AddUser (string userId, string groupId): void	POST identity/gro-ups/{group-Id}/members	POST /API/bpm/-actorMember	-	/user/create	-	POST identity/gro-ups/{group-Id}/members

Operación	Activiti	Bonita	Bizagi	Camunda	JBPM	Flowable
RemoveUser(string userId, string groupId): void	DELETE identity/users/{userId}	DELETE /API/bpm/actorMember/:id	-	/user/{id}	-	DELETE identity/users/{userId}
GetRoles(string like):IEnumerable<Role>	-	GET /API/identity/role	-	/authorizati-on	-	-
GetRole(string roleId):Role	-	GET /API/identity/role/:roleId	-	/authorization/{id}	-	-
DeleteRole(string roleId):void	-	DELETE /API/identity/role/:roleId	-	/authorization/{id}	-	-
ModifyRole(Role role):void	-	PUT /API/identity/role/:roleId	-	/authorization/{id}	-	-
AddRole(Role role):void	-	POST /API/identity/role	-	/authorization/create	-	-
AddUserRole(string userId, string roleId):void	-	POST /API/bpm/actorMember	-	/authorization/create	-	-
RemoveUserRole(string userId, string roleId):void	-	DELETE /API/bpm/actorMember/:id	-	/authorization/{id}	-	-

Table 3.8: Mapeo de servicios

4 Portal Genérico

En este capítulo se define una solución genérica que usa como base las API de los distintos BPMS [Wes07] estudiada en la sección anterior. Se presenta una arquitectura abstracta en donde se mencionan las distintas capas de la misma. También se presenta el modelo genérico para soportar las funcionalidades de los BPMS [Wes07] considerados. Al modelo genérico presentado en el trabajo anterior se agregan algunos conceptos asociados al estudio en en el capítulo previo.

4.1 Arquitectura

En la FIGURA 4.1 se presenta la arquitectura definida para el portal genérico.

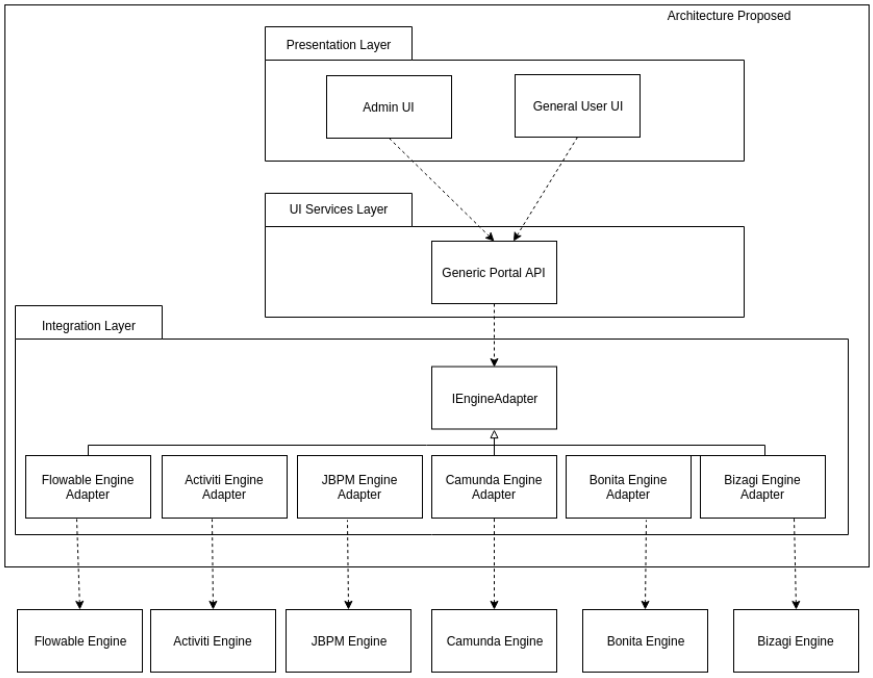


Figure 4.1: Arquitectura Portal

Esta arquitectura respeta la propuesta del trabajo [DCA16]. La misma consiste en tres capas (como se puede ver en la FIGURA 4.1): presentación, servicios e integración.

En la capa de presentación es en donde se encuentra todo lo referente al portal web genérico. Separando las distintas funcionalidades por un lado destinadas a un usuario común y por otro a los administradores.

En la capa de servicios se expresa en base a un modelo de información unificado presentado en la SECCIÓN 3.7 y 4.3.1, y una API de servicios en la SECCIÓN 3.8.

Finalmente la capa de integración es la que se comunica propiamente con los motores y permite realizar las operaciones invocadas desde la presentación. En otras palabras es la que hace la traducción entre una operación expuesta por la API generica, y una o varias expuestas por la API del motor de ejecución respectivo.

4.2 Capa de Presentación

A partir del análisis de los portales y servicios de las diferentes herramientas y cubriendo las funcionalidades para el tipo de interfaz que se desea manejar, el modelo que se toma para esta, es el mismo que el del trabajo [DCA16]

El modelo fue realizado en WebRatio, un IDE que soporta el modelado de aplicaciones web y que utiliza como lenguaje una extensión de IFML basada en WebML. En él se definen tres Site views (Vistas del sitio) diferentes: Administrator, Common y Public; una para cada tipo de usuario administrador, normal y anónimo, respectivamente.

En la FIGURA 4.2 se muestra una de las áreas que se encuentra dentro de Administrator Site View.

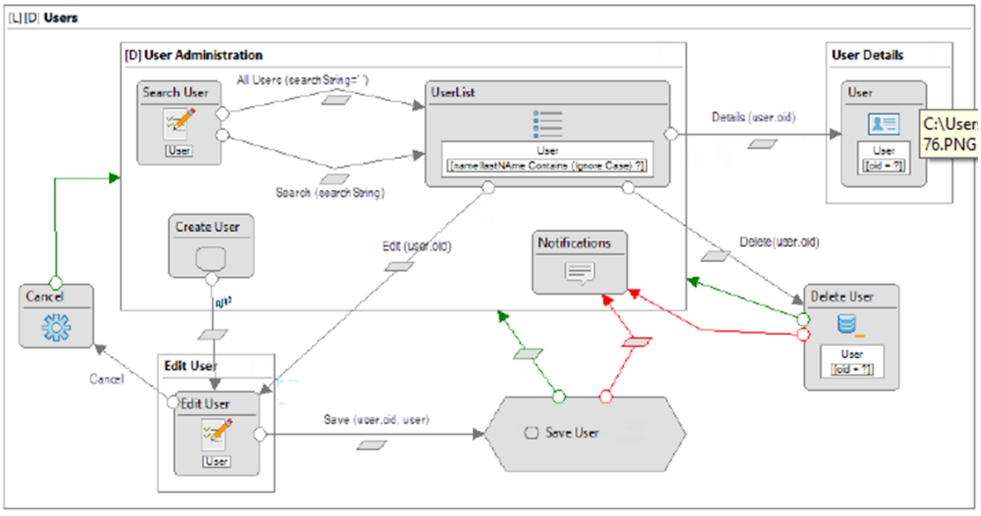


Figure 4.2: Diagrama del área de usuarios dentro del siteview del público del administrador [DCA16]

Esta área llamada Users contiene las páginas relacionadas a este concepto, denominadas administración de Usuario, detalles de usuario y editar usuario. A su vez, dentro de estas páginas se observan componentes que permiten a un usuario Administrador realizar tareas como búsqueda y listado de usuarios, ver los detalles de un usuario, eliminar un usuario y editar y crear usuarios.

En la FIGURA 4.3 se presenta el área de casos de definida dentro de Common User Site View.

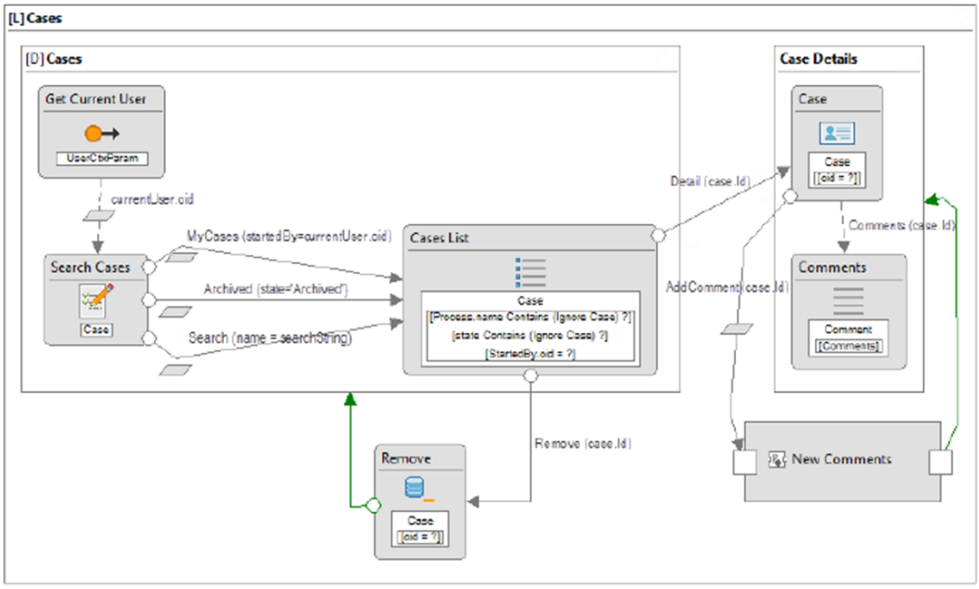


Figure 4.3: Diagrama del área de casos dentro del siteview del usuario Normal [DCA16]

Como se puede observar, las páginas que integran esta área permite a un usuario normal ver la lista de casos en los que actualmente está trabajando, listar los casos archivado y buscar un caso por nombre. Además puede eliminar un caso y agregar comentarios al mismo.

4.3 Capa de Servicios

4.3.1 Modelo Genérico

A continuación se muestra el modelo de dominio. Por simplicidad se separo en destinas partes, clasificando cada concepto según si pertenecen a ejecución (color verde), administración (color naranja) y evaluación e histórico (color lila).

La FIGURA 4.4 hace referencia al modelo presentado en el trabajo [DCA16] con la incorporación de dicha clasificación. En las siguientes imágenes se muestran los conceptos necesarios para proveer soporte a las nuevas funcionalidades estudiadas, en cada imagen se repite el concepto que se conecta con el modelo original.

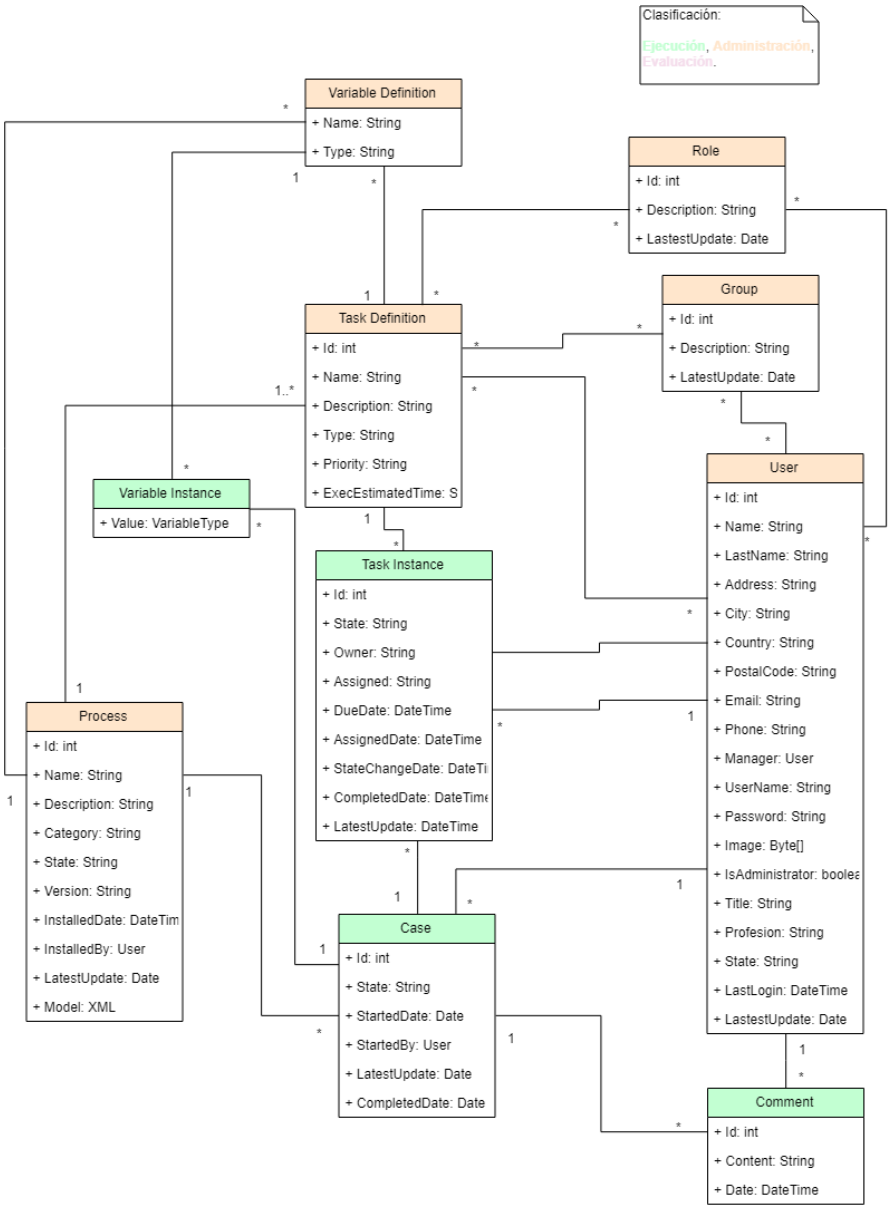


Figure 4.4: Modelo de datos existente de la API genérica

Como se observa en el modelo de las figuras FIGURA 4.4, FIGURA 4.5, FIGURA 4.6, FIGURA 4.7, FIGURA 4.8, FIGURA 4.9, aparecen todos los conceptos unificados de cada BPMS [Wes07].

En FIGURA 4.4 se mantienen los mismos conceptos que el trabajo [DCA16] de Process y Case, dónde este último es una instancia del primero, que sea crea cuando un usuario lo inicia a partir de un Proceso. Un proceso, a su vez, posee definiciones de tareas que son instanciadas dentro de un caso.

Muchas veces es necesario proveer cierta información para iniciar un caso o para realizar una tarea o también, almacenar datos de resultados de condiciones que pueden afectar el flujo de un proceso y es aquí donde aparece el concepto de Variable Definition, a nivel de proceso o de definición de tarea, por lo que su instancia Variable Instance será a nivel de caso o de instancia de tarea.

Un usuario puede pertenecer a grupos y tener roles, los cuales son determinantes a la hora de definir las tareas en que un usuario puede participar. Si bien estos dos conceptos son similares y podrían unificarse, el hecho de no hacerlo se acerca más a la realidad que se da en las organizaciones.

Los usuarios pueden, también, agregar comentarios a los casos.

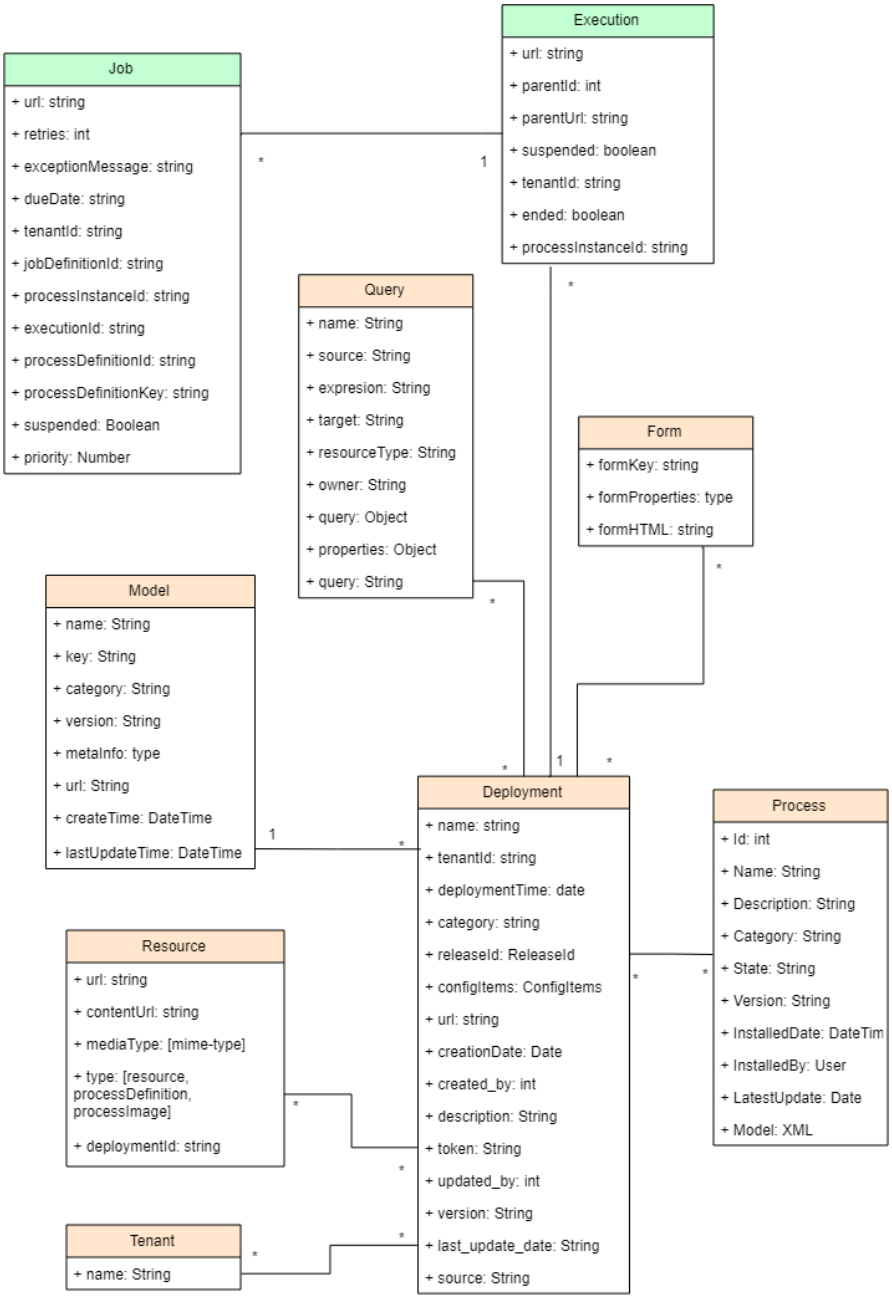


Figure 4.5: Sección del modelo de datos relacionada a Proceso

En FIGURA 4.5 Aparece el concepto de Deployment asociado a Process.Este concepto puede verse como una forma de mantener centralizada información relevante de un conjunto de procesos permitiendo a los usuarios la interacción con estos. Tal es el caso de formularios (Form), consultas (Query), datos de una ejecución en particular (Execution) e información del modelo de procesos (Model).

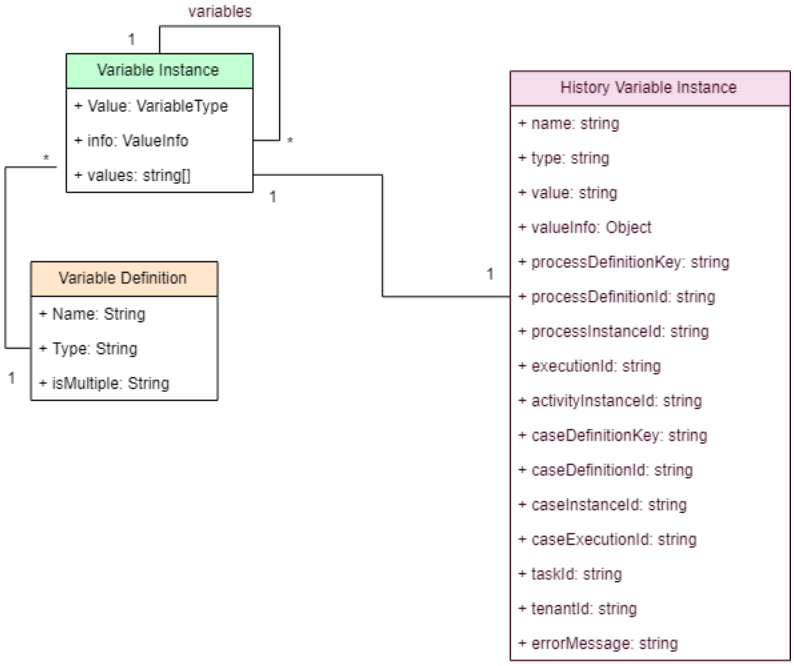


Figure 4.6: Sección del modelo de datos relacionada a Variable Instance

En FIGURA 4.6 Como se describió antes, los valores de las variables (Variable Instance), están asociados a casos o instancias de tareas.Estos valores pueden cambiar a lo largo de la ejecución, por lo que se guarda el histórico de valores que la variable ha tomado en cada ejecución (History Variable Instance). Se definen además variables complejas, que pueden tener una estructura de árbol, y por esto es que puede darse la relación de una variable de instancia con muchas de estas.

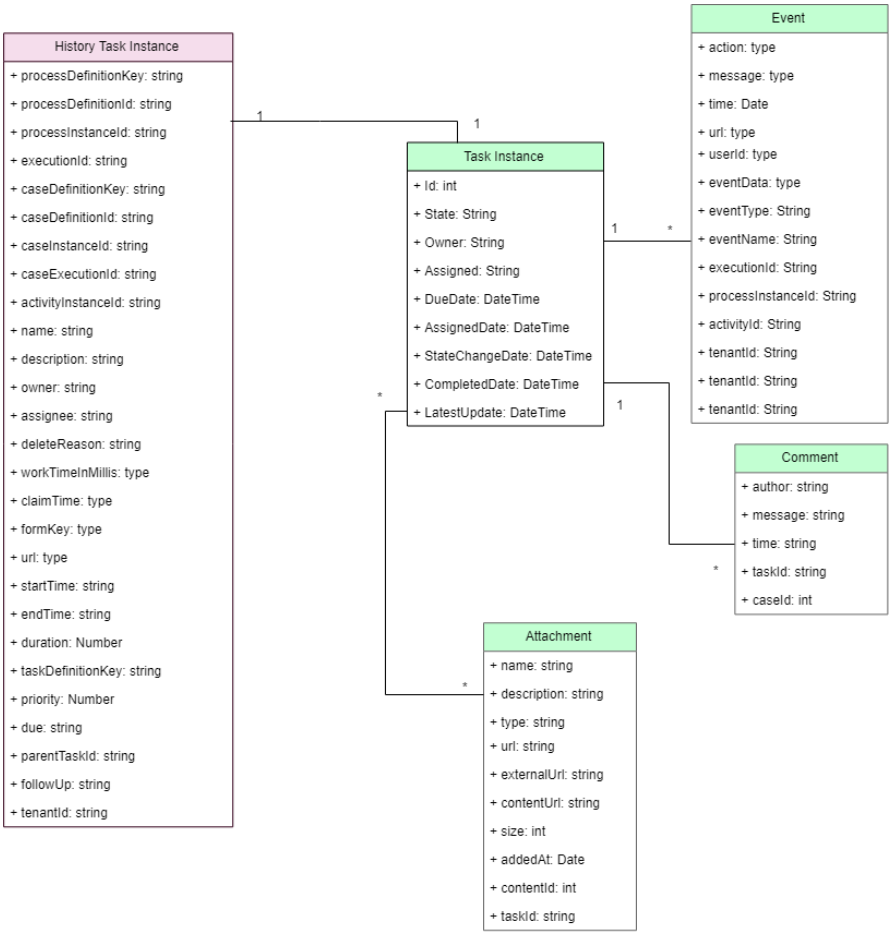


Figure 4.7: Sección del modelo de datos relacionada a Task Instance

En FIGURA 4.7 se define que a las instancias de tareas también se pueden agregar comentarios, y además adjuntar distintos formatos de archivos. La información de modificaciones que puedan existir sobre una tarea, tales como agregar un rol de usuario que pueda participar en ella, se representa en el concepto evento. También se guarda el histórico de valores de una tarea. (History Task Instance).

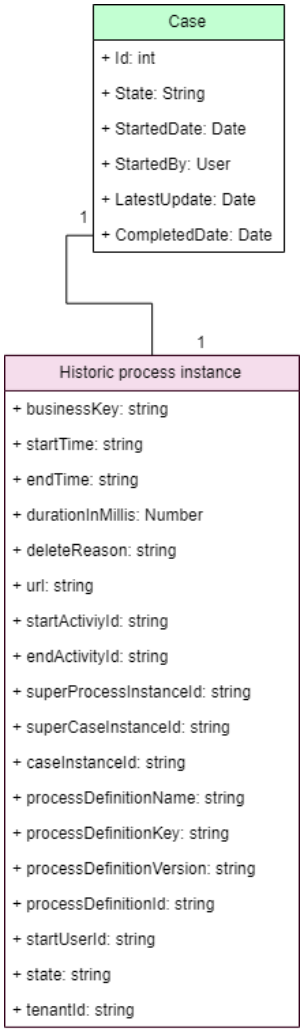


Figure 4.8: Sección del modelo de datos relacionada a Case

En FIGURA 4.8 se define de igual forma que en los casos anteriores, el histórico para un caso (History process Instance).

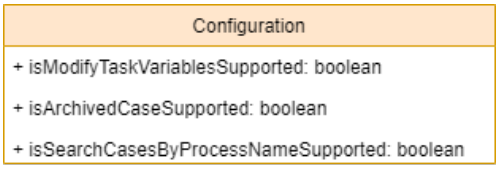


Figure 4.9: Sección del modelo de datos relacionada a Configuración

En FIGURA 4.9 aparece el concepto de Configuration, el cuál fue agregado con el objetivo de mantener información de si se soporta o no cierta operación en cada BPMS [Wes07], como las de modificar variables de tareas, archivar casos y buscar casos por nombre de proceso.

4.3.2 API Genérica

La definición de API genérica presentada a continuación es la misma que la del trabajo [DCA16]. Esta fue definida de acuerdo a funcionalidades básicas o deseables para el acceso al motor de gestión de procesos que son las que manejan los conceptos presentados en el modelo genérico definido en [DCA16], presentado en la sección previa.

Método	**Descripción**
Login(String user, String password):void	Permite al usuario identificarse en el sistema y poder acceder a las funcionalidades a las cuales tiene permitida
ChangePassword(String oldPassword, String newPassword):bool	Permite al usuario cambiar su password
GetProcessDefinitions(String name, String category, bool? active): IEnumerable<Process>	Retorna la lista de definiciones de proceso, permitiendo filtrar por nombre, categoría y estado.
GetProcessDefinition(int processId): Process	Retorna una definición de proceso en particular.
SuspendProcess(int processId): void	Pasa el proceso con id "processId" al estado Suspendido, por lo que no podrán crearse casos a partir de él, hasta que sea reactivado.
ActivateProcess(int processId): void	Pasa el proceso con id "processId" a estado Activo, permitiendo que se creen procesos a partir de él.
AssociateCategory(int processId, string category): void	Asocia una categoría a un proceso.
RemoveCategory(int processId, string category): void	Desasocia una categoría a un proceso.

GetStartVariables(int processId, string): IEnumerable<VariableInstance>	Retorna la lista de variables para iniciar una instancia del proceso enviado como parámetro.
startCase(int processId, IEnumerable<VariableInstance> casevariables): void	Permite crear una instancia del proceso cuyo id es enviado como parámetro.
GetAllCases(string processName, string CreatedBy, string userInvolved, string state): IEnumerable<Case>	Retorna la lista de casos. Los posibles filtros son: nombre de proceso, Id del usuario que lo creó, Id de un usuario que participó o está asignado en alguna tarea y estado.
GetCase(int caseId): Case	Retorna un caso cuyo id es igual al enviado como parámetro.
DeleteCase(int caseId): void	Elimina el caso enviado como parámetro.
GetTasks(string name, bool? assignee, bool? candidateUser): IEnumerable<TaskInstance>	Retorna la lista de tareas. Los posibles filtros son: el nombre de la tarea, tareas asignadas o no asignadas, tareas que pueden ser tomadas por el usuario que realiza el pedido
GetTask(int taskId): TaskInstance	Retorna la tarea cuyo id es enviado como parámetro.
TakeTask(int taskId): bool	Asigna la tarea al usuario que realiza la invocación.
ReasignTask(string taskId, string userId): bool	Reasigna la tarea con id "taskId" al usuario con id "userId".
GetTaskVariables(string taskId): IEnumerable<VariableInstance>	Retorna la lista de variables asociadas a la tarea enviada como parámetro.
ModifyTaskVariables(string taskId, IEnumerable<VariableInstance> taskvariables): bool	Permite modificar los valores de las variables de la tarea enviada como parámetro.
SaveTask(string taskId, IEnumerable<VariableInstance> taskvariables): bool	Pasa la tarea a estado cerrado.
GetTaskComments(int taskId): IEnumeable<Comment>	Retorna la lista de comentarios asociados a la tarea enviada como parámetro.
AddCommentToTask(int taskId, string userId, string comment): void	Agrega un comentario a una tarea, agegando como autor al usuario enviado como parámetro.
GetUsers(string like): IEnumerable<User>	Retorna la lista de usuarios existentes.
GetGroups(string like): IEnumerable<Group>	Retorna la lista de grupos existentes.

GetGroup(string groupId): Group	Retorna el grupo cuyo id es enviado como parámetro.
DeleteGroup(string groupId): void	Elimina el grupo cuyo id es enviado como parámetro.
ModifyGroup(Group group): void	Modifica el grupo enviado como parámetro.
AddGroup(group Group): void	Permite agregar un nuevo grupo a la lista de grupos.
AddUser(string userId, string groupId): void	Permite agregar un usuario al grupo.
RemoveUser(string userId, string groupId): void	Permite eliminar un usuario del grupo.
GetRoles(string like): IEnumerable<Role>	Retorna la lista de roles existentes.
GetRole(string roleId): Role	Retorna el rol cuyo id es enviado como parámetro .
DeleteRole(string roleId): void	Elimina el rol cuyo id es enviado como parámetro.
ModifyRole(Role role): void	Modifica el rol enviado como parámetro.
AddRole(Role role): void	Permite agregar un nuevo rol a la lista de roles.
AddUser(string userId, string roleId): void	Permite agregar un rol al usuario.
RemoveUser(string userId, string roleId): void	Permite eliminar un rol del usuario.
GetConfiguration(): Configuration	Retorna la configuración de cada BPMS [Wes07] particular de acuerdo a que funcionalidades soporta.
GetAdapters(): IEnumerable<String>	Retorna una lista con los identificadores de cada adaptador.

Table 4.1: Definición de la API Genérica

4.4 Capa de Integración

Esta capa es la encargada de comunicarse con los distintos motores de ejecución. Aquí se traducen las operaciones brindadas por la API genérica a las de cada uno de estos (SECCIÓN 3.8), implicando un mapeo de conceptos con los definidos en el Modelo genérico, según se presentó en CUADRO 3.7, así como de atributos. Cada método de esta capa recibirá un conjunto de atributos donde estarán incluidos los necesarios para realizar la correspondiente petición hacia el motor en concreto. Cabe destacar que dicho conjunto podría contener mas

atributos, en cuyo caso los restantes no serán tenidos en cuenta.

Por otra parte, se define una interfaz de configuración con todos los métodos definidos en la API genérica. Cada uno de estos métodos devuelve un booleano que indica si este motor provee de dicha funcionalidad. De esta forma la capa de presentación puede identificar que funcionalidades están implementadas.

Respecto al mapeo de errores se manejan cuatro tipos, los mismos que se definieron en el trabajo [DCA16] . Existe relacionamiento con los errores HTTP que puede devolver un motor al momento de hacer una petición, en caso de que por algún motivo la operación no pueda concluir con éxito. A continuación se listan estos:

1. **Unauthorized Exception:** Si se obtuvo un error 401 o 403
2. **NotFoundException:** Si se obtuvo un error 404
3. **InvalidParameterException:** Si se obtuvo un error 400
4. **InternalServerError:** Si se obtuvo un error 500

5 Prototipo

En este capítulo se presenta el prototipo que se va a implementar según lo especificado en el CAPÍTULO 4. Se comienza definiendo en la SECCIÓN 5.1 el alcance del mismo, luego se explica la implementación de la arquitectura en la SECCIÓN 5.4 y se ejemplifica en que consiste la instalación de la misma en la SECCIÓN 5.5.

5.1 Alcance

A continuación en el CUADRO 4.1 se presentan todos los métodos que se definieron en la API presentada en la SECCIÓN 4.3.2 y se identifican cuales de estos se van a implementar en el prototipo, considerando en la columna **Método** cada uno de estos, y en la columna **Dentro del alcance** si se lo implementa. Para definir esto se tomo como referencia los métodos asociados a la ejecución, que utiliza un usuario final no administrador. Cabe destacar que dicha implementación va a ser únicamente para los BPMS [Wes07]: Camunda, JBPM y Bonita, considerando que para cada uno de estos existe una versión gratuita y de código abierto. En consecuencia con esto se tuvo en cuenta que Activiti ya se implementó en el trabajo [DCA16], Flowable, como se hizo mención en la SECCIÓN 3.2, tiene la misma estructura que Activiti y finalmente que Bizagi es un producto pago.

Método	**Dentro del alcance**
Login(String user, String password):void	Si
ChangePassword(String oldPassword, String newPassword):bool	No
GetProcessDefinitions(String name, String category, bool? active): IEnumerable<Process>	Si
GetProcessDefinition(int processId): Process	Si

SuspendProcess(int processId): void	No
ActivateProcess(int processId): void	No
AssociateCategory(int processId, string category): void	No
RemoveCategory(int processId, string category): void	No
GetStartVariables(int processId, string): IEnumerable<VariableInstance>	Si
startCase(int processId, IEnumerable<VariableInstance> casevariables): void	Si
GetAllCases(string processName, string CreatedBy, string userInvolved, string state): IEnumerable<Case>	Si
GetCase(int caseId): Case	Si
DeleteCase(int caseId): void	No
GetTasks(string name, bool? assignee, bool? candidateUser): IEnumerable<TaskInstance>	Si
GetTask(int taskId): TaskInstance	Si
TakeTask(int taskId): bool	Si
ReasignTask(string taskId, string userId): bool	Si
GetTaskVariables(string taskId): IEnumerable<VariableInstance>	Si
ModifyTaskVariables(string taskId, IEnumerable<VariableInstance> taskvariables): bool	Si
SaveTask(string taskId, IEnumerable<VariableInstance> taskvariables):bool	Si
GetTaskComments(int taskId): IEnumeable<Comment>	No
AddCommentToTask(int taskId, string userId, string comment): void	No
GetUsers(string like): IEnumerable<User>	Si
GetGroups(string like): IEnumerable<Group>	No
GetGroup(string groupId): Group	No
DeleteGroup(string groupId): void	No
ModifyGroup(Group group): void	No
AddGroup(group Group): void	No
AddUser(string userId, string groupId): void	No
RemoveUser(string userId, string groupId): void	No
GetRoles(string like): IEnumerable<Role>	No
GetRole(string roleId): Role	No
DeleteRole(string roleId): void	No
ModifyRole(Role role): void	No

AddRole(Role role): void	No
AddUser(string userId, string roleId): void	No
RemoveUser(string userId, string roleId): void	No
GetConfiguration(): Configuration	Si
GetAdapters(): IEnumerable<String>	Si

Table 5.1: Definición de Alcance

5.2 Arquitectura concreta

La arquitectura abstracta, como se comentó en SECCIÓN: 4.1, es la misma que se desarrolló en el trabajo [DCA16]. En cuanto a la implementación de la misma se optaron por algunas series de cambios que se creyeron convenientes. Se hizo un énfasis en la modularidad de los componentes, la baja acoplación entre los mismos, y poder brindar un enfoque multi-tenant en un único producto.

Mediante esta implementación, se buscó tener en un único servidor de aplicaciones la posibilidad de manejar todos los motores que se deseen, con la facilidad de agregar nuevos de estos utilizando la implementación existente.

En cuanto al componente de Frontend se decidió desacoplarlo del Backend. Actualmente se tienen dos módulos WAR, el primero que es quien se encarga del Frontend y el segundo, en conjunto con una serie de jars, del Backend.

En esta arquitectura se brinda soporte a que distintos clientes, que usen distintos BPMS [Wes07], puedan acceder al portal web usando una única instancia de esta plataforma. Finalmente, que ésta sea capaz de dirigirlos al motor que les corresponda en tiempo de ejecución.

En la FIGURA 5.1 se muestra la implementación completa y todos los componentes involucrados en el sistema. En esta parte se puede ver que el navegador se comunica con un servidor (componente web) que es el que publica todos los artefactos del front-end. Todos estos, al momento de solicitar el index, viajan al browser y desde este se realizan las proximas invocaciones http hacia la api rest generica, finalmente se renderizan los resultados en el browser.

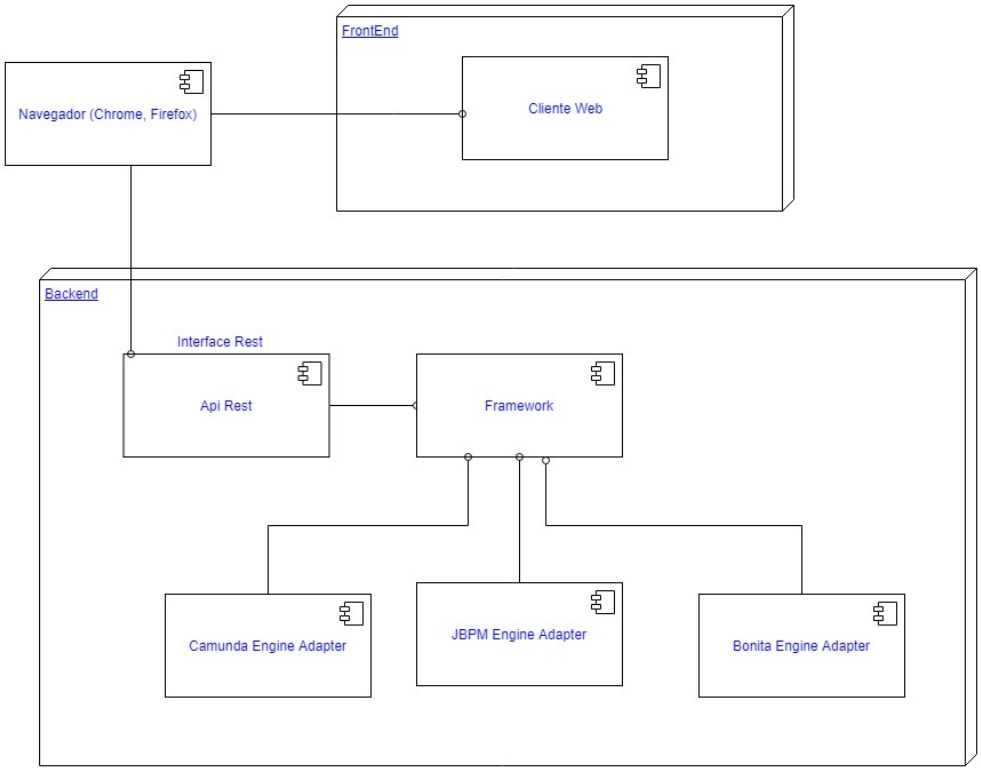


Figure 5.1: Modelo de Componentes

5.3 Tecnologías Utilizadas

Las tecnologías utilizadas fueron las mismas que se utilizaron en el trabajo [DCA16], considerando que las mismas son adecuadas y que al ser un trabajo de extensión, se creyó conveniente mantenerlas dentro de lo posible.

El frontend fue desarrollado utilizando HTML5, Bootstrap, así como el framework AngularJS.

El backend se desarrollo con Java, en donde se expuso la API REST HTTP, la cual se desarrollo utilizando Jersey como implementación de JAX-RS. De igual manera, para consumir los servicios REST expuestos por cada uno de los motores utilizados, se utilizó el cliente HTTP de Jersey. También se utilizo la API Reflection de Java.

La creación y compilación de los proyectos fue llevada a cabo con Maven.

Como servidor de aplicaciones se eligió Tomcat.

Dentro de las herramientas que fueron útiles en el desarrollo fueron Eclipse, Postman, consola de Chrome, GitHub.

5.4 Implementación

Como ya fue presentado en la SECCIÓN 5.2, el diseño de la arquitectura se divide en distintos módulos en donde cada uno de ellos se trabaja de forma independiente. Se cuenta con la aplicación web y con la API REST, que además cuenta con un modulo al cual se denomino Framework y de este dependen los distintos adaptadores, los cuales se conectan con cada motor. En las siguientes secciones se va a explicar como se implementaron cada uno de estos.

5.4.1 Descripción frontend

La aplicación Web, como se adelantó, se basa en el trabajo [DCA16]. Está implementado en AngularJS y se despliega en un servidor de aplicaciones Tomcat.

Los cambios realizados al trabajo [DCA16] fueron:

- Se desacoplo este componente del backend.
- La selección del motor al momento de hacer login, como se visualiza en la FIGURA 5.2 para de esta forma dar soporte a distintos motores a la vez.
- El soporte para nuevos tipos de variables de instancia al momento de dar inicio a un proceso o a una tarea. De esta forma es posible contar con variables con estructura de árbol, es decir variables que a su vez pueden tener una lista de estas, en caso que el motor lo soporte.
- Se modificaron las etiquetas de algunas variables para, de esta forma, facilitar la integración con la implementación del backend.

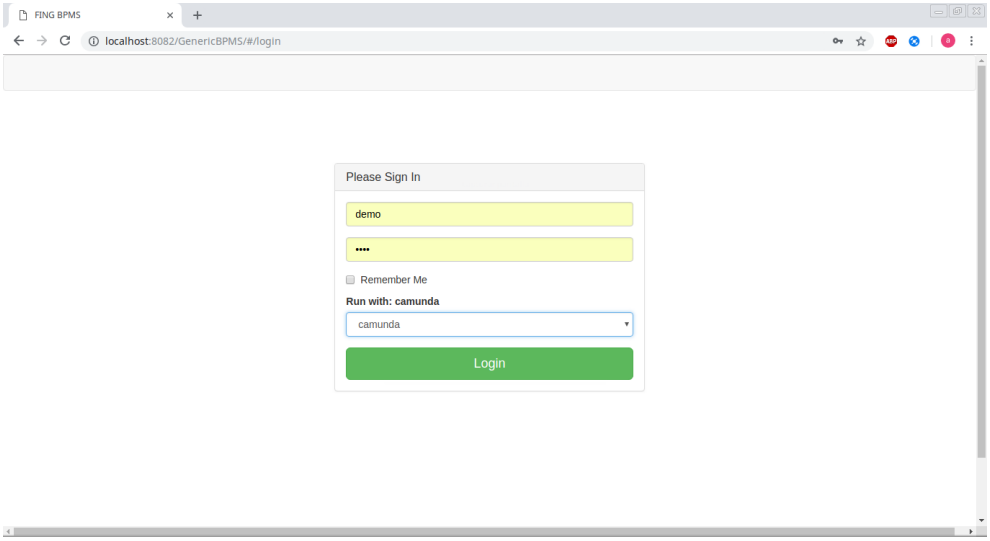


Figure 5.2: Login - Portal Genérico

En la FIGURA 5.3 se muestra a modo de ejemplo, la pantalla principal del portal web. Se pueden visualizar las tres pestañas correspondientes a Procesos, Casos y Tareas, donde se muestra la respectiva información de cada uno, según se definió en la SECCIÓN 4.2. En esta figura esta seleccionada la primer pestaña, en donde se visualiza todos los procesos instalados en el BPMS [Wes07] correspondiente, y una información asociada al proceso seleccionado.

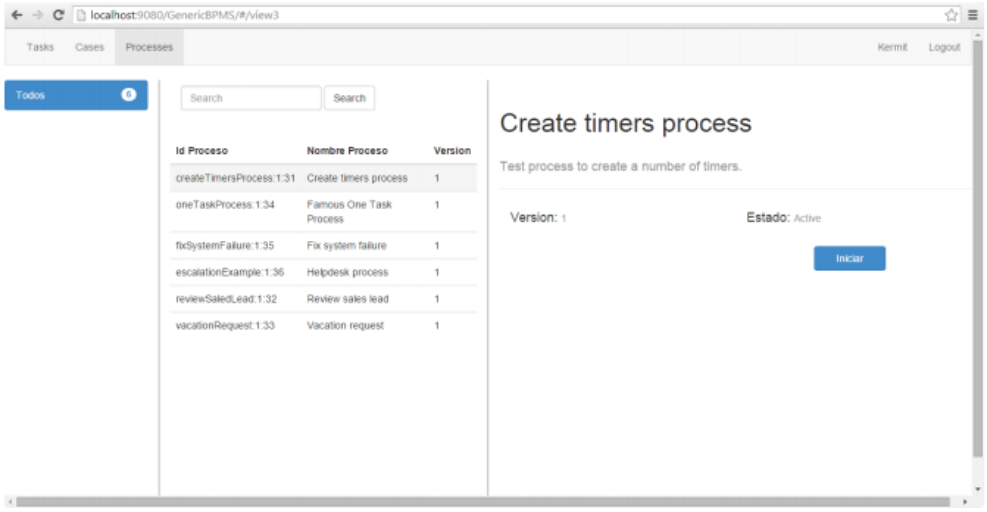


Figure 5.3: Gestión de procesos - Portal Genérico

Finalmente cabe destacar que el frontend es parametrizable según la interfaz de configuración definida en la SECCIÓN 4.4.

5.4.2 Descripción backend

Este componente fue al que se le dedicó mayor tiempo. Se hizo especial énfasis en separar la comunicación con el motor del resto de la implementación. Esto se hizo pensando en la posibilidad de extender este trabajo a distintos motores como también acotando la posibilidad de errores al momento de tener que modificar un único adaptador. En la FIGURA 5.4 se muestra unificado todos los componentes que se implementaron a nivel de software.

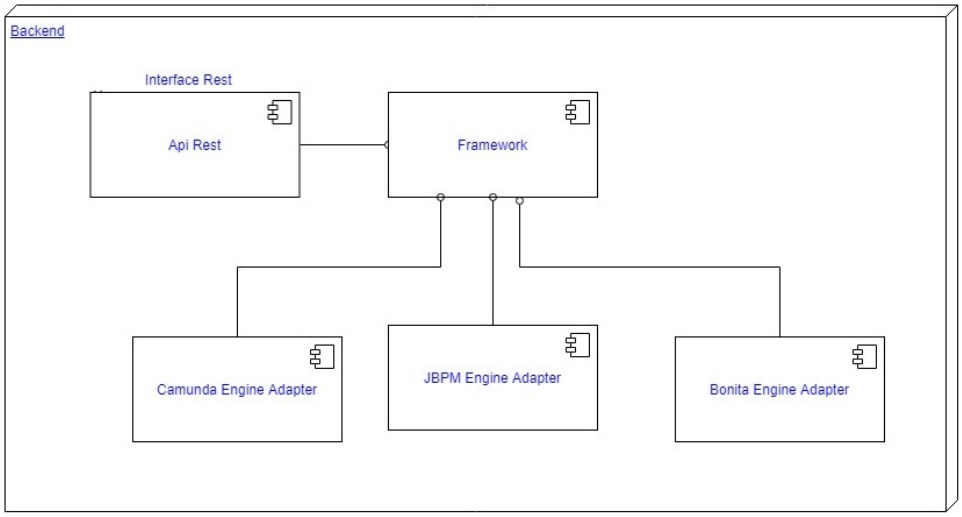


Figure 5.4: Modelo componentes backend

En concreto la implementación actual, y con la que se presenta el trabajo se detalla a continuación:

API genérica

Esta tiene la responsabilidad de recibir las peticiones REST que llegan desde el cliente, y trasladar el request al adaptador del motor pertinente (esto se verifica según la cookie previamente seteada por cada adapter). Está implementada usando la especificación Java EE, usa un EJB stateless para manejar cada una de las respuestas que vienen desde el servidor.

Framework

Este componente se diseñó en función de tener centralizado todo lo necesario para implemen-

tar un nuevo adaptador. El mismo cuenta con el modelo de datos presentado en la SECCIÓN 4.3.1, también cuenta con la interfaz presentada en la SECCIÓN 4.3.2 que define todos los servicios que debe de implementar un adaptador. Por último, cuenta con el siguiente conjunto de librerías útiles que usan tanto la API genérica como los adaptadores:

- json
- javax.ejb-api
- gson
- jersey-media-json-jackson
- jersey-server
- javax.annotation-api
- javax.ws.rs-api
- jersey-client
- jsoup
- jersey-common
- jersey-container-servlet

Este componente es además también el responsable de brindarle a la API genérica un mapa que contiene todas las clases que implementan a dicha interfase con el identificador que devuelve cada una de estas clases al ejecutar el método obtenerNombreMotor(). Por otra parte cabe destacar que estas clases van a ser instanciadas usando Java Reflection. Existe un archivo, que debe de estar ubicado en la carpeta conf del Tomcat cuyo nombre es ”adaptadores.properties”. Este archivo cuenta con tres propiedades por motor, y se podrían ir agregando a medida que se se quiera ingresar un nuevo motor. Las propiedades son:

ubicacionMotor1 = adapter.camunda.IBpmsEngineCamunda

urlMotor1 = http://localhost:8081/camunda/api/engine/engine/default/

AuthenticationUrl1 = http://localhost:8081/camunda/api/admin/auth/user/default/login/welcome

Donde:

ubicacionMotor1 es el directorio donde se encuentra la clase del "motor1" que implementa la interfase (En el ejemplo, la clase del adaptador de Camunda que implementa la interfase). Considerar que reflection cambia los "." por "/". En el ejemplo, debe existir una clase de nombre IBpmsEngineCamunda en algun jar dentro del libs del Tomcat en la ubicación adapter/camunda.

urlMotor1 es la url en donde se encuentra publicado el motor de ejecución.

AuthenticationUrl1 es la dirección de autenticación del motor1 si existiera. Esto permite flexibilidad al momento de mantener los usuarios. Se podría ocasionalmente tener otro servidor que se encargue de todo lo que corresponde a las cuentas de los usuarios.

En la FIGURA 5.5 se muestra parte del código del framework, el cual mediante reflection carga las instancias de las clases que implementan los adaptadores

```
public static Map<String, IAdapter> crearAdaptadores() {
    Map<String, IAdapter> retorno = new HashMap<String, IAdapter>();
    ClassLoader classLoader = FabricaDeAdaptadores.class.getClassLoader();
    File configDir = new File(System.getProperty("catalina.base"), "conf");
    File configFile = new File(configDir, "adaptadores.properties");
    InputStream stream;
    try {
        stream = new FileInputStream(configFile);
        Properties props = new Properties();
        props.load(stream);
        Integer iterador = new Integer(1);
        while (props.containsKey("ubicacionMotor"+String.valueOf(iterador.intValue())))
        {
            String ubicacionMotor = props.get("ubicacionMotor"+String.valueOf(iterador.intValue())).toString();
            String urlMotor = props.get("urlMotor"+String.valueOf(iterador.intValue())).toString();
            String urlAuthenticationMotor = props.get("AuthenticationUrl"+String.valueOf(iterador.intValue())).toString();

            System.out.println("AuthenticationUrl"+String.valueOf(iterador.intValue())+ urlAuthenticationMotor );
            System.out.println("ubicacionMotor"+String.valueOf(iterador.intValue())+ ubicacionMotor );
            System.out.println("urlMotor"+String.valueOf(iterador.intValue())+ urlMotor);
            //adapter.camunda.IBpmsEngineCamunda
            Class<?> aClass = classLoader.loadClass(ubicacionMotor);

            IAdapter iadapter = (IAdapter) aClass.newInstance();
            iadapter.setUrlMotor(urlMotor);
            iadapter.setUrlAuthentication(urlAuthenticationMotor);
            retorno.put(iadapter.obtenerNombreMotor().toLowerCase(), iadapter);
            iterador = new Integer(iterador.intValue() +1 );

        }

    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
```

Figure 5.5: Código del framework que se ejecuta para instanciar los adaptadores

Se destaca con este diseño, el bajo costo de agregar un nuevo motor a la solución actual. Para lograrlo, se tiene este único jar el cual contiene todo lo necesario para desarrollar la conexión con un nuevo motor. Se deberá implementar la interfaz proporcionada por este, colocar cada una de las respectivas propiedades en el archivo "adaptador.properties" y agregar esta implementación del nuevo adaptador como un jar en el directorio libs del Tomcat, sin tener que modificar nada de lo ya implementado.

Adaptadores

Cada uno de estos componentes de software implementan la interfaz que se encuentra en el framework. Cada uno de los métodos de esta van a ser invocados desde la API genérica y cada adaptador tiene la responsabilidad de ejecutar la operación hacia su respectivo motor y de devolver la respuesta indicada.

Para el funcionamiento de esta arquitectura se agregaron los siguientes métodos a la interfase que define el framework:

1- **String obtenerNombreMotor():** Este método devuelve un String con el nombre con el que se identifica el motor.

2- **void setearUrlMotor(String url):** Este método se ejecuta al momento de iniciar una nueva instancia de la clase que implementa la interfase del motor. Es el que setea la url en donde el motor se encuentra.

3- **void setearUrlAuthentication(String url):** Este método es responsable de setear la url de autenticación. Podría ser la misma que la del método anterior.

También es responsabilidad de cada adaptador setear una cookie al momento de hacer login la cual le indica a la API genérica cual es el motor que ejecutó esta operación, para que en las futuras invocaciones esta pueda identificar que motor debe atender el pedido. La cookie se debe llamar NOMMOTOR y debe de tener como valor el string que devuelve el método obtenerNombreMotor.

5.5 Flujo de la aplicación

En la FIGURA 5.6 se muestra el flujo que tiene un usuario y como impacta este en el portal genérico y en el respectivo motor. La primer operación que se ve en la misma es un llamado asincrónico que se produce al momento en que se carga la página de login.

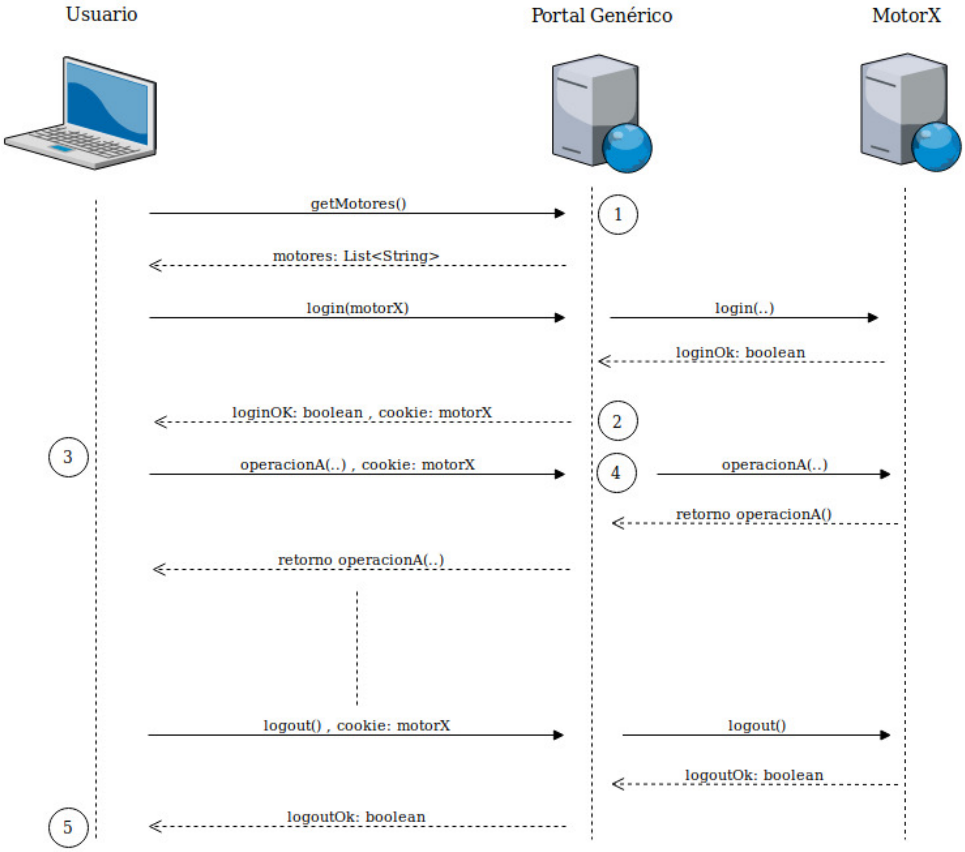


Figure 5.6: Flujo de la aplicación

En el punto 1 de la figura se pide al framework todos los adaptadores existentes. Para obtenerlos el mismo lo hace a partir del archivo de configuración ”adaptadores.properties”. Crea una nueva instancia de la clase que implementa la API genérica para cada uno de presentes. Posteriormente obtiene el nombre de cada uno el cual es proporcionado por dichas instancias al ejecutar el método obtenerNombreMotor(), asociando dichas instancias a sus respectivos nombres.

En el punto 2 de la figura se muestra que el adaptador es el responsable de setear la cookie NOMMOTOR con el valor que devuelve el metodo obtenerNombreMotor() en la respuesta HTTP.

En el punto 3 se setean las cookies en el navegador para las futuras invocaciones.

En el punto 4 de la figura la API le pide al Framework el adaptador que se identifica con el

valor que trae la cookie nomMotor. A este adaptador le ejecuta el metodo solicitado en el request.

En el punto 5 de la figura se borran las cookies del navegador.

6

Caso de Estudio

En este capítulo se muestra la ejecución del proceso de negocio representado en el diagrama FIGURA 6.1. El mismo es un proceso simple pero tiene aspectos relevantes que permiten ser visualizados de igual forma a la hora de integrar el portal web con un motor de ejecución de procesos particular.
El proceso es llamado ”Pedido de Vacaciones” y es iniciado por un usuario con rol de Empleado. Luego realiza la tarea “ingresar pedido de vacaciones“ y este pedido debe ser evaluado por un usuario con rol de administrador que debe tomar la decisión de aprobarlo o no. En caso de aprobarlo, se envía un email que confirma esta acción. En el caso contrario, el empleado tiene la opción de ajustar el pedido para que vuelva a ser evaluado por el administrador o de terminar el mismo.

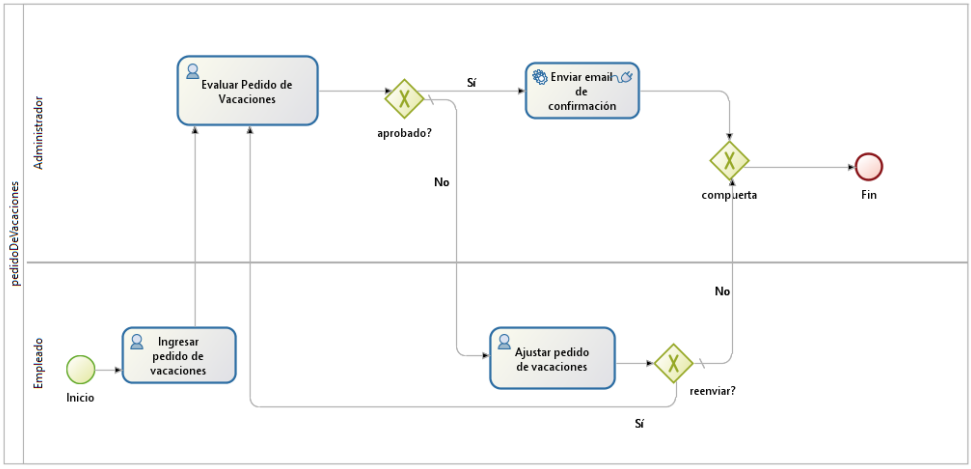


Figure 6.1: Modelo del proceso de negocio ”Pedido de vacaciones”

La ejecución de este proceso será en los motores Camunda [Cam18a], Bonita [Bon18] y jBPM [JBP18a].

Los usuarios definidos para la ejecución son 'bel' con rol Empleado y 'admin' con rol Administrador en el caso de Bonita, 'Alejandro' con rol Empleado y 'demo' con rol Administrador en el caso de Camunda, y por último 'kieserver' con rol Empleado y 'mary' con rol Administrador en el caso de JBPM.

El proceso comienza con el inicio de sesión de los usuarios. Como se ve en la FIGURA 6.2, se permite seleccionar cualquiera de los motores existentes e ingresar los datos de usuario y contraseña para hacer el login. Los posibles errores que se pueden presentar en esta instancia se dan cuando el usuario o contraseña no son válidos y cuando el motor seleccionado no está disponible, ya sea por una falla en el servidor o porque no se hizo la configuración correspondiente asociada a este motor, mostrándose para cada uno de estos casos el mensaje de error correspondiente.

En el caso presentado se hace con los motores mencionados y con los respectivos usuarios de rol empleado por primera vez.

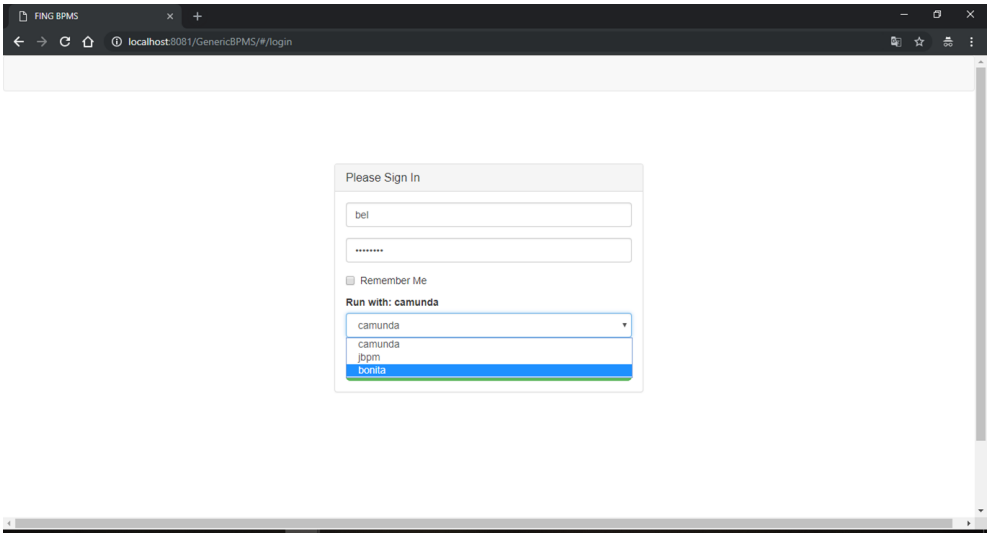
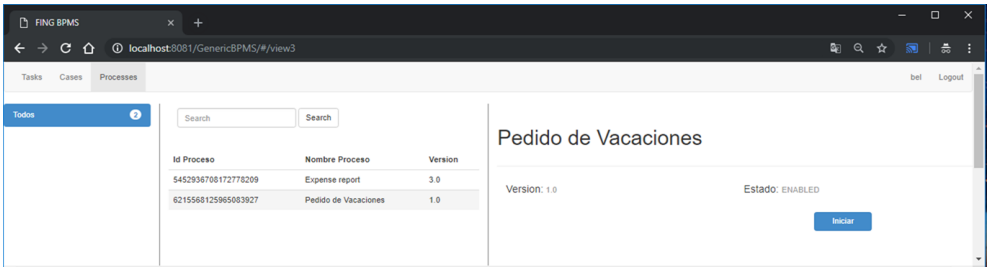
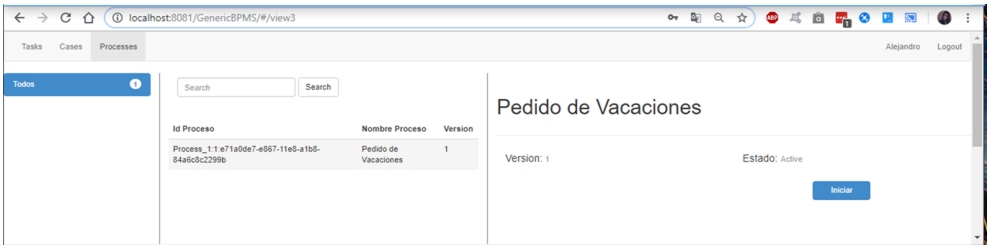


Figure 6.2: Login

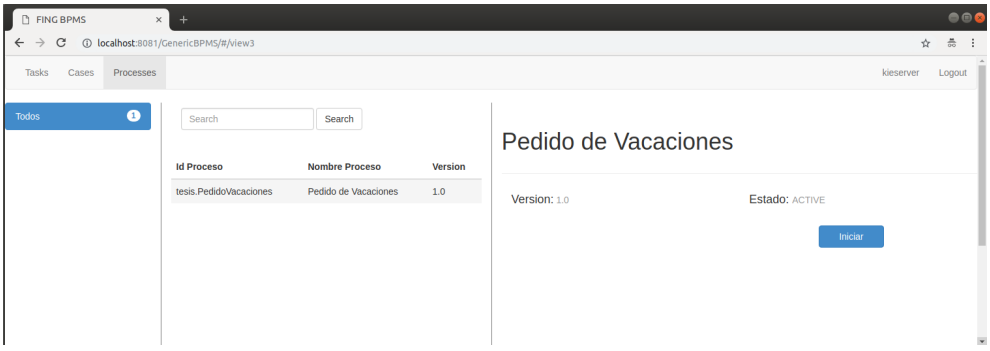
En una primer instancia, los usuarios empleados seleccionan el proceso "Pedido de Vacaciones" a partir del cual se crearán los casos, como se observa en la FIGURA 6.3. Cuando es necesario instanciar un proceso con variables, se desplegará un formulario para ingresar los valores.



(a) Bonita



(b) Camunda



(c) jBPM

Figure 6.3: Lista de procesos existentes

Los servicios asociados a la pantalla de la FIGURA 6.3 se presentan en el CUADRO 6.1.

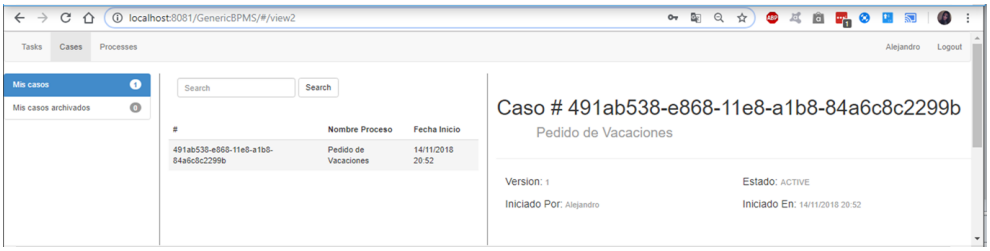
Obtener definiciones de procesos
API genérica
GET `api/processes`
jBPM
GET `/kie-server/services/rest/server/queries/processes/definitions`
Camunda
GET `/camunda/api/engine/engine/default/process-definition`
Bonita
GET `/API/BPM/process?p=0`

Obtener una definición de proceso
API genérica
GET `api/processes/{id}`
jBPM
GET `/kie-server/services/rest/server/queries/processes/definitions/{processId}`
Camunda
GET `/camunda/api/engine/engine/default/process-definition/{processId}`
Bonita
GET `/API/bpm/process/{processId}`
Iniciar un proceso
API genérica
POST `api/cases/start/{processId}`
jBPM
POST `/kie-server/services/rest/server/containers/{ContainerId}/processes/prueba.Proceso1/instances`
Camunda
POST `/camunda/api/engine/engine/default/process-definition/{processId}/start`
Bonita
POST `/API/bpm/case`, body: { “processDefinitionId": “processId“ }
Obtener variables para iniciar proceso
API genérica
GET `api/processes/{id}/startVariables`
jBPM
GET `/kie-server/services/rest/server/containers/{ContainerId}/processes/definitions/prueba.Proceso1/variables`
Camunda
GET `/camunda/api/engine/engine/default/process-definition/{processId}/form-variables`
Bonita
GET `/API/bpm/process/{processId}/contract`

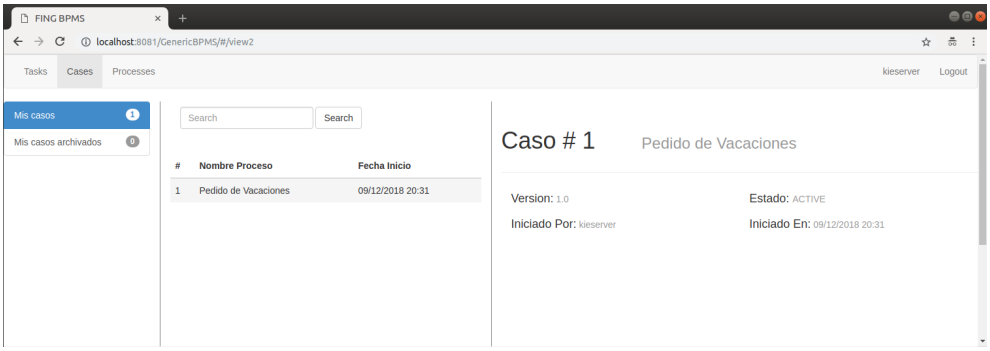
Table 6.1: Servicios utilizados en FIGURA 6.3

En la FIGURA 6.4 se presenta la pantalla donde se listan todos los casos existentes donde vemos el caso ”Pedido de vacaciones”, iniciado anteriormente. Aquí se puede ver la diferencia en los identificadores provistos por cada uno de los motores.

(a) Bonita



(b) Camunda



(c) jBPM

Figure 6.4: Lista de casos existentes

Los servicios que se utilizan para obtener la información que se muestra en la pantalla de la FIGURA 6.4 se presentan en el CUADRO 6.2.

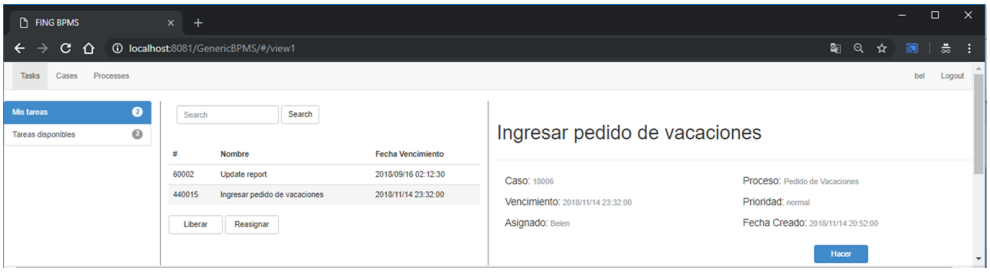
Obtener casos activos/archivados
API genérica
GET `api/cases?state=active` GET `api/cases?state=archived`
jBPM
GET `/kie-server/services/rest/server/queries/processes/instances?initiator=mary&status=2` GET `/kie-server/services/rest/server/queries/processes/instances?status=1`
Camunda
GET `/camunda/api/engine/engine/default/history/process-instance?finished=true`

GET `/camunda/api/engine/engine/default/history/process-instance?unfinished=true`
Bonita
GET `/API/bpm/case?p=0&f=started_by=currentUserId`
GET `/API/bpm/archivedCase?p=0&f=started_by=currentUserId`
Obtener un caso activo/archivado
API genérica
GET `api/cases/{id}?isArchived=true`
GET `api/cases/{id}?isArchived=false`
jBPM
GET `/kie-server/services/rest/server/containers/{ContainerId}/processes/instances/caseId`
Camunda
GET `/camunda/api/engine/engine/default/history/process-instance/{processID}`
GET `/camunda/api/engine/engine/default/process-instance/{processID}/variables`
Bonita
GET `/API/bpm/case/{caseId}`
GET `/API/bpm/archivedCase/{caseId}`

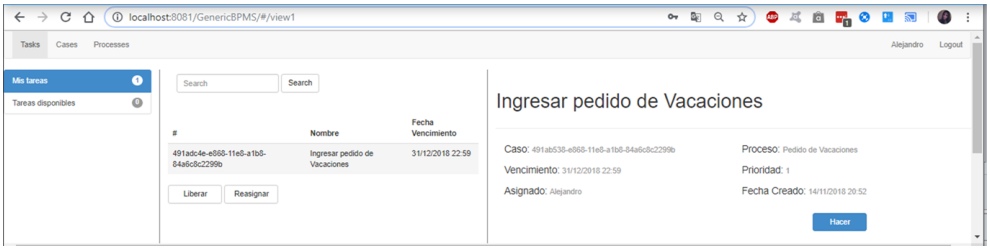
Table 6.2: Servicios utilizados en FIGURA 6.4

Una particularidad en estas operaciones es que para que se satisfagan en el adaptador de Bonita se invocan dos operaciones distintas dependiendo si se trata de casos archivados o no. En cambio en Camunda, se puede obtener la información relativa a los distintos tipos de casos proporcionándole este dato como query param.

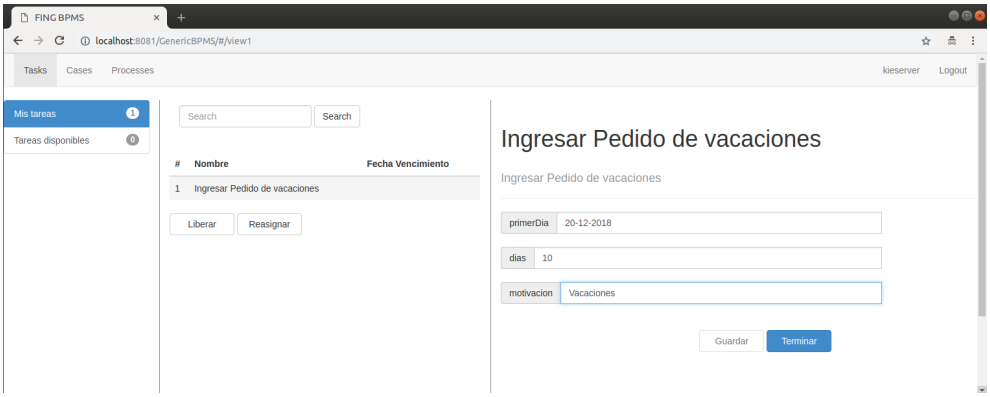
Si se selecciona la pestaña Tareas se podrá ver las tareas disponibles para ser tomadas, y las ya tomadas como se muestra en la FIGURA 6.5. Como los usuarios actualmente logueados tienen únicamente el rol de Empleado, la única tarea disponible para tomar es la de ”Ingresar pedido de vacaciones”. Cabe destacar que esta tarea en principio va a estar como disponible para los usuarios que puedan tomarla, una vez tomada existe la posibilidad de liberarla o de reasignarla a otro usuario con el rol que corresponda.



(a) Bonita



(b) Camunda



(c) jBPM

Figure 6.5: Tareas existentes para ejecutar

Los servicios para obtener y modificar datos de las tareas asociados a la pantalla de la FIGURA 6.5 se presentan en el CUADRO 6.3.

Obtener tareas disponibles/asignadas
API genérica
GET `api/tasks?assignee=false&candidateUser=true`
GET `api/tasks?assignee=true&candidateUser=false`
jBPM
GET `/kie-server/services/rest/server/queries/tasks/instances/owners`
GET `/kie-server/services/rest/server/queries/tasks/instances/pot-owners?status=Ready`

Camunda
GET /camunda/api/engine/engine/default/task?asignee={userId}
GET /camunda/api/engine/engine/default/task?candidateUder={userId}
Bonita
GET /API/bpm/humanTask?p=0&f=user_id=currentUserId
GET /API/bpm/humanTask?p=0&f=assigned_id=currentUserId
Obtener una tarea
API genérica
GET api/tasks/{id}
jBPM
GET /kie-server/services/rest/server/queries/tasks/instances/{taskId}
Camunda
GET /camunda/api/engine/engine/default/task/{taskId}
GET /camunda/api/engine/engine/default/history/process-instance/{processId}
Bonita
GET /API/bpm/humanTask/{taskId}
Tomar una tarea
API genérica
PUT api/tasks/{id}/take
jBPM
PUT /kie-server/services/rest/server/containers/{ContainerId}/tasks/{taskId}/states/ claimed
Camunda
POST /camunda/api/engine/engine/default/task/{taskID}/claim
Bonita
PUT /API/bpm/userTask/{taskId} body: {“assigned_id": “user_id" }
Obtener variables para hacer tarea
API genérica
GET api/tasks/{id}/variables
jBPM
GET /kie-server/services/rest/server/containers/{ContainerId}/tasks/{taskId}/contents/ input
Camunda
GET /camunda/api/engine/engine/default/task/{taskId}/form-variables
Bonita
GET /API/bpm/userTask/{taskId}/contract

Table 6.3: Servicios utilizados en FIGURA 6.5

Los detalles de las tareas se pueden observar en el panel derecho, seleccionando las mismas. Es posible que al seleccionar una tarea, la misma ya no se encuentre, debido a que, por ejemplo, un usuario con permisos especiales eliminó el caso en dónde se encontraba esta tarea. En este caso, se visualizará un error como el que se muestra en la FIGURA 6.6.

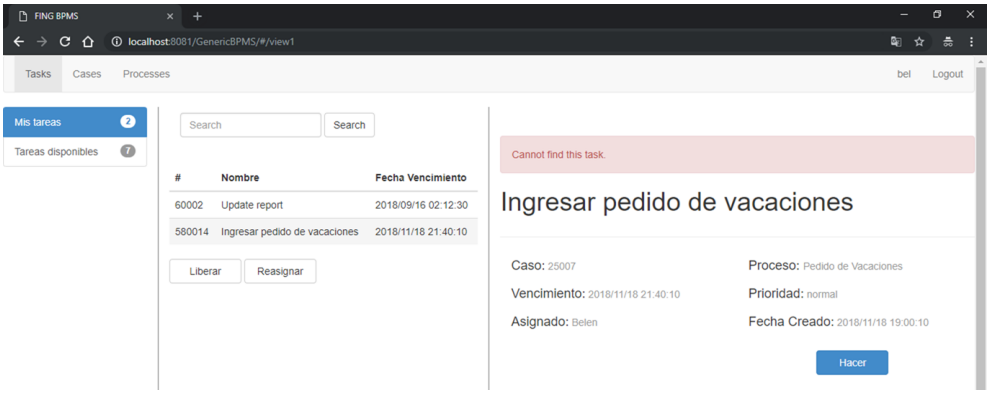
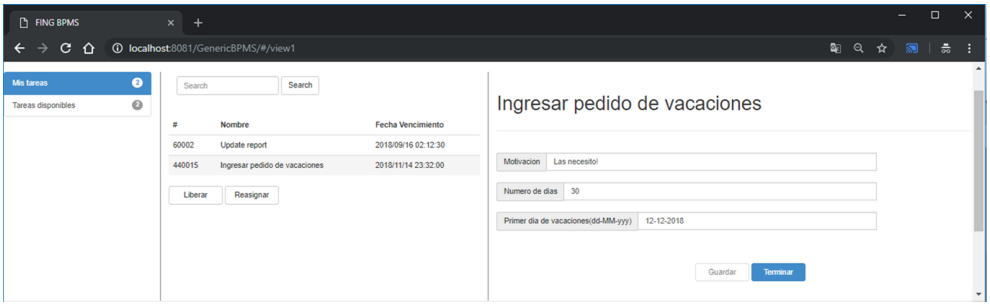


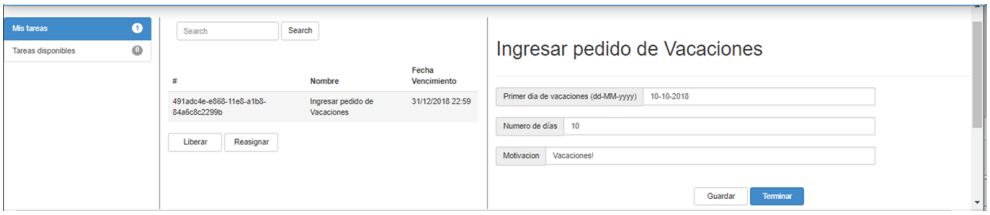
Figure 6.6: Error al no encontrar la tarea seleccionada

En la FIGURA 6.7, se puede visualizar el formulario que se debe completar para realizar la tarea, una vez presionado el botón 'Hacer'. Los campos de formularios son de tipo String y enumerado y se construyen a partir de las variables asociadas a la tarea, que tienen un atributo de tipo, un nombre e identificador. Las variables de tipo booleanas se manejan como enumerados. El servicio que se utiliza para obtener las variables de instancia que se mapean a cada uno de estos campos es el que se muestra en tabla CUADRO 6.3.

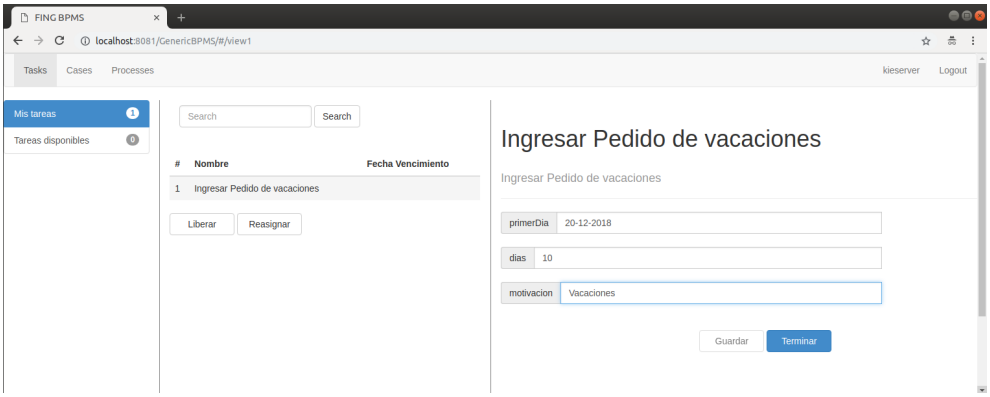
Para lograr obtener estas variables, los adaptadores realizan las traducciones correspondientes entre la estructura particular a la estructura general que maneja las variables.



(a) Bonita



(b) Camunda



(c) jBPM

Figure 6.7: Formulario para la tarea de ingresar el pedido de vacaciones

El servicio que se invoca para realizar la tarea y enviar las variables al presionar el botón terminar de la pantalla de la FIGURA 6.7 se presentan en el CUADRO 6.4.

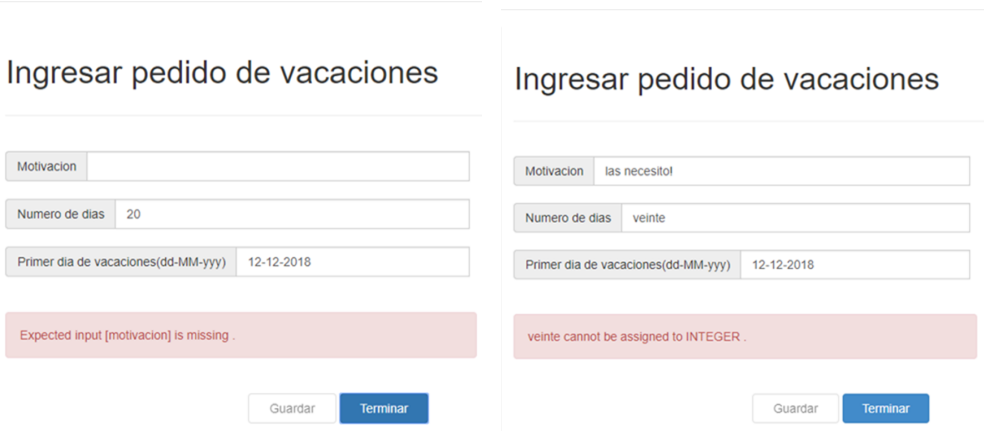
Terminar tarea
API genérica
POST `api/tasks/{id}/save`
jBPM
PUT `/kie-server/services/rest/server/containers/{ContainerId}/tasks/{taskId}/states/claimed`
PUT `/kie-server/services/rest/server/containers/{ContainerId}/tasks/{taskId}/states/started`

Camunda
POST `/camunda/api/engine/engine/default/task/{taskId}/submit-form`
Bonita
POST `/API/bpm/userTask/{taskId}/execution`

Table 6.4: Servicio utilizado en FIGURA 6.7

Los adaptadores realizan las conversiones correspondientes entre estas variables con valores de tipo String a tipos enteros o booleanos, de ser posible, para luego realizar correctamente las invocaciones hacia su motor. Para llevar a cabo esta conversión es necesario tener en cuenta el tipo de la variable que se encuentra en la definición de variable asociada.

Tomando como ejemplo la Figura 6.8, en dónde al ingresarse en un campo un valor que no puede convertirse a entero, el mensaje de error que retorna el motor de ejecución, al intentar realizar la operación asociada a la mencionada anteriormente, provee de esta explicación. También si hay variables requeridas, retornará un error.



(a) Error en campo requerido

(b) Error en campo de tipo entero

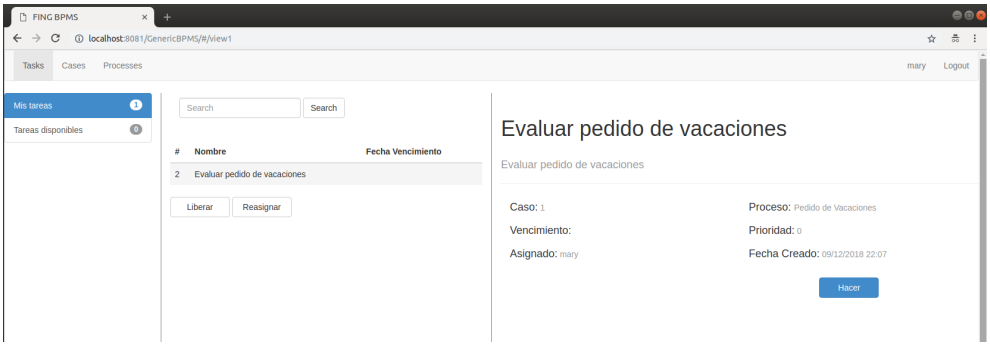
Figure 6.8: Errores en campos

Los adaptadores también manejan estructuras particulares de acuerdo a cada motor de procesos y por esto es necesario realizar un mapeo entre la estructura genérica de variables a la particular.

Una vez terminadas las tareas, se procede a desloguearse y a loguearse con los usuarios con rol de Administrador en cada motor. Al ingresar se ve la tarea ”Evaluar Pedido de vacaciones” dentro de "Tareas disponibles". Una vez tomadas estas tareas se visualizan los detalles que se muestran en la FIGURA 6.9

(a) Bonita

(b) Camunda

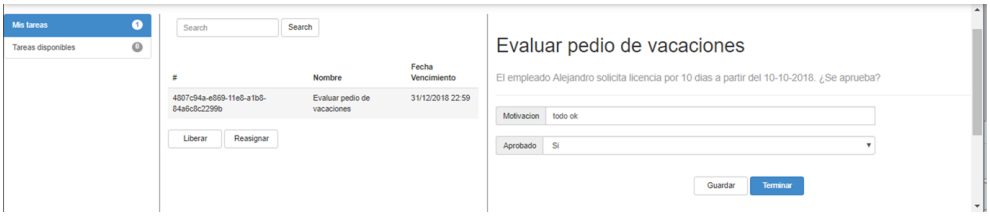


(c) jBPM

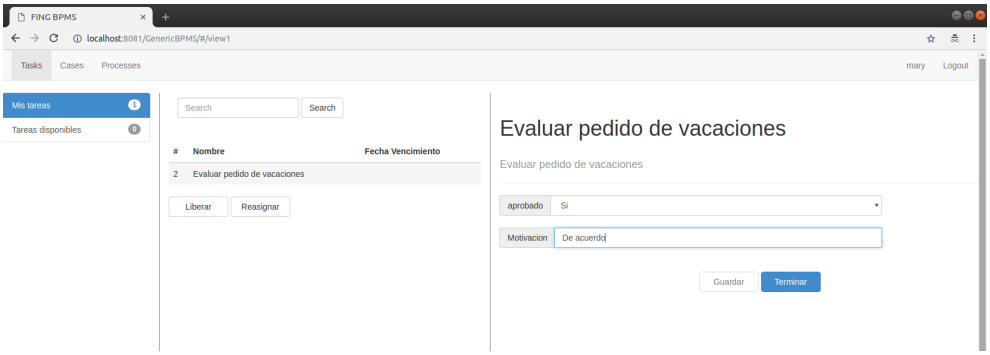
Figure 6.9: Evaluar pedido de vacaciones

De igual forma que el caso anterior, los administradores proceden a realizar las tareas completando los formularios con los datos necesarios, mostrados en la FIGURA 6.10.

(a) Bonita



(b) Camunda



(c) jBPM

Figure 6.10: Formulario de la tarea de evaluar pedido de vacaciones

En caso de no aprobar el pedido de vacaciones la siguiente tarea sera ”Ajustar pedido de vacaciones” presentada en la FIGURA 6.11 para el usuario que solicito las vacaciones, donde el mismo deberá ajustar los datos en caso de querer reenviar la solicitud o terminar el flujo. Si este usuario decide volver a solicitarla, el proceso vuelve al estado de la FIGURA 6.9. Por otro lado, si se aprueban las vacaciones, se ejecuta la tarea automática ”Enviar email de confirmación” donde se le notifica al usuario solicitante, que fue aprobado su pedido de vacaciones enviándose el mail que se ve en la FIGURA 6.12.

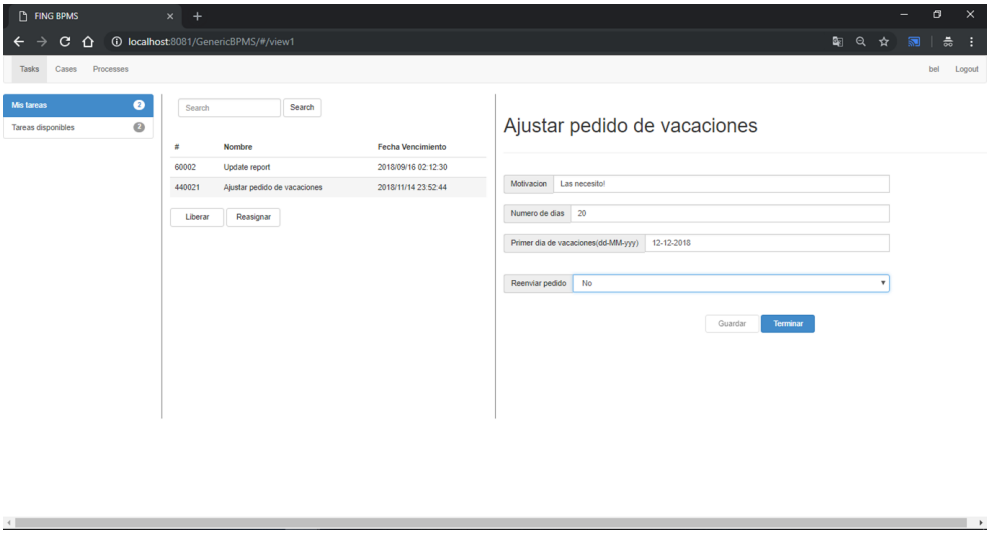


Figure 6.11: Ajustar pedido de vacaciones

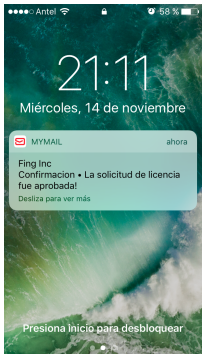


Figure 6.12: Confirmación del pedido de licencia

7 Conclusiones y Trabajo a Futuro

7.1 Conclusión

En este trabajo se desarrolló un portal genérico para la ejecución de procesos de negocio, a partir de la definición de un modelo de datos y API genérica que se adapta a seis implementaciones concretas de BPMS.

Uno de los objetivos de este trabajo fue relevar las propuestas existentes sobre la unificación de conceptos de BPM y compararla con la propuesta existente en el trabajo [DCA16]. En dicho relevamiento, presentado en el CAPITULO 2, encontramos que la mayoría de las propuestas difieren con [DCA16] en cuanto al nivel de abstracción mayor con que realizan la identificación de conceptos manejados en los BPMS. Además, en que esta propuesta [DCA16] proporciona una vista unificada de conceptos y relaciones con foco en la ejecución de un proceso de negocio, junto con una vista altamente desacoplada de los componentes principales de un BPMS, ya que el portal web que implementa, junto con su modelo de datos subyacente y API que permite administrar estos conceptos, son genéricos.

Siguiendo con los objetivos, evaluamos los portales y APIs de servicios de acceso al motor de procesos de los BPMS Activiti, Bizagi, Bonita, Flowable, Camunda y jBPM, presentado en el CAPITULO 3. En cuanto a dicha evaluación observamos que si bien cada portal ofrece su propia interfaz de usuario con algunas particulares propias, en lo que respecta a la ejecución, sus servicios son todos similares. Esto se corresponde con el análisis hecho en la SECCIÓN 2.2 en donde se visualiza que todos estos sistemas tienen un núcleo común. Esto posibilitó la definición de un único sistema (portal web y API genérica) para el manejo de diferentes motores de procesos de negocio, cumpliendo con el resultado esperado.

Respecto al objetivo de generar automáticamente el portal web genérico utilizando la API genérica ampliada, como la misma no agrega nuevas funcionalidades, sino que la expansión es la de soportar mas motores de ejecución, no se tuvo la necesidad de modificar la generación

automática presentada en el trabajo [DCA16] como se comento en la SECCIÓN 4.2.

En el CAPITULO 4 extendimos y adaptamos el modelo de datos y API genéricos presentados en el trabajo [DCA16]. Para esto realizamos un modelo de datos para cada uno de los BPMS antes mencionados, los cuales fueron presentados en el CAPITULO 3. Profundizamos el análisis, contemplando conceptos relacionados con la Administración y la Evaluación. Continuando con esto logramos unificar a todos estos, obteniendo el modelo genérico presentado en este capitulo. De este análisis concluimos que fue posible obtener un único modelo conceptual, dado que a pesar que cada BPMS maneja sus propios conceptos, existe relacionamiento entre los mismos.

El análisis presentado en el párrafo anterior se puso en práctica cuando implementamos el prototipo del portal genérico de BPMS, mostrado en el CAPITULO 5. De esta forma se demuestra que la abstracción de conceptos es posible, dado que existe cierto relacionamiento entre los mismos, y se pueden tratar de forma similar, a través de la API REST que provee cada BPMS. Esto último simplificó bastante la integración. Por otra parte, hicimos especial énfasis en que el trabajo pueda continuarse y profundizares, sin necesidad de un rediseño. Esto va de la mano con un diseño modular de bajo acoplamiento. En donde, en caso de querer agregar un nuevo motor, se deberá implementar únicamente el adaptador que ”adapta”, valga la redundancia, las operaciones expuestas por la API genérica y las del motor propiamente.

Por último desarrollamos un caso de estudio que permite mostrar la ejecución de un proceso de negocio utilizando el portal genérico y su conexión con los tres motores Camunda, Bonita y jBPM, a través del API genérica. De esta forma pudimos visualizar empíricamente que independientemente del motor en el cual estemos ejecutando nuestro modelo, la interfaz es la misma.

7.2 Trabajo a futuro

Ya llegando al final del trabajo, y aclarando que se intentó abarcar las distintas características de cada BPMS en el tiempo estipulado, se cree interesante mencionar algunas cosas que podrían resultar como mejora en el producto final.

Podría ser interesante utilizar mas motores propietarios, de forma de validar que el modelo de datos y API no es sólo común a las soluciones presentadas en este trabajo.

En el prototipo se contemplaron tres motores de ejecución, si bien todo el trabajo se planteó como una continuidad del trabajo [DCA16] y se respetó la interfaz definida en este, sería interesante integrar el adaptador allí implementado. Además, implementar los adaptadores correspondientes a los motores que faltan a la solución actual.

También se pueden abarcar más funcionalidades, no solo respecto a la ejecución sino a la parte de histórico y administración del BPMS. Una funcionalidad interesante a agregar tiene que ver con que actualmente es necesario hacer un deploy desde el menú del BPMS en el

que se desea ejecutar finalmente el modelo y este paso podría evitarse si se extendiera la API genérica para da soporte a esto.

Por otra parte se vieron características particulares en algunos BPMS que se destacaron frente a otros. Por ejemplo, el Dashboard que presenta jBPM parece ser una herramienta muy útil y amigable para con el usuario. Este tipo de herramientas podría resultar de sumo interés agregarla al portal genérico, siguiendo como guía lo que expone jBPM e implementando lo necesario para que funcione en los otros motores.

Relacionado al front-end, una mejora sería incluir validadores y editores para campos de valores de variables, dependiendo del tipo de variable. Actualmente se manejan estos como de tipo String. Además, resultaría interesante desarrollar una versión móvil como dispone Bizagi para seguir la ejecución del modelo, si bien la web es responsivo.

Referencias

[Wes07] M Weske. *Business Process Management: Concepts, Language, Architectures*. Springer, 2007.

[OMG13] OMG. "Business Process Model and Notation". In: (2013). URL: http://www.omg.org/spec/BPMN/2.0.1/PDF.

[DCA16] A Delgado, D Calegari, and A Arrigonni. *Towards a generic BPMS user portal definition for the execution of business processes*. pág 39-59. Electr. Notes Theor. Comput. Sci., 2016.

[Act18] Activiti. "Sitio web oficial". In: (2018). URL: https://www.activiti.org/.

[Bon18] Bonita. "Sitio web oficial". In: (2018). URL: https://www.bonitasoft.com/.

[Biz18] Bizagi. "Sitio web oficial". In: (2018). URL: https://www.bizagi.com/es.

[JBP18a] JBPM. "Sitio web oficial". In: (2018). URL: https://www.jbpm.org/.

[Cam18a] Camunda. "Sitio web oficial". In: (2018). URL: https://camunda.com/.

[Flo18] Flowable. "Sitio web oficial". In: (2018). URL: https://www.flowable.org/.

[Cro] CrossVale. "CrossVale Integrate-Automate-Deliver". In: (). URL: https://crossvale.com/.

[AG] Software AG. "Software AG". In: (). URL: http://www1.softwareag.com/corporate/products/webmethods_process/bpm/capabilities/default.asp.

[OMGa] CMMN OMG. "Case Management Model and Notation". In: (). URL: https://www.omg.org/cmmn/.

[OMGb] DMN OMG. "Decision Model and Notation". In: (). URL: https://www.omg.org/spec/DMN/.

[Red18] Redhat. "Sitio web oficial". In: (2018). URL: https://www.redhat.com/es.

[KR06] B. Axenath Kindler E and V. Rubin. “The Role of Business Processes in Service Oriented Architectures”. In: *A Meta-Model For the Integration of Business Process Modelling Aspects* (2006), pp. 4477–4479.

[OMG00] OMG. “Workflow Management Facility Specification”. In: *Workflow Management Facility Specification* (2000).

[Son97] John Wiley Sons. “Workflow Handbook”. In: *Workflow Reference Model* (1997), pp. 243–293. URL: https://www.wfmc.org/docs/tc003v11.pdf.

[SY14] J. Su Sun Y and J. Yang. “A Key to Business-Process-as-a-Service (BPaaS)”. In: *Separating Execution and Data Management* (2014), pp. 374–382.

[Mue04] M. zur Muehlen. “Foundation, Design and Implementation of Workflow-driven Process Information Systems”. In: *Workflow based Process Controlling* (2004).

[Cam18b] Camunda-Rest. “Servicios Rest Camunda”. In: (2018). URL: https://docs.camunda.org/manual/7.10/reference/rest/.

[JBP18b] JBPM-Rest. “Servicios Rest jBPM”. In: (2018). URL: https://docs.jboss.org/jbpm/release/7.13.0.Final/jbpm-docs/html_single/#_kie.ksrestapi.