
PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

# PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

## FACULTAD DE CIENCIAS E INGENIERÍA


# Análisis, Diseño y Construcción de una Herramienta para Modelado de Procesos: MJS Process Designer

Tesis para optar el Título de Ingeniero Informático

Presentada por:

**Meylin Cinthia Camarena Gil**
**Jackeline Marina Pedreschi Núñez**
**Sandro Salvador Rondón Suasnabar**

**LIMA – PERU**

**2008**

## Resumen

Las organizaciones desarrolladoras de software en general, han comprendido que la clave de la entrega exitosa de un producto radica en la efectiva gestión de su proceso software, ya que existe una correlación directa entre la calidad del proceso y la calidad del producto obtenido a partir de éste [75]. Incluso los organismos gubernamentales han emprendido una serie de proyectos cuyo objetivo es la adecuada documentación de sus procesos, como lo es el caso de la Oficina Nacional de Gobierno Electrónico e Informática en el Perú.

La definición de procesos es uno de los primeros esfuerzos que toda organización debe completar para poder iniciar la mejora de sus procesos, en particular, aquellas que se dedican al desarrollo de software. Es así, que nacen una serie de notaciones y lenguajes de definición de procesos (LDP), los cuales llevan ya varias décadas usándose en numerosos campos de la industria, pero que son recientemente aplicados en el ámbito de la Ingeniería de Software.

Un lenguaje de definición de procesos tiene como objetivo principal definir las entidades básicas que conforman un proceso y las reglas de interacción y operación que puedan existir entre ellas.

Los lenguajes de definición de procesos se apoyan en las herramientas de software para facilitar el trabajo del usuario. Es así, que en la actualidad existen una gran variedad de

herramientas para el modelado de procesos, las cuales pueden hacer uso de lenguajes de definición de procesos formales o propios de la herramienta.

XPDL (XML Process Definition Language) es un lenguaje de definición de procesos propuesto por la WfMC (Workflow Management Coalition). El objetivo de este lenguaje es proporcionar un lenguaje formal que permita la importación y exportación de las definiciones de procesos tanto a nivel gráfico y semántico entre una gran variedad de herramientas que hagan uso del mismo lenguaje.

La popularización de conceptos tales como: explosión de niveles, definición de metodologías, gestión de versiones de procesos, entre otros; ha traído como consecuencia que dichos conceptos sean incluidos en algunas herramientas de administración de procesos que se encuentran disponibles en el mercado. Sin embargo, dichas herramientas incluyen algunos de estos conceptos mediante la implementación de un lenguaje de definición propio, lo cual dificulta su posible interoperabilidad con otras herramientas.

En este trabajo de tesis se presenta el desarrollo de una herramienta software basada en el lenguaje XPDL, que incluye como parte de sus funcionalidades: la definición de procesos, la explosión de actividades, la definición de metodologías y la gestión de versiones de los procesos y metodologías. Para lograr dicho objetivo, se ha desarrollado una extensión al XPDL que permita el manejo de los conceptos de: explosión, metodologías y versionado de procesos, dado que son conceptos que no se encuentran incluidos de forma nativa en el mencionado lenguaje.

Cabe resaltar que el presente proyecto es parte del componente de desarrollo de herramientas que viene realizando el Grupo de Investigación y Desarrollo en Ingeniería de Software y Sistemas de Información de la PUCP como parte del Proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Ibero América).

*A mis padres y hermanos por brindarme siempre su incondicional apoyo y confianza en todas mis decisiones.*

*Meylin Camarena*

*A mis padres Carlos y Ana, por brindarme su aliento y comprensión a lo largo de toda mi vida. A mis abuelos Consuelo y Salvador y a mi tía María Consuelo, por cuidarme cuando era pequeño. A Jackeline por entrar en mi vida y brindarme su apoyo incondicional.*

*Sandro Rondón*

*A Dios y a la Virgen María, por su amor y protección. A mis padres y hermanas, por su amor, comprensión, confianza y sabios consejos. A Sandro, por estar siempre a mi lado brindándome su apoyo y amor incondicional.*

*Jackeline Pedreschi*

*Agradecemos a la Pontifica Universidad Católica del Perú por habernos acogido como alumnos y por la gran formación profesional que nos ha brindado.*

*Al Ing. Abraham Dávila, nuestro asesor, por su orientación, apoyo y dedicación para la realización del presente trabajo de tesis. Del mismo modo, por motivarnos a la elaboración de papers acerca de nuestro trabajo y a la participación en diversos Congresos a nivel nacional e internacional.*

*Y a todos aquellos que nos apoyaron directa o indirectamente en la realización de este trabajo.*

***Reconocimientos:***

*El presente trabajo está enmarcado dentro del proyecto 506AC0287 COMPETISOFT (Mejora de Procesos Para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Ibero América) del programa CYTED (Ciencia y Tecnología para el Desarrollo) y apoyado parcialmente por la Dirección Académica de Investigación y el Departamento de Ingeniería de la Pontificia Universidad Católica del Perú.*

# Índice general

ANEXOS

# Índice de ilustraciones

# Índice de cuadros

# Índice de archivos fuente

# Introducción

En la actualidad la industria de software tienen como uno de sus objetivos la mejora constante de los procesos de desarrollo de software que utilizan en su actividad diaria. Esta mejora se logra mediante la implementación de algún modelo relativo a la calidad de procesos. Esta mejora se hace necesaria debido a que el desarrollo de los sistemas ha alcanzado un alto grado de complejidad acorde con las nuevas necesidades de los usuarios. Adicionalmente, la alta competitividad, eficiencia y velocidad de respuesta que requiere el mercado, obliga a las industrias a implementar ciclos de desarrollo más cortos basados en presupuestos bajos y que además permitan gestionar gran cantidad de datos y realizar actividades complejas.

Como consecuencia de lo anterior, el modelado de los procesos de desarrollo de software ha adquirido vital importancia en las diferentes organizaciones que existen en el mercado. La ingeniería y re-ingeniería de los procesos de software apuntan principalmente a elevar la calidad de los procesos de desarrollo.

Para poder modelar un proceso se requiere de un conjunto de reglas sintácticas y semánticas que definan los elementos y las relaciones que forman parte del flujo de un proceso. Hoy en día existe una gran variedad de Lenguajes de Definición de Procesos que ayudan a cumplir con este objetivo, sin embargo muchos de ellos no definen todos los elementos necesarios que permitan plasmar las distintas variaciones que pueden existir entre un modelo y otro.

A raíz de esto, diferentes organizaciones tales como la WfMC (Workflow Management Coalition), han concentrado sus esfuerzos en la elaboración de lenguajes estándares para la definición de procesos, siendo uno de los más conocidos en el mercado el lenguaje XPDL (XML Process Definition Language). Dicho lenguaje permite definir los elementos comunes utilizados en los diferentes marcos de procesos actuales, y además, brinda la posibilidad de extender dichos elementos en base a las necesidades particulares del usuario.

Sin embargo, un LDP sin una herramienta de software adecuada que la soporte, no constituye una opción viable debido a lo engorroso que resulta su manejo y actualización. Debido a esto la herramienta debe permitir la definición de sus procesos, la administración de las definiciones existentes y un adecuado mecanismo de control de la evolución de sus procesos de forma sencilla para el usuario.

Como una alternativa de solución ante las necesidades expuestas, el siguiente trabajo de tesis presenta el desarrollo de una herramienta de software que permita la definición grafica y semántica de los procesos basada en el lenguaje XPDL versión 2.0. Adicionalmente esta herramienta incluye el manejo de la explosión de actividades en niveles, la definición de metodologías y la gestión de versiones de los procesos. Estos conceptos no se encuentran definidos nativamente en el lenguaje XPDL, por lo que se consideró necesario elaborar una extensión con la finalidad de incluirlos dentro del lenguaje.

En el primer capítulo de este documento, se presenta el marco teórico correspondiente a las definiciones utilizadas en el presente trabajo.

En el segundo capítulo se presenta el análisis de las necesidades actuales que aquejan a las organizaciones respecto a la definición de sus procesos, un breve análisis de los productos actualmente existentes en el mercado y la alternativa de solución propuesta ante estas necesidades.

En el tercer capítulo se presenta el análisis y diseño de la herramienta, que incluye: las tecnologías utilizadas, la arquitectura implementada, la descripción de los módulos considerados en la herramienta y los planes de prueba respectivos.

Asimismo, en el cuarto capítulo este documento presenta la implementación de la herramienta, los estándares utilizados en su codificación y las pruebas de cada uno los componentes.

Finalmente en el quinto capítulo se presentan las observaciones, conclusiones y recomendaciones para un trabajo futuro.

# 1. Marco Teórico

En este primer capítulo, se presenta el marco teórico relacionado con la definición de procesos y los conceptos necesarios para poder comprender el trabajo realizado.

## 1.1. Definición de Procesos

La actividad de cualquier organización, sin importar la naturaleza o el rubro en la que se encuentre enfocada, puede ser concebida como un conjunto de procesos. De este modo, cualquier organización puede ser considerada como un sistema de procesos relacionados directa o indirectamente entre si, en donde la salida (output) de un proceso puede actuar como entrada (input) de otro.

Según la ISO/IEC 12207 [1], un proceso se define como las entradas que se transforman en salidas mediante un conjunto de actividades relacionadas. Adicionalmente, un proceso está conformado por otros elementos, tales como: puntos de inicio y fin de proceso, participantes, aplicaciones, artefactos; así como la información relacionada a cada uno de los elementos.

La complejidad de un proceso dentro de una organización puede variar debido a varios factores, como por ejemplo: importancia del proceso dentro de la organización y envergadura del proceso, entre otros [2]. Debido a esto, un proceso puede estar definido en su expresión más básica, es decir, compuesto solamente por actividades simples; hasta procesos más complejos que incluyan referencias a otros procesos a manera de subprocesos. Esto último con el fin de incluir su flujo de trabajo como parte del flujo principal [3].

Según Curtis, Kellner & Over [4], los cinco objetivos principales en la definición de los procesos, son:

a. Facilitar el entendimiento humano y la comunicación.
b. Soportar los procesos de mejora.
c. Soportar la administración de los procesos.
d. Proveer una guía automatizada en la ejecución de procesos.
e. Proporcionar la ayuda automatizada de la ejecución.

El presente trabajo de tesis se enfoca en la definición de procesos de una organización desarrolladora de software. Un proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo" [5].

El ciclo de vida de un proceso de desarrollo de software en el proceso unificado (RUP) comprende cuatro fases [6]:

a. Concepción: Define el alcance del proyecto en base a las necesidades expresadas por el usuario.
b. Elaboración: Define el plan de proyecto, especificándose las características, diseño y arquitectura del mismo.
c. Construcción: Se elabora el producto en base a la planificación del proyecto y diseño elaborado en la etapa de elaboración
d. Transición: Transfiere el producto a los usuarios finales.

Actualmente las organizaciones desarrolladoras de software han incluido como parte de su estrategia de desarrollo organizacional la evaluación y mejora continua de sus procesos, ya que han comprendido que la calidad del producto obtenido depende de la calidad del proceso [7] lo cual radica en una efectiva gestión de sus procesos de software [8].

### 1.2. Modelado de Procesos

Debido a la gran complejidad de los procesos y subprocesos en una organización, los procesos son frecuentemente difíciles de comprender y administrar. Un modelo de proceso permite organizar y documentar la información del proceso facilitando su compresión. Al modelar un proceso se busca desarrollar una descripción lo más exacta posible del proceso así como de las actividades y demás elementos que lo conforman [9].

Al modelar un proceso mediante una representación gráfica (diagrama de proceso), se puede visualizar las interrelaciones existentes entre las distintas actividades que lo conforman,

posibles puntos de conexión con otros procesos o subprocesos, los roles o participantes encargados de la ejecución de las actividades, entre otros. Del mismo modo, permite identificar posibles problemas existentes así como oportunidades de mejora. Esto conlleva a que la organización pueda automatizar, integrar, monitorizar y optimizar de forma continua los procesos que administra [10].

El modelado de procesos es una actividad importante en donde se representa la estructura y el comportamiento deseado de un sistema permitiendo así identificar con facilidad las interrelaciones existentes entre las actividades, analizar cada uno de los elementos existentes y sus relaciones, identificar oportunidades de simplificación y reutilización o sacar a la luz problemas existentes dando oportunidad al inicio de acciones correctivas [5].

El modelado de procesos permite realizar un mejor análisis de los procesos existentes, en base al cual se puede realizar la descomposición de procesos de trabajo en actividades discretas así como la identificación de las actividades que aportan un valor añadido y las actividades que sirven de soporte a estas. También se puede visualizar qué sucede en cada una de las etapas del proceso, cuándo sucede y porqué.

La complejidad inherente al proceso de software puede ser dominado gracias a una comprensión profunda del proceso en sí mismo y mediante un soporte automatizado del proceso, es decir, no basta con tener disponible un modelo de proceso sino también es necesario contar con las herramientas adecuadas para definirlo, modificarlo y analizarlo.

### 1.2.1. Principios de Modelado

Según Jacobson, entre los principios básicos del modelado de procesos se encuentran los siguientes [5]:

- **La elección del modelo a crear tiene profunda influencia en cómo se ataca el problema y se da forma a la solución elegida**. La elección correcta del modelo representa una gran ayuda a la hora de resolver el problema, por el contrario, la elección errónea del modelo puede derivar en el intento de aplicar una solución que en la práctica es imposible de llevar a cabo.

- **Cada modelo puede ser expresado en diversos niveles de detalle.** El nivel de granularidad con el cual se elaborará un modelo de proceso depende de las necesidades de análisis del usuario.

- **Los mejores modelos son los que reflejan la realidad lo más cercano posible.** Si bien es cierto que al modelar se tiende a simplificar la realidad observada, el modelo

elaborado queda obsoleto si en este proceso de simplificación se obvia o enmascara algún detalle de importancia dentro del proceso.

- **Un modelo independiente no es suficiente para representar una realidad.** La mejor aproximación se logra con la combinación de los resultados de varios modelos independientes relacionados entre si.

### 1.2.2. Lenguajes de Procesos

Un lenguaje de proceso es usado para definir las entidades básicas que conforman un proceso (actividades, participantes, subprocesos, entre otros) y las reglas de interacción y operación que puedan existir entre estas entidades [11]. Un lenguaje de proceso estándar debería permitir a los usuarios utilizar productos de diferentes vendedores para definir e implementar los procesos de su organización sin causar problemas de incompatibilidad cuando dicha definición es interpretada por un producto distinto en el que fue creado [12].

Según Ould, los lenguajes para modelar los procesos se categorizan de acuerdo a los objetivos que persiguen [13], entre los cuales se tiene:

- **Para definirlo:** Permite comunicar la composición de un proceso y las interrelaciones entre sus elementos.

- **Para analizarlo:** Implica un análisis, cuantitativo o cualitativo, de un proceso que permite decidir cambios en la planificación de actividades, en la asignación de tareas a empleados, incremento o reducción del grado de paralelismo, entre otros.

- **Para ejecutarlo:** A partir de una definición de proceso se puede conseguir que un sistema software coordine su ejecución, esto es, gestione automáticamente las tareas necesarias para que los procesos puedan ser llevados a cabo. Los lenguajes de ejecución brindan los elementos necesarios para especificar como se desarrollará el flujo de un proceso durante su ejecución.

## 1.3. Lenguajes de Definición de Procesos

Para poder realizar el modelado con coherencia es necesario contar con un conjunto de reglas de representación y entendimiento, que vienen recogidas en un lenguaje.

Un lenguaje de definición de procesos (LDP) es, en general, un conjunto de reglas sintácticas y semánticas que definen los elementos y las relaciones que son parte de la definición de los procesos. Un LDP puede cubrir desde la parte estática hasta la parte dinámica de los procesos y enfatizar los aspectos para los cuales fue creado [14].

Un lenguaje de definición de procesos se puede clasificar en tres tipos [15]:

- **Formal:** Aquel que cuenta con sintaxis y semántica formales.
- **Semi-formal:** Aquel que cuenta con una notación formal, por lo general gráfica, pero no tiene semántica formal.
- **Informal:** Aquel lenguaje que no cuenta con sintaxis ni semántica formal.

La importancia de contar con un lenguaje para la definición de procesos se hace más notoria cuando ocurre algún cambio en la organización, tales como: cambios tecnológicos, cambios en los procedimientos o cambios en las principales aplicaciones. Un modelo que ha sido desarrollado basado en un LDP suele ser más fácil de adaptarse a los cambios.

En la actualidad existen diversos lenguajes tales como: XPDL [3], BPEL [16], SEPM [18], Little JIL/Juliette [19], y YAWL [17], entre otros. A continuación se realiza una breve descripción de algunos de los lenguajes que existen en el mercado:

- **XPDL (XML Process Definition Language) [3]:** Lenguaje basado en XML para la definición de un flujo de trabajo que puede ser usado para almacenar o intercambiar modelos de procesos de negocio entre distintas herramientas.

- **BPEL (Business Process Execution Language) [16]:** Lenguaje basado en XML diseñado para definir procesos de negocio que interactúan con entidades externas mediante operaciones de un servicio Web.

- **YAWL (Yet Another Workflow Language) [17]:** Es el primer lenguaje basado en patrones de proceso (workflow patterns) que provee un formato simple para el modelado de sistemas cuya interrelación de procesos describe un flujo de control complejo. Este lenguaje puede ser usado tanto por herramientas orientadas a la ejecución de procesos, así como aquellas cuyo propósito es el de definir procesos de forma gráfica gracias a los objetos visuales que posee dicho lenguaje.

- **ALF (AlphaFlow's Process Definition Language) [21]:** Permite importar definiciones de procesos en formato XML. Los nombres de los elementos definidos en este lenguaje están basados en los nombres utilizados en la herramienta AlphaFlow.

- **jDPL (jBOSS Process Definition Language) [20]:** Notación en formato XML que permite, de acuerdo a un esquema XML determinado, empaquetar todos los archivos asociados en una definición de proceso. Este lenguaje está relacionado estrechamente con el plugin jBPM para Eclipse.

- **SPEM (Software Process Engineering Metamodel) [18]:** Es un estándar del OMG [22] cuyo objetivo principal es proporcionar un marco formal para la definición de procesos de desarrollo de sistemas y de software así como para la definición y descripción de todos los elementos que los componen.

Actualmente XPDL y BPEL son los más usados en el mercado [24], sin embargo ambos presentan enfoques distintos, siendo BPEL un lenguaje orientado a los aspectos de la ejecución de un proceso [23] y XPDL es un lenguaje orientado a la definición de procesos de forma gráfica y semántica [23].

### 1.4. XPDL (XML Process Definition Language)

Los conceptos básicos que son la base de XPDL versión1.0 fueron formulados por la WfMC (Workflow Management Coalition) y las compañías que desarrollaban herramientas de administración de procesos de negocio y de workflow. Estos conceptos fueron incorporados en un meta-modelo y detallados en un glosario; los cuales fueron utilizados en la especificación de las diversas interfaces para los conceptos que formaban parte de un sistema de workflow [25].

Una pieza esencial en la administración de procesos es el intercambio de las definiciones de procesos entre diversas herramientas y también entre distintos vendedores. Es así que nace la primera versión de un lenguaje estándar de intercambio denominado WPDL (Workflow Process Definition Language), publicada por la WfMC en 1994 [25].

La creciente popularidad de XML y su uso para definir los formatos de documentos para Internet, combinados con algunos años de la experiencia acumulada usando WPDL en workflow y herramientas de BPM, condujo a la creación de XPDL 1.0, el cual fue lanzado oficialmente en octubre del 2002 [25].

XPDL conservó la semántica utilizada en WPDL pero definió una nueva sintaxis usando un esquema de XML. Sin embargo, ni WPDL ni XPDL 1.0 propusieron una representación gráfica específica para el modelado de procesos a pesar que el metamodelo subyacente para un proceso estaba conformado de actividades (nodos) y caminos conectores entre ellos (transiciones) [25].

BPMN fue desarrollado por BPMI (Business Process Management Initiative) con la finalidad de adoptar las técnicas empleadas en las herramientas de esquematización, así como unificar y extender los gráficos utilizados en ellas para expresar la semántica de los procesos. BPMN 1.0 fue lanzado en mayo de 2004. Además de la notación gráfica, BPMN

incorporó un número de mecanismos específicos para el modelado de procesos tales como: eventos y mensajes entre los mismos [25].

Debido a que el lanzamiento de BPMN fue posterior al de XPDL 1.0, los conceptos de diagramado propuestos por dicha notación gráfica no estaban incluidos dentro del lenguaje XPDL. A raíz de esto, la WfMC define la implementación de XPDL 2.0, incorporando estos mecanismos de diagramado y ofreciendo además un meta-modelo extendido que unifica XPDL y BPMN [25].

En la ilustración 1-1, se muestran los principales elementos gráficos de la notación BPMN y que son soportados por el lenguaje XPDL [26].

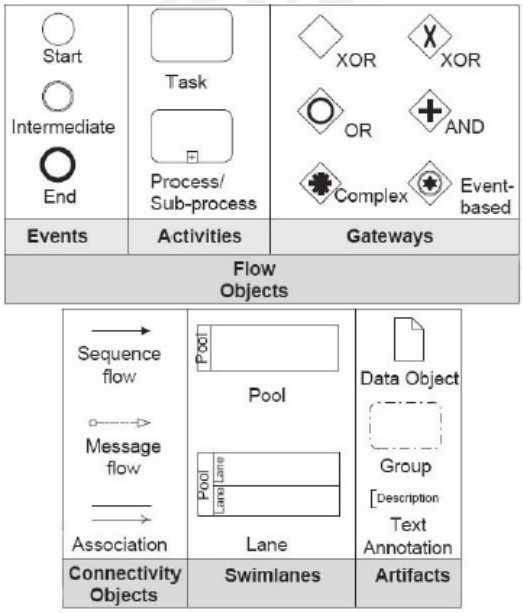


**Ilustración 1-1: Símbolos BPMN [26]**

En conclusión, XPDL es un lenguaje estandarizado basado en XML y que utiliza la notación gráfica BPMN para la definición de un flujo de trabajo. Dicho lenguaje puede ser usado para almacenar o intercambiar modelos de procesos entre distintas herramientas.

La WfMC dentro de su programa de estandarización en el área de procesos, ha identificado cinco interfaces funcionales [3]:

- Definición de procesos e importación/exportación de los mismos.
- Interoperabilidad entre distintos sistemas de workflow.
- Interacción con otros tipos de aplicaciones.
- Interacción con las interfaces de escritorio de los usuarios.
- Sistema para monitorizar los procesos, que proporcionen métricas que faciliten la gestión de los mismos.

El lenguaje XPDL forma parte de la primera interfaz, Definición de Procesos de Negocio, la cual da soporte a la definición y a la importación/exportación de procesos. Esto permite que un modelo de proceso definido en una determinada aplicación pueda ser usado por otras aplicaciones de modelado y/o por otras aplicaciones que se encarguen de la instanciación de los procesos definidos.

En el lenguaje XPDL se puede plasmar los dos aspectos que definen un proceso [3], los cuales son:

### 1.4.1. Modelado Gráfico [3]

XPDL versión 2.0 contiene extensiones capaces de representar todos los aspectos manejados en la notación gráfica BPMN. Esta información gráfica es opcional y dependiente de la herramienta modeladora de procesos que genera el archivo XPDL. Dicha información es representada a través de los elementos NodeGraphicsInfo y ConnectorGraphicsInfo, los cuales pueden aparecer múltiples veces en cada elemento de XPDL dependiendo del número de herramientas modeladoras que hayan agregado información gráfica al archivo XPDL.

a. **NodeGraphicsInfo**

Este elemento puede ser usado por las siguientes entidades del lenguaje XPDL:

- Artifact
- Pool
- Lane
- Activity

En base a la notación gráfica BPMN, la cual puede ser representada a través del lenguaje XPDL, una actividad puede tener diversas representaciones gráficas, siendo la forma más básica la actividad de tipo Task (ilustración 1-2).

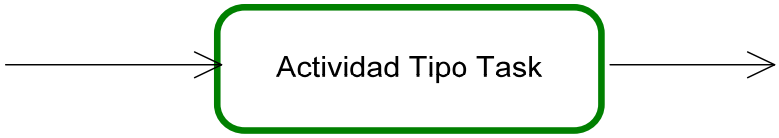


**Ilustración 1-2: Task BPMN [3]**

Cada herramienta es identificada a través del atributo ToolId, permitiendo así que cada una de estas herramientas pueda agregar su propia información gráfica. Por lo tanto, cada herramienta puede exhibir y representar el mismo archivo XPDL de diversas maneras, ya

que cada una utiliza su propia información gráfica, pero conserva la información gráfica de las otras herramientas.

En el cuadro 1-1, se listan los atributos del elemento NodeGraphicsInfo con su respectiva descripción por cada uno de ellos.

| Atributo | Descripción |
|---|---|
| **BorderColor** | Representa el color del borde del elemento gráfico y es representado a través de una cadena de texto. |
| **Coordinates** | Representa las coordenadas de X e Y de la esquina izquierda superior del elemento. |
| **FillColor** | Define el color del elemento y es expresado a través de una cadena de texto. |
| **Height** | Almacena la altura del elemento. |
| **LaneId** | En el caso que el elemento sea del tipo Lane, este atributo se usa para almacenar su identificador. |
| **ToolId** | Representa al identificador de la herramienta, el cual podría corresponder al nombre de la herramienta que genera el archivo XPDL. |
| **IsVisible** | Este atributo posee dos valores: true y false. Si tiene el valor true, indica que el nodo debe mostrarse gráficamente. |
| **Page** | Representa el nombre de la página en la cual el nodo debería ser mostrado. |
| **Shape** | Almacena la forma del elemento gráfico representado. |
| **Width** | Guarda información sobre el ancho del elemento. |

**Cuadro 1-1: NodeGraphicsInfo [3]**

Haciendo uso del elemento NodeGraphicsInfo se puede almacenar dicha información tal como se muestra en el archivo fuente 1-1:

```
<NodeGraphicsInfos>
   <NodeGraphicsInfo  Width="75.0" Height="50.0" BorderColor="#3B8C3D"
        FillColor="#FFFFFF" ToolId=”MJSProcess”  Shape=”Rectangulo Redondeado”>
        <Coordinates XCoordinate="128.0" YCoordinate="96.0"/>
   </NodeGraphicsInfo>
</NodeGraphicsInfos>
```

**Archivo Fuente 1-1: NodeGraphicsInfo XPDL**

b. **ConnectorGraphicsInfo**

Este elemento puede ser usado para representar la conexión gráfica entre los elementos que pertenecen a un proceso. Las entidades del lenguaje XPDL que poseen este tipo de elemento son:

- Transition
- Association
- Message Flow

En la ilustración 1-3 se visualiza una conexión gráfica entre la actividad 1 y la actividad 2, lo que en BPMN se conoce como Flujo de Secuencia.


**Ilustración 1-3: Transición BPMN [3]**

En el cuadro 1-2, se listan los atributos del elemento ConnectorGraphicsInfo con su respectiva descripción por cada uno de ellos.

| Atributo | Descripción |
|---|---|
| **BorderColor** | Representa el color del borde del elemento gráfico y es representado a través de una cadena de texto. |
| **Coordinates** | Representa las coordenadas de X e Y de la esquina izquierda superior del elemento. |
| **FillColor** | Define el color del elemento y es expresado a través de una cadena de texto. |
| **ToolId** | Es el identificador de la herramienta y el cual podría corresponder al nombre de la herramienta que genera el archivo XPDL |
| **IsVisible** | Este atributo posee dos valores: true y false. Si es true, indica que el nodo debe mostrarse gráficamente. |
| **Page** | Representa el nombre de la página en la cual el nodo debería ser mostrado. |
| **Shape** | Almacena la forma del elemento gráfico representado. |
| **Style** | Representa el estilo en que es representada la conexión. |

**Cuadro 1-2: ConnectorGraphicsInfo [3]**

Haciendo uso del elemento ConnectorGraphicsInfo se puede almacenar dicha conexión gráfica tal como se presenta en el archivo fuente 1-2:

```
<ConnectorGraphicsInfos>
  <ConnectorGraphicsInfo BorderColor="#193EA1" FillColor="#193EA1"
    ToolId=”MJSProcess” Style=”Continua”>
     <Coordinates XCoordinate="203.5" YCoordinate="122.82308197021484"/>
     <Coordinates XCoordinate="228.5" YCoordinate="123.2227783203125"/>
  </ConnectorGraphicsInfo>
<ConnectorGraphicsInfos>
```

**Archivo Fuente 1-2: ConnectorGraphicsInfo XPDL**

### 1.4.2. Modelado Semántico [3]

XPDL versión 2.0 ofrece un meta-modelo que describe las entidades de nivel superior contenidas en una definición de proceso, sus relaciones y atributos (incluyendo algunos que pueden ser definidos para fines de simulación o monitoreo). Asimismo, define una serie de convenciones para la agrupación de definiciones de procesos y el uso en común de entidades usadas por uno o más procesos.

Cada una de las entidades definidas dentro del meta-modelo posee una serie de atributos que pueden ser de carácter: obligatorio, opcional o extendido. Estos últimos son los que permiten al usuario añadir características adicionales a dichas entidades.

Un modelo de procesos incluye varias entidades cuyo rango de acción puede ir más allá del ámbito de un único proceso. En particular, las definiciones de participantes, aplicaciones y los datos relevantes pueden ser referenciadas por un gran número de procesos.

El meta modelo que ofrece el lenguaje XPDL propone el uso de un repositorio en común que contenga estas entidades que son usadas por más de una definición de proceso. Para poder llevar a cabo este objetivo y lograr una eficiente transferencia de datos entre el repositorio de datos y las definiciones de procesos es que se introduce el concepto de Paquete. Dicho elemento actúa como un contenedor de entidades comunes entre diferentes definiciones de procesos con la finalidad de evitar su redefinición de forma individual dentro de cada modelo.

Un paquete contiene además de entidades, los atributos en común de cada una de las definiciones de proceso que agrupa. Cada definición de proceso contenida en el paquete puede hacer uso de cualquier cualidad común del mismo.

A nivel de paquete se pueden definir las siguientes entidades: aplicaciones, participantes, tipos de datos, variables, artefactos y atributos extendidos, tal y como se aprecia en la ilustración 1-4.

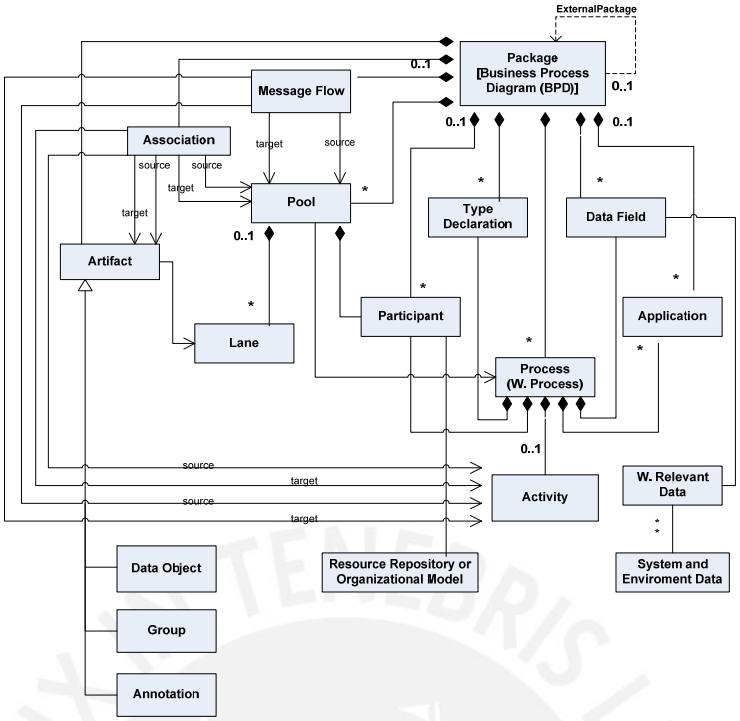


**Ilustración 1-4: Meta-Modelo del Paquete [3]**

Por otro lado, a nivel de proceso se pueden definir las siguientes entidades: Actividades, transiciones, participantes, parámetros formales, variables, aplicaciones, bloque de actividades, conectores, asociaciones, sub-procesos y atributos extendidos, tal y como se muestra en la ilustración 1-5.

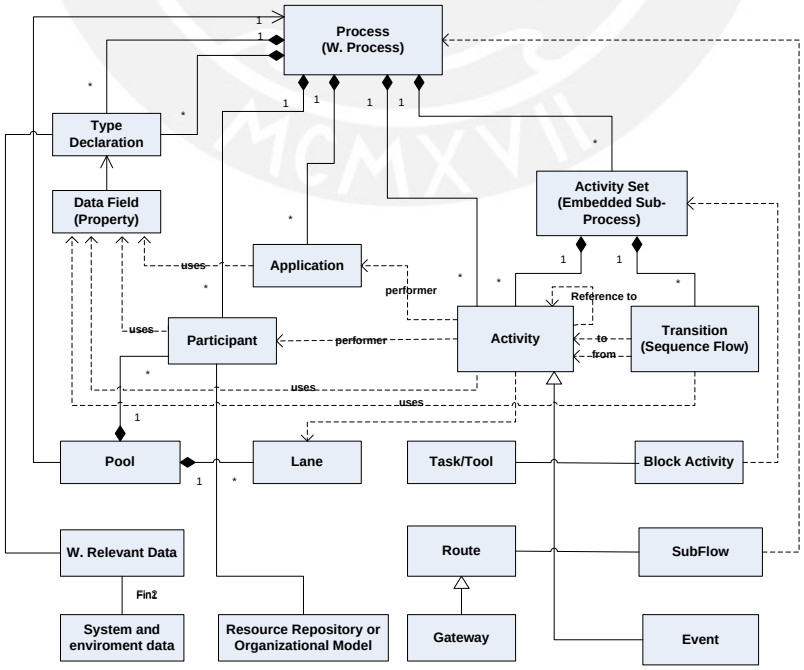


**Ilustración 1-5: Meta-Modelo del Proceso [3]**

#### 1.4.2.1 Definición de las principales entidades del metamodelo – XPDL

A continuación se realiza una breve descripción de las principales entidades que ofrece el Meta-Modelo del lenguaje XPDL [3]:

**a. Actividad (Activity):**

Una definición de proceso consiste en una o más actividades donde cada una de ellas constituye una unidad lógica y autónoma de trabajo dentro del proceso. Una actividad representa el trabajo que será realizado por una combinación de recursos (especificado por la asignación del participante) y/o de las aplicaciones informáticas (especificadas por la aplicación asignada).

Una actividad puede ser de diversos tipos:

- **Subproceso (SubFlow):** En este caso la actividad vendría a ser un contenedor para la ejecución de una definición de proceso diferente a la que pertenece. El proceso identificado dentro del sub-proceso contiene su propia definición de actividades, transiciones, aplicaciones y recursos.

- **Bloque de Actividad (Block Activity):** La cual está conformada por un conjunto de actividades unidas a través de transiciones.

- **Conector (Route Activity):** Este tipo de actividad no realiza ningún trabajo de procesamiento y por lo tanto no tiene ningún recurso y/o aplicación asignado. La función que cumple es apoyar en las decisiones de enrutamiento entre las transiciones de entrada y/o salida que conectan a más de una actividad.

Adicionalmente una actividad puede representar un **evento,** que viene a ser un suceso que ocurre durante el curso de un proceso y que puede afectar el flujo del mismo.

**b. Transición (Transition)**

La relación entre dos actividades se establece mediante el elemento transición. Un conjunto de actividades y las relaciones existentes entre ellas conforman un flujo de control. Cada transición de forma individual tiene tres propiedades básicas que guardan la siguiente información: actividad origen desde donde parte la transición, actividad destino de la transición y la condición a la cual está asociada en caso exista. Su alcance de operación es local a la definición de proceso que contiene las actividades asociadas.

Una transición puede ser del tipo condicional o incondicional. Para el primer caso dicha condición está relacionada a la evaluación de una o más expresiones para habilitar o deshabilitar su flujo.

Las transiciones dentro de un proceso pueden dar lugar a una ejecución secuencial o paralela de actividades individuales. La información requerida para relacionar condiciones de unificación (join) o separación (split) se define dentro de la actividad apropiada.

Aquellas transiciones más complejas que no pueden ser expresadas usando la transición elemental y las funciones de split y/o join, se forman usando las actividades de tipo route. Este tipo de actividad puede ser especificada como paso intermedio entre las actividades reales permitiendo combinaciones adicionales de la función split y/o join.

**c. Participante (Participant):**

Esta entidad almacena información del ente que actúa como el ejecutor de una o más actividades en la definición de un proceso. Los recursos que pueden ser asignados para realizar una actividad específica se definen como un atributo de la actividad denominado Participant Assigment, el cual liga la actividad al sistema de recursos disponibles que pueden ser asignados.

La definición de un participante no se refiere necesariamente a un ser humano o a una sola persona, sino que también puede ser identificado por un conjunto de personas o al uso de sistemas autómatas capaces de desempeñar una actividad requerida.

**d. Aplicación (Application):**

Contiene la información de las aplicaciones o servicios que se pueden invocar para apoyar o automatizar la tarea asociada a cada actividad. Tales aplicaciones pueden ser herramientas construidas, áreas departamentales de una empresa o procedimientos puestos en ejecución.

**e. Data Relevante (Data Field):**

Hace referencia a los datos que se crean y se utilizan dentro de cada proceso durante la ejecución del mismo. Estos datos pueden ser referenciados por las actividades o las aplicaciones.

Estas entidades se pueden utilizar para pasar información persistente o resultados intermedios entre actividades y/o para la evaluación de expresiones condicionales en el caso de las transiciones o en la asignación del participante. XPDL incluye la definición de varios tipos de datos que pueden ser básicos o complejos, tales como: string, integer, date, array, entre otros. Cada variable referencia a un tipo de dato en particular.

**f. Entradas / Salidas (Input Sets / Output Sets):**

Definen los requerimientos de información de entrada y salida de una actividad. Por lo general una entrada o una salida puede ser un artefacto del tipo documento.

A pesar de que XPDL cubre la mayoría de elementos estándar necesarios para la definición de un proceso, después de realizar un análisis de las características de este lenguaje, se ha encontrado que dicho lenguaje no contempla aspectos tales como:

a. No posee una lista estandarizada de valores posibles para asignar a los atributos que conforman las estructuras gráficas de una definición de proceso, por ejemplo: una lista de colores. De esta forma al importar una definición en una aplicación distinta a la que fue generada, podrían no plasmarse adecuadamente las características gráficas definidas en el modelo original.

b. No incorpora de manera nativa elementos necesarios para definir los siguientes conceptos: explosión de actividades en niveles, definición de metodologías y gestión de versiones.

El presente proyecto de tesis tiene como uno de sus objetivos desarrollar una extensión del lenguaje XPDL con la finalidad de poder incluir los elementos necesarios que permitan acoplar los conceptos mencionados en el punto b.

### 1.5. Tecnologías Java Web

Uno de los aspectos más importantes de la plataforma Java es el acelerado ritmo con el que evolucionan las tecnologías y arquitecturas diseñadas para soportarla. Constantemente aparecen nuevas librerías, herramientas, frameworks, entre otros. Esto dificulta la elección de las tecnologías con las cuales se va a trabajar debido a que muchas de ellas cumplen una misma función. Adicionalmente, se debe buscar que las tecnologías seleccionadas interactúen de tal forma que se pueda aprovechar toda la flexibilidad y potencia que ofrecen.

A continuación se hace un breve resumen de algunos de los patrones, frameworks y tecnologías que pueden ser utilizados en la creación de una aplicación Java Web.

#### 1.5.1. Patrones de Software

Un patrón define una solución aplicada con éxito para un mismo problema dentro de un contexto dado, de tal modo que se puede reutilizar esta solución más adelante sin tener que volver a pensarla otra vez [27]. Se denominan patrones de software a los patrones que son aplicados durante el desarrollo de un sistema de software o aplicación [28].

La utilización de patrones de software en el desarrollo de una aplicación brinda las siguientes ventajas: Permite el ahorro de tiempo al programador, quien enfocará sus esfuerzos en el

desarrollo de la lógica de la aplicación; brinda una arquitectura uniforme a la aplicación, facilitando así su mantenimiento, modificación y expansión, entre otros [29].

Un patrón presenta una estrategia definida en busca de solucionar un problema específico, debido a esto es posible la utilización de uno o más patrones durante el desarrollo de una aplicación [30].

Uno de los patrones de diseño que en la actualidad es la opción más aceptable y recomendable en el desarrollo de aplicaciones Web es el patrón MVC (Modelo Vista Controlador) [31].

**Patrón Modelo Vista Controlador (MVC) [32]:**

El patrón MVC es un patrón de arquitectura de software que separa en tres componentes distintos los siguientes elementos: los datos de una aplicación, la interfaz de usuario y la lógica de control [33]. En la ilustración 1-6 se muestra el esquema de funcionamiento que define dicho patrón en base a las tres capas anteriormente mencionadas.

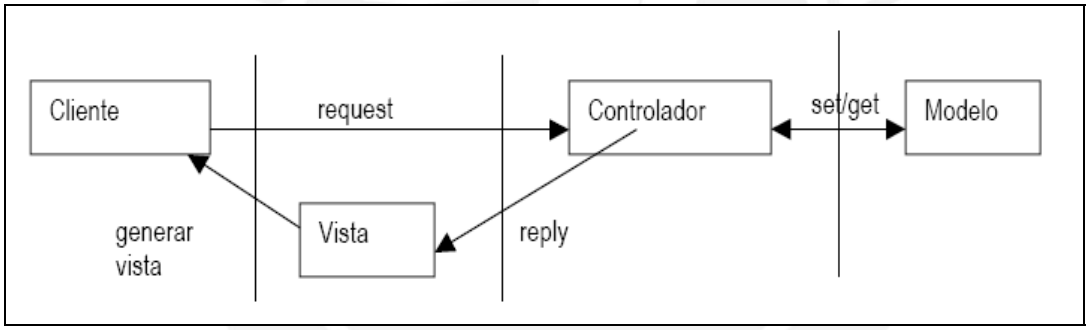


**Ilustración 1-6: Esquema del patrón MVC [32]**

- **Capa Vista:** Se encarga de procesar la información que recibe del controlador y presentarla al usuario en un formato adecuado. Para las aplicaciones desarrolladas en plataforma Web el formato elegido es HTML. En esta capa solamente se deben realizar operaciones simples tales como condicionales, bucles, formatos, entre otros.

- **Capa Modelo:** Es la encargada de guardar los datos en un medio persistente como son las bases de datos, archivos XML, entre otros. Así mismo, en ella se realizan las operaciones propias de la lógica del negocio.

- **Capa Controlador:** Es la capa intermedia que escucha los cambios realizados en la capa Vista y se los envía a la capa Modelo. Esta última se encarga del procesamiento de datos, los cuales son devueltos a la capa Vista a través del Controlador. Todo este proceso forma un ciclo que se repite cada vez que el usuario realiza una acción en la interfaz. La

capa controladora es la que determina qué vista se debe de presentar y qué información es la que se envía.

La arquitectura MVC fue diseñada para que los cambios realizados en una aplicación afecten lo menos posible a la programación que ya se encuentra implementada. Esto es posible gracias a que su arquitectura desacopla en diferentes capas los datos, la lógica del negocio y la lógica de presentación. Todo esto hace posible la actualización y desarrollo de cada uno de los componentes de forma independiente [32].

### 1.5.2. Marco de Trabajo

Un marco de trabajo (framework) brinda ayuda en las distintas áreas del proceso de desarrollo de una aplicación [34]. Los frameworks están diseñados para permitir el desarrollo fácil y rápido de las aplicaciones. Gracias a esto, los programadores ya no tendrán que preocuparse por manejar programación de bajo nivel, como por ejemplo: concurrencia masiva, manejo de transacciones, seguridad, entre otros.

Adicionalmente, los frameworks ayudan a que las aplicaciones cuenten finalmente con características superiores debido a que un framework forma parte de la aplicación cuando ésta ya se encuentra en producción [35].

Actualmente existen disponibles un gran número de frameworks para implementar aplicaciones Java Web, sin embargo no existe un framework que brinde las funcionalidades necesarias para ser utilizado durante todo el proceso de desarrollo de software. Esto se debe a que por lo general un framework está orientado a una tarea específica de dicho proceso.

A continuación se hace un repaso de los tipos de frameworks existentes en el mercado:

**A. Frameworks de Presentación:**

Los frameworks de presentación son los responsables de recolectar la información generada por parte del usuario y llevarla hacia la capa de negocio en donde será procesada. Del mismo modo, recoge la data procesada que proviene de la capa de negocio para presentarla al usuario a través de la interfaz utilizada por la aplicación [36].

Entre los principales frameworks de presentación se encuentran:

**a. Struts**

Struts es un framework de código abierto desarrollado bajo el proyecto Apache Jakarta. Fue creado para facilitar el desarrollo de aplicaciones Web basadas en Java Servlets y Java Serves Pages (JSP) [37].

Características principales [38]:

- Posee una arquitectura basada en el patrón MVC.
- Provee un conjunto de etiquetas JSP personalizadas que simplifican el proceso de la creación de los JSP.
- Permite la integración con el framework Tiles para el control de distribución de los componentes que soporta la creación de plantillas que pueden ser reutilizadas. Esto permite que se pueda modificar el aspecto de las aplicaciones de manera sencilla.
- Permite la integración con el framework Validator para la validación de los datos que son ingresados en los formularios. Dichas validaciones son definidas externamente al código fuente en un archivo XML.

**b. Spring**

Spring es un framework de aplicaciones Java/J2EE que fue desarrollado bajo los términos de la licencia Apache en su versión 2.0 [39].

Características principales:

- Está basado en una configuración de JavaBeans bastante simple.
- Facilita la gestión del ciclo de vida de los componentes.
- Hace uso del patrón de Inversión de Control (IoC).
- Ofrece diversas plantillas que facilitan el uso de Hibernate, iBatis, JDBC, entre otros; adicionalmente se integra "de fábrica" con Quartz, Velocity, Freemarker, Struts y Webwork2.
- Posee un plugin para ser utilizado en el entorno de programación Eclipse.
- Ofrece un ligero contenedor de beans para los objetos de la capa de negocio y persistencia de datos, repositorio de DataSources JDBC y sesiones Hibernate.
- Define el contexto de la aplicación mediante el uso de archivos XML, convirtiéndose en una potente herramienta para manejar objetos Singleton o “factorias” que necesitan su propia configuración.

**c. Java Server Faces (JSF)**

Java Server Faces es el estándar de la especificación J2EE 5 presentado por Sun para la capa de presentación Web [40].

Características principales [41]:

- Posee una arquitectura basada en el patrón MVC.
- Nace como una evolución natural de los frameworks actuales hacia un sistema de componentes. Su arquitectura de componentes define un método común para la construcción de componentes gráficos simples como: botones, cajas de textos, checkbox, entre otros; hasta la construcción de componentes más complejos como: menús, pestañas, árboles, entre otros.

- Permite crear componentes gráficos personalizados según las necesidades del usuario.
- Permite mostrar los componentes gráficos en distintas formas, lo cual va a depender del cliente Web (exploradores, celulares, PDA, etc.) que se esté utilizando para ver la aplicación.

**B. Frameworks de Persistencia**

El lenguaje de programación Java está diseñado para el desarrollo de aplicaciones siguiendo el paradigma de la programación orientada a objetos. Sin embargo, dado que la gran mayoría de sistemas administradores de base de datos tienen un modelo relacional, se hace necesario elaborar un mapeo entre los objetos que se manejan en Java y las tablas que pertenecen a este tipo de sistemas de bases de datos [42]. Este mapeo no es del todo simple, pues se presentan algunas dificultades tales como [43]:

- Diferencias en la estructura de datos utilizada, mientras que los objetos de programación poseen una estructura jerárquica, las tablas de base de datos poseen una estructura tabular.
- En la programación orientada a objetos se manejan conceptos propios tales como herencia, polimorfismo y encapsulamiento; los cuales no se encuentran contemplados de la misma forma en un modelo de base de datos relacional.

Debido a estas dificultades, es que surgen diversos frameworks de persistencia que sirven de interfaces entre los objetos de la aplicación y las tablas de los sistemas de base de datos relacionales, con el fin de poder ayudar al programador a realizar las tareas de persistencia de datos de una forma menos engorrosa.

Entre los frameworks de persistencia open source más conocidos en el mercado se encuentran:

**a. JDBC (Java Data Base Connectivity)**

JDBC es una interfaz de programación Java que proporciona métodos para el manejo de operaciones sobre base de datos haciendo uso del lenguaje SQL.

La tarea principal de JDBC es ocultar las características específicas de cada base de datos y preocuparse solamente por las tareas de consulta, inserción y/o actualización de los datos [44].

Gracias al conjunto de interfaces Java y a los métodos de gestión de conexiones que brinda JDBC para los diferentes modelos de base de datos, el programador ya no tiene que preocuparse por implementar dichas clases de conexión; sino que sólo se enfoca en la elaboración de las sentencias adecuadas para el manejo de la información [45].

**b. Hibernate**

Hibernate es un framework para la capa de persistencia de tipo objeto/relacional y un generador de sentencias SQL [46]. Nace en el año 2001 con el fin de ofrecer una solución completa para la persistencia en aplicaciones Java. Su objetivo es "liberar al desarrollador del 95% de las tareas comunes relacionadas a la persistencia de datos" [47].

Características principales [48]:

- Brinda un marco de trabajo que evita la necesidad de utilizar la interfaz JDBC.
- Soporta la mayoría de los sistemas de base de datos SQL como son: MySQL, PostgreSQL, Oracle, DB2, entre otros.
- Tiene un lenguaje de consulta propio: HQL (Hibernate Query Language), el cual está diseñado como una extensión mínima del SQL. Este lenguaje HQL está orientado a objetos, lo que proporciona un puente para el mapeo de objetos Java y los objetos de la base de datos. A este concepto se le denomina ORM (Object Relational Mapper).
- Ofrece un soporte robusto para transacciones complejas tales como relaciones de uno a muchos y de muchos a muchos. Además, Hibernate se encarga de generar automáticamente complejos joins para optimizar la consulta de los datos.
- Ofrece una amplia y variada documentación, además de contar con una comunidad de seguidores muy activa.

**c. iBatis**

iBatis es un framework de persistencia desarrollado por Apache Software Foundation en el año 2001.

La finalidad de este framework es facilitar la implementación del acceso a base de datos para aplicaciones Java. De esta forma el desarrollador se preocupa específicamente de la manipulación de las sentencias SQL requeridas para su aplicación [49].

iBatis está formado por dos componentes: la capa DAO y los SQLMaps. La primera se encarga de realizar las conexiones con la base de datos, mientras que la segunda ayuda a separar las llamadas a la fuente de datos del código propio de la aplicación. Esto se debe a que dichas sentencias SQL se encuentran mapeadas en un archivo XML externo [44].

Características principales [49]:

- Permite elaborar sentencias dinámicas gracias a la flexibilidad que provee la estructura basada en archivos XML que maneja.
- iBatis no es un ORM.
- No es independiente del proveedor. Si se cambia de base de datos entonces será necesario hacer el cambio del archivo de mapeo.
- Puede ser implementado en Java, .Net y Ruby on Rails.

### 1.5.3. Parser Manejadores de Archivos XML

Un parser es una herramienta de administración de archivos XML que en base a la especificación de la estructura y las reglas de conformación de un archivo XML, implementa operaciones de lectura, escritura y creación de archivos en dicho formato [50].

Entre los XML Parser de naturaleza open source más conocidos tenemos:

**A. JDOM**

JDOM es un API (Application Programming Interface) de procesamiento de documentos XML basado y optimizado en base al lenguaje Java, por lo cual utiliza todas las características y ventajas que dicho lenguaje proporciona [51].

Características principales [52]:

- Permite la creación, manipulación y serialización de documentos XML.
- No incluye un parser propio, pero puede utilizar alguno existente.
- Brinda integración con el lenguaje Java.
- Facilidad de uso.
- Manejo de documentos como árboles, lo cual permite el acceso aleatorio a cualquier parte del documento.

**B. SAX**

SAX es el acrónimo de “Simple API for XML”. Como su nombre lo indica es una API basada en eventos correspondientes a las diversas características encontradas en el documento XML [53].

Características principales [52]:

- Permite la creación y manipulación de documentos XML.
- No incluye un parser propio, pero puede utilizar alguno existente que soporte SAX v2.
- Define métodos para obtener y agregar valores a los diferentes atributos por medio de un XML Reader.

**C. XMLBeans**

XMLBeans define la compilación del esquema (*.xsd) asociado al archivo XML, generando las clases e interfaces necesarias para el acceso y modificación de los datos propios de dicho esquema [53].

Características principales [54]:

- Provee una vista del documento XML basado en objetos Java, creados en base a la estructura nativa del XML.
- Permite trabajar con el documento XML en su totalidad al trabajar íntegramente en memoria.

- Permite el acceso a los elementos del documento XML a través de métodos get y set, los cuales trabajan con tipos de datos propios de su esquema y generados al momento de su compilación.
- Soporta todas las definiciones del esquema de XML.
- Brinda acceso rápido al documento XML.

### 1.5.4. Tecnologías de Presentación

Las tecnologías de presentación permiten elaborar aplicaciones más interactivas para el usuario a fin de que éste se encuentre familiarizado con el sistema.

Entre las tecnologías de presentación de naturaleza open source más conocidas en el mercado se tiene:

**A. JSTL**

La librería JSTL es un componente dentro de la especificación Java 2 Enterprise Edition (J2EE) y es controlada por Sun MicroSystems [55].

Características principales [56]:

- Posee un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal utilizada comúnmente para escribir páginas JSP.
- JSTL se integra de manera limpia y uniforme a las etiquetas HTML, debido a que las etiquetas que utiliza están definidas en formato XML.
- Las etiquetas JSTL pueden referenciar objetos que se encuentren en los ambientes Request y Session sin conocer el tipo del objeto y sin necesidad de realizar una operación de cast.

**B. AJAX**

Acrónimo de “Asynchronous JavaScript and XML”. Es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (Rich Internet Applications) [57]. Estas aplicaciones se ejecutan en el cliente, es decir en el navegador del usuario, manteniendo comunicación asíncrona con el servidor en un segundo plano. De esta forma es posible realizar cambios sobre una página Web cargada en el navegador sin necesidad de recargarla en su totalidad. Esto permite aumentar la interactividad, velocidad y usabilidad del sistema [58].

### 1.5.5. Servidores de Aplicaciones Web y Servidores Web

En informática se denomina servidor de aplicaciones a un equipo que forma parte de una red de computadores, en el cual se ejecuta una o más aplicaciones para ser utilizadas por los clientes definidos dentro de la red. Como consecuencia del éxito del lenguaje de

programación Java, el término servidor de aplicaciones usualmente hace referencia a un servidor de aplicaciones J2EE [59].

Un servidor Web es el responsable de aceptar las peticiones de tipo HTTP provenientes de un cliente (browser) y enviarles una respuesta adecuada a través del mismo protocolo. Generalmente este tipo de respuestas se manifiestan a través de páginas Web [60].

Entre los servidores de aplicaciones y servidores Web de naturaleza open source más conocidos se encuentran:

**A. JBoss**

JBoss AS es el primer servidor de aplicaciones de código abierto certificado para J2EE 1.4 disponible en el mercado. JBoss cuenta con la capacidad de funcionar en un ambiente de producción ofreciendo una plataforma de alto rendimiento para aplicaciones de e-business.

Haciendo uso de una arquitectura orientada a servicios y con una licencia de código abierto, JBoss AS puede ser descargado, utilizado, incrustado y distribuido sin restricción alguna. Por este motivo, JBoss AS es la plataforma más popular de middleware para desarrolladores, vendedores independientes de software y grandes empresas [61].

**B. Apache Tomcat**

Tomcat es la implementación de referencia para las Java Server Pages (JSP) y las especificaciones Java Servlet. Esto significa que es el servidor Java disponible que más se ajusta a los estándares establecidos para la distribución de aplicaciones basadas en JSP.

En sus inicios existió la percepción de que el uso de Apache Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. En la actualidad ya no existe dicha percepción y Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad [62].

### 1.5.6. Sistemas Administradores de Base de Datos

Un Sistema Administrador de Base de Datos es un software que controla la administración, almacenamiento y organización de la información de una base de datos. Permite la creación de bases de datos, realizar consultas sobre las mismas, almacenar grandes volúmenes de información y controlar el acceso a ellos [63].

Entre los sistemas administradores de base de datos open source más conocidos se tiene:

### A. PostgreSQL

PostgreSQL es un motor de base de datos de software libre que proporciona un manejador de conexión nativa JDBC [64].

Características principales [65]:

- Implementa un motor que asegura la integridad de los datos.
- Está diseñado para soportar grandes volúmenes de datos.
- Es soportado por varios sistemas operativos como Windows y Linux, entre otros.
- Brinda la posibilidad de realizar transacciones, sub-selecciones, triggers, vistas y bloqueos sofisticados.
- Ofrece integridad referencial de claves externas.

### B. MySQL

MySQL es un servidor de bases de datos relacional que se distribuye bajo la licencia GPL (General Public License), es decir, es libre para aquellas aplicaciones que se distribuyen con el mismo tipo de licencia [66].

Características principales [67]:

- Es multiplataforma.
- Soporta el acceso a las bases de datos de forma simultánea por varios usuarios y/o aplicaciones.
- Soporta las características más comunes de otros DMBS como: procedimientos almacenados, triggers, vistas, transacciones, entre otros.

# 2. Análisis de la Situación Actual

En este segundo capítulo se describe la situación actual de las organizaciones frente a las necesidades de modelar sus procesos y la importancia de contar con una herramienta que les permita elaborar dichas definiciones. Además, se mencionan una serie de herramientas para la definición de procesos, las cuales fueron evaluadas con la finalidad de enumerar las principales ventajas y carencias que presentan en base a las posibles necesidades de las organizaciones actuales. Como punto final, se menciona las principales características que contendrá la herramienta propuesta.

## 2.1. Situación Actual

Las organizaciones realizan sus actividades a través de un conjunto de procesos de diversa naturaleza, como por ejemplo: procesos administrativos, procesos de fabricación, procesos para la ejecución de proyectos, entre otros. Lamentablemente, en la actualidad las empresas medianas y pequeñas no suelen ser conscientes de la necesidad de tener claramente identificados sus procesos; en particular, las empresas desarrolladoras de software.

En este contexto, muchas agrupaciones mundiales han centrado sus esfuerzos en tratar de desarrollar especificaciones estandarizadas que permitan a las organizaciones la adecuada definición de todos los elementos y el flujo de trabajo que siguen sus procesos. Para ello, se

utilizan diversas herramientas software que permiten a las organizaciones hacer este trabajo mucho más llevadero.

Las principales necesidades que se busca cubrir con una herramienta de administración de procesos son:

- Contar con una herramienta que permita definir y administrar los procesos de forma fácil y entendible para el usuario.
- Manejar un lenguaje estándar que permita que las definiciones de procesos elaboradas en la herramienta puedan ser interpretadas por otras herramientas afines.
- Gestionar las versiones de los cambios que se dan a través del tiempo en las definiciones de procesos.
- Definir metodologías que adapten los procesos existentes a las diferentes realidades que se puedan dar en una organización.

## 2.2. Evaluación de herramientas existentes para la gestión de procesos

En la actualidad existen diversas herramientas que permiten modelar y/o definir los procesos de una organización utilizando diversas estrategias y metodologías. Una de las principales características es el grado de interoperabilidad e interacción planificada con otras herramientas afines.

| Nombre de la Herramienta | Compañía Desarrolladora | Lenguaje de Proceso | Plataforma | Versión | Tipo de Licencia |
|---|---|---|---|---|---|
| Together Workflow Editor | http://www.together.at/together/prod/twe/index.html | XPDL | Stand Alone | 2.2-1 | Evaluación |
| JOpera for Eclipse | http://www.jopera.ethz.ch/ | Propio | Stand Alone | 2.1.2 | Libre |
| ObjectWeb Bonita | http://bonita.objectweb.org/ | XPDL | Web | 3.0 | GNU (LGPL) |
| WorkflowGen | http://www.workflowgen.com/workflow/workflow_software_features.htm | XPDL | Web | -- | Comercial |
| BPWin | http://www.bpwin.ru | Propio | Stand Alone | 4.0 | Comercial |
| XFLOW Process Management System | http://xflow.sourceforge.net/ | Propio | Web | -- | Libre |

**Cuadro 2-1: Herramientas que existen en el mercado**

Mientras algunas herramientas se adaptan generalmente hacia diversos ámbitos de uso y diferentes exigencias del consumidor creando para ello sus propios lenguajes de definición, otras tratan de estandarizar dichas definiciones a través del uso de un lenguaje de definición formal que les permita potenciar su interoperabilidad con otros productos.

Para el presente trabajo se realizó la evaluación de una muestra de las herramientas para el modelado de procesos que existen en el mercado. Cabe mencionar que muchas de ellas no solamente permiten realizar el modelado, sino que además permiten llevar a cabo su ejecución mediante la creación de instancias de los procesos definidos. Es así, que la evaluación realizada solamente abarcó aquellas herramientas que presentaban características relacionadas con el objetivo del presente trabajo. En el cuadro 2-1 se presenta una breve descripción de cada una de las herramientas evaluadas.

A continuación se nombran las principales características encontradas en cada una de estas herramientas:

**Together Workflow Editor [68]**

- Hace uso del lenguaje XPDL para almacenar la definición de sus procesos.
- Posee un editor gráfico que permite diagramar el flujo de un proceso a través de un conjunto de herramientas de diagramado que están basadas en la notación gráfica BPMN (Business Process Management Notation).
- Mantiene la información de un proceso a través de ventanas de diálogo y/o formularios.
- Genera un archivo XML con la definición de los procesos creados por los usuarios.
- Realiza una validación gráfica de los procesos, basándose en las reglas establecidas por los patrones de procesos.
- Permite importar la definición de un proceso a partir de un archivo XML que cumpla con el estándar definido por el lenguaje XPDL.
- Puede compartir fácilmente la definición de los procesos generados con otras herramientas afines al lenguaje utilizado.

**JOpera for Eclipse [69]**

- Posee un lenguaje propio para almacenar la definición de los procesos.
- Permite definir el flujo de un proceso de forma gráfica.
- Permite la gestión de versiones de procesos, haciendo posible que el usuario visualice la evolución que estos han sufrido a través del tiempo.
- No es una herramienta propiamente dicha, sino que es un plugin que se integra al IDE Eclipse.
- Su interoperabilidad con otros programas es un poco compleja ya que no posee un lenguaje universal o estándar.

**ObjectWeb Bonita [70]**

- Hace uso del lenguaje XPDL para almacenar la definición de sus procesos.
- Permite definir el flujo de un proceso de forma gráfica, sin embargo las herramientas de diagramado que utiliza son limitadas. La herramienta utiliza BPMN como notación gráfica.
- Mantiene la información de un proceso a través de ventanas de diálogo y/o formularios.
- Posee un sistema de seguridad basado en roles, los cuales poseen diferentes niveles de acceso que permiten desde la visualización de un proceso, hasta la aprobación o rechazo del mismo.
- Permite importar la definición de un proceso a partir de un archivo XML que cumpla con el estándar definido por el lenguaje XPDL.

**WorkflowGen [71]**

- Hace uso del lenguaje XPDL para almacenar la definición de sus procesos.
- Posee un diseñador gráfico.
- Mantiene la información de un proceso a través de ventanas de diálogo y/o formularios.
- Posee un sistema de seguridad en base a perfiles, lo que permite tener un mejor control sobre el nivel de acceso de los usuarios que acceden a la herramienta.
- Permite la administración de versiones de los procesos definidos.

**BPWin [72]**

- Utiliza técnicas de modelado específicas, tales como: IDEF0, IDEF3 y DFD.
- Permite definir el flujo de un proceso de forma gráfica basándose en las notaciones gráficas de Yourdon o Sarson.
- Brinda mecanismos de explosión de niveles, lo que permite que un proceso puede ser representado en n-niveles de jerarquía.
- Incluye mecanismos de consistencia del número de entradas y salidas existentes entre los niveles contiguos de un proceso explosionado.
- Su integración sólo es posible con otros productos que manejen la misma técnica de modelado utilizada por la herramienta**.**

**XFLOW [73]**

- Soporta el versionamiento de procesos.
- Permite la explosión de procesos en n-niveles de jerarquía.
- Guarda las definiciones de un proceso en archivos XML.
- Posee un formato de definición de lenguaje propio, lo que hace que tenga una integración pobre con otras herramientas.
- No posee una interfaz gráfica para la definición de los procesos, los usuarios tienen que definir sus procesos de forma manual en un archivo XML de acuerdo a los estándares y reglas definidos por la herramienta.

## 2.3. Características no encontradas en las herramientas evaluadas

Luego de haber realizado la evaluación de estas herramientas se encontraron las siguientes carencias:

- Ninguna de ellas posee un mecanismo que permita a las organizaciones adaptar las definiciones de sus procesos bases al entorno de trabajo sobre el cual se está llevando a cabo un proyecto.

- Algunas de las herramientas evaluadas permiten la explosión de una actividad en diferentes niveles, lo cual permite hacer más legibles aquellos flujos que son complejos. Sin embargo, las herramientas evaluadas que ofrecen esta característica hacen uso de su propio lenguaje de definición, lo cual hace más complicado su interoperabilidad con otras herramientas (Cuadro 2-1).

- La funcionalidad de gestión de versiones de las definiciones de procesos que hace posible que las organizaciones puedan tener un mejor control sobre la evolución de sus procesos y el concepto de explosión de una actividad en n-niveles, son conceptos que no están contemplados por todas las herramientas evaluadas y aquellas que sí la ofrecen hacen uso de su propio lenguaje de definición.

## 2.4. Características principales de la herramienta propuesta

Las principales características que la herramienta propuesta contiene, se mencionan a continuación:

- La herramienta MJS Process Designer permitirá la creación de paquetes, los cuales actúan como contenedores de definiciones de procesos, metodologías, versiones de procesos y metodologías y los elementos comunes a todos ellos.

- Se podrá definir gráficamente un proceso utilizando las herramientas de diagramado, las cuales se basan en la notación gráfica BPMN. El usuario podrá diagramar los siguientes tipos de elementos: actividades, subprocesos, bloques de actividades, transiciones, asociaciones y conectores lógicos.

- Como parte del proceso de diagramado la herramienta realizará la validación de los flujos de trabajo definidos por el usuario según las reglas básicas de diagramado de procesos. Entre las reglas que se tomarán en cuenta para el presente trabajo de tesis se tienen: obligatoriedad de nodos de inicio y fin de proceso, consistencia entre el número de entradas y salidas, entre otras.

- La herramienta permitirá la explosión de actividades pertenecientes a una definición de procesos en varios sub-niveles. Como parte del proceso de explosión se validará la coherencia entre las entradas y salidas provenientes de niveles contiguos.

- Se podrá definir metodologías basadas en las definiciones de procesos existentes en un mismo paquete. Adicionalmente, se mantendrá la consistencia de los elementos que conforman la metodología con respecto a las modificaciones que pudiesen realizarse en los procesos originales a los que pertenecen.

- Otra de las bondades que la herramienta ofrece es el manejo y administración de versiones a nivel de procesos y metodologías. Además, permitirá establecer e identificar cual es la versión actual o válida del proceso o metodología.

- La herramienta permitirá la exportación de las definiciones de procesos y metodologías definidas sobre ella mediante la generación de un archivo XML basado en el lenguaje de especificación de procesos XPDL.

- Debido a que el lenguaje XPDL no contempla nativamente las siguientes definiciones: explosión de actividades en sub-niveles, metodologías y versiones; se consideró necesario realizar una extensión al lenguaje XPDL con la finalidad de incluirlas dentro del mismo.

- La herramienta se integrará a través de archivos XML con otras herramientas del Proyecto COMPETISOFT – PUCP (Ver sección 2.5).

## 2.5. Proyecto COMPETISOFT – Herramientas

La industria de software representa una actividad económica de suma importancia para todos los países del mundo, especialmente para los iberoamericanos, ya que ofrece múltiples fuentes de ingresos y empleo y se perfila como una de las oportunidades más importantes en los países en vía de desarrollo. Sin embargo, en los países iberoamericanos, las empresas de desarrollo de software – normalmente pequeñas y medianas empresas (PYMEs)- no están preparadas para competir internacionalmente debido a que se enfrentan a una serie de problemas tales como la dependencia tecnológica y metodológica (especialmente de EEUU), la falta de formación sobre los procesos del ciclo de vida del software y sobre la calidad del mismo.

En el 2005, muchos investigadores y profesionales reconocieron la importancia de un marco de referencia común para la mejora y certificación de pequeñas y medianas organizaciones en países iberoamericanos. Es así que se propone el proyecto COMPETISOFT al Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo (CYTED), un grupo creado en

1984 para la multilateral cooperación científica y tecnológica, el cual es soportado por 21 países latinoamericanos, España y Portugal.

Para desarrollar el proyecto COMPETISOFT, se estudiaron diferentes iniciativas latinoamericanas orientadas a ofrecer un marco de referencia para mejorar la productividad de las organizaciones que desarrollan software, tales como: MoProSoft (México), el Modelo de Mejora de Procesos Brasileño, Agile SPI (Colombia), Métrica v3 (España), entre otros. El proyecto COMPETISOFT pretende unificar estas diferentes alternativas, aprovechando la sinergia y los conocimientos de estos países para conseguir un marco metodológico de ámbito verdaderamente iberoamericano.

Actualmente, el proyecto COMPETISOFT (Proyecto CYTED # 3789) involucra a 13 países (entre los cuales se encuentra el Perú) y 96 investigadores.

En el Perú, las empresas vienen introduciendo distintos marcos de referencia para la mejora de sus procesos, sin embargo estos no calzan con las distintas realidades que se presentan en las empresas peruanas, por lo que el Perú se convierte en un escenario propicio para implantar un marco metodológico orientado a las pequeñas y medianas empresas como el que propone COMPETISOFT.

COMPETISOFT PUCP es un proyecto desarrollado por el Grupo de Investigación y Desarrollo en Ingeniería de Software de la Universidad Católica (GIDIS-PUCP), el cual tiene como objetivo principal implantar un modelo de mejora de procesos de software en empresas dedicadas a este rubro en el Perú, planteándose las siguientes metas:

- Desarrollar herramientas de apoyo para la implantación del modelo de mejora.
- Desarrollar instrumentos de evaluación diagnóstica para COMPETISOFT.
- Evaluar el nivel de empresas peruanas según este modelo.
- Desarrollar mapeo de procesos entre modelos para apoyar a las empresas a elegir sus respectivos caminos de mejora.

El conjunto de proyectos que forman parte de COMPETISOFT PUCP se encuentran categorizados de la siguiente forma:

- Proyecto de Desarrollo de una herramienta de soporte para la Gestión, Operación y Auditoria de Procesos de Software en PYME.
- Proyectos de Mejora de Procesos con PYME desarrolladora de Software en el Perú.
- Proyecto de Mapeo de Procesos entre COMPETISOFT y Modelos de referencias.

Dentro de la primera categoría se encuentra el proyecto PS-PUCP, proyecto que congrega un conjunto de herramientas que permiten reflejar y manejar los procesos de una

organización, evaluar dichos procesos en base a marcos de referencia y proveer una auditoria a través del monitoreo y mediciones en todos los pasos de un proceso.

MJS Process Designer forma parte de este proyecto, teniendo como objetivo principal definir un proceso a través de un editor gráfico, mantener su información a través de formularios, todo ello siguiendo un estándar general basado en el lenguaje XPDL con la finalidad de que el usuario pueda definir cualquier tipo de proceso.

Dentro del proyecto PS-PUCP se encuentran adicionalmente dos herramientas, las cuales son:

1. Herramienta para Gestión de Proyectos basada en XPDL (EVCS Project Manager)
   Es una herramienta basada en el lenguaje XPDL, que permite administrar proyectos de software y crear instancias de un proceso o metodología a través de una interfaz grafica.

2. Herramienta para Auditoria de Procesos basada en XPDL (LMB Process Audit)
   Herramienta basada en el lenguaje XPDL, que permite mapear los procesos de una organización a un modelo de referencia y además realizar la auditoria de dichos procesos a través de un editor gráfico.

# 3. Análisis y Diseño

En este capítulo se muestran los aspectos más resaltantes de cada etapa del ciclo de vida del proyecto, desde su concepción como producto hasta el diseño de su arquitectura. Antes de entrar a la sección de diseño e implementación, se presenta la definición del producto en base a las necesidades detectadas en el mercado y definidos en el proyecto COMPETISOFT-PUCP. Para la etapa de análisis se presenta la lista de requerimientos asociada al sistema, la especificación de los casos de uso de la herramienta y el diagrama de análisis de la aplicación. Para la etapa de diseño se presenta el diagrama de clases de diseño, la extensión realizada al lenguaje XPDL y la arquitectura de la herramienta. Finalmente se presenta una descripción detallada de los módulos que componen la herramienta y el plan de pruebas a realizar sobre las principales funcionalidades de la misma.

## 3.1. Definición del producto

Como resultado del análisis de la situación actual, se diseñó la herramienta MJS Process Designer en busca de satisfacer las necesidades señaladas en capítulos anteriores.

A pesar de existir una gran variedad de herramientas en el mercado, no se encontró ninguna que abarque los conceptos de explosión de actividades en niveles, manejo de metodologías y gestión de versiones haciendo uso de un lenguaje formal. Debido a esto y como parte del desarrollo de la herramienta, se consideró necesario realizar la extensión del lenguaje de

definición de procesos XPDL con la finalidad de incluir en él las definiciones anteriormente referidas. Con ello, se logra contar con una herramienta que además de satisfacer las necesidades del usuario, también asegure su interoperabilidad con otras herramientas que manejen el mismo lenguaje. El análisis y desarrollo de la extensión realizada se explicará con mayor detalle a lo largo del presente capítulo.

El modelado gráfico utilizado en los diferentes módulos que componen la herramienta le permite al usuario modelar la definición de un proceso de forma sencilla; facilitando la modificación del modelo en forma dinámica.

En el módulo de explosión de niveles se considera que una actividad podrá ser explosionada en un nuevo nivel, el cual podrá estar conformado por dos o más actividades relacionadas entre sí. Este proceso se podrá repetir las veces que el usuario considere necesario sin restricción alguna, por lo que el modelo podrá reflejar un mayor detalle del flujo de trabajo según las necesidades del usuario.

Otra de las bondades que la herramienta brinda al usuario es la posibilidad de personalizar un flujo de trabajo o un marco de referencia generado en la herramienta en base a las necesidades del proyecto al cual se encuentra asociado. A este proceso se le denomina definición de metodologías y consiste en extraer una o más actividades pertenecientes a un proceso o marco de referencia con la finalidad de conformar un nuevo flujo de trabajo.

La gestión de versiones de definiciones de procesos y metodologías tiene un papel fundamental en la correcta gestión y administración de los procesos dentro de una organización. En el módulo de administración de versiones, el usuario podrá crear y administrar versiones de los procesos y metodologías definidos en el sistema a lo largo del tiempo.

Para asegurar la interoperabilidad de la herramienta MJS Process Designer se implementará la funcionalidad de exportar la definición de un paquete en un archivo XML utilizando el lenguaje de definición de procesos XPDL.

La herramienta MJS Process Designer se desarrollará en plataforma Web para facilitar la distribución de la aplicación y el acceso de los usuarios desde cualquier ubicación. Para ello, se utilizará un repositorio de datos común, en el cual se registrarán las definiciones de procesos manejadas por las empresas que tengan acceso a la aplicación.

MJS Process Designer será desarrollada de tal forma que se pueda integrar en un conjunto de herramientas de administración de procesos que abarque a todos los miembros del proyecto PS-PUCP.

## 3.2. Requerimientos del Software

A continuación se presentan los principales requerimientos de la herramienta MJS Process Designer:

El sistema permitirá la creación de paquetes, procesos y entidades relacionadas a cada uno de ellos. Adicionalmente, el sistema permitirá la definición de subprocesos y set activities, los cuales podrán ser usados dentro de la definición de procesos pertenecientes al mismo paquete.

El modelado de procesos se realizará por medio de una interfaz gráfica, la cual estará basada en la notación gráfica BPMN. Además, el sistema permitirá validar el flujo de trabajo diagramado en base a un conjunto de reglas básicas de diagramado de procesos.

Las definiciones de procesos modeladas en la herramienta MJS Process Designer podrán incluir enrutamiento de actividades de tipo secuencial y paralelo mediante el uso de elementos conectores (AND, OR, XOR y conector simple).

El usuario podrá definir parámetros de entrada y salida asociados a las actividades que conforman una definición de proceso.

Una actividad perteneciente a un proceso podrá ser explosionada en un nuevo nivel conformado por dos o más actividades relacionadas entre sí. El sistema validará la consistencia de las entradas y salidas compartidas entre niveles contiguos.

El usuario podrá definir metodologías en base a los procesos que haya registrado previamente en el sistema. La herramienta MJS Process Designer mantendrá la consistencia de los elementos pertenecientes a las metodologías frente a las posibles modificaciones que pudiesen realizarse en los procesos base de los cuales fueron extraídos dichos elementos.

El sistema permitirá la gestión de versiones de procesos y metodologías.

El usuario podrá exportar las definiciones de procesos, metodologías y demás elementos definidos dentro de un paquete en un archivo XML. La estructura del archivo XML generado por el sistema estará basado en el esquema del lenguaje de definición de procesos XPDL.

## 3.3. Casos de Uso

A continuación se presenta la especificación de los casos de uso principales:

| *Diagramado de Proceso* | |
|---|---|
| **Descripción** | El propósito de este caso de uso es realizar el modelado gráfico de un proceso. |
| **Actores** | Usuario Modelador. |
| **Precondición** | Se encuentra abierto un proceso en el sistema. |

**Flujo Básico**

1. El actor selecciona en el árbol sobre el nombre del proceso que se desea hacer el diagramado.
2. El sistema activa el área de modelado gráfico.
3. El actor escoge un elemento para ser agregado al modelado gráfico del proceso.
4. El actor selecciona cualquiera de las opciones disponibles en la barra de herramientas del área de modelado (inicio, fin, actividades, conectores, transiciones, asociaciones, texto).
5. El actor dibuja en el área de trabajo el elemento que seleccionó:

5.1. Si el elemento es inicio, fin, actividad o condición (Cualquiera de ellos en adelante serán denominados Nodos):

5.1.1. El actor ubicará el lugar donde se colocará el nodo.

5.1.2. Si el nodo escogido es del tipo inicio, el sistema verificará que no se haya dibujado anteriormente la condición de inicio.

5.1.3. El sistema dibujará automáticamente la respectiva representación gráfica del tipo de nodo escogido: Inicio, fin, actividad (básica, bucle, suproceso, setactivity) o conector (simple, AND, OR, XOR).

5.2. Si es una transición:

5.2.1. El actor seleccionará alguno de los nodos para indicar el inicio de la transición.

5.2.2. El actor seleccionará otro nodo para indicar el fin de la transición.

5.2.3. El sistema verificará que el nodo fin de la transición no sea el mismo que el nodo inicio de la transición.

5.2.4. El sistema dibujará la transición de la siguiente forma:

5.2.4.1 Si el tipo de transición elegida es del tipo lineal, el sistema dibujará una línea recta uniendo el nodo de inicio y el nodo de fin de la transición.

5.2.4.2 Si el tipo de transición elegida es del tipo polilínea, el sistema dibujará una línea cuadrangular uniendo el nodo de inicio y el nodo de fin de la transición.

5.3. Si es una asociación del tipo IN o OUT:

5.3.1. El actor seleccionará algún nodo.

5.3.2. El sistema verificará que el nodo seleccionado sea del tipo actividad básica o actividad bucle.

5.3.3. El sistema dibujará la asociación de acuerdo con los siguientes criterios:

5.3.3.1 Si es una asociación del tipo IN, el sistema dibujará automáticamente una línea de un tamaño definido en el lado izquierdo del nodo escogido con la cabeza de la flecha ingresando a la actividad.

5.3.3.2 Si es una asociación del tipo OUT, el sistema dibujará automáticamente una línea de un tamaño definido en el lado derecho del nodo escogido con la cabeza de la flecha saliendo de la actividad.

5.4. Si es una asociación del tipo IN/OUT:

5.4.1. El actor seleccionará el nodo que será el nodo de salida de la asociación.

5.4.2. El sistema verificará que el nodo seleccionado sea del tipo actividad básica o actividad bucle.

5.4.3. El actor realizará seleccionará el nodo que será el nodo de entrada para la asociación.

5.4.4. El sistema verificará que el nodo seleccionado sea del tipo actividad básica o actividad bucle.

5.4.5. El sistema verificará que el nodo de entrada de la asociación no sea el mismo que el nodo de salida de la asociación.

5.4.6. El sistema dibujará una flecha de doble sentido uniendo el nodo de entrada y el nodo de salida de la asociación.

5.5. Si es un texto:

5.5.1. El actor seleccionará el lugar donde desea colocar un texto.

5.5.2. El sistema mostrará una ventana de entrada para ingresar el texto.

5.5.3. El actor ingresa el texto en la ventana.

5.5.4. El sistema dibuja en el modelado gráfico del proceso el texto ingresado.

6. El actor sigue realizando los pasos 4 y 5 hasta que termine de realizar el modelado gráfico del proceso.

7. El actor guardará el modelado gráfico del proceso (Referencia al caso de uso Guardar Proceso).

| **Poscondición** | Se realizó el diagramado del proceso. |
|---|---|

**Flujo Alternativo "Modificar tamaño del nodo"**

1. El actor sitúa el puntero del mouse en el cuadrado de redimensionamiento (parte inferior derecho) del nodo que se desee modificar.
2. El actor irá arrastrando el puntero del mouse hasta que el tamaño del nodo sea el deseado.
3. El sistema cambiará el tamaño del nodo redimensionado.

**Flujo Alternativo "Modificar el alto de la transición polilínea"**

1. El actor sitúa el puntero del mouse en el cuadrado de redimensionamiento (parte inferior izquierda) sobre la transición del tipo polilínea que desee modificar.
2. El actor irá arrastrando el puntero del mouse hasta que el alto de la transición sea el deseado.
3. El sistema cambiará el alto de la transición polilínea.

**Flujo Alternativo "Modificar ubicación del nodo"**

1. El actor selecciona el nodo que desea cambiar de ubicación.
2. El sistema cambia a color rojo el borde exterior del nodo escogido para indicar que ha sido seleccionado.
3. El actor irá arrastrando dentro del área del editor gráfico el nodo escogido hasta que se encuentre en la ubicación deseada.
4. El sistema moverá la ubicación grafica del nodo.
5. El sistema moverá automáticamente las posiciones de las transiciones y asociaciones asociadas al nodo elegido.

| Flujo Alternativo “Modificar ubicación del texto” |
|---|
| 1. El actor selecciona el texto que desea cambiar de ubicación.<br>2. El sistema cambia a color rojo el borde exterior del texto escogido para indicar que ha sido seleccionado.<br>3. El actor irá arrastrando dentro del área del editor gráfico el texto escogido hasta que se encuentre en la ubicación deseada.<br>4. El sistema moverá la ubicación grafica del texto. |
| **Flujo Excepcional “Mismo nodo de inicio y nodo de fin”** |
| Ocurre en el paso 5.2. o 5.4., cuando el actor ha escogido el mismo nodo de inicio y nodo de fin para la transición o la asociación del tipo IN/OUT.<br>1. El sistema muestra un mensaje de error “No puede escoger el mismo nodo de inicio y fin. Escoja otro”.<br>2. El actor selecciona la opción “Aceptar”. |
| **Flujo Excepcional “Elemento de inicio ya dibujado”** |
| Ocurre en el paso 5.1. cuando el actor ha tratado de dibujar el elemento inicio, cuando ya existía uno creado en el proceso.<br>1. El sistema muestra un mensaje de error “Inicio ya creado en el proceso”.<br>2. El actor selecciona la opción “Aceptar”. |
| **Flujo Excepcional “Asociación no permitida”** |
| Ocurre en el paso 5.3. o 5.4. cuando el actor ha escogido algún nodo que no es del tipo actividad básica o actividad bucle para que se el nodo de entrada o fin de la asociación.<br>1. El sistema muestra un mensaje de error: “Error: una asociación solamente se puede asociar a una actividad básica o una actividad bucle”.<br>2. El actor selecciona la opción “Aceptar”. |

| ***Mantenimiento de Proceso*** | |
|---|---|
| **Descripción** | El propósito de este caso de uso es el de permitir al usuario realizar el mantenimiento semántico del proceso. |
| **Actores** | Usuario modelador / Usuario supervisor. |
| **Precondición** | Para que este caso de uso se ejecute debe haberse definido un proceso. |
| **Flujo Básico** | |
| 1. El actor selecciona en el árbol de procesos aquel proceso sobre el que se realizará el mantenimiento.<br>2. El actor selecciona la opción “Inf” que se encuentra al lado derecho del nombre del proceso seleccionado.<br>3. El sistema mostrará en el sector inferior izquierdo un menú de opciones que permitirán dar mantenimiento a la información del proceso, entre las cuales se encuentran:<br>▪ General<br>▪ Definición de Cabecera | |

- Redefinición de Cabecera
- Parámetros formales
- Participantes
- Variables del Proceso
- Aplicaciones
- Atributos Extendidos
- Set Activities

4. El actor selecciona una de las siguientes opciones de mantenimiento.

4.1. Si el actor selecciona la opción “General”, se mostrará un formulario donde podrá ingresar los siguientes campos: Identificador personalizado, nombre, descripción, nivel de acceso, tipo de proceso, estado, proceso personalizado, ejecución personalizada, condición de fin personalizada, habilitar compensación de instancia y suprimir error de comunicación.

4.2. Si el actor selecciona la opción “Definición de Cabecera”, se mostrará un formulario para ingresar la información de la definición de cabecera (Referencia al Mantenimiento de Definición de Cabecera).

4.3. Si el actor selecciona la opción “Redefinición de Cabecera” se mostrará un formulario para ingresar la información de la redefinición de cabecera (Referencia al Mantenimiento de Redefinición de Cabecera).

4.4. Si el actor selecciona la opción “Parámetros Formales” se mostrará una ventana donde el usuario podrá crear nuevos parámetros asociados al proceso. (Referencia al Mantenimiento de Parámetro Formal).

4.5. Si el actor selecciona la opción “Participantes” se mostrará una ventana donde el usuario podrá crear nuevos participantes asociados al proceso. (Referencia al Mantenimiento de Participante).

4.6. Si el actor selecciona la opción “Aplicaciones” se mostrará una ventana donde el usuario podrá crear nuevas aplicaciones asociadas al proceso. (Referencia al Mantenimiento de Aplicación).

4.7. Si el actor selecciona la opción “Variables de Proceso” se mostrará una ventana donde el usuario podrá crear nuevas variables asociadas al proceso. (Referencia al Mantenimiento de Variable).

4.8. Si el actor selecciona la opción “Atributos Extendidos” se mostrará una ventana donde el usuario podrá crear nuevos atributos extendidos para el proceso. (Referencia al Mantenimiento de Atributo Extendido).

4.9. Si el actor selecciona la opción “Conjunto de Actividades” se mostrará una ventana donde el usuario podrá crear nuevos conjuntos de actividades asociados al proceso. (Referencia al Mantenimiento de Conjunto de Actividad).

5. El actor ejecuta el paso 4 las veces que requiera hasta que termine de realizar el mantenimiento del proceso.

| | |
|---|---|
| **Poscondición** | Se realiza el mantenimiento del proceso. |

**Flujo Excepcional “Datos Incompletos”**

Se inicia en el paso 4 del flujo básico si no se han ingresado datos requeridos.

1. El sistema muestra un mensaje de error “Faltan ingresar datos” y muestra el detalle del error.
2. El actor selecciona la opción “Aceptar” y se regresa al paso anterior.

| *Explosionar Actividad* | |
|---|---|
| **Descripción** | El propósito de este caso de uso es explosionar una actividad en un nuevo nivel compuesto por dos o más actividades. |
| **Actores** | Usuario modelador / Usuario supervisor |
| **Precondición** | Para que este caso de uso se ejecute debe haberse realizado el caso de uso “Abrir Paquete”. |

**Flujo Básico**

1. El actor selecciona un proceso del árbol que lista los procesos pertenecientes a un paquete en el módulo “Modelado de Procesos”.
2. El actor selecciona la opción “Ir Vista Explosión”.
3. El sistema muestra una pantalla dividida en tres sectores:
   - En el sector izquierdo superior se encuentra el árbol que contiene el proceso seleccionado y los niveles que han sido creados hasta el momento producto de la explosión de actividades pertenecientes al proceso.
   - En el sector izquierdo inferior mostrará la lista de mantenimientos disponibles para el modelado semántico del elemento seleccionado en el editor gráfico.
   - En el sector derecho se encuentra el área de trabajo con las opciones: Editor Gráfico y Mantenimiento. En la primera de ellas el usuario podrá realizar el modelado gráfico del nuevo nivel y en la segunda podrá realizar el mantenimiento de la información semántica del elemento seleccionado.
4. El actor selecciona el proceso en el árbol del módulo.
5. El sistema carga en el editor gráfico (área de trabajo) el modelado gráfico del proceso.
6. El actor selecciona una actividad de tipo base perteneciente al proceso
7. El actor selecciona la opción de explosión de actividad.
8. El sistema crea un nuevo nodo en el árbol del módulo correspondiente al nivel producto de la explosión de la actividad seleccionada.
9. El actor selecciona el nodo correspondiente al nivel creado.
10. El sistema muestra el área de modelado grafico en blanco.
11. El actor realiza el modelado grafico del nivel creado (Referencia al caso de uso Modelar un proceso).
12. El actor selecciona la opción “Guardar”.
13. El sistema realiza la validación de la consistencia de los artefactos definidos en el nivel y los artefactos heredados del nodo explosionado.
14. El sistema guarda el modelado gráfico del nivel (Referencia al caso de uso Guardar Proceso).

| | |
|---|---|
| **Poscondición** | Se realizó el modelado gráfico del nivel. |

**Flujo Alternativo “Relacionar asociación padre ”**

Ocurre cuando el actor realiza el mantenimiento de la información relacionada a una avocación del nivel.

1. El actor selecciona un elemento asociación en el modelado gráfico del nivel.
2. El actor selecciona la opción de mantenimiento de la barra de herramientas.
3. El sistema muestra los mantenimientos disponibles para el elemento asociación:

General y Asociar asociación padre.

4. El actor selecciona la opción: Asociar asociación padre.
5. El sistema muestra un formulario con el listado de asociaciones de la actividad que ha sido explosionada.
6. El actor elige una de las asociaciones listadas y selecciona la opción guardar.
7. El sistema registra la relación de la asociación del nivel con asociación padre.

**Flujo Alternativo "Inconsistencia con artefactos heredados del nodo padre"**

Ocurre en el paso 13 cuando no se han asociado todos los artefactos asociados a la actividad explosionada con los artefactos definidos en el nivel creado.

1. El sistema muestra un mensaje de error "Inconsistencia de artefactos heredados de nodo padre".
2. El actor selecciona la opción "Aceptar".

| Mantenimiento de metodología | |
|---|---|
| **Descripción** | El propósito de este caso de uso es el de permitir crear y modificar metodologías definidas en un paquete. |
| **Actores** | Usuario modelador / Usuario supervisor |
| **Precondición** | Para que este caso se ejecute debe haberse definido un paquete. |

**Flujo Básico**

1. El caso de uso se inicia cuando el actor selecciona la opción "Ir vista metodología".
2. El sistema muestra una pantalla dividida en tres sectores:
   - En el sector izquierdo superior se encuentra el árbol que contiene como raíz el nombre de la metodología creada y como nodos los distintos procesos seleccionados para crear la metodología, así como las actividades asociadas a dichos procesos.
   - En el sector izquierdo inferior mostrará la lista de mantenimientos disponibles para el modelado semántico del elemento seleccionado en el editor gráfico.
   - En el sector derecho se encuentra el área de trabajo con las opciones: Editor gráfico y Mantenimiento. En el primero de ellos se visualizará el modelado gráfico de la metodología creada y en el segundo se visualizará la información semántica del elemento seleccionado en una metodología.
3. El actor selecciona la opción de "Crear Nueva Metodología".
4. El sistema carga en el área de mantenimiento el formulario de creación de metodología con los siguientes campos: Identificador personalizado, nombre, descripción, lista de procesos disponibles para ser utilizados como procesos de referencia, procesos asociados a la metodología.
5. El actor ingresa los datos de la metodología y selecciona los procesos base de donde se extraerán los elementos que conforman la metodología.
6. El actor selecciona la opción "Guardar".
7. El sistema valida los datos ingresados.
8. El sistema crea la nueva metodología.
9. El sistema carga el árbol de la metodología, el cual tiene como nodo raíz el nombre de

| | |
|---|---|
| la metodología y como nodos los procesos asociados a ella con sus respectivas actividades. | |
| **Poscondición** | Se crea una nueva metodología. |

**Flujo Alternativo "Modificar Metodología"**

1. El caso de uso se inicia cuando se tiene abierto una metodología en el sistema.
2. El actor selecciona la opción "Inf" que se encuentra al lado derecho del nombre de la metodología que desea modificar.
3. El sistema mostrará en el sector inferior izquierdo la opción que permitirá dar mantenimiento a la información de la metodología y el actor seleccionará dicha opción.
4. El sistema carga en el área de trabajo un formulario con la información de la metodología a través de los siguientes campos: Identificador personalizado, nombre, descripción, lista de procesos seleccionados, lista de procesos disponibles para ser utilizados como procesos de referencia.
5. El actor modifica los datos de la metodología y selecciona los procesos base de donde se extraerán los elementos que conforman la metodología.
6. El actor selecciona la opción "Guardar".
7. El sistema valida los datos ingresados.
8. El sistema modifica los datos de la metodología.

| | |
|---|---|
| **Poscondición** | Se guardan las modificaciones sobre la metodología seleccionada. |

**Flujo Excepcional "Datos Incompletos"**

Se inicia en el paso 7 del flujo básico y paso 7 del flujo alternativo si no se han ingresado los datos requeridos.

1. El sistema muestra un mensaje de error "Faltan ingresar datos" y muestra el detalle del error.
2. El actor selecciona la opción "Aceptar" y se regresa al paso anterior.

| ***Generar versión proceso*** | |
|---|---|
| **Descripción** | El propósito de este caso de uso es crear una nueva versión de un proceso. |
| **Actores** | Usuario Supervisor. |
| **Precondición** | El usuario debe haberse autentificado en el sistema. |

**Flujo Básico**

1. El actor selecciona la opción Ir Vista Versión.
2. El sistema muestra una pantalla dividida en tres sectores:
   - En el sector izquierdo superior se encuentra el árbol que contiene como raíz el nombre del proceso seleccionado y como nodos las distintas versiones que existen del mismo.
   - En el sector izquierdo inferior se mostrará la lista de mantenimientos disponibles

| | |
|---|---|
| para la visualización de la información más no la modificación.<br>▪ En el sector derecho se encuentra el área de trabajo con las opciones: Editor gráfico y Mantenimiento. En el primero de ellos se visualizará el modelado gráfico de la versión seleccionada y en el segundo se visualizará la información semántica del elemento seleccionado en una versión.<br>3. El actor selecciona un nodo correspondiente a una versión del proceso.<br>4. El sistema carga en el editor gráfico (área de trabajo) el modelado gráfico de la versión seleccionada.<br>5. El actor selecciona la opción de “Nueva versión”.<br>6. El sistema crea un nuevo nodo en el árbol del módulo correspondiente a la nueva versión del proceso creada. | |
| **Poscondición** | Se creó una nueva versión del proceso. |

### 3.4. Diagrama de Clases de Análisis

En la etapa inicial del trabajo de tesis se planteó el diseño de clases de análisis de la herramienta como se muestra en la ilustración 3-1.

Cada entidad definida en los meta modelos de paquete y proceso del lenguaje XPDL (ilustraciones 1-4 y 1-5), es representada tal como se muestra en la ilustración 3-1 a través de una clase. Del mismo modo, para modelar relaciones existentes entre dichas entidades se utilizó el elemento asociación de UML.

Las entidades definidas en el lenguaje XPDL poseen un conjunto de atributos que pueden ser de carácter obligatorio u opcional. En el diagrama de clases de análisis inicial se planteó que cada clase tendría como parte de sus atributos solamente aquellos que fuesen de carácter obligatorio en el lenguaje XPDL. Para el manejo de los atributos opcionales se decidió implementar una clase generalizada de uso común.

Esta clase de uso común debería contar con las siguientes características:

- Permitir la definición de un atributo identificado por un nombre.
- Permitir almacenar un valor asociado a dicho atributo.
- Estar relacionada con las demás clases a fin de poder definir sus atributos opcionales.

Las características anteriormente mencionadas ya vienen inmersas en la entidad “Extended Attribute” del lenguaje XPDL. Dicha entidad permite agregar características adicionales que no han sido contempladas de forma nativa para los elementos definidos en dicho lenguaje.

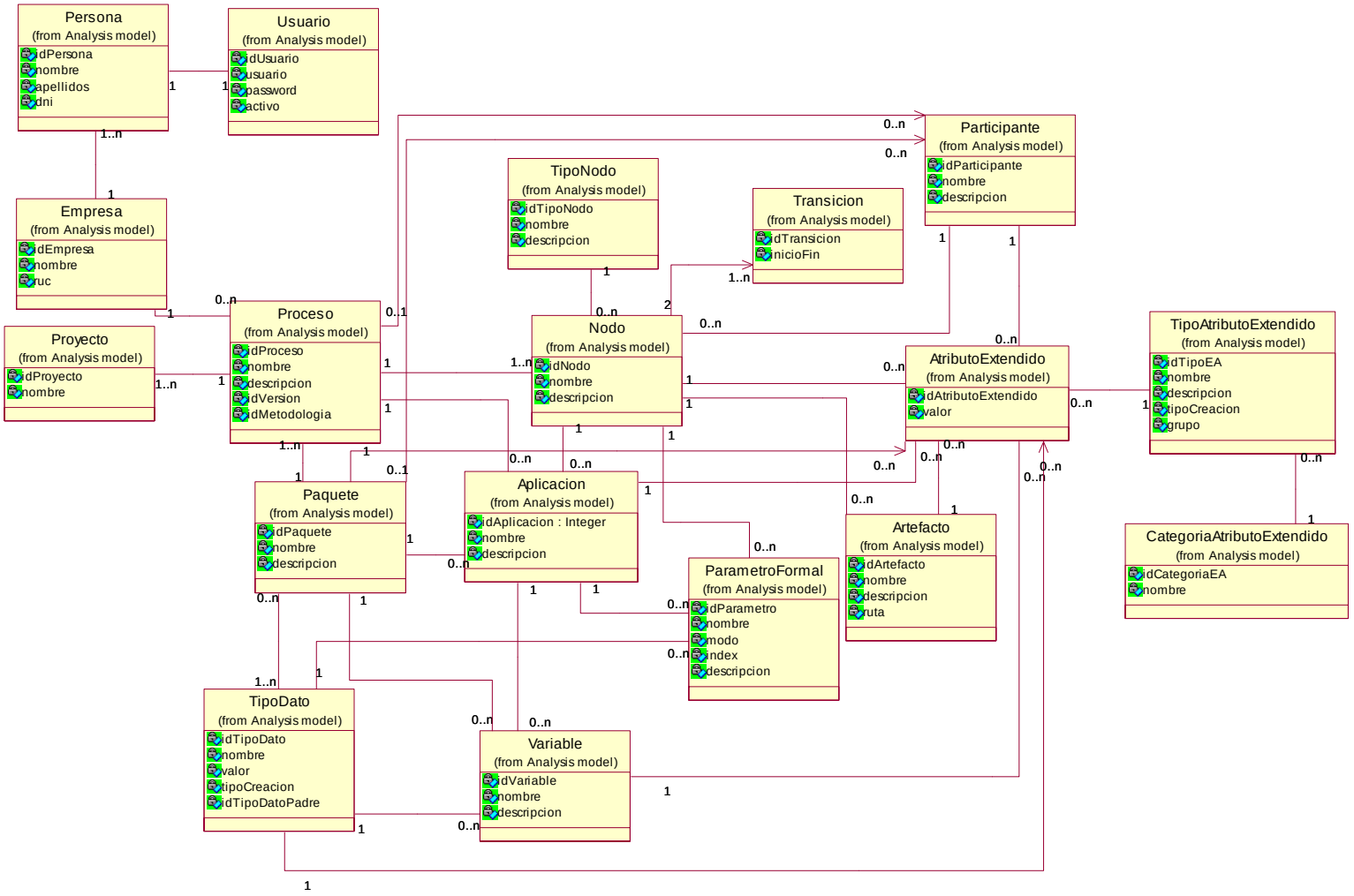


**Ilustración 3-1: Diagrama de Clases de Análisis Inicial**

En el diagrama de clases inicialmente planteado se representó la entidad “Extended Attribute” a través de la clase “AtributoExtendido”. Sin embargo, la definición de esta clase no era suficiente para llevar a cabo la generalización requerida de los atributos, ya que esta clase también debía soportar las siguientes características:

- Permitir la categorización de los atributos de las diferentes clases a fin de poder determinar a que entidad pertenecen.
- Permitir diferenciar los atributos que poseen nombres similares, pero que a su vez se encuentren definidos en entidades diferentes.
- Permitir diferenciar los atributos propios de las entidades del lenguaje XPDL de los atributos definidos por un usuario.

Como parte de la solución inicial, adicionalmente a la clase “AtributoExtendido” se crearon las clases: “CategoriaAtributoExtendido” y “TipoAtributoExtendido”.

La clase “CategoriaAtributoExtendido” se encargaría de almacenar los nombres de las entidades que se definen en el lenguaje XPDL, como por ejemplo: Paquete, Proceso, Participante, entre otros. Todo ello con la finalidad de permitir la categorización de los atributos e identificar la entidad propietaria de los mismos.

La clase “TipoAtributoExtendido” se encargaría de almacenar los nombres de los atributos definidos en las entidades del XPDL o de aquellos definidos por el usuario. Para poder determinar la procedencia de estos atributos, se definió dentro de esta clase el atributo “tipoCreacion”. Adicionalmente, se definió el atributo “grupo” para categorizar los elementos que no figuran en los metamodelos de paquete y proceso, pero que pertenecen a otras entidades del XPDL. Por ejemplo: el atributo Responsable forma parte de la Definición de Cabecera, la cual a su vez está relacionada con la entidad Proceso.

Este diseño de diagrama de clases inicial brindaba las siguientes ventajas:

- Reducir del número de tablas a administrar debido a que solamente se definirían las clases de las entidades principales del XPDL. Las demás entidades se expresarían utilizando las siguientes clases: “AtributoExtendido”, “CategoriaAtributoExtendido” y “TipoAtributoExtendido”.

- Soportar todos los atributos definidos en el lenguaje XPDL gracias al manejo generalizado que se planteaba.

Las deficiencias que presentaba el diagrama de clases inicial salieron a la luz cuando se empezó a elaborar el prototipo funcional de uno de los mantenimientos de la herramienta.

Este prototipo no sólo incluía el diseño de interfaces, sino que además contemplaba la implementación de dicha funcionalidad.

El manejo generalizado de los atributos hacía difícil la inserción, modificación y eliminación de los atributos propios de las entidades XPDL. Esto se identificó al tener que crear sentencias SQL bastante elaboradas para llevar a cabo estas operaciones básicas.

Otro inconveniente encontrado fue la fuerte carga transaccional que estaría dirigida en gran parte hacia las tablas de base de datos asociadas a las clases: “AtributoExtendido”, “CategoriaAtributoExtendido” y “TipoAtributoExtendido”. Esto afectaría directamente el tiempo de respuesta de la base de datos al realizar una transacción, lo que ocasionaría descontento y malestar en el usuario de la herramienta.

Como consecuencia de los inconvenientes encontrados se rediseñó el diagrama de clases con la finalidad de poder superar dichos inconvenientes.

A continuación se presenta el diagrama de clases de análisis final de la herramienta MJS Process Designer. Para facilitar su entendimiento y visualización se ha dividido el diagrama de clases en cuatro grupos: Paquete, Proceso, Metodología y Administración/Seguridad.

### 3.4.1. Diagrama de Clases: Metamodelo Paquete

La ilustración 3-2 corresponde al diagrama de clases del metamodelo Paquete, en donde se muestran todas las clases involucradas en dicho metamodelo.

Entre las principales clases se encuentran las siguientes:

- **DefinicionCabecera:** Clase donde se definen los atributos generales de la entidad Paquete.
- **RedefinicionCabecera:** Clase que permite redefinir la información correspondiente a la entidad Paquete.
- **Script:** Clase que identifica el lenguaje script usado en las expresiones XPDL pertenecientes a los procesos que se encuentran dentro de un mismo paquete.
- **Proceso:** Clase donde se definen los atributos generales de la entidad Proceso del lenguaje XPDL.
- **TipoDato:** Clase que almacena información sobre los diferentes tipos de datos manejados en las definiciones de proceso.
- **Artefacto:** Clase que representa una información que es utilizada o producida durante el desarrollo de un proceso.

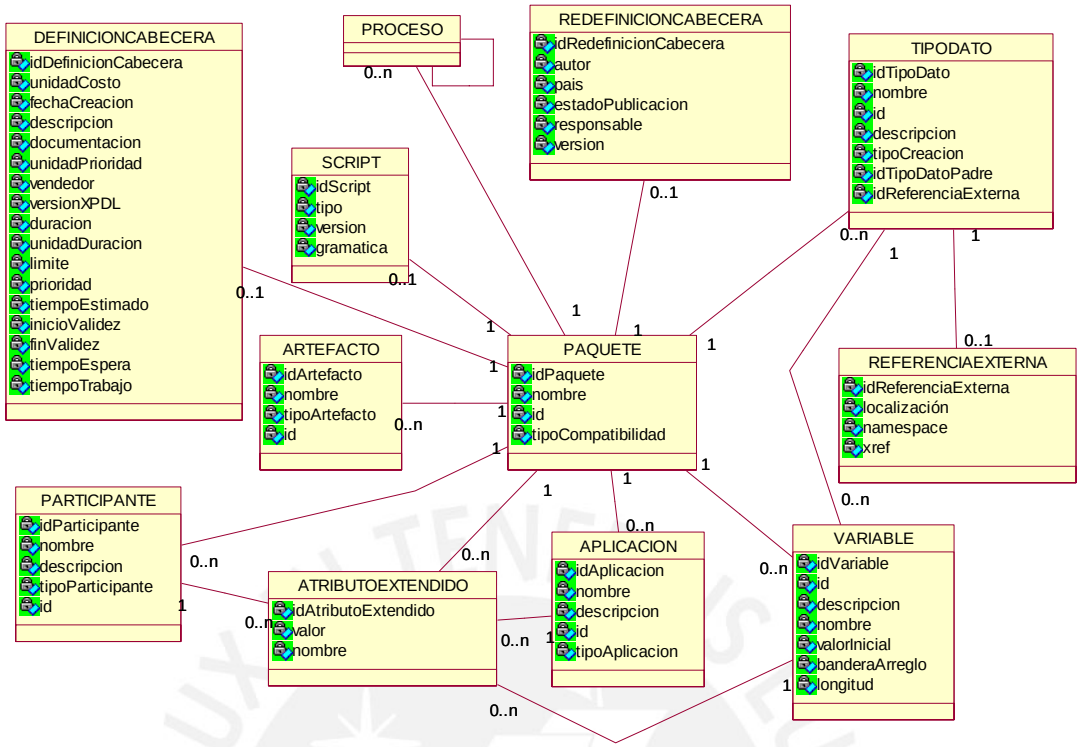


**Ilustración 3-2: Diagrama de Clases del Metamodelo Paquete**

### 3.4.2. Diagrama de Clases: Metamodelo Proceso

La ilustración 3-3 corresponde al diagrama de clases del metamodelo Proceso, en donde se muestran todas las clases involucradas en este metamodelo.

Entre las principales clases se encuentran las siguientes:

- **Proceso:** Clase donde se definen los atributos principales de un proceso. Dicha clase se relaciona consigo misma para establecer la relación que existe entre dos procesos: un proceso principal o padre y un proceso referenciado por el anterior como subproceso. El flujo de trabajo del subproceso se considera parte del flujo de trabajo del proceso principal.
- **Nodo:** Clase donde se definen los atributos de los elementos gráficos que intervienen en el proceso, entre los cuales se encuentran: actividades, conectores, nodo de inicio y nodo de fin.
- **Transición:** Clase que almacena información sobre las conexiones existentes entre los nodos.
- **SetActivity:** Clase que representa un conjunto de actividades (Nodo).

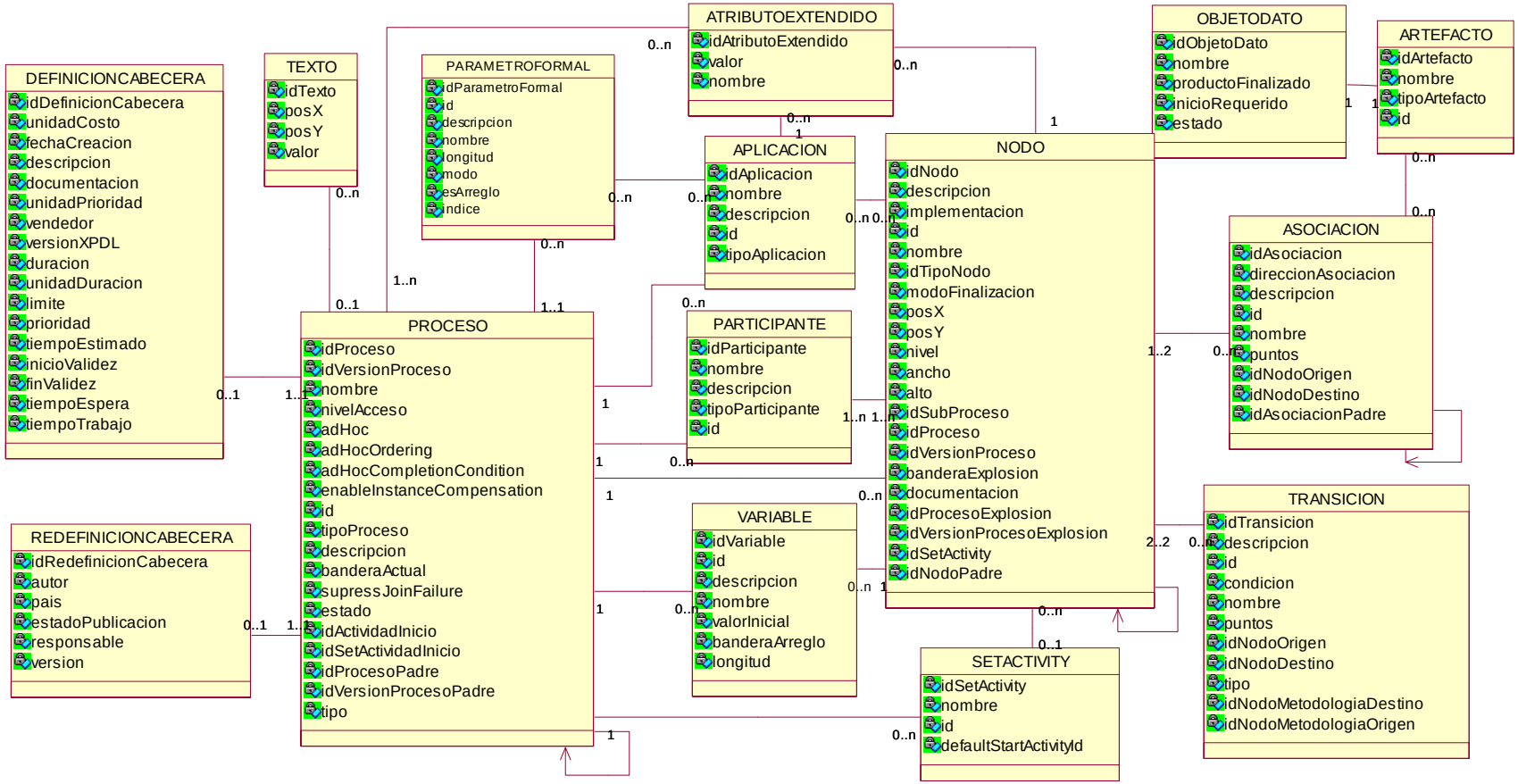


**Ilustración 3-3: Diagrama de Clases de Proceso**

La clase Nodo se relaciona consigo misma para representar la relación del nodo padre con los nodos hijos que surgen al momento de realizar la explosión de una actividad en un nuevo nivel.

Un nodo sólo puede estar relacionado a un proceso, pero un proceso está conformado por un conjunto de nodos.

- **Transición:** Clase donde se definen los atributos de las transiciones definidas en un proceso. Se relaciona con la clase Nodo para referenciar cuáles son los dos nodos que une.
- **Asociación:** Clase donde se definen los atributos de las entradas y salidas relacionadas a los nodos de tipo actividad. Esta clase se relaciona consigo misma para definir la relación de las entradas y salidas de un nodo explosionado con las entradas y salidas de los nodos hijos.
- **Texto:** Clase donde se almacenan los atributos de los textos creados en los modelados gráficos.
- **SetActivity:** Clase donde se definen los atributos de los conjuntos de actividades que se crean en un proceso. La estructura de un SetActivity es muy similar a la estructura de un proceso, por lo tanto puede ser manejada como tal.

### 3.4.3. Diagrama de Clases: Metamodelo Metodología

La ilustración 3-4 corresponde al diagrama de clases del metamodelo Metodología, en donde se muestran todas las clases involucradas en dicho metamodelo.

Entre las principales clases se encuentran las siguientes:

- **Metodología:** Clase donde se definen los atributos de una metodología. Adicionalmente una metodología puede estar asociada a uno o más proyectos.
- **ProcesoxMetodologia:** Clase intermedia que permite relacionar la metodología con los procesos base que la conforman.
- **NodoMetodologia:** Clase donde se definen los atributos de los nodos que conforman una metodología, tanto los nodos extraídos de los procesos base así como los nuevos nodos definidos en la metodología (inicio, fin y conectores).
- **AsociacionxMetodologia:** Clase donde se definen los atributos de las asociaciones definidas en los NodoMetodologia.
- **TransicionxMetodologia:** Clase donde se definen las transiciones definidas en una metodología.
- **Transición:** Clase donde se definen los atributos de las transiciones que conectan los NodoMetodologia.

### 3.4.4. Diagrama de Clases Metodología

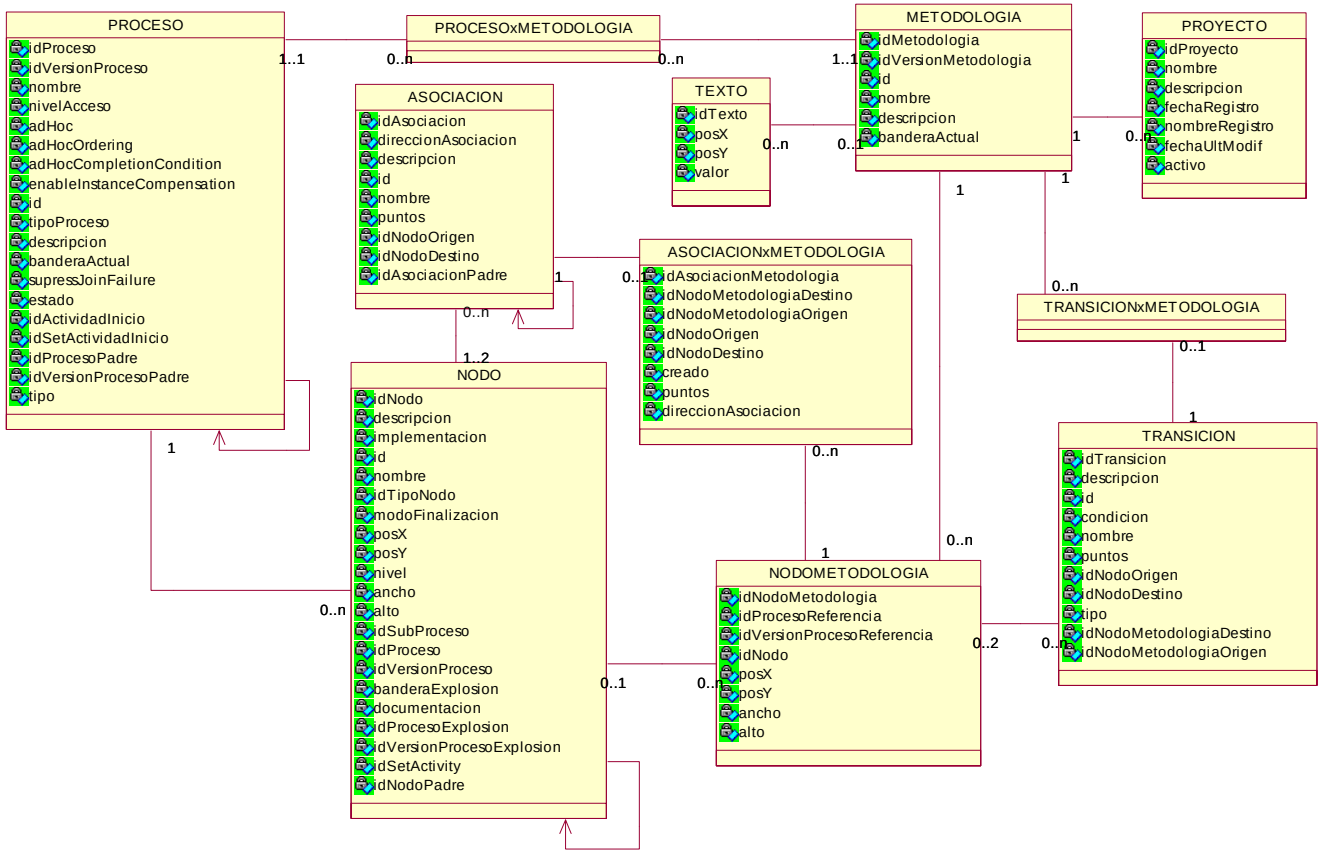


**Ilustración 3-4: Diagrama de Clases de Metodología**

### 3.4.5. Diagrama de Clases: Metamodelo Administración / Seguridad

Finalmente, en la ilustración 3-5 se muestra el diagrama de clases del metamodelo Administración/Seguridad. La herramienta MJS Process Designer tiene entre sus funcionalidades brindar una seguridad básica para el ingreso al sistema, la cual está basada en la autentificación a través del uso de un usuario y una contraseña.

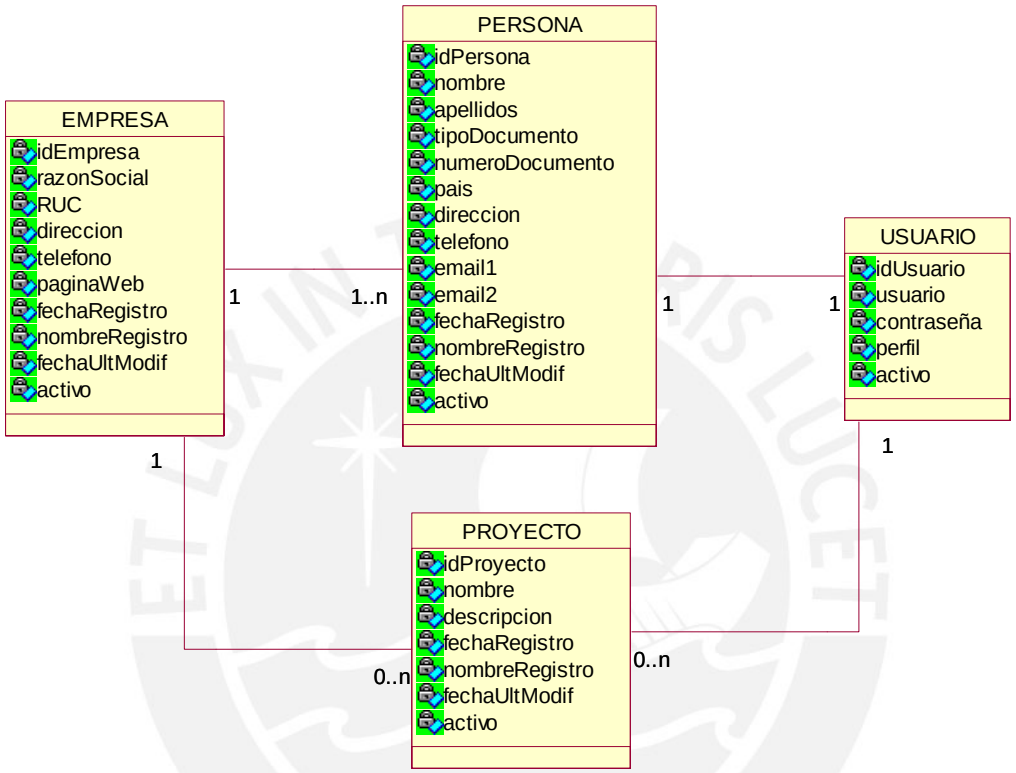


**Ilustración 3-5: Diagrama de Clases Administración / Seguridad**

Entre las principales clases se encuentran las siguientes:

- **Empresa:** Clase donde se definen los datos de una empresa registrada en el sistema.
- **Usuario:** Clase donde se definen los datos necesarios para ingresar al sistema, como son: usuario y contraseña. De igual forma se define el perfil que tiene un usuario dentro de dicho sistema.
- **Persona:** Clase donde se definen los datos de una persona. Dicha persona pertenece a una empresa y tiene asignado un usuario para ingresar al sistema.
- **Proyecto:** Clase donde se definen los atributos de un proyecto. Una empresa puede administrar uno o más proyectos propios. Cada proyecto estará asociado a un usuario específico de la empresa.

### 3.5. Extensión del Lenguaje XPDL

Como se mencionó en el primer capítulo, el lenguaje XPDL no brinda los elementos necesarios para incluir dentro de una definición de procesos los conceptos de versionamiento, metodología y explosión de actividades en niveles.

La evaluación de las funcionalidades disponibles en una muestra de herramientas modeladoras de procesos que existen actualmente en el mercado reflejó la escasez de dichos conceptos en este tipo de herramientas.

Con la finalidad que la herramienta MJS Process Designer provea dichos conceptos de gran utilidad para los usuarios, es que se decide incluirlos como parte de las funcionalidades propias del sistema.

Inicialmente se consideró elaborar un lenguaje de definición de procesos propio similar al lenguaje XPDL, en el cual se incluirían los conceptos que no se encontraban definidos de forma nativa en este último. Sin embargo, la desventaja que presentaba esta alternativa era la dificultad para el intercambio de definiciones de proceso con otras herramientas que no utilizaran el mismo lenguaje de definición.

En base a lo anteriormente expuesto y con el fin de asegurar la interoperabilidad de la herramienta MJS Process Designer, es que se decide utilizar el lenguaje formal XPDL dentro de la implementación de la herramienta y realizar una extensión sobre el mismo, a fin de incluir aquellos conceptos no contemplados de forma nativa [14].

A continuación se listan aquellos elementos XPDL que han sido extendidos:

**A. WorkflowProcess (Flujo de trabajo)**

Este elemento contiene los atributos y elementos necesarios para guardar la definición de un flujo de trabajo, entre los cuales se encuentran: actividades, transiciones, aplicaciones, data relevante, atributos propios, entre otros.

Luego de realizar un análisis acerca del concepto de metodología y el producto obtenido al explosionar una actividad, se llegó a la conclusión que ambos conceptos presentan entre sus características propias algunas muy similares a las encontradas en la definición de un flujo de trabajo. Dado que XPDL tiene entre sus entidades el elemento WorkflowProcess, se decide utilizar dicho elemento para representar las definiciones anteriormente mencionadas.

Con el fin de poder diferenciar entre un proceso base (definido por XPDL), un proceso de tipo metodología y un proceso de tipo explosión, se decide agregar un elemento denominado ProcessCategory dentro del elemento WorkflowProcess.

En la ilustración 3-6 se puede apreciar que el elemento ProcessCategory contiene otros tres elementos denominados ProcessBasic, ProcessMethodology y ProcessExplosion; los cuales permiten guardar información propia acerca de cada uno de estos tres tipos de procesos.

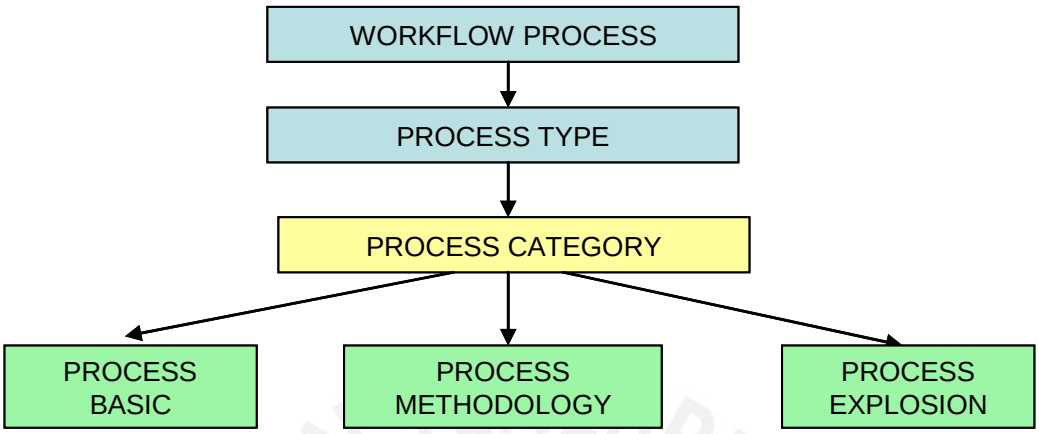


**Ilustración 3-6: Extensión WorkflowProcess**

A continuación se detallan cada uno de los elementos por los que está compuesto el elemento ProcessCategory:

**a. ProcessBasic:**

Este elemento permite identificar un proceso de tipo base o proceso definido nativamente por el lenguaje XPDL.

De la misma forma, debido a que el lenguaje XPDL no ofrece de forma nativa los elementos necesarios para poder almacenar información acerca de la versión de un proceso, es que se decide crear un elemento denominado VersionInformation (ilustración 3-7), el cual posee dos atributos:

- **IdVersion:** Número de versión del proceso.
- **CurrentVersion:** Número entero que indica si el proceso en mención es la versión vigente del proceso.

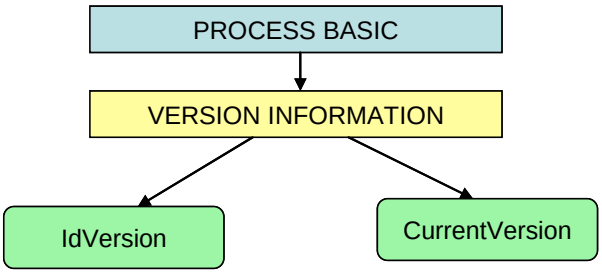


**Ilustración 3-7: Elemento Process Basic**

Este elemento VersionInformation es usado dentro del elemento ProcessBasic y dentro del elemento ProcessMethodology (elemento que se explicará más adelante) con el fin de poder realizar una gestión de versiones a nivel de procesos y metodologías.

**b. ProcessMethodology:**

Este elemento (ilustración 3-8) permite identificar un proceso de tipo metodología y a su vez agrupa una serie de elementos que se encargan de guardar información propia de un proceso de este tipo, entre los cuales se tiene:

- **VersionInformation:** Este elemento permite guardar información acerca de la versión de una determinada metodología.
- **ProcessesReference:** Este elemento agrupa un conjunto de elementos denominados ProcessReference, los cuales permiten guardar información acerca de los procesos de donde se extraen las actividades que conforman una metodología.
- **ProcessReference:** Este elemento posee dos atributos:
  - **IdProcReference:** Identificador del proceso referenciado.
  - **IdVersionProcReference:** Número de versión del proceso referenciado.

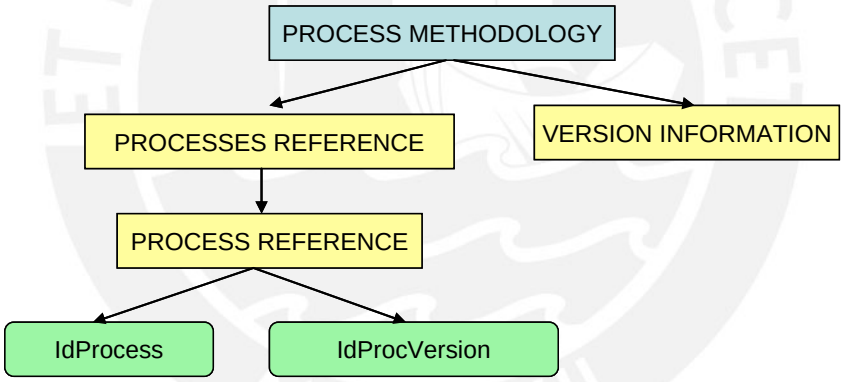


**Ilustración 3-8: Elemento ProcessMethodology**

**c. ProcessExplosion:**

Este elemento (ilustración 3-9) permite identificar un proceso que fue generado como consecuencia de la explosión de una actividad en un nuevo nivel. Dicho elemento contiene otros dos elementos que se encargan de guardar información propia de este tipo de proceso, los cuales son:

- **ProcessParent:** Este elemento guarda información sobre el proceso al cual pertenece la actividad que fue explosionada, para lo cual posee los siguientes atributos:
  - **IdProcess:** Identificador del proceso padre.
  - **IdProcVersion:** Número de versión del proceso padre.

- **ActivityParent:** Este elemento guarda información sobre la actividad que fue explosionada y que dio origen al nuevo proceso de tipo explosión. Los atributos que posee son:
  - **IdActivity:** Identificador de la actividad padre.
  - **Level:** Número de nivel en donde se encuentra dicha actividad.

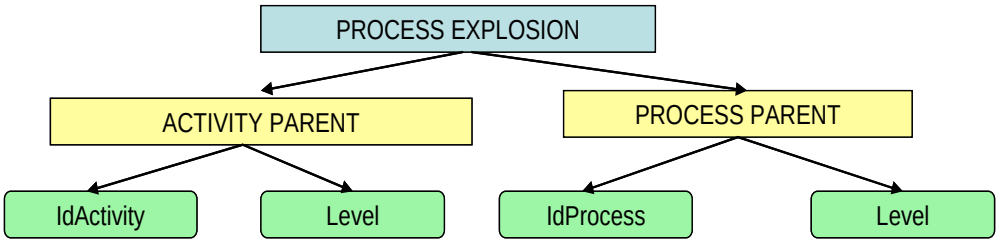


**Ilustración 3-9: Elemento ProcessExplosion**

A continuación, en el archivo fuente 3-11 se muestra la extensión del elemento WorkflowProcess en el correspondiente formato XPDL:

```
<xsd:complexType name="ProcessType">
   <xsd:sequence>
   <!-- Acá se colocan atributos propios del XPDL - ProcessType -->
   <!—- Acá se agregan los atributos añadidos por mjsprocess -->
   <xsd:element ref="mjs:ProcessCategory" minOccurs="1" />
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="ProcessCategory">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element name="ProcessBasic">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="mjs:VersionInformation"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="ProcessMethodology">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="mjs:VersionInformation" minOccurs="1" />
                  <xsd:element ref="mjs:ProcessesReference" minOccurs="1" />
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="ProcessExplosion">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="mjs:ProcessParent" minOccurs="1" />
                  <xsd:element ref="mjs:ActivityParent" minOccurs="1" />
               </xsd:sequence>
```

```
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

<xsd:element name="VersionInformation">
    <xsd:complexType>
        <xsd:attribute name="IdVersion" type="xsd:int" use="required" />
          <xsd:attribute name="CurrentVersion">
            <xsd:simpleType>
              <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="1" />
                <xsd:enumeration value="0" />
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
      </xsd:complexType>
</xsd:element>

<xsd:element name="ProcessesReference">
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="mjs:ProcessReference" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="ProcessReference">
    <xsd:complexType>
        <xsd:attribute name="IdProcess" type="xsd:string" use="required" />
        <xsd:attribute name="IdProcVersion" type="xsd:int" use="required" />
     </xsd:complexType>
</xsd:element>

<xsd:element name="ProcessParent">
    <xsd:complexType>
        <xsd:attribute name="IdProcess" type="xsd:string" use="required" />
        <xsd:attribute name="IdProcVersion" type="xsd:int" use="required"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="ActivityParent">
    <xsd:complexType>
        <xsd:attribute name="IdActivity" type="xsd:string" use="required" />
        <xsd:attribute name="Level" type="xsd:int" use="required"/>
    </xsd:complexType>
</xsd:element>
```

**Archivo Fuente 3-1: Extensión WorkflowProcess**

**B. Activity (Actividad)**

Este elemento contiene las características principales de una actividad. Del mismo modo que en el elemento WorkflowProcess, se considera necesaria la creación de un elemento que permita categorizar una actividad de acuerdo al tipo de proceso al cual pertenece.

Es así, que se crea un nuevo elemento denominado ActivitySourceType, el cual contiene otros tres elementos (ilustración 3-10) denominados ActivityBasic, ActivityMethodology y ActivityExplosion.

Los tres elementos anteriormente mencionados, además de cumplir la función de categorización de una actividad, guardan información propia acerca de cada uno de estos tres tipos de actividades.

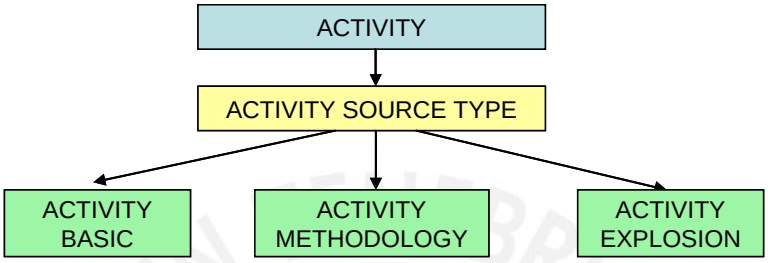


**Ilustración 3-10: Extensión Activity**

A continuación se detallan cada uno de los elementos por los que está compuesto el elemento **ActivitySourceType**:

**a. ActivityBasic:**

Este elemento no posee ningún atributo adicional, solamente actúa como elemento categorizador de una determinada actividad.

**b. ActivityMethodology:**

Este elemento permite identificar una actividad que pertenece a un proceso de tipo metodología y a su vez agrupa una serie de elementos que se encargan de guardar información propia de una actividad de este tipo (ilustración 3-11), entre los cuales se tiene:

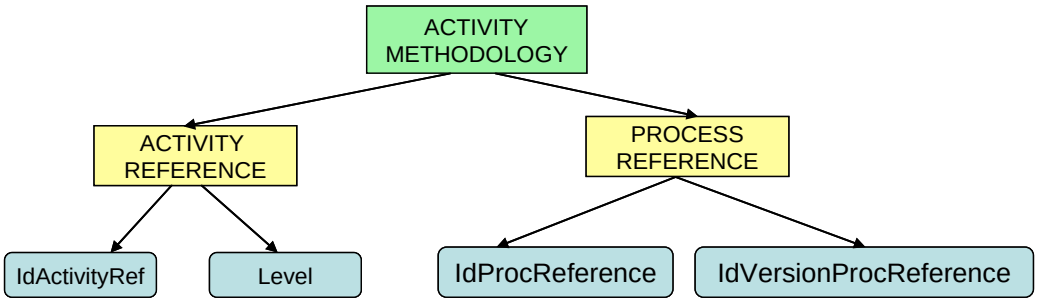


**Ilustración 3-11: Elemento ActivityMethodology**

- **ActivityReference:** Referencia a la actividad que fue extraída de un proceso base para conformar una metodología, para lo cual posee los siguientes atributos:
  - **IdActivityRef:** Identificador de la actividad referenciada.
  - **Level:** Número de nivel al que pertenece la actividad desde donde fue extraída.
- **ProcessReference:** Este elemento contiene información sobre el proceso al cual pertenece la actividad referenciada.

**c. ActivityExplosion:**

Este elemento permite identificar una actividad que pertenece a un proceso de tipo explosión y a su vez posee un atributo adicional denominado **Level** que permite identificar el nivel dentro del proceso en el cual se encuentra la actividad.

A continuación, en el archivo fuente 3-2 se muestra la extensión del elemento Activity en el correspondiente formato XPDL:

```
<xsd:element name="Activity">
   <xsd:complexType>
      <xsd:sequence>
         <!-- Acá se agregan los atributos añadidos por mjsprocess-->
         <xsd:element ref="mjs:ActivitySourceType" minOccurs="1" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="ActivitySourceType">
   <xsd:complexType>  <xsd:choice>
         <xsd:element name="ActivityBasic" />
         <xsd:element name="ActivityMethodology">
            <xsd:complexType>  <xsd:sequence>
                  <xsd:element ref="mjs:ActivityReference" minOccurs="1" />
                  <xsd:element ref="mjs:ProcessReference" minOccurs="1" />
            </xsd:sequence>  </xsd:complexType>
         </xsd:element>
         <xsd:element name="ActivityExplosion">
            <xsd:complexType>
               <xsd:attribute name="LevelNumber" use="required" />
            </xsd:complexType>
         </xsd:element>
    </xsd:choice>  </xsd:complexType>
</xsd:element>

<xsd:element name="ActivityReference">
   <xsd:complexType>
      <xsd:attribute name="IdActivityRef" type="xsd:string" use="required" />
      <xsd:attribute name="Level" type="xsd:int" use="required" />
   </xsd:complexType>
</xsd:element>
```

**Archivo Fuente 3-2: Extensión Activity**

**C. Input/Output (Entrada/Salida)**

Estos elementos contienen información sobre las entradas y salidas relacionadas a una actividad y a su vez hacen referencia a un determinado artefacto.

Para su extensión se ha incluido el concepto de **IOIdentifier**, el cual permite identificar una entrada o salida y a su vez diferenciar si estas fueron heredadas de un nivel anterior (en caso de una actividad de tipo explosión) o vinieron incluidas con la actividad extraída (en caso de una metodología).

El elemento **IOIdentifier** posee dos atributos:

- **PersonalId:** Identificador propio de la entrada o salida.
- **OwnerId:** Identificador del cual se hereda la entrada o salida.

A continuación, en el archivo fuente 3-3 se muestra la extensión del elemento Input/Output en el correspondiente formato XPDL.

```
<xsd:element name="Input">
   <xsd:complexType>
      <xsd:sequence>
         <!—- Acá se colocan los atributos propios del XPDL -->
         <!-- Acá se agregan los atributos añadidos por mjsprocess-->
         <xsd:element ref="mjs:IOIdentifier" minOccurs="0" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>


<xsd:element name="Output">
   <xsd:complexType>
      <xsd:sequence>
         <!—- Acá se colocan los atributos propios del XPDL -->
         <!-- Acá se agregan los atributos añadidos por mjsprocess-->
         <xsd:element ref="mjs:IOIdentifier" minOccurs="0" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>


<!—Elemento IOIdentifier definido por mjsprocess -->
<xsd:element name="IOIdentifier">
   <xsd:complexType>
      <xsd:attribute name="PersonalId" type="xsd:int" use="required" />
      <xsd:attribute name="OwnerId" type="xsd:int" use="required" />
   </xsd:complexType>
</xsd:element>
```

**Archivo Fuente 3-3: Extensión Input/Output**

### 3.6. Arquitectura de la Solución

En sus inicios la herramienta MJS Process Designer proponía contar con una arquitectura basada en una plataforma cliente - servidor, la cual implementaba el diagrama de capas presentado en la ilustración 3-12.

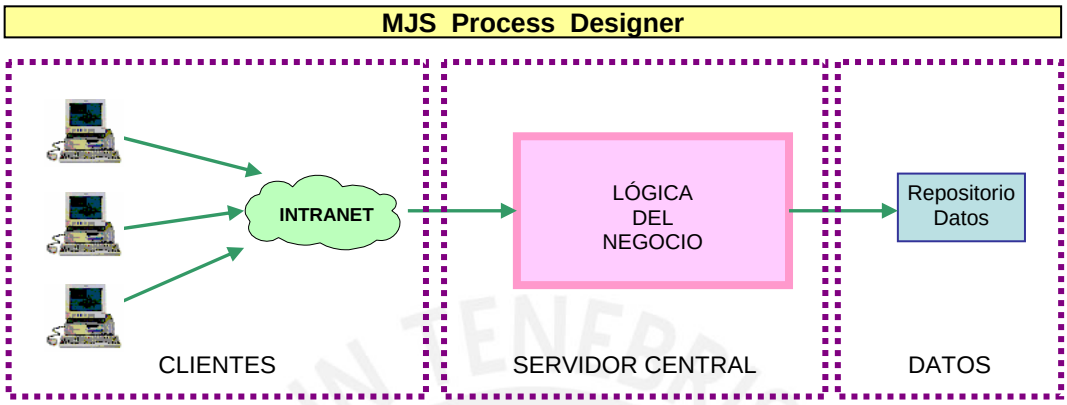


**Ilustración 3-12: Diagrama de Capas inicial de la Herramienta**

Esta arquitectura proporcionaba las siguientes ventajas:

- Los aspectos de seguridad y transaccionalidad estarían centralizados en el Servidor Central. De esta forma el control y administración de los mismos estaría solamente bajo la responsabilidad de la capa lógica del negocio.
- La lógica del negocio no estaría replicada en los clientes, lo cual permitiría que las modificaciones y mejoras a nivel de reglas del negocio se encuentren disponibles para el conjunto de usuarios que hicieran uso de la herramienta. Con esto se reduciría el costo de recursos y tiempo en el mantenimiento de la aplicación.
- Los accesos, los recursos y la integridad de los datos serían controlados por el servidor. Con ello se evitaría que un programa cliente defectuoso o no autorizado pudiese dañar el sistema.
- La arquitectura brindaría una gran escalabilidad al permitir aumentar la capacidad de clientes y servidores por separado.

Todas estas ventajas representaban puntos muy favorables para el mantenimiento y soporte del sistema a nivel de la lógica del negocio, pero ¿qué sucedería si se realizara alguna actualización en la interfaz del usuario? o ¿qué pasaría si el usuario no se encontrase dentro de la red interna donde se encuentra instalada la herramienta?

En el primer caso se tendría que volver a reinstalar la aplicación en todas las máquinas que utilizaran dicha herramienta, lo cual demandaría inversión de tiempo e interrupción de las labores cotidianas de los usuarios. En el segundo caso, el usuario no tendría acceso a la herramienta fuera de la red interna, lo cual limitaría su uso a un área geográfica muy

restringida. Todo esto representaba una desventaja frente al constante crecimiento y evolución del uso de aplicaciones a través de Internet.

Es así que se cambió la propuesta inicial de utilizar una plataforma cliente - servidor por una plataforma Web, ya que ello permitiría salvaguardar las dificultades anteriormente mencionadas. Al mismo tiempo, el trabajar sobre una plataforma Web facilita el uso de diversas tecnologías que dan la posibilidad de desarrollar un aplicativo mucho más estructurado e interactivo para el usuario.

En cuanto al número de capas físicas se mantiene el mismo número del diseño original, sin embargo, se plantea la división de la capa del negocio en tres capas lógicas, las cuales serán explicadas con un mayor detalle más adelante. El modelo de arquitectura de capas final para la herramienta queda plasmado en la ilustración 3-13.

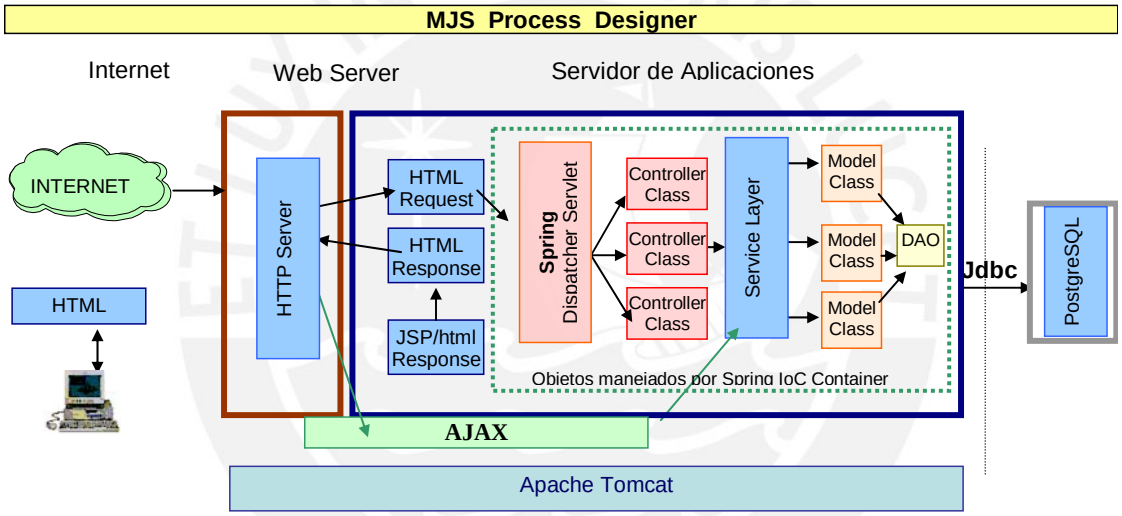


**Ilustración 3-13: Diagrama de Capas Final de la herramienta**

La arquitectura de la herramienta está diseñada bajo el patrón MVC (Modelo – Vista – Controlador), diseño que se puede apreciar en la ilustración 3-13.

A continuación se explican cada una de las capas que posee la arquitectura planteada:

**a. Capa de Vista**

Está formada por el conjunto de páginas Web que son accesadas por el usuario a través de un navegador Web (Browser).

**b. Capa de Controlador**

Está formada por las clases “Controller Class”, las cuales se encargan de recepcionar la petición de un usuario (HTML Request) y decidir qué información se le transmitirá y a través de qué vista como respuesta a dicha petición (HTML Response).

**c. Capa de Modelo**

Está formada por las clases Service y DAO (Data Access Object), las cuales en conjunto se encargan de realizar las operaciones de lógica del negocio y de acceso a la base de datos.

La elección de este patrón de diseño se debe a las fuertes ventajas que ofrece para el desarrollo de la herramienta, entre las cuales se tiene:

- Facilidad de desarrollo y acortamiento del "time to market" gracias al paralelismo de tareas debido a la existencia de una clara separación entre los componentes del sistema. Esta característica permite que los componentes puedan ser implementados de forma separada y al mismo tiempo por más de una persona.
- Permite tener diferentes vistas de usuario sin tener que realizar cambios radicales en la lógica del negocio. Actualmente existe una gran variedad de equipos que se pueden interconectar a una aplicación Web, por lo que es necesario implementar diversos tipos de vistas para una misma aplicación, con la finalidad que el usuario pueda acceder a ella.
- Facilita el cambio de base de datos, debido a que solamente se necesitaría modificar la capa de modelo.

Dentro de la capa de Modelo o capa de Lógica del Negocio se hace uso de otros dos patrones de diseño J2EE adicionales, los cuales son:

- **Business Delegate**

  Business Delegate actúa como una abstracción de la lógica del negocio en el lado del cliente y por lo tanto oculta la implementación de los servicios que brinda. De esta manera se reduce el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio del sistema.

  Si bien es cierto que en la arquitectura propuesta las peticiones de los usuarios no interactúan directamente con la lógica del negocio sino que lo hacen a través de la capa Controlador, el uso del patrón Business Delegate en conjunto con las clases de tipo Controller garantiza aún más la separación entre las funciones referentes a las capas de Vista y Lógica del Negocio.

- **Data Access Object (DAO)**

  El acceso a los datos varía dependiendo del tipo de almacenamiento (bases de datos, ficheros planos, archivos XML, entre otros) y de la implementación del vendedor, tal es así que en un momento dado una herramienta puede requerir utilizar un tipo determinado de base de datos y en un corto de tiempo requerir el uso de otro tipo de fuente de datos.
  La solución a este problema es el uso del patrón DAO, el cual permite abstraer y encapsular todos los accesos a la fuente de datos. Los objetos DAO se encargan del

manejo de la conexión con la fuente de datos para realizar operaciones de consulta y/o almacenar información.

Como se mencionó anteriormente, gracias al uso del patrón MVC algún cambio en las reglas del negocio o el acceso a los datos sólo se vería impactado en toda la capa de Lógica del Negocio. Con el uso de los patrones Business Delegate y Data Access Object, se logra que estos dos tipos de impactos ocurran en ámbitos reducidos, tal es así que los cambios en las reglas del negocio se verían reflejados en la capa de servicios y los cambios en el acceso a datos se verían reflejados en la capa DAO.

En conclusión, el uso en conjunto de estos dos últimos patrones de diseño dentro de la capa del negocio brinda una mayor facilidad para el desarrollo y mantenimiento de la herramienta.

### 3.6.1. Diagrama de Despliegue

En la ilustración 3-14 se presenta el diagrama de despliegue en donde se visualizan los nodos físicos que posee la herramienta.

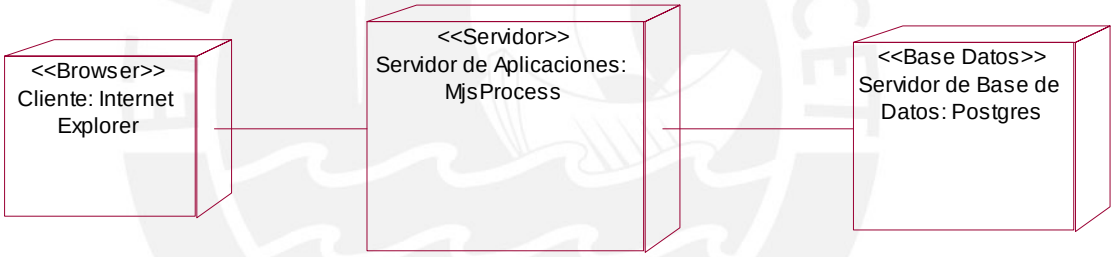


**Ilustración 3-14: Diagrama de Despliegue**

- **Cliente**

  Corresponde a la interfaz de usuario a través de la cual los clientes acceden a la herramienta haciendo uso de un navegador Web, que para el caso de la herramienta MJS Process Designer es el Internet Explorer 6.0.

- **MJSProcess**

  Nodo principal de la herramienta que contiene las reglas del negocio para las funciones de modelado de procesos, gestión de versiones, creación de metodologías, explosión de las actividades en niveles y generación de archivos XPDL.

- **Base de Datos**

  Repositorio que contiene la información registrada por los usuarios acerca de los procesos que han definido durante el transcurso del tiempo.

### 3.6.2. Diagrama de Componentes

A continuación se presentan los componentes principales que se encuentran en la herramienta MJS Process Designer (ilustración 3-15). Posteriormente, en el cuadro 3-1 se detalla la descripción de cada uno de ellos.

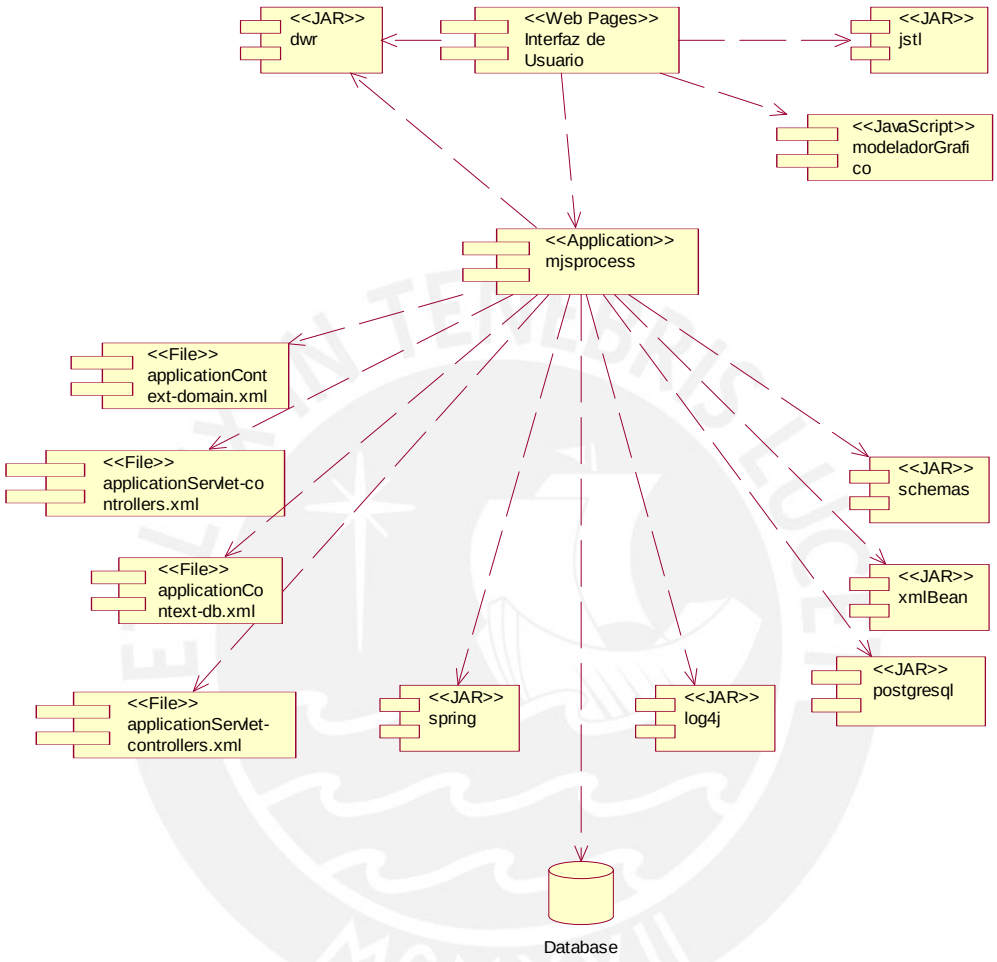


**Ilustración 3-15: Diagrama de Componentes**

| CAPA DE VISTA | |
|---|---|
| **Interfaz de Usuario** | Conjunto de archivos de extensión .jsp que conforman la interfaz Web con la que interactúa el usuario. |
| **JSTL** | Es una librería que implementa funciones de uso frecuente en aplicaciones JSP y sirve como lenguaje de expresión para referenciar objetos y sus propiedades sin necesidad de usar código Java. |
| **ModeladorGrafico** | Conjunto de funciones JavaScript que haciendo uso del lenguaje VML (Vector Markup Language) proporcionan al usuario funciones |

| | |
|---|---|
| | de modelado gráfico que permiten diagramar los siguientes elementos: Actividad, conector, transición, eventos de inicio y fin, entre otros. |
| **CAPA MODELO** | |
| **MJSProcess** | Componente principal de la herramienta que agrupa las funcionalidades básicas: Modelado de procesos, creación de metodologías, explosión de actividades en niveles y gestión de versiones. |
| **ApplicationContext-Domain.xml** | Archivo de configuración que se encarga de mapear los objetos del negocio usados por la herramienta. |
| **ApplicationServlet-handlerMapping.xml** | Archivo de configuración que contiene el mapeo de los controladores con los archivos JSP a través de los cuales son invocados. |
| **ApplicationServlet-controllers.xml** | Archivo de configuración que contiene el registro de los controladores usados por la herramienta. |
| **Schemas.jar** | Contiene los métodos adecuados que permiten que la herramienta genere el archivo XPDL correspondiente a la definición de un proceso. |

**Cuadro 3-1: Componentes del Sistema MJS Process Designer**

### 3.6.3. Tecnologías usadas en la herramienta

En el primer capítulo se hace referencia a una lista de tecnologías Java Web, en donde se define cuál es el propósito de cada una de ellas y cuáles son sus características principales. Dentro de la arquitectura de la herramienta se han utilizado algunas de estas tecnologías debido a las diferentes ventajas que ofrecen para el cumplimiento de las características funcionales de la misma.

A continuación se listan cuáles son las tecnologías usadas dentro de cada una de las capas de la arquitectura de la herramienta MJS Process Designer y a su vez se menciona una breve descripción del porqué de su elección:

#### A. Frameworks de Presentación

En el primer capítulo se explicó el concepto de framework de presentación para el desarrollo de aplicaciones Web. También se enumeraron las características de los principales framework disponibles en el mercado, tales como: Struts, JSF y Spring.

Estos últimos son los que presentan mayor difusión en el mercado de software [74], mejor documentación disponible y el respaldo de las respectivas organizaciones por las que fueron desarrollados.

Para el presente trabajo de tesis se eligió Spring como framework de presentación debido a la ventaja que representa su capacidad de trabajar en unión de otras API o frameworks como Struts, JSF, Hibernate, entre otros; lo cual no siempre se cumple en el caso de elegir Struts o JSF.

Adicionalmente Spring, mediante el patrón de diseño de Inversión de Control (IoC), simplifica la tarea del desarrollador al asumir el manejo y control de los beans y transacciones utilizadas dentro de la aplicación; a diferencia de Struts y JSF que dejan mayor responsabilidad en el lado del desarrollador.

**B. Framework de Persistencia**

Tal como se explicó en el primer capítulo, existen en la actualidad diversos frameworks de persistencia que se pueden utilizar en el desarrollo de aplicaciones Java Web con el fin de facilitar el mapeo con las bases de datos relacionales.

Entre los frameworks de persistencia que podían ser utilizados en la implementación de la herramienta MJS Process Designer se encontraban: JDBC, Hibernate e iBatis; los cuales fueron escogidos debido a su naturaleza open source y a la integración que brinda Spring para trabajar con ellos.

Para la elección del framework de persistencia que se utilizaría durante el desarrollo de la herramienta se optó por comparar dos tipos de frameworks: Hibernate, perteneciente a la categoría ORM (Object Relational Mapper) y JDBC perteneciente a una categoría diferente. Debido a que cada uno de ellos presenta ventajas y desventajas en su uso, se optó por escoger aquel framework que se ajustara más a las necesidades de la aplicación.

Entre las características a tomar en cuenta para la elección del framework de persistencia se encuentran:

- **Facilidad de uso**: Hibernate utiliza su propio lenguaje HQL para la elaboración de sentencias de acceso a base de datos, lo que conlleva a que se requiera más tiempo para su aprendizaje. Adicionalmente, el uso de un lenguaje propio obliga a la elaboración de sentencias SQL más complejas de lo normal. En cambio, JDBC utiliza en sus sentencias el código nativo de SQL, el cual por ser de uso estándar es más fácil de entender y utilizar.

- **Estandarización:** El uso de un lenguaje propio por parte de Hibernate ocasiona que las aplicaciones estén sujetas a este framework en todo momento, ya que ante un posible cambio se deberá realizar una traducción de las sentencias generadas en HQL a sentencias SQL estándar. En cambio, JDBC es la interfaz nativa para conectarse a las

bases de datos desde aplicaciones desarrolladas en lenguaje Java, por lo que la ejecución de sus sentencias es más rápida ya que no necesita realizar ninguna traducción.

A pesar de las ventajas que ofrece JDBC frente a Hibernate, al usar este framework se produce cierta pérdida de la portabilidad de la aplicación con respecto a la base de datos. Esto se debe a que cada DBMS (Sistema Administrador de Base de Datos) implementa el lenguaje estándar SQL de una manera distinta, por lo que las sentencias de acceso a la base de datos estarían ligadas al DBMS utilizado.

Una manera de mitigar este inconveniente es a través del uso de las interfaces DAO, las cuales encapsulan y aíslan el acceso a los datos de la lógica del negocio. Con ello, ante cualquier cambio de la base de datos utilizada por la aplicación, sólo sería necesario modificar la capa de acceso a datos (DAO) sin alterar la capa de lógica del negocio.

**C. Servidor Web y de Aplicación**

En el mercado existe una gran variedad de servidores Web y servidores de aplicación de tipo open source, siendo dos de los principales JBoss y Apache Tomcat. Existe una diferencia entre estos servidores, mientras que JBoss es un servidor de aplicaciones que puede manejar EJB, Tomcat es un servidor Web que sirve a su vez como contenedor de servlets y JSP.

Dado que para la implementación de la herramienta no se ha considerado el uso de EJB, sino solamente el uso de JSP y servlets, se optó por usar Tomcat como servidor Web y contenedor de estos dos últimos elementos.

Otro motivo por el cual se decidió usar Tomcat en lugar de JBoss es la diferencia de tiempos de inicialización requeridos por estos servidores al realizar algún cambio durante la etapa de desarrollo. Mientras que Tomcat necesita aproximadamente diez segundos para ser inicializado, JBoss requiere un mínimo de dos minutos para la misma operación, lo cual genera tiempos muertos durante el desarrollo del proyecto.

**D. Sistema Administrador de Base de Datos**

Para el presente proyecto se realizó una comparación entre PostgreSQL y MySQL encontrándose las siguientes diferencias:

- **Integridad referencial:** PostgreSQL implementa el uso de rollback, subconsultas y transacciones haciendo su funcionamiento mucho más eficaz en comparación con MySQL, el cual carece del soporte para estos tipos de transacciones.

- **Velocidad:** La principal ventaja de MySQL es su velocidad a la hora de realizar las operaciones, esto lo convierte en uno de los gestores de base de datos que ofrecen mayor velocidad de respuesta.

- **Escalabilidad:** PostgreSQL posee una mayor escalabilidad que MySQL debido a que soporta mayor cantidad de peticiones simultáneas de manera correcta, con lo que permite que múltiples usuarios puedan acceder a la aplicación al mismo tiempo.

Dado que ninguno de estos dos DBMS cumplía en su totalidad con los objetivos no funcionales de la herramienta, se tuvo que elegir aquel administrador de base datos que se orientara hacia las características más trascendentales para ella: escalabilidad e integridad referencial. Es así que se decide optar por PostgreSQL como Sistema Administrador de Base de Datos.

### E. Tecnologías de Presentación

Para el presente proyecto se optó por utilizar las tecnologías JSTL y AJAX con la finalidad de crear una aplicación interactiva y de fácil uso para el usuario.

#### a. JSP/JSTL

Se optó por utilizar JSTL durante la implementación de la herramienta por las siguientes razones:

- Evita la inclusión de código Java (scriptlets) dentro del código fuente JSP. Con ello se logra un mayor orden en la implementación de la capa de presentación.
- Evita la inclusión de las clases propias de la lógica del negocio dentro del código de los formularios, debido a que ya no será necesario realizar operaciones de cast en la recuperación de datos desde los objetos request y session.
- Facilita el mantenimiento de los formularios al separar la lógica del negocio (código Java) de la capa de presentación (JSP y HTML), facilitando el trabajo de los diseñadores.

#### b. AJAX

Se optó por utilizar AJAX durante la implementación de la herramienta por las siguientes razones:

- Mejora la interactividad de la herramienta evitando que el usuario tenga que esperar hasta que llegue la totalidad de los datos desde el servidor para poder visualizar el formulario.
- Facilita la portabilidad de la herramienta al no requerir el uso de los componentes como Flash y Applet.
- Aumenta la velocidad de la aplicación al evitar que se recargue toda la página cada vez que se realice alguna operación.

### 3.7. Diagrama de clases de Diseño

Las clases de diseño utilizadas en el sistema MJS Process Designer se clasifican en cuatro tipos:

- **Tipo Controller:** Clases que sirven de intermediarias entre las peticiones del cliente y la lógica del negocio.
- **Tipo Service:** Interfaces que contienen toda la funcionalidad referente a la lógica del negocio del sistema y que son implementadas por clases de tipo **ServiceDb.**
- **Tipo DAO:** Interfaces que se encargan del acceso a base de datos para realizar operaciones transaccionales y de consulta. Las clases de tipo **DAOJdbc** se encargan de implementar dichas interfaces.
- **Tipo Entity:** Clases que representan las entidades de negocio utilizadas por el sistema.

La relación entre este tipo de clases (ilustración 3-16) está basada en el patrón MVC, el cual plasmado en el sistema MJS Process Designer se muestra a continuación.

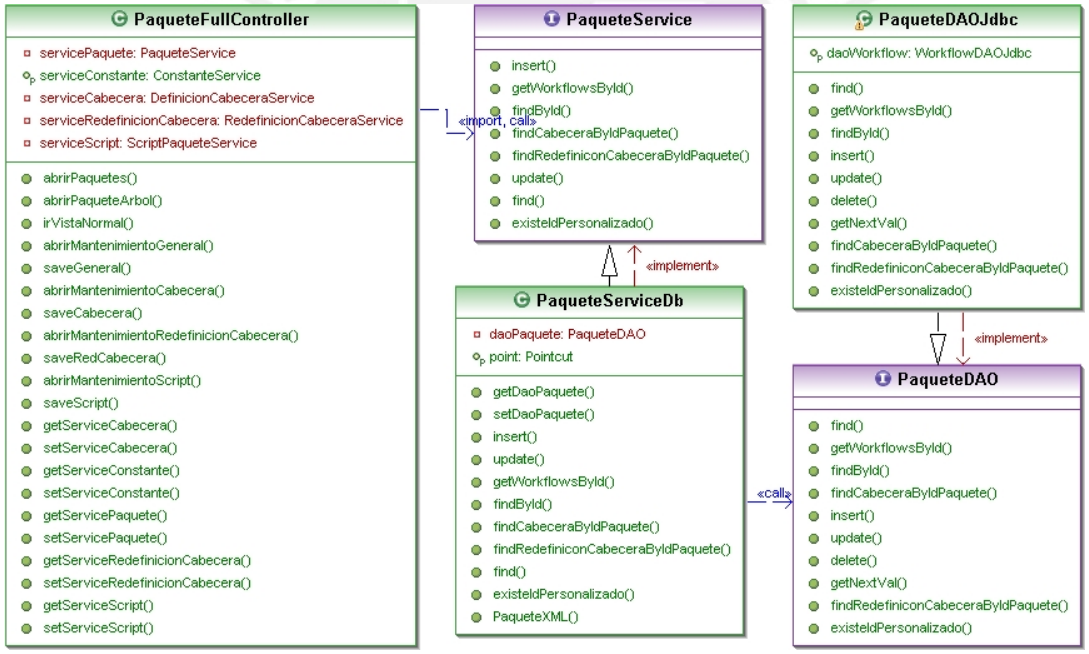


**Ilustración 3-16: Tipos de Clase de Diseño**

En la ilustración 3-17 se muestra el diagrama de clases para el metamodelo Paquete. Para facilitar el entendimiento y visualización de este diagrama, solamente la clase principal Paquete muestra el esquema de conexión completo, mientras que para las demás clases se muestra únicamente el esquema hasta el nivel de interfaces Service y/o Data Access Object (DAO) dado que se sigue el mismo patrón para todas.

Los demás diagramas se detallan en el Anexo E: Documento de Diseño Detallado.

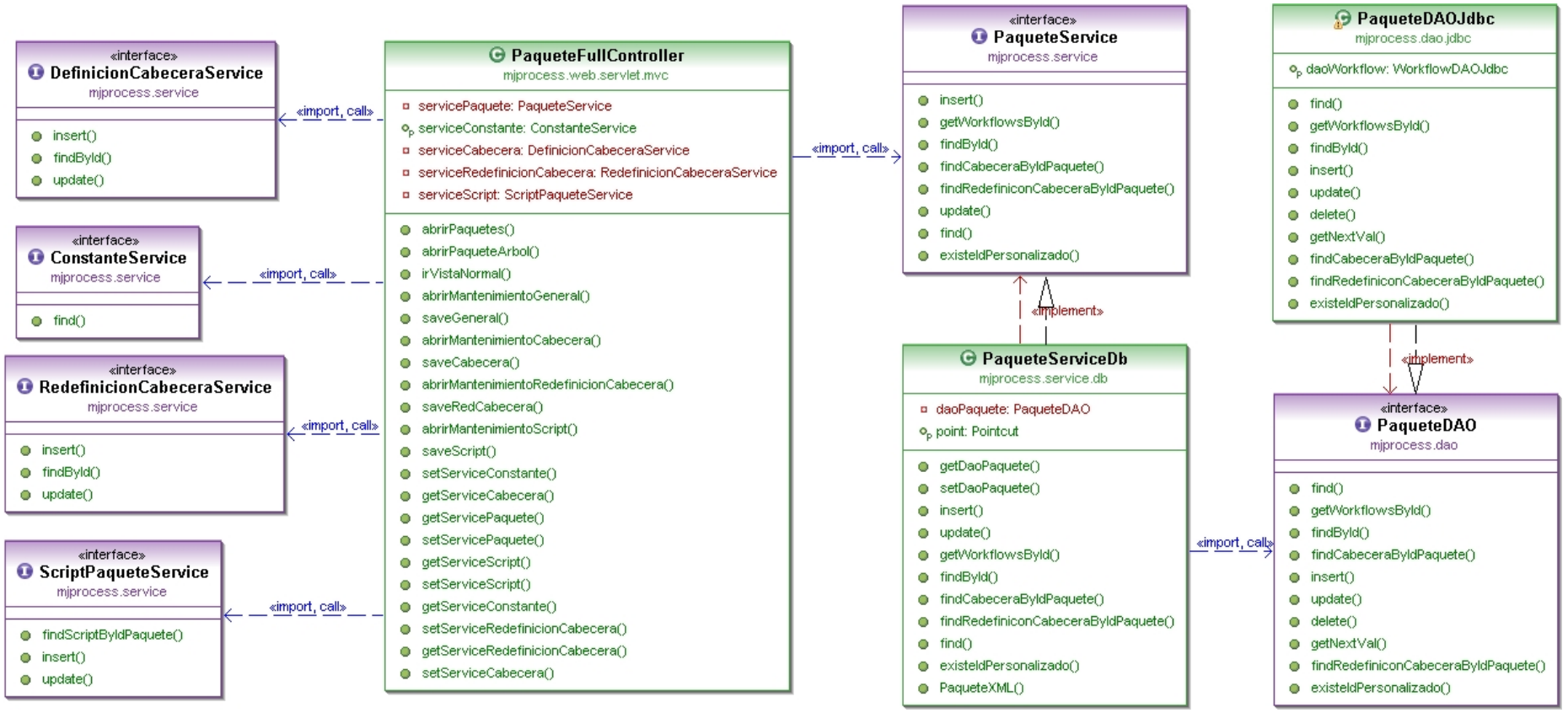


**Ilustración 3-17: Diagrama de Clase de Diseño para el metamodelo Paquete**

### 3.8. Diagrama de Secuencias

En el diagrama de secuencia se muestra la interacción entre las clases que conforman la herramienta. Con estos diagramas se puede observar la interacción de estas clases dentro de un escenario determinado.

En la ilustración 3-18 se presenta el diagrama de secuencia para el caso de uso "Crear Paquete". En el Anexo E: Documento de Diseño Detallado se presenta los diagramas de secuencias principales del sistema MJS Process Designer.

### 3.9. Módulos de la herramienta

A continuación se presenta la descripción de cada uno de los módulos que conforman la herramienta MJS Process Designer:

#### 3.9.1. Modelador de Procesos

La actividad de toda organización, sin importar el ramo o área a la que pertenezca, está basada en un conjunto de procesos que definen cómo y cuándo se llevan a cabo las diferentes actividades o tareas a realizar.

Una correcta administración y gestión de la definición de procesos está basada en la facilidad con la que el usuario puede modelar y dar mantenimiento a los procesos que se manejan dentro de una organización. Con ello, el usuario puede realizar una revisión constante de los procesos e identificar posibles puntos factibles de optimización.

La gestión de procesos le permite a la organización implementar una política de mejora continua de los mismos y como consecuencia de ello, aumentar la eficacia y productividad de la organización.

El módulo Modelador de Procesos es considerado el principal módulo de la herramienta debido a que permite al usuario modelar las definiciones de los procesos manejados en la organización a la que pertenece. La herramienta MJS Process Designer brinda las funcionalidades necesarias para el modelado de definiciones de procesos en general, pero para el presente trabajo de tesis, la herramienta se enfocará en los procesos de una organización desarrolladora de software.

La herramienta utiliza como entidad raíz o base el elemento paquete del XPDL, el cual actúa como contenedor de las definiciones de procesos y de los elementos relacionados a estas. El uso de la herramienta MJS Process Designer implica como primer paso la creación de una nueva definición de paquete o abrir una definición previamente registrada en el sistema.

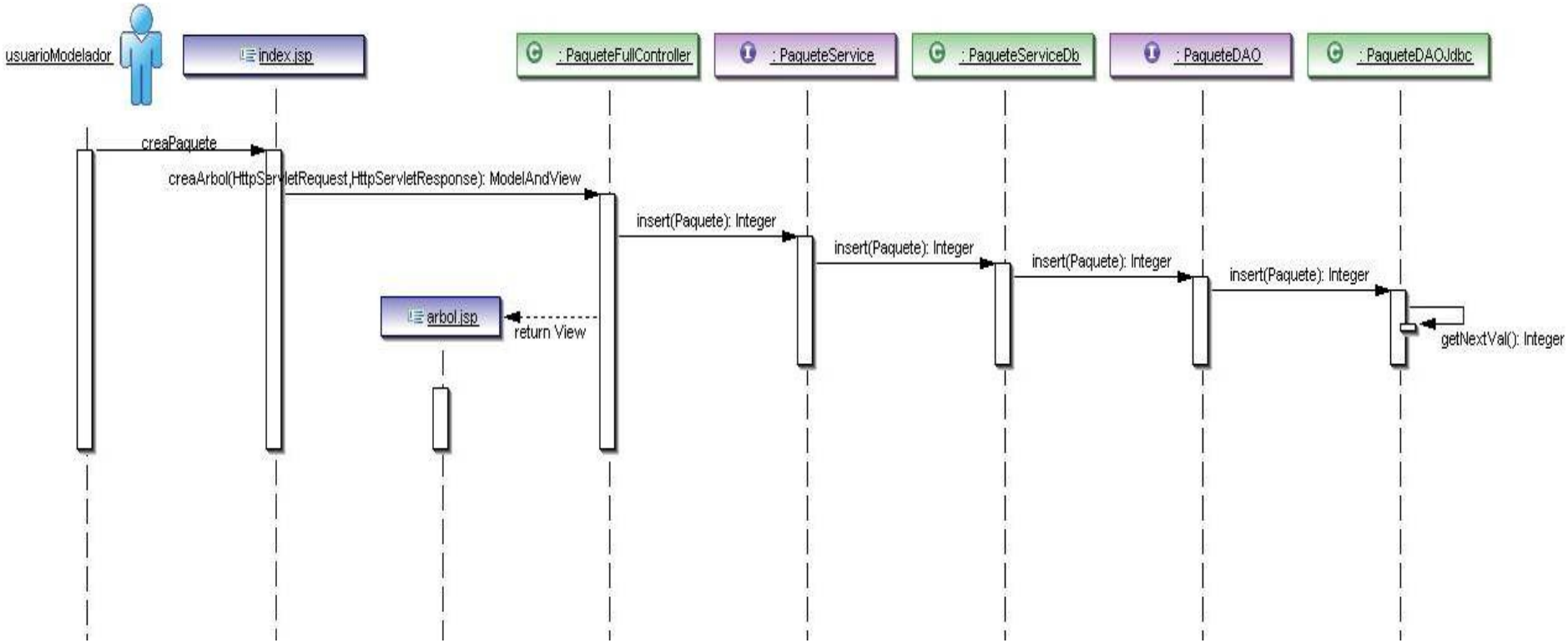


**Ilustración 3-18: Diagrama de secuencia para “Crear Paquete”**

Una vez que el usuario cargue la definición de un paquete en el sistema, se podrá visualizar un árbol en el cual se listan todas las versiones vigentes de las definiciones de los procesos y set activities registrados dentro del mismo. Si el usuario desea ver el registro histórico de las versiones existentes de una definición de proceso deberá utilizar el módulo Gestor de Versiones, el cual será explicado más adelante.

El módulo Modelador de Procesos brinda al usuario dos tipos de interfaces con la finalidad de realizar el modelado gráfico y semántico de los procesos y metodologías. Estos tipos de interfaces son:

**a. Interfaz Gráfica:**

Esta interfaz permite al usuario realizar el modelado gráfico del proceso, en otras palabras, diagramar el flujo de trabajo del mismo. El modelador gráfico de la herramienta MJS Process Designer utiliza la notación gráfica BPMN, la cual abarca la mayoría de elementos definidos en el lenguaje XPDL.

Uno de los componentes que brinda la interfaz gráfica es la barra de herramientas de diagramado, en la cual el usuario tendrá disponibles los distintos tipos de elementos posibles de graficar dentro de una definición de proceso. Los elementos gráficos que pueden ser incluidos en la definición de un flujo de trabajo son los siguientes: Actividad, evento de inicio y fin, entrada y salida, conector, transición y texto libre.

El modelador de procesos permite al usuario utilizar los siguientes tipos de actividades:

- **Actividad Básica:** Este elemento es considerado como el componente básico de todo proceso, debido a que por definición representa cualquier actividad realizada dentro de un flujo de trabajo.
- **Actividad Loop:** Este elemento representa una actividad que podrá repetir su ejecución dentro de un flujo de trabajo en más de una ocasión de acuerdo a una determinada condición.
- **Actividad Subproceso:** Este elemento permite al usuario asociar a un flujo de trabajo la referencia a otro proceso, cuya ejecución se considerará dentro de la ejecución del proceso principal.
- **Actividad BlockActivity:** Este elemento hace referencia a una agrupación de actividades y transiciones que conforman un nuevo flujo de trabajo.

El flujo de un proceso se define en base a transiciones, las cuales unen las actividades que conforman el proceso, definiendo así una secuencia u orden de ejecución para las mismas. La herramienta MJS Process Designer proporciona al usuario dos tipos de transiciones: Lineal y poli-línea; esto con el fin de facilitar el diagramado de procesos complejos y grandes.

Adicionalmente, la herramienta proporciona cuatro tipos de conectores: conector simple, conector AND, conector OR y conector XOR; los cuales permiten que el usuario defina diferentes flujos de ejecución para un mismo proceso en base a la evaluación de parámetros previamente definidos.

La definición de procesos incluye también los artefactos utilizados y/o generados (inputs y outputs) por las actividades que lo conforman. Para el presente trabajo de tesis se considera como artefactos a los documentos que son generados durante la ejecución de un proceso de desarrollo de software; por ejemplo: Documento de Análisis, Catálogo de Requisitos, Lista de exigencias, Manual de Usuario, entre otros.

Durante el proceso de diagramado se consideró necesario grabar las entidades graficadas hasta el momento antes de realizar el mantenimiento de la información referente a ellas. Esto con el fin de asegurar la consistencia del modelo gráfico visualizado y la información almacenada en el repositorio de datos.

Una vez grabada la información grafica de las entidades modeladas, el usuario podrá seleccionar un elemento en el área de diagramado y acceder a la lista de los mantenimientos que tiene asociado. Dependiendo de la opción de menú elegida se procederá a cargar el formulario en el área de mantenimiento.

Adicionalmente, como parte del modelado gráfico se ha considerado llevar a cabo la validación de algunas reglas básicas de diagramado, con el fin de asegurar la consistencia del modelo de procesos registrados en la herramienta. Entre las reglas de modelado que serán validadas se encuentran las siguientes:

- Obligatoriedad y unicidad de nodos de inicio y fin de proceso.
- Validación de flujos de trabajo incompletos.
- Existencia de actividades aisladas o sueltas en el modelo de proceso.

**b. Interfaz de Formularios:**

Esta interfaz permite al usuario realizar el mantenimiento semántico de las definiciones de paquete, proceso y elementos relacionados a ellos.

En el lenguaje XPDL existen algunos elementos que pueden ser definidos en dos niveles distintos: nivel de paquete o nivel de proceso. Los elementos definidos a nivel de paquete podrán ser utilizados por los procesos que se encuentren contenidos en dicho paquete; en cambio, un elemento que es definido a nivel de proceso sólo podrá ser utilizado dentro del ámbito del mismo.

Entre los principales mantenimientos de la entidad paquete se encuentran:

- La definición y redefinición de cabecera, en donde se registra la data de identificación del paquete.
- El mantenimiento de aplicaciones, las cuales podrán ser utilizadas por las actividades definidas en un proceso perteneciente al paquete.
- El mantenimiento de participantes, los cuales podrán ser asignados como recursos para la ejecución de una actividad.
- El mantenimiento de artefactos, los cuales podrán ser utilizados como entradas y/o salidas de las actividades pertenecientes a un proceso.

Entre los principales mantenimientos de la entidad proceso se encuentran:

- La definición y redefinición de cabecera, en donde se registra la data de identificación del proceso.
- El mantenimiento de aplicaciones, las cuales podrán ser utilizadas por las actividades que conforman el proceso.
- El mantenimiento de participantes, los cuales podrán ser asignados como recursos para la ejecución de una actividad.
- El mantenimiento de set activities, los cuales podrán ser referenciados dentro de un proceso mediante las actividades de tipo block activity.

Adicionalmente, la interfaz de formularios provee al usuario un área de ayuda, en donde se podrá visualizar las descripciones de cada uno de los campos pertenecientes a un formulario.

La definición de un proceso puede pasar por tres estados de publicación a lo largo de su ciclo de vida, los cuales son: “Bajo revisión”, “Bajo prueba” y “Publicado”.

### 3.9.2. Explosión de Niveles

Los procesos dentro de una organización pueden variar en complejidad dependiendo de la importancia que tengan dentro de la lógica del negocio manejada por la organización. Como consecuencia de ello es posible encontrar desde modelos de procesos complejos y recargados hasta modelos simples y básicos. Un proceso puede ser parte de otro proceso mayor que lo abarque, es así, que un proceso de negocio puede ser visto en varios niveles de granularidad.

Uno de los objetivos de modelar es reflejar de forma precisa y detallada el flujo de trabajo a llevarse a cabo en un proceso, pero igual de importante es lograr que el modelo de proceso generado sea de fácil entendimiento para el usuario.

Es así que se define la implementación del módulo de Explosión de Niveles por actividad, con la finalidad de brindarle al usuario la capacidad de poder modelar un proceso con mayor nivel de detalle, así como facilitar las tareas de auditoria y seguimiento sobre los procesos de la organización.

El usuario podrá explosionar una actividad de tipo básico en un nuevo nivel compuesto por dos o más actividades. Esta operación se podrá realizar las veces que el usuario lo requiera, obteniéndose con cada nuevo nivel un mayor detalle con respecto al nivel superior inmediato. Adicionalmente, la herramienta permitirá validar la consistencia de las entradas y salidas compartidas entre niveles contiguos.

Para la implementación del módulo de Explosión de Niveles se consideró necesario realizar las siguientes extensiones al lenguaje XPDL: ActivityParent, ActivitySourceType y ActivityExplosion, las cuales fueron explicadas anteriormente.

El módulo de Explosión de Niveles tiene las siguientes funcionalidades:

- Permite explosionar una actividad en un nuevo nivel compuesto por dos o más actividades.
- Permite realizar el mantenimiento del modelado gráfico y semántico de cada uno de los elementos que conforman el nuevo nivel.
- Permite la visualización del modelo de proceso en un árbol jerárquico, en el cual se mostrarán los niveles resultantes de la explosión de alguna actividad.

Los mantenimientos gráficos y semánticos de los elementos pertenecientes a los niveles creados se realizarán de manera idéntica a los explicados en el módulo Modelador de Procesos.

### 3.9.3. Definición de Metodologías

Dentro de una organización se pueden presentar diferentes entornos o contextos de trabajo en los cuales se llevan a cabo los procesos involucrados en su actividad diaria. Incluso a nivel de proyectos no siempre se utiliza un mismo flujo de trabajo debido a que no todos los proyectos tienen el mismo alcance. Como consecuencia de lo anteriormente expuesto surge la necesidad de adaptar las definiciones de procesos a los diferentes entornos de trabajo mediante la definición de metodologías.

Una metodología se compone de un conjunto de actividades extraídas de diferentes definiciones de procesos base pertenecientes a un mismo paquete. Estas actividades al ser relacionadas de forma adecuada, se transforman en un flujo de proceso personalizado que se adapta de forma más eficiente al contexto de trabajo de una organización o proyecto específico.

Es así que se define la implementación del módulo de Definición de Metodologías con la finalidad de brindarle al usuario la capacidad de personalizar los flujos de trabajo base existentes en su organización según las necesidades del momento.

Este módulo puede ser accedido a través de la vista Metodología de la herramienta MJS Process Designer, la cual brinda las siguientes funcionalidades:

- Permite seleccionar los procesos base de los que se extraen las actividades que conforman la metodología.
- Permite el mantenimiento del modelado gráfico y semántico de la metodología.
- Permite la visualización del árbol de metodología, en el cual se listan los procesos base referenciados por la metodología y las actividades que lo conforman agrupadas de acuerdo al nivel al cual pertenecen.

Al igual que el proceso, una metodología tiene disponible tres estados de publicación: “Bajo revisión”, “Bajo prueba” y “Publicado”. Por defecto, al crearse una nueva metodología el estado de publicación asignado es “Bajo revisión”.

El modelado gráfico de la metodología es similar al modelado gráfico de un proceso, con la diferencia que en la barra de herramientas de dibujado no están habilitadas las opciones para crear actividades. Estas restricciones de dibujado se deben a que las únicas actividades que pueden conformar una metodología son aquellas que se encuentran disponibles dentro de los procesos base seleccionados.

Otra característica particular en la definición de una metodología es que las actividades escogidas mantienen todas las asociaciones definidas en su proceso original, las cuales no podrán ser eliminadas al haber sido definidas en su proceso de origen. Sin embargo, el usuario podrá agregar nuevas asociaciones a las actividades que forman parte de una metodología.

El mantenimiento de cada uno de los elementos que conforman el flujo de una metodología es similar al realizado en el módulo Modelador de Procesos. La información de las actividades y asociaciones que son extraídas de los procesos base solamente podrá ser visualizada sin poder modificarse.

Cabe resaltar que la herramienta mantendrá en todo momento la consistencia ante posibles variaciones en los elementos de los procesos base referenciados. De esta forma, cualquier modificación realizada sobre los elementos extraídos de los procesos originales se verá reflejada en la definición de la metodología automáticamente.

Para la implementación del módulo de Definición de Metodologías se consideró necesario realizar las siguientes extensiones del lenguaje XPDL: ProcessMethodology, ActivityMethodology, ProcessReference, ActivityReference; las cuales fueron explicadas anteriormente.

### 3.9.4. Gestión de Versiones

Los procesos desarrollados en una organización, al igual que las metodologías definidas, pueden variar a lo largo del tiempo debido a diversos motivos, tales como: Modificaciones en las reglas del negocio manejadas por la empresa, cambios en los roles o participantes asociados a alguna actividad, refinamiento de procesos, reemplazo o actualización de actividades, entre otros.

En base a lo anteriormente expuesto, se hace necesaria la generación y administración de versiones de los procesos y las metodologías definidos en una organización. Dicha gestión brinda las siguientes ventajas:

- Permite un correcto control de cambios.
- Asegura la consistencia histórica de los procesos y metodologías manejados en la empresa.
- Facilita la constante revisión y mejora de procesos y metodologías.
- Determina con exactitud cuál es la versión de proceso o metodología vigente en la organización.

Es así que se define la implementación del módulo de Gestión de Versiones con la finalidad de brindarle al usuario la capacidad de generar nuevas versiones y visualizar el cambio que han sufrido los procesos y metodologías a través del tiempo.

Para poder utilizar las funcionalidades de este módulo se debe acceder a la vista Versión. Esta vista contiene un árbol donde se listan todas las versiones existentes del proceso o metodología con el que se esté trabajando.

Con el fin de mantener la coherencia histórica entre las distintas versiones, la herramienta controlará que sólo se puedan realizar modificaciones a un proceso o una metodología que tenga activo el indicador de versión actual. De esta forma, si se quisiera modificar alguna versión anterior se deberá crear previamente una nueva versión basada en la versión que se desee cambiar.

Para la implementación del módulo de Gestión de Versiones se consideró necesario realizar la creación de un nuevo elemento XPDL denominado VersionInformation, el cual fue explicado anteriormente.

### 3.10. Plan de Pruebas

Para probar el correcto funcionamiento de la herramienta MJS Process Designer se elaboró un plan de pruebas que permita identificar errores durante su funcionamiento, esto con el objetivo de poder corregirlos antes de finalizar la elaboración del sistema.

Con el fin de probar los casos de uso especificados en el ERS (Anexo C: Especificación de Requisitos de Software) se elaboró un conjunto de pruebas de tipo funcional.

A continuación se presentan algunos de los principales casos de pruebas realizados. La lista completa de estos casos de prueba se encuentra en el Anexo G: Plan de Pruebas.

| Guardar definición proceso | |
|---|---|
| **ID Prueba:** | UC 25-1 |
| **Objetivo :** | Diagramar el modelado gráfico de un proceso y guardarlo. |
| **Precondición:** | La vista de proceso está activa. Existe un paquete abierto y dentro de éste existe al menos un proceso creado. |
| **Proceso:** | 1. Seleccionar un proceso en el árbol de procesos.<br>2. Dibujar los elementos gráficos del proceso (actividades, transiciones, asociaciones, entre otros) en el área de diagramado.<br>3. Presionar el botón “Guardar”.<br>4. Esperar a que termine de guardar (el avance del guardado se puede apreciar en la barra de estado situada en la parte inferior de la pantalla). |
| **Resultado Esperado:** | Se guarda la definición gráfica del proceso. |

| Explosionar una actividad | |
|---|---|
| **ID Prueba:** | UC 30-1 |
| **Objetivo :** | Explosionar una actividad en sub-niveles. |
| **Precondición:** | La vista de proceso está activa. Existe un paquete abierto y dentro de éste existe al menos un proceso creado con su modelado gráfico. |
| **Proceso:** | 1. Seleccionar un proceso en el árbol de procesos.<br>2. Seleccionar la opción “Ir a la vista de explosión”.<br>3. Seleccionar el proceso en el árbol de procesos de explosión.<br>4. Seleccionar la actividad que va a explosionar en el área de diagramado.<br>5. Seleccionar la opción “Explosionar actividad”. |
| **Resultado Esperado:** | Se crea en el árbol de procesos de explosión un nuevo proceso del tipo explosión. |

| Diagramado de metodología | |
|---|---|
| **ID Prueba:** | UC 23-1 |
| **Objetivo :** | Realizar el diagramado gráfico de un nodo perteneciente a un proceso base. |
| **Precondición:** | Debe estar definida una metodología y activa el área de diagramado. |
| **Proceso:** | 1. Elegir del árbol el nodo que se desee diagramar en la metodología.<br>2. Seleccionar la opción “Seleccionar”. |
| **Resultado Esperado:** | Se coloca el nodo elegido en el área de diagramado de la metodología con todas las asociaciones que tiene relacionada en el proceso base. |

| Generar versión proceso | |
|---|---|
| **ID Prueba:** | UC 33-1 |
| **Objetivo :** | Crear una nueva versión de un proceso. |
| **Precondición:** | La vista de proceso está activa. Existe un paquete abierto y dentro de éste existe al menos un proceso creado. |
| **Proceso:** | 1. Seleccionar un proceso en el árbol de procesos.<br>2. Seleccionar la opción “Ir a la vista de versiones”.<br>3. Seleccionar en el árbol de versiones, la versión del proceso sobre el que va crear la nueva versión.<br>4. Seleccionar la opción “Nueva versión”.<br>5. Esperar a que termine de versionar (el avance del versionado se puede apreciar en la barra de estado situada en la parte inferior de la pantalla). |
| **Resultado Esperado:** | Se crea en el árbol de versiones una nueva hoja correspondiente a la nueva versión del proceso. |

# 4. Construcción y Pruebas

En este capítulo se presenta la fase final del proyecto, que corresponde a las etapas de construcción y pruebas del sistema.

En la primera sección del presente capítulo se detalla la lógica utilizada para la implementación de los componentes que constituyen la herramienta MJS Process Designer. Posteriormente, se hace mención a las buenas prácticas seguidas durante la etapa de diseño y desarrollo de la misma. Finalmente, se presentan las pruebas integrales de cada uno de los módulos de la herramienta, las cuales fueron ejecutadas de forma satisfactoria garantizando así el correcto funcionamiento del sistema.

## 4.1. Implementación de Componentes

A continuación se presentan algunas de las principales consideraciones que se tomaron en cuenta durante la etapa de construcción de la herramienta MJS Process Designer.

### 4.1.1. Estructuración Interna del Proyecto Web

Dado que la herramienta MJS Process Designer fue elaborada dentro del entorno de desarrollo Eclipse, el tipo de proyecto que se utilizó fue el denominado Dynamic Web Project, cuya estructuración interna se muestra en la ilustración 4-1.

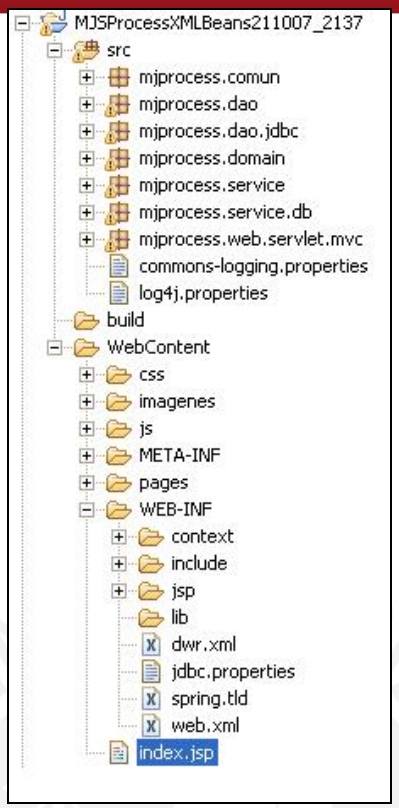


**Ilustración 4-1: Estructura Interna del Proyecto**

A continuación se explica en detalle la estructura interna del proyecto:

**a. /src/:**

En este directorio se encuentran las clases Java, las cuales están dividas en siete paquetes:

- **mjsprocess.comun:** En este paquete se encuentran los archivos Java que son de uso común por las diferentes clases de la aplicación, como por ejemplo: Constants.java (archivo de constantes).
- **mjsprocess.domain:** Este paquete agrupa todos los beans de entidad usados por la aplicación, los cuales representan las diferentes entidades que forman parte de la lógica del negocio.
- **mjsprocess.dao:** Acá se encuentran las interfaces Java que definen los métodos necesarios para el acceso a la base de datos.
- **mjsprocess.dao.jdbc:** En este paquete se encuentran las clases que implementan las interfaces de acceso a la base de datos.
- **mjsprocess.service:** Está conformado por las interfaces Java que definen los métodos necesarios para el manejo de la lógica del negocio de la herramienta.
- **mjsprocess.service.db:** Este paquete contiene las clases que implementan las interfaces que se encargan del manejo de la lógica del negocio de la herramienta.
- **mjprocess.web.servlet.mvc:** En este directorio se encuentran las clases de tipo controlador, las cuales sirven de intermediarios entre el usuario y el sistema.

**b. /WebContent/css:**

En este directorio se encuentran almacenadas las hojas de estilo usadas por la herramienta.

**c. /WebContent/imagenes/:**

Contiene las imágenes e iconos usados por la herramienta.

**d. /WebContent/js/:**

Agrupa un conjunto de archivos JavaScript que contienen funciones que son usadas en la capa de presentación de la herramienta. En este directorio se encuentran también el conjunto de archivos que conforman el motor de modelado gráfico de la herramienta MJS Process Designer.

**e. / WebContent /WEB-INF/context/**:

Contiene el conjunto de archivos XML, los cuales son utilizados como archivos de configuración para el uso del framework Spring.

**f. / WebContent /WEB-INF/jsp/**:

En este directorio residen los archivos JSP utilizados por la aplicación, los cuales forman parte de la interfaz de usuario del aplicativo.

**g. /WebContent/WEB-INF/lib/**:

Agrupa los archivos JAR que están siendo utilizados por la herramienta.

**h. / WebContent /WEB-INF/web.xml**:

Este archivo contiene elementos de configuración del sistema, como por ejemplo: Página de inicio, ubicación y mapeo de servlets, manejo de errores, entre otros.

**i. / WebContent /WEB-INF/dwr.xml**:

En este archivo se declaran las clases Java que podrán ser llamadas desde una función JavaScript con la finalidad de utilizar la tecnología AJAX.

### 4.1.2. Configuración de Spring

Una de las características principales de Spring es que permite configurar y crear aplicaciones complejas a partir de un conjunto de componentes sencillos que son declarados en archivos XML.

Spring actúa como un contenedor que se encarga de almacenar los objetos de una aplicación, manejar su ciclo de vida, declarar cómo es que estos objetos van a ser creados, cómo es que van a ser configurados y en qué forma se van a asociar entre ellos.

A continuación se procederá a explicar cuales fueron las configuraciones realizadas en el sistema MJS Process Designer para el manejo de las clases JavaBean, Service, Controller y DAO a través del framework Spring.

### A. Instalación de Spring

Para empezar a utilizar Spring es necesario agregar en el proyecto Web su respectivo JAR, el cual puede ser descargado de forma gratuita a través de la página Web de dicho framework. Este JAR debe ser agregado en el directorio WEB-INF/lib/ perteneciente al proyecto creado en el IDE de Eclipse.

### B. Configuración del archivo web.xml

Web.xml es el archivo de configuración común a toda aplicación Web en Java. Este archivo se encuentra ubicado dentro de la carpeta WEB-INF y es el encargado de describir el comportamiento de las aplicaciones.

```
<!-- LISTENERS -->
<listener> <listener-class> org.springframework.web.context.ContextLoaderListener
</listener-class></listener>
<listener> <listener-class> org.springframework.web.util.Log4jConfigListener
</listener-class> </listener>
<listener> <listener-class> org.springframework.web.util.IntrospectorCleanupListener
</listener-class> </listener>

<!-- CONTEXT -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value> /WEB-INF/context/applicationContext*.xml </param-value>
</context-param>
:
<!-- SERVLETS -->
<servlet>
        <servlet-name>MJProcessv1</servlet-name>
        <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value> /WEB-INF/context/applicationServlet*.xml </param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
</servlet>
:
<!-- SERVLETS MAPPING -->
<servlet-mapping>
       <servlet-name>MJProcessv1</servlet-name>
       <url-pattern>*.htm</url-pattern>
</servlet-mapping>
:
<!-- TAGS -->
<taglib>
        <taglib-uri>/spring</taglib-uri>
        <taglib-location>/WEB-INF/spring.tld</taglib-location>
</taglib>
```

**Archivo Fuente 4-1: Web.xml**

En el archivo fuente 4-1 se muestra un extracto del archivo web.xml perteneciente a la herramienta MJS Process Designer, el cual ha sido divido en cinco secciones para facilitar su descripción.

**Sección de Listener:**

Son tres listener principales que deben ser especificados en el archivo web.xml:

- **ContextLoaderListener:** Permite cargar los archivos de configuración utilizados por Spring al momento de inicializar la aplicación.
- **Log4jConfigListener**: Permite utilizar la librería log4j que sirve para poder dejar trazas durante la ejecución de la aplicación, lo cual es de mucha utilidad durante la etapa de desarrollo.
- **IntrospectorCleanupListener:** Se encarga de limpiar el cache del JDK al momento de finalizar la aplicación.

**Sección de Contexto:**

El elemento **context-param,** como su nombre lo indica, sirve para señalar todos los parámetros globales que serán utilizados por la aplicación. Uno de los parámetros especificados en este archivo es la ruta en donde se encuentran los archivos de configuración de Spring y cómo es que son nombrados.

Como se puede observar en el archivo fuente 4-1, para el caso de la herramienta MJS Process Designer son todos aquellos archivos que se encuentran en el directorio /WEB-INF/context/ y cuyos nombres empiezan con “applicationContext”.

**Sección de Servlet**

En esta sección se requiere definir el servlet de Spring, el cual será el encargado de atender los eventos que se producen en la aplicación. Para la presente herramienta, el servlet se denomina **MJSProcessv1**, el cual es una instancia de la clase DispatcherServlet que actúa como un controlador de Spring MVC.

La configuración de dicho servlet se encuentra definida en los archivos que empiecen con “applicationServlet”, los cuales se encuentran ubicados en el directorio /WEB-INF/context/.

**Sección Servlet Mapping**

En esta sección se definen todas las URL que serán manejadas por el servlet principal de la aplicación. Para la presente herramienta, la expresión “*.htm” significa que el servlet manejará todas aquellas URL que contengan páginas de tipo HTML, por ejemplo: Páginas de tipo JSP, HTML puro, entre otras.

**Sección Tag**

En esta sección se agregan todas aquellas librerías de etiquetas (taglib) que serán usadas por la aplicación. En el archivo fuente 4-1 se puede apreciar que ha sido agregado el taglib que provee Spring para que dichas etiquetas puedan ser utilizadas en las páginas JSP.

### C. Configuración de los archivos Application Servlet

Spring, al inicializar el servlet DispatcherServlet buscará los archivos cuyos nombres sigan el patrón [servlet-name]*.xml para localizar la información de configuración de los beans manejados por la aplicación.

Los archivos de configuración manejados por la herramienta MJS Process Designer son los siguientes:

#### a. Archivo applicationServlet.xml

En el archivo fuente 4-2 se muestra un extracto del archivo applicationServlet.xml utilizado por la herramienta MJS Process Designer, el cual está dividido en dos secciones: View Resolver y Excepciones.

```
<!--  View Resolver -->
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass">
            <value>org.springframework.web.servlet.view.JstlView</value></property>
     <property name="order"><value>1</value></property>
     <property name="prefix"><value>/WEB-INF/jsp/</value> </property>
     <property name="suffix"><value>.jsp</value></property>
</bean>

<!-- Excepciones  -->
<bean id="exceptionResolver"
        class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
      <property name="exceptionMappings">
      <props> <prop key="java.lang.Exception">error</prop> </props>
   </property>
</bean>
```

**Archivo Fuente 4-2: ApplicationServlet.xml**

**Sección View Resolver**

Los elementos ViewResolver son aquellos que permiten asignar un nombre lógico a las vistas utilizadas en la aplicación.

Para el desarrollo de la presente herramienta se ha definido el bean denominado viewResolver como una instancia de la clase InternalResourceViewResolver, la cual es la clase más simple que utiliza Spring para el manejo de páginas de tipo JSP. A través de este bean se configura

que las vistas de la aplicación trabajen con las etiquetas que ofrece la librería JSTL y se indica cuáles serán los prefijos y sufijos agregados a los nombres lógicos de las mismas.

De esta forma cuando se solicite a la herramienta MJS Process Designer una vista “index”, el bean viewResolver proveerá la vista WEB-INF/jsp/index.jsp.

**Sección Excepciones**

En esta sección se declara un bean de excepción perteneciente a la clase SimpleMappingExceptionResolver, el cual se encarga de redireccionar el aplicativo hacia una determinada vista cuando los controladores lancen alguna excepción. Tal como se muestra en el archivo fuente 4-2, el bean exceptionResolver se encarga de manejar las excepciones del tipo java.lang.exception, las cuales son redireccionadas hacia la vista cuyo identificador se denomina “error”.

**b. Archivo applicationServlet-controllers.xml**

En este archivo se definen los beans de tipo Controller utilizados en la aplicación. Los tipos de controladores utilizados por la herramienta MJS Process Designer son los que derivan de la clase SimpleController y los que derivan de la clase MultiActionController. El primer tipo de controlador posee un único método a ejecutar al momento de ser invocado, mientras que el segundo tipo puede tener más de un método asociado.

```
<bean id="PaqueteFullController"
        class="mjprocess.web.servlet.mvc.PaqueteFullController">
    <property name="methodNameResolver" ref="paqueteControllerResolver"/>
    <property name="servicePaquete" ref="PaqueteService" />
    <property name="serviceConstante" ref="ConstanteService" />
    <property name="serviceCabecera" ref="DefinicionCabeceraService" />
    <property name="serviceScript" ref="ScriptPaqueteService" />
    <property name="serviceRedefinicionCabecera" ref="RedefinicionCabeceraService" />
</bean>
....
<bean id="paqueteControllerResolver"
class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
          <property name="paramName">
                    <value>
                         accion
                    </value>
          </property>
</bean>
```

**Archivo Fuente 4-3: ApplicationServlet-controllers.xml**

Spring define una interfaz denominada MethodNameResolver, la cual debe ser implementada por todas las clases que se utilicen para identificar el método a llamar. Dichas clases deben ser referenciadas a través de una propiedad cuyo nombre sea methodNameResolver.

En el archivo fuente 4-3 se puede observar que el bean paqueteControllerResolver que deriva de la clase ParameterMethodNameResolver, permite identificar el nombre del método a invocar. En el desarrollo de MJS Process Designer se utilizó el parámetro denominado "accion", el cual es obtenido del objeto request que llega al controlador.

Es importante tener en cuenta que la definición de un bean requiere indicar como mínimo el identificador (id) y la clase a la que pertenece (class). Adicionalmente, si dicha clase posee atributos, estos deben ser definidos a través de la etiqueta <property>, en la cual se debe indicar el nombre del atributo (name) y el nombre del bean o clase de Spring (ref) al cual se hace referencia.

De acuerdo al patrón de diseño utilizado por la herramienta MJS Process Designer, los objetos de tipo Service son invocados a través de los objetos tipo Controller. Como se observa en el archivo fuente 4-3, el bean PaqueteFullController hace referencia a varios beans de tipo Service, entre los que se encuentran: PaqueteService, ConstanteService, entre otros. Al mismo tiempo, las clases a las que pertenecen los beans que son configurados en este archivo deben implementar los métodos get y set por cada uno de los property definidos, tal y como se muestra en el archivo fuente 4-4. Esto con la finalidad de cumplir con los requisitos básicos para el uso del patrón de Inversión de Control.

```
public class PaqueteFullController extends MultiActionController{

	private final Log log = LogFactory.getLog(getClass());
	private PaqueteService servicePaquete;
	private ConstanteService serviceConstante;
	private DefinicionCabeceraService serviceCabecera;
	private RedefinicionCabeceraService serviceRedefinicionCabecera;
	private ScriptPaqueteService serviceScript;

	public DefinicionCabeceraService getServiceCabecera() {
		return serviceCabecera;
	}
	public void setServiceCabecera(DefinicionCabeceraService serviceCabecera) {
		this.serviceCabecera = serviceCabecera;
	}
	public ConstanteService getServiceConstante() {
		return serviceConstante;
	}
	public void setServiceConstante(ConstanteService serviceConstante) {
		this.serviceConstante = serviceConstante;
	}
	public PaqueteService getServicePaquete() {
		return servicePaquete;
	}
	public void setServicePaquete(PaqueteService servicePaquete) {
		this.servicePaquete = servicePaquete;
	}
```

**Archivo Fuente 4-4: ApplicationServlet-controllers.xml**

**c. Archivo applicationServlet-handlerMapping.xml**

La funcionalidad básica que provee el HandlerMapping es determinar según el contenido del request, que acciones deben llevarse a cabo.

La herramienta MJS Process Designer define un bean denominado urlMapping que deriva de la clase BeanNameUrlHandlerMapping, el cual permite mapear una dirección URL directamente a un bean.

El archivo fuente 4-5 es un ejemplo del uso del handlerMapping en la herramienta, en él se indica que las acciones de las páginas nuevo_paquete.htm y mantPaq_general.htm son manejadas por el controlador PaqueteFullControler.

```
<bean id="urlMapping"
          class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="urlMap">
     <map>
       <entry key="/nuevo_paquete.htm"><ref bean="PaqueteFullController" /> </entry>
       <entry key="/mantPaq_General.htm"><ref bean="PaqueteFullController" /></entry>
          ....
     </map>
  </property>
</bean>
```

**Archivo Fuente 4-5: ApplicationServlet-handlerMapping.xml**

**D. Configuración de los archivos ApplicationContext**

ApplicationContext es un contenedor avanzado propio de Spring que es utilizado para definir los beans manejados por la aplicación, relacionarlos y definir cómo van a ser manejadas las peticiones de los clientes.

Estas configuraciones pueden ser realizadas en un único archivo XML denominado comúnmente applicationContext, sin embargo, con el fin de facilitar la administración de las configuraciones se definen varios archivos de contexto de aplicación de acuerdo a una funcionalidad común.

A continuación se listan los archivos que en su conjunto forman la configuración del applicationContext de la herramienta MJS Process Designer:

**a. Archivo applicationContext.xml**

En este archivo se hace uso de la clase PropertyPlaceHolderConfigurer, la cual se encarga de indicar a Spring que tiene que cargar ciertas configuraciones desde de un archivo de propiedades externo (archivo fuente 4-6).

Uno de estos archivos es el jdbc.properties, el cual contiene los datos necesarios para realizar la conexión a la base de datos. Entre los datos que se encuentran en este archivo se tienen: nombre del driver de la base de datos, usuario, contraseña y dirección Web.

```
<bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list> <value>/WEB-INF/jdbc.properties</value> </list>
    </property>
</bean>
```

**Archivo Fuente 4-6: ApplicationContext.xml**

**b. Archivo applicationContext-Domain.xml**

Este archivo contiene la lista de beans de tipo DAO y Service utilizados por la aplicación (archivo fuente 4-7).

```
<!-- Lista de DAO -->
<bean id="PaqueteDAO" class="mjprocess.dao.jdbc.PaqueteDAOJdbc">
    <property name="dataSource"   ref="dataSource"/>
</bean>
...
<!-- Lista de Service -->
<bean id="PaqueteService" class="mjprocess.service.db.PaqueteServiceDb" >
    <property name="daoPaquete" ref="PaqueteDAO"/>
</bean>
...
```

**Archivo Fuente 4-7: ApplicationContext-Domain.xml**

De acuerdo al patrón de diseño utilizado por la herramienta MJS Process Designer son los objetos de tipo Service los que invocan a los objetos de tipo DAO. Como se observa en el archivo fuente 4-7 es el bean PaqueteService quien hace referencia al bean PaqueteDAO. Al mismo tiempo las clases a las que pertenecen los beans que son configurados en este archivo deben implementar los métodos get y set por cada uno de los property definidos.

### 4.1.3. Integración de AJAX con Spring:

Para poder hacer uso de la tecnología AJAX de forma sencilla y manteniendo el estándar de trabajo definido por el framework Spring, se decidió utilizar el API DWR (Direct Web Remoting). Este API brinda una interfaz sencilla para la integración con Spring, al mismo tiempo, permite realizar llamadas remotas y de forma directa desde código JavaScript (que se ejecuta en un navegador Web) a objetos Java que se encuentran en el servidor.

Para poder realizar este tipo de llamadas remotas se requiere que la configuración del API DWR se encuentre tanto en el lado del servidor como en el lado del cliente. Dicha configuración se realiza de la siguiente manera:

**A. Configuración en el lado del Servidor**

DWRServlet es el motor del API DWR. En esta clase se centralizan todas las posibles funcionalidades que ofrece Direct Web Remoting, las cuales van desde la generación del código JavaScript a utilizar en el cliente hasta el marshalling de tipos de datos, incluyendo por supuesto la invocación a los métodos remotos.

Como cualquier otro servlet, DWRServlet debe ser declarado y mapeado hacia alguna URL en el archivo web.xml de la aplicación Web, tal y como se muestra en el archivo fuente 4-8.

```
<!-- ////////////////////////////////////////////////////////////// -->
<!--                          SERVLETS                              -->
<!-- ////////////////////////////////////////////////////////////// -->

<servlet>
    <servlet-name>dwr-invoker</servlet-name>
    <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>


<!-- ////////////////////////////////////////////////////////////// -->
<!--                       SERVLETS MAPPING                         -->
<!-- ////////////////////////////////////////////////////////////// -->

<servlet-mapping>
    <servlet-name>dwr-invoker</servlet-name>
    <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

**Archivo Fuente 4-8: web.xml**

**B. Exportación de los Objetos Java que serán llamados desde código JavaScript**

Todas las declaraciones necesarias para realizar llamadas remotas a los métodos de los objetos Java que se encuentran en el servidor se realizan en el archivo **dwr.xml**. Cabe mencionar que para la presente herramienta estos objetos son los que se encuentran en la capa Service, ya que son ellos los que se encargan de controlar la lógica del negocio.

Las clases a las que pertenecen los objetos Java cuyos métodos podrán ser llamados desde el cliente son definidas con la etiqueta **<create>,** tal como se muestra en el archivo fuente 4-9.

En esta etiqueta se indica cómo se deben crear y manejar dichas instancias a través de los siguientes atributos:

- **Creador:** Referencia a la entidad que se encarga de instanciar los objetos. Para la presente herramienta es el mismo framework quien se encarga de llevar a cabo dicha instanciación, dado que estos objetos son manejados por Spring.
- **JavaScript:** Es el nombre con el que será referenciado el objeto desde alguna función de tipo JavaScript.

El elemento Create también incluye una serie de parámetros necesarios, los cuales son:

- **BeanName:** Nombre del bean declarado en los archivos de configuración Spring que será exportado.
- **Class:** Nombre de la clase a la cual pertenece el bean a exportar.

```
<?xml version="1.0" encoding="UTF-8"?>
<dwr>
  <allow>
      <create creator="spring"  javascript="creadorProceso" >
        <param name="beanName" value="WorkflowService"/>
        <param name="class" value="mjprocess.service.db.WorkflowServiceDb" />
      </create>
  </allow>
</dwr>
```

**Archivo Fuente 4-9: dwr.xml**

**C. Uso de los métodos de los objetos exportados en el lado del Cliente**

DWR proporciona para cada uno de los métodos exportados por el servidor una función JavaScript que actúa como stub del cliente. Los stubs del cliente ocultan al desarrollador las complejidades de la comunicación cliente-servidor (especialmente, el manejo del objeto XMLHttpRequest).

Las funciones JavaScript, que hacen referencia a los métodos de una misma clase exportada, se agrupan en un mismo fichero de extensión .js. Cada uno de estos ficheros se obtiene de la URL: **<servletdwrpath>/interface/<nombreclase>.js.**

Para poder hacer uso de estas funciones JavaScript se siguen los siguientes pasos:

- **Inclusión de los ficheros .js:** Se debe incluir en la cabecera de las páginas Web que utilizan los métodos de los objetos exportados la URL mostrada en el archivo fuente 4-10.

```
<head>
      <script type="text/javascript"
         src="<%=request.getContextPath()%>/dwr/interface/creadorProceso.js">
      </script>
</head>
```

**Archivo Fuente 4-10: Declaración de uso de AJAX en un archivo JSP**

- **Invocación a los métodos exportados:** Dado que las llamadas a los métodos remotos deben ser asíncronas, no se puede llamar directamente a un método y obtener un resultado. Para que una llamada remota con DWR funcione es necesario pasarle como parámetro una función de callback, la cual se encarga de procesar la respuesta del servidor.

  En el archivo fuente 4-11 se observa que la función **processData** es la llamada de tipo callback que se encarga de realizar las acciones necesarias con el resultado obtenido de la invocación a uno de los métodos de los objetos exportados. En el presente ejemplo el objeto Java invocado recibe el nombre de creadorProceso (archivo fuente 4-9), del cual es invocado el método “crear”.

```
//Crea un proceso de tipo normal
function creaProceso()
{
      var idPaquete = getIdPaquete();
      var data = creadorProceso.crear(procesaData,idPaquete,1);
}

function procesaData(data)
{
      window.parent.frames["ifrArbol"].agregaProceso(data);
}
```

**Archivo Fuente 4-11: Ejemplo de invocación de método utilizando AJAX**

### 4.1.4. Generación del Archivo XML:

Para poder exportar las distintas definiciones registradas en la herramienta (paquete, proceso y metodología) se utiliza el formato de archivo definido por el lenguaje XPDL.

La exportación de las definiciones del paquete utilizando la tecnología XMLBeans requiere la elaboración de un archivo denominado “esquema” (con extensión .xsd). Este último define la estructura y reglas con la que se exportarán las definiciones registradas en la herramienta.

Debido a que el desarrollo de la herramienta MJS Process Designer incluye la extensión del lenguaje XPDL, se tuvo que generar un nuevo esquema basado en el esquema propio de dicho lenguaje y que además incluía la estructura de las extensiones realizadas al mismo.

Este nuevo esquema tuvo que ser compilado utilizando el JAR XBean propio del XMLBeans, obteniéndose como resultado un archivo JAR. Dicha librería contenía el conjunto de clases necesarias para generar el archivo XML de la definición de un paquete.

Para que la herramienta MJS Process Designer realice la exportación de las definiciones de un paquete se siguieron los siguientes pasos:

- Se agregó como una de las librerías externas el archivo de esquemas.jar generado mediante la tecnología XMLBeans.
- Se desarrolló la lógica necesaria para la exportación de las definiciones en un archivo XPDL utilizando las funciones proporcionadas por el JAR agregado en la aplicación.

### 4.2. Buenas Prácticas usadas en la herramienta

Durante el proceso de análisis, diseño e implementación de la herramienta MJS Process Designer se utilizaron un conjunto de buenas prácticas con la finalidad de garantizar la calidad del producto final obtenido. A continuación se mencionan las buenas prácticas de desarrollo de software utilizadas en la herramienta:

- Se incluyó la documentación del código fuente como parte del proceso de desarrollo de la herramienta, con la finalidad de facilitar el entendimiento por parte del desarrollador que requiera realizar una modificación a la herramienta en el futuro.

- El tiempo de respuesta de la herramienta frente a una acción realizada por el usuario influye en gran medida en la satisfacción del mismo con respecto a la herramienta, como consecuencia de ello se deben tomar las medidas necesarias para minimizar en lo posible el tiempo de espera del usuario.

  Una de las medidas adoptadas para lograr este objetivo consistió en la implementación del modelado gráfico haciendo uso de funciones elaboradas en lenguaje JavaScript y lenguaje VML. Gracias a esto el proceso de diagramado se lleva a cabo en el lado del cliente, con lo que se asegura una respuesta rápida y dinámica de la aplicación. De esta forma, solamente se accede al servidor cuando el usuario graba el modelo en la base de datos.

  Adicionalmente se consideró no utilizar expresiones CSS debido a que estas son evaluadas continuamente durante la interacción del usuario con la página Web, lo cual aumentaría el tiempo de respuesta de la aplicación.

- El tiempo de carga inicial de los formularios de la herramienta es considerado otro de los factores que influye en la satisfacción del usuario, por ello es recomendable adoptar las medidas necesarias con la finalidad de disminuir el mismo.

  Entre las buenas prácticas que se adoptaron para lograr este objetivo se encuentra la implementación de las funciones JavaScript como archivos externos, los cuales a su vez serán referenciados por las páginas JSP y HTML que requieran utilizarlos. Esto permite que el navegador Web almacene los archivos de JavaScript dentro de su memoria cache, por lo que la próxima vez que se acceda a la página solamente se descargará el código HTML.

  Otra de las medidas para agilizar la carga inicial consistió en colocar la declaración de estilos CSS en la definición de cabecera de la página HTML, con lo que se permite la carga progresiva de los elementos incluidos en la página. De lo contrario, si la referencia a los estilos utilizados se ubicara en la parte inferior del archivo, la carga de elementos se encontraría bloqueada debido a que el pintado estaría en espera de la referencia de los estilos utilizados.

- La inclusión de código de Java nativo dentro de una página HTML (scriptles) permite realizar operaciones más complejas de las que se pueden elaborar con el lenguaje JavaScript, pero a su vez hace más ilegible y desordenado el código fuente de la página Web. Esto obliga a que el usuario posea cierto dominio del lenguaje Java para entender o modificar dicha página.

  Una alternativa a este inconveniente, es el uso de la librería JSTL, la cual brinda un adecuado manejo de la información que es enviada desde el servidor a las páginas Web que se encuentran en la capa de presentación sin necesidad de incluir scriptles y manteniendo el estándar de codificación Web.

- El uso del patrón de diseño MVC asegura la separación de la lógica del negocio con la capa de presentación de la herramienta.

### 4.3. Pruebas de Componentes

Con la finalidad de probar el cumplimiento de los requerimientos establecidos para la herramienta MJS Process Designer se eligió el modelado de uno de los procesos principales de desarrollo de la metodología Métrica v3.

Por definición, Métrica v3 es una metodología que contempla el desarrollo de Sistemas de Información para las distintas tecnologías que actualmente están disponibles y los aspectos de gestión que aseguran que un proyecto cumpla sus objetivos en términos de calidad, coste y plazos.

Métrica v3 está conformada por cuatro procesos principales de desarrollo, los cuales a su vez están conformados por actividades y estas últimas se subdividen en tareas. Los procesos principales que conforman Métrica v3 son:

- Estudio de Viabilidad del Sistema.
- Análisis.
- Diseño.
- Construcción.

### 4.3.1. Modelador de Procesos

Para la prueba del módulo Modelador de Procesos se escogió modelar el primer proceso de Métrica v3: Estudio de Viabilidad del Sistema, el cual está compuesto por las siguientes actividades:

- Establecimiento del alcance del sistema.
- Estudio de la situación actual.
- Definición de requisitos del sistema.
- Estudio de alternativas de solución.
- Valoración de las alternativas.
- Selección de la solución.

Para poder registrar una definición de proceso, el usuario deberá abrir una definición de paquete previamente registrada en el sistema o crear una nueva. Una vez abierto o creado el paquete, el usuario podrá visualizar la lista de mantenimientos disponibles en la herramienta, los cuales le permitirán realizar el mantenimiento de la información y elementos relacionados al paquete. Entre los principales mantenimientos se encuentran:

- Definición de cabecera.
- Redefinición de cabecera.
- Script.
- Aplicaciones.
- Participantes.
- Atributos extendidos.
- Tipos de datos.
- Variables de paquete.

Para poder visualizar uno de los mantenimientos del listado anterior, el usuario deberá hacer clic sobre la opción acerca de la cual requiere mantener la información. Con ello, el formulario asociado se cargará automáticamente en la pestaña de Mantenimiento, en donde el usuario podrá registrar o modificar la información relacionada al paquete (ilustración 4-2).

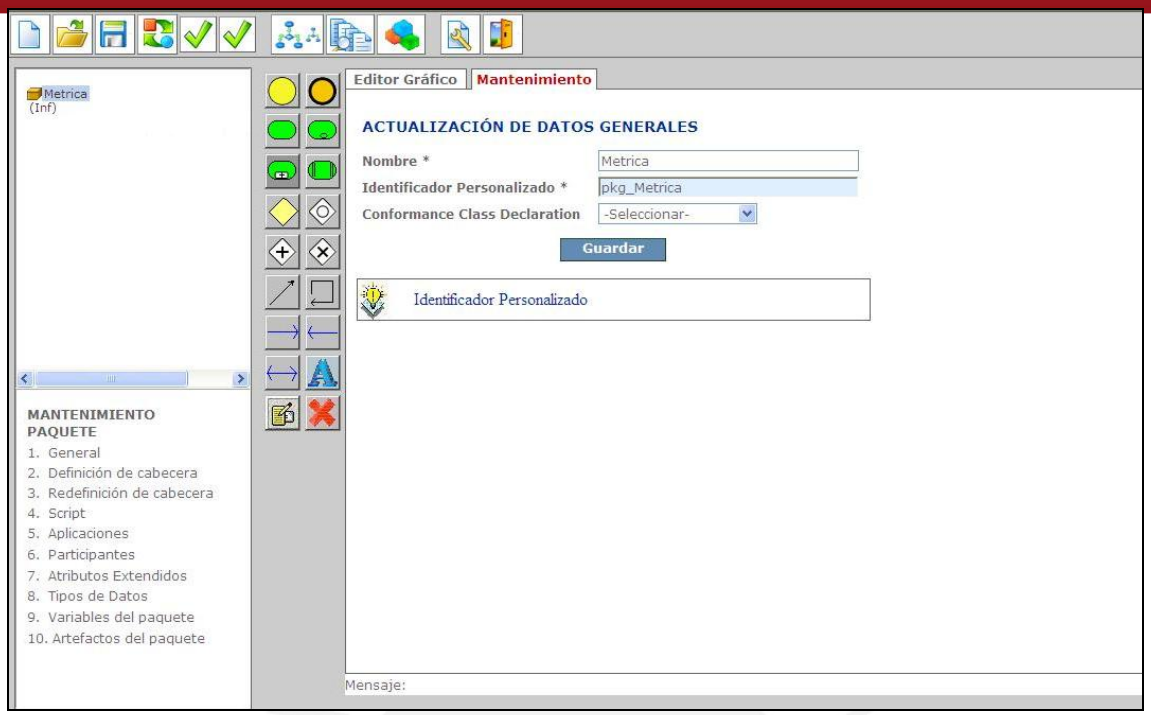


**Ilustración 4-2: Mantenimiento de Datos Generales de Paquete**

Posteriormente el usuario procederá a modelar el proceso “Estudio de la viabilidad del sistema”, para esto deberá crear una nueva definición de proceso en la herramienta. Una vez creado el proceso el usuario podrá visualizar la lista de mantenimientos disponibles en la herramienta, los cuales le permitirán realizar el mantenimiento de la información de los elementos relacionados al proceso.

Entre los principales mantenimientos se encuentran:

- Cabecera de Proceso.
- Redefinición de cabecera.
- Parámetros formales.
- Participantes.
- Variables del Proceso.
- Aplicaciones.
- Atributos extendidos.
- Set Activities.

Para poder visualizar uno de los mantenimientos del listado anterior, el usuario deberá hacer clic sobre la opción acerca de la cual requiere mantener la información. Con ello, el formulario asociado se cargará automáticamente en la pestaña de Mantenimiento, en donde el usuario podrá registrar o modificar la información relacionada al proceso, tal como se muestra en la ilustración 4-3.

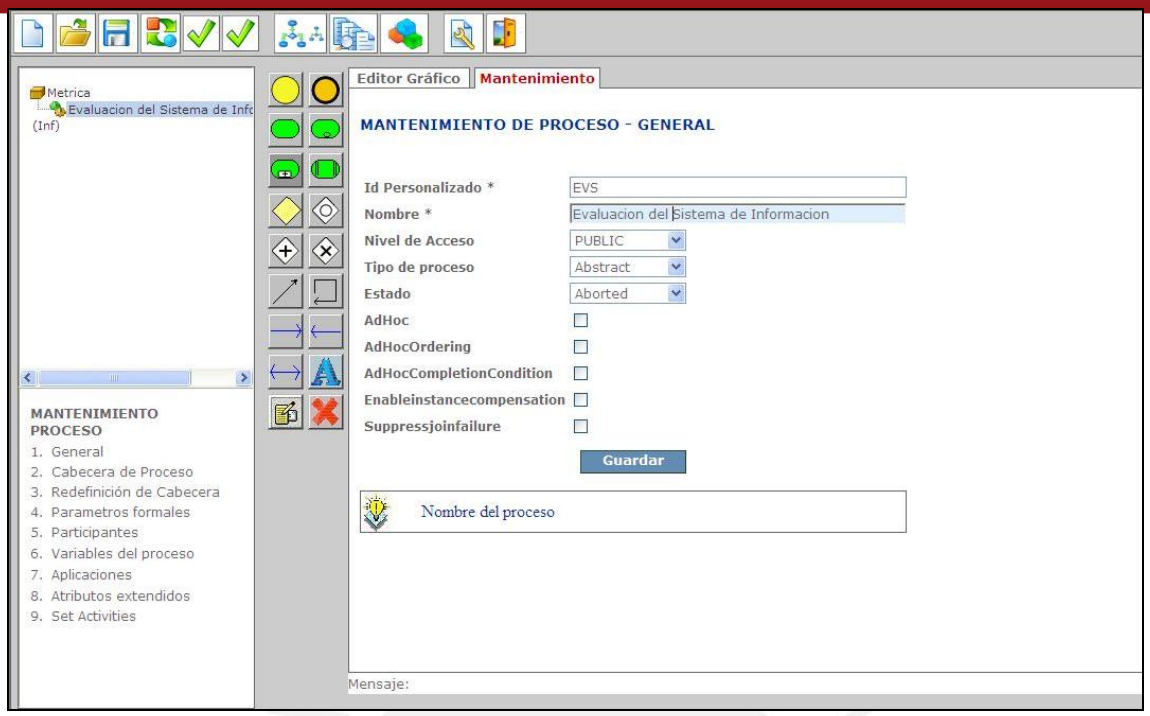


**Ilustración 4-3: Mantenimiento de Datos Generales de Proceso**

Finalizado el mantenimiento semántico de las entidades que el usuario crea conveniente, se procederá a realizar el modelado gráfico del flujo de trabajo. Para ello, se deberá activar el área de dibujado haciendo clic sobre el nombre del proceso. Como siguiente paso, el usuario diagramará el flujo de proceso utilizando los diferentes elementos gráficos disponibles en la barra de herramientas ubicada al lado izquierdo del área de diagramado, tal como se muestra en la ilustración 4-4.

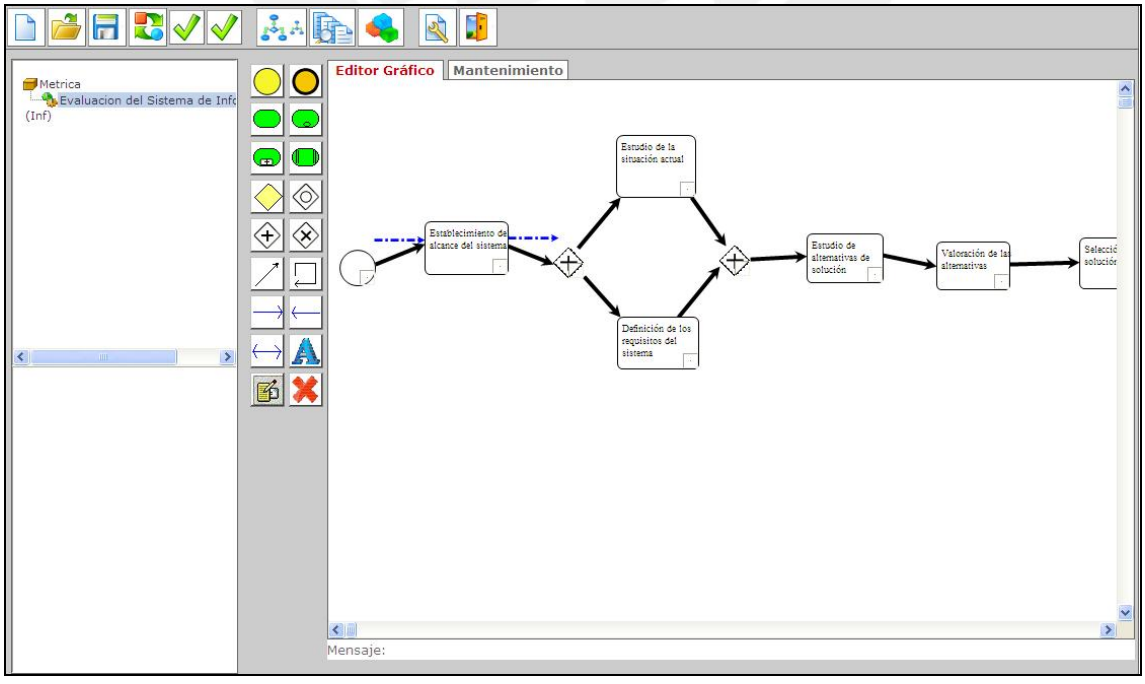


**Ilustración 4-4: Modelado gráfico del proceso**

La herramienta permite definir flujos de trabajo paralelos utilizando los diferentes tipos de conectores lógicos proporcionados; para el ejemplo de prueba (ilustración 4-4) se utilizó el conector AND para reflejar la ejecución sincronizada de las actividades “Estudio de la situación actual” y “Definición de los requisitos del sistema”.

Al finalizar el diagramado, el usuario deberá grabar en el sistema el estado actual del flujo de trabajo modelado con la finalidad de asegurar la consistencia al iniciar el mantenimiento de la información de los elementos diagramados (Modelado Semántico).

Para realizar el mantenimiento de una actividad perteneciente al flujo de trabajo, el usuario deberá seleccionarla y hacer clic en el icono de acceso al menú de mantenimiento disponible en la barra de herramientas. Para poder visualizar uno de los mantenimientos del listado, el usuario deberá hacer clic sobre el nombre respectivo, con ello el formulario asociado se cargará automáticamente en la pestaña de Mantenimiento (ilustración 4-5).

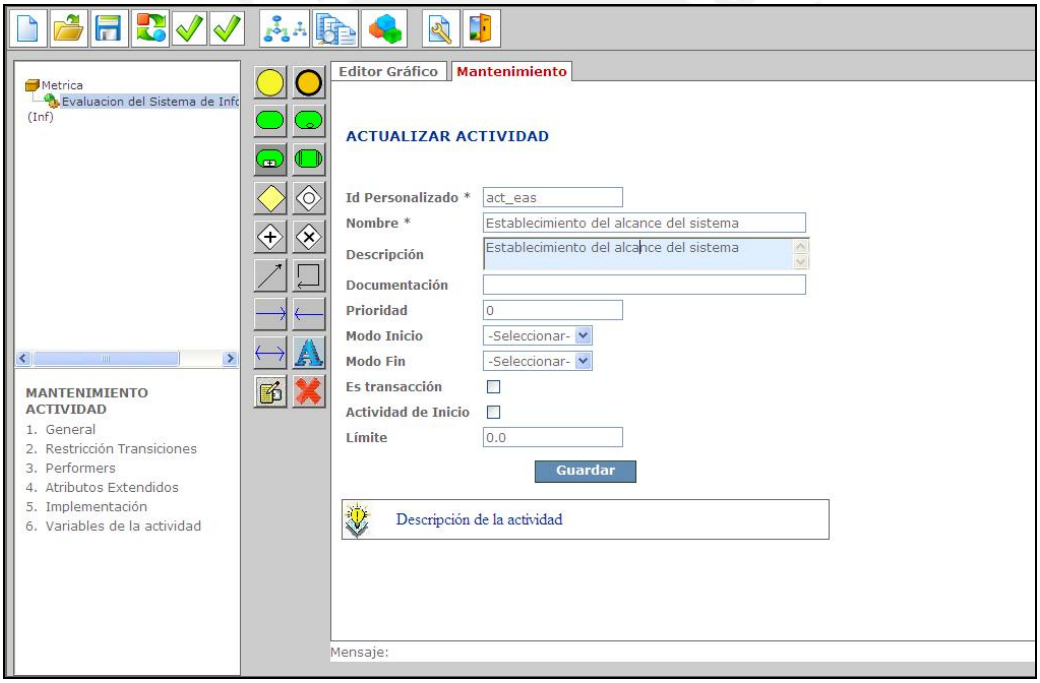


**Ilustración 4-5: Mantenimiento de Datos Generales de Actividad**

Al completar el mantenimiento gráfico y semántico se da por terminado el modelado del proceso, teniendo como producto final una nueva definición de proceso lista para su aprobación y su posterior uso.

### 4.3.2. Explosión de Niveles

Para la prueba del módulo de Explosión de Niveles se escogió explosionar la actividad “Establecimiento del Alcance del Sistema” perteneciente al proceso de “Estudio de Viabilidad del Sistema” en las siguientes actividades:

- Estudio de la solicitud.
- Identificación del alcance del sistema.
- Especificación del alcance del EVS.

Para explosionar una actividad en un nuevo nivel, el usuario deberá seleccionar el proceso al que pertenece la actividad y hacer clic en la opción “Ir a Vista de Explosión”. La herramienta mostrará un árbol listando el proceso seleccionado, en el cual se irán agregando los nodos correspondientes a los nuevos niveles creados producto de la explosión de una actividad (ilustración 4-6).

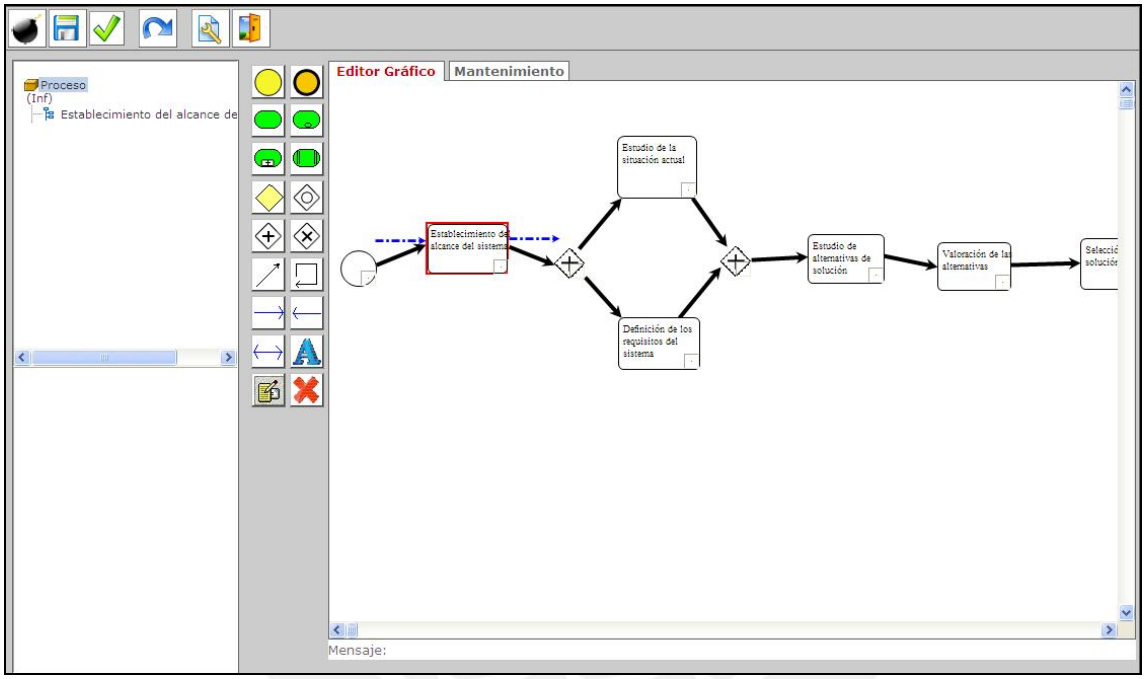


**Ilustración 4-6: Módulo de explosión de niveles**

Si el usuario desea explosionar una actividad deberá cargar el modelado gráfico del proceso al que pertenece en la pestaña “Editor Gráfico”. Como siguiente paso deberá seleccionar la actividad a explosionar (la cual deberá ser obligatoriamente de tipo básico) y hacer clic en la opción “Explosionar Nodo”.

Como resultado de los pasos anteriores, el sistema registrará un nuevo nivel asociado a la actividad y se agregará un nuevo nodo al árbol de la vista.

Para realizar el modelado gráfico del nuevo nivel, el usuario deberá hacer clic sobre el nodo que corresponde a dicho nivel en el árbol de explosión, con lo que se habilitará la pestaña “Editor Gráfico”. Este modelado es idéntico al explicado en la prueba del módulo “Modelador de Proceso” (ilustración 4-7).

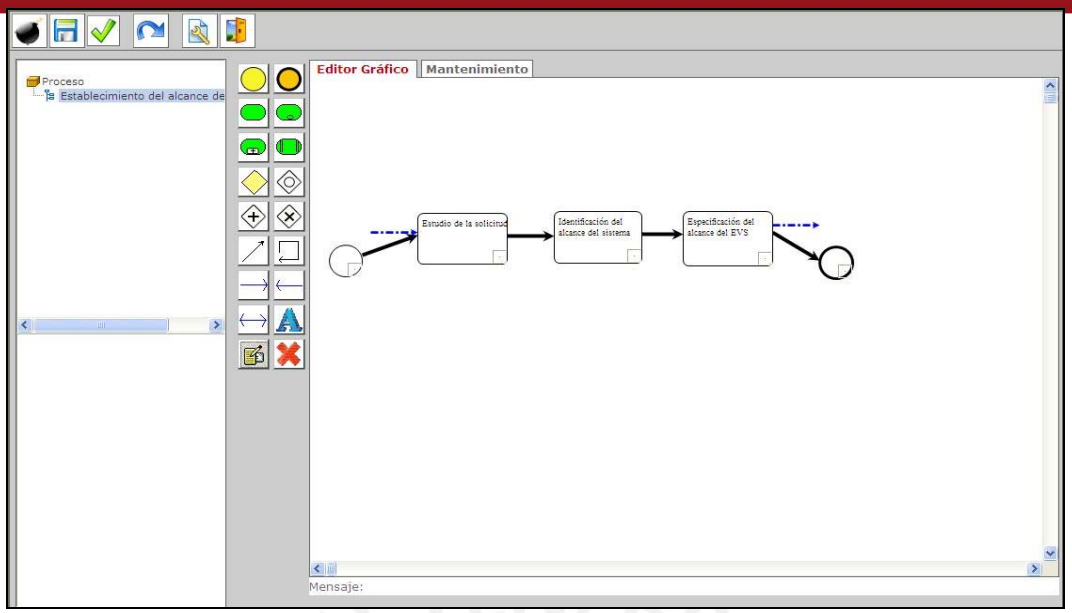


**Ilustración 4-7: Modelado gráfico del nuevo nivel**

### 4.3.3. Módulo de Metodologías

Para la prueba del módulo de Metodologías se decidió realizar una versión simplificada del proceso original de Métrica v3.

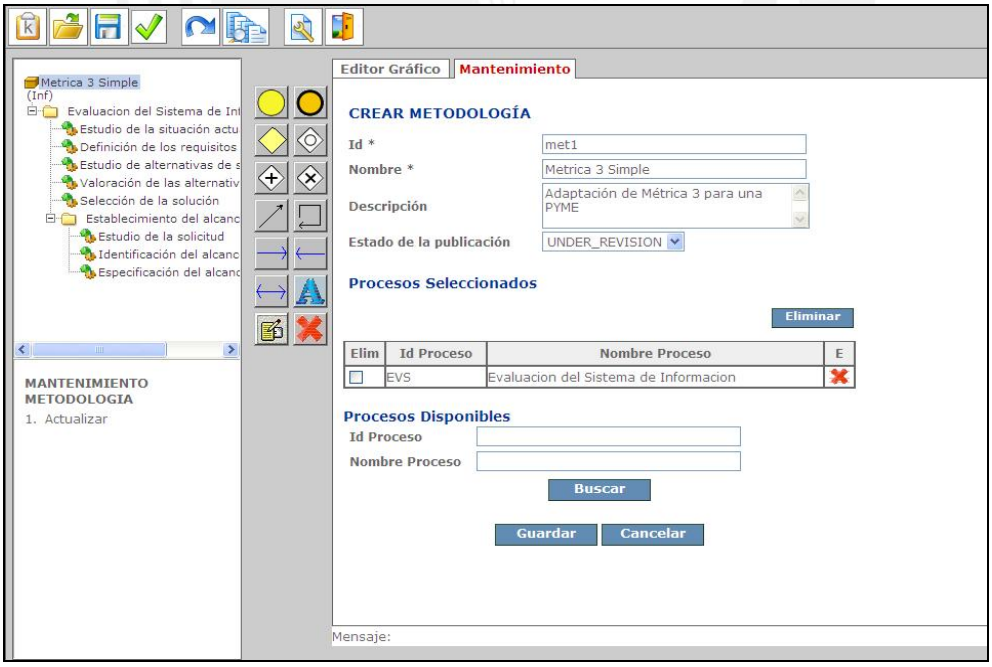


**Ilustración 4-8: Formulario de creación de metodología**

Esta metodología estará conformada por las siguientes actividades:

- Especificación del alcance del EVS.
- Definición de requisitos del sistema.

- Estudio de alternativas de solución.
- Selección de la solución.

Para crear una metodología el usuario deberá hacer clic en la opción “Ir a Vista de Metodología”. El usuario hará clic sobre la opción “Nueva Metodología”, a continuación la herramienta mostrará un formulario en el cual se ingresarán los siguientes datos: Identificador, nombre y descripción. Adicionalmente se deberán seleccionar los procesos base de los cuales se extraerán las actividades que conformarán la nueva metodología.

Una vez creada la metodología, la herramienta cargará en el árbol de metodología la lista de los procesos base seleccionados con sus respectivos nodos (ilustración 4-8).

Para realizar el modelado gráfico de la metodología se deberá activar la pestaña “Editor Gráfico” haciendo clic sobre el nombre de la metodología. Si el usuario desea agregar una actividad a la metodología deberá seleccionarla de los procesos listados en el árbol de la vista actual, para ello deberá hacer clic en el nodo que le corresponde y utilizar la opción “Seleccionar”.

El módulo de metodologías permite la definición de un flujo de trabajo en base a las actividades seleccionadas utilizando los conectores y transiciones disponibles en la barra de herramientas (ilustración 4-9).

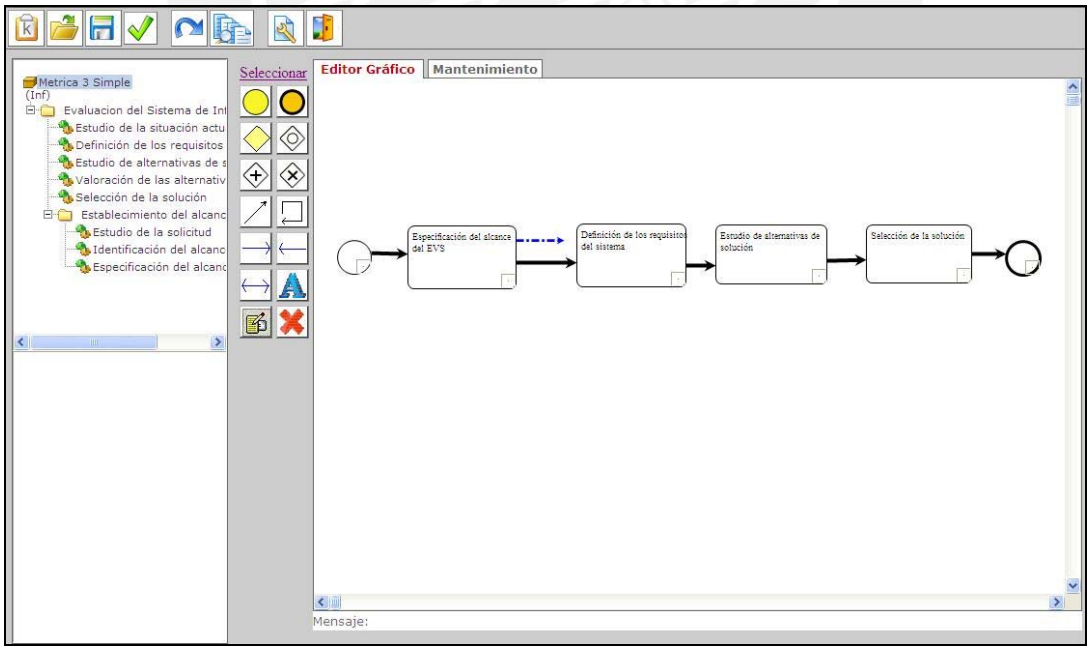


**Ilustración 4-9: Modelado gráfico de la metodología**

En la vista de metodología no se podrá actualizar la información de las actividades que conforman una metodología, dicha información sólo estará disponible en modo lectura (ilustración 4-10).

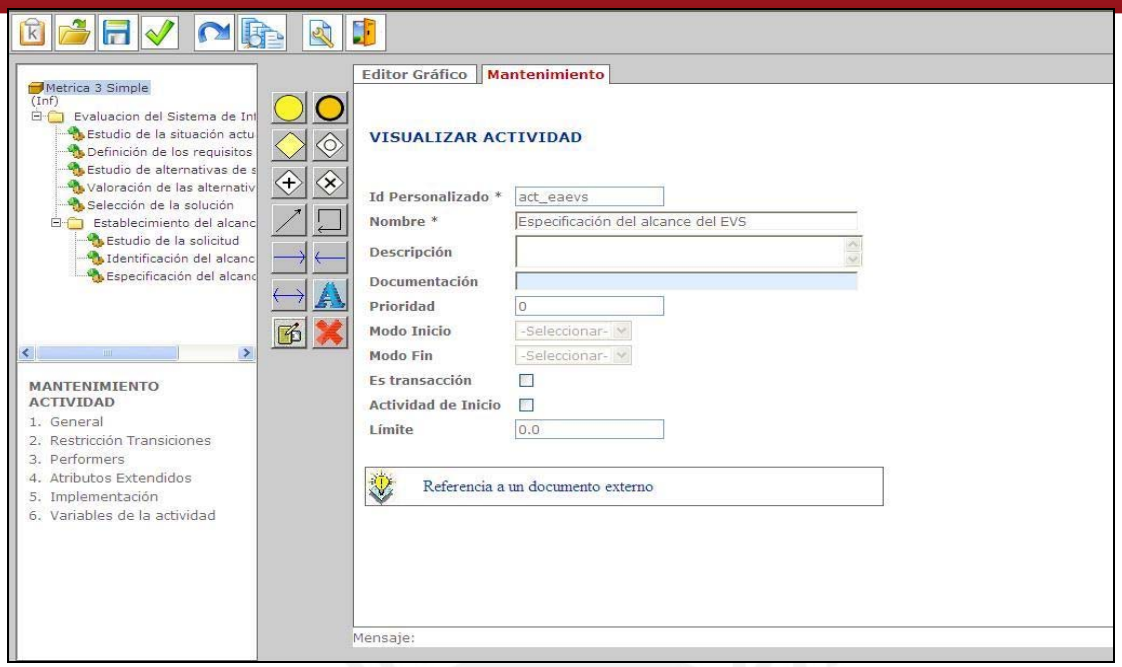


**Ilustración 4-10: Mantenimiento de actividad de una metodología**

### 4.3.4. Módulo de Versionamiento

Para la prueba del módulo de Versionamiento se definió crear una nueva versión del proceso original de Métrica v3.

Para versionar un proceso el usuario deberá seleccionar el proceso del árbol de la vista actual y hacer clic en la opción “Ir a Vista de Versionamiento”. La herramienta mostrará un árbol asociado al proceso seleccionado en el cual se listan todas las versiones existentes para dicho proceso (ilustración 4-11).

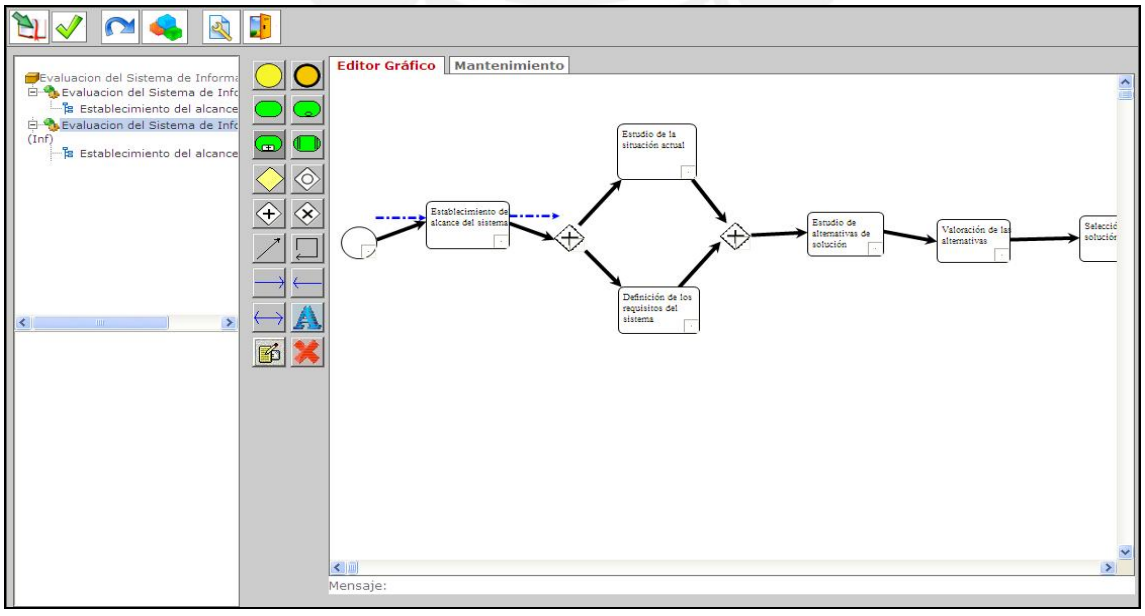


**Ilustración 4-11: Vista de Versionamiento**

Si el usuario desea versionar un proceso deberá seleccionar la versión que será tomada como base en el árbol de la vista y hacer clic en la opción “Nueva Versión”.

Una vez finalizado el versionamiento del proceso, la herramienta agregará en el árbol de versiones un nuevo nodo, el cual corresponderá a la nueva versión del proceso. Si el usuario desea ver el modelado gráfico de la versión generada deberá hacer clic en el nodo correspondiente a esa versión.

La nueva versión creada será tomada como versión vigente del proceso, el cual podrá ser modificado en el módulo Modelador de Procesos.

### 4.3.5. Generación del Archivo XPDL

Para exportar la definición de un paquete registrado en el sistema el usuario deberá abrir una definición previamente registrada y hacer clic en la opción “Generar archivo XPDL”. Como siguiente paso, la herramienta procederá a generar dicho archivo con la definición del paquete según el lenguaje XPDL.

A continuación, en el archivo fuente 4-12 se muestra el archivo XPDL generado para el ejemplo anterior:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xpd:Package Id="pkg_Metrica" Name="Metrica"
  xmlns:xpd=http://www.wfmc.org/2004/XPDL2.0alpha
  xmlns:mjs="http://www.example.org/Jamesa">
  <xpd:Associations>
    <xpd:Association Id="aso_10" Name="asociacion_6" AssociationDirection="From"
      Source="act_eaevs" />
    <xpd:Association Id="aso_9" Name="null" AssociationDirection="To" Target="act_esol" />
    <xpd:Association Id="aso_8" Name="asociacion_2" AssociationDirection="From"
       Source="act_eas" />
    <xpd:Association Id="aso_7" Name="asociacion_1" AssociationDirection="To"
      Target="act_eas" />
    <xpd:Association Id="aso_6" Name="asociacion_6" AssociationDirection="From"
      Source="act_eaevs" />
    <xpd:Association Id="aso_5" Name="null" AssociationDirection="To" Target="act_esol" />
    <xpd:Association Id="aso_2" Name="asociacion_2" AssociationDirection="From"
      Source="act_eas" />
    <xpd:Association Id="aso_1" Name="asociacion_1" AssociationDirection="To"
      Target="act_eas" />
  </xpd:Associations>
  <xpd:Artifacts>
    <xpd:Artifact Id="art_1" Name="Catalogo de Requisitos" ArtifactType="DataObject">
      <xpd:DataObject Id="doc_1" Name="Catalogo de Requisitos" RequiredForStart="true"
        ProducedAtCompletion="false" />
    </xpd:Artifact>
    <xpd:Artifact Id="art_2" Name="Plan de Trabajo" ArtifactType="DataObject">
      <xpd:DataObject Id="doc_2" Name="Plan de Trabajo" RequiredForStart="false"
        ProducedAtCompletion="true" />
```

```
    </xpd:Artifact>
    <xpd:Artifact Id="art_3" Name="Catalogo de Usuarios" ArtifactType="DataObject">
      <xpd:DataObject Id="doc_3" Name="Catalogo de Usuarios" RequiredForStart="false"
        ProducedAtCompletion="true" />
    </xpd:Artifact>
    <xpd:Artifact Id="art_4" Name="Descrip. General del Sistema" ArtifactType="DataObject">
      <xpd:DataObject Id="doc_4" Name="Descripcion General del Sistema"
        RequiredForStart="false" ProducedAtCompletion="true" />
    </xpd:Artifact>
  </xpd:Artifacts>
  <xpd:WorkflowProcesses>
    <xpd:WorkflowProcess Id="EVS" Name="Evaluacion del Sistema de Informacion"
      ProcessType="Abstract" Status="Aborted" AccessLevel="PUBLIC">
      <mjs:ProcessCategory>
        <mjs:ProcessBasic>
          <mjs:VersionInformation IdVersion="0" CurrentVersion="0" />
        </mjs:ProcessBasic>
      </mjs:ProcessCategory>
      <xpd:ProcessHeader>
        <xpd:Created>17/05/2008</xpd:Created>
        <xpd:Limit>0.0</xpd:Limit>
        <xpd:TimeEstimation>
          <xpd:WaitingTime>0.0</xpd:WaitingTime>
          <xpd:WorkingTime>0.0</xpd:WorkingTime>
          <xpd:Duration>0.0</xpd:Duration>
        </xpd:TimeEstimation>
      </xpd:ProcessHeader>
      <xpd:RedefinableHeader PublicationStatus="UNDER_REVISION" />
      <xpd:Activities>
        <xpd:Activity Id="sta_1" Name="nodo_1">
          <xpd:Event><xpd:StartEvent /></xpd:Event>
          <mjs:ActivitySourceType>
            <mjs:ActivityBasic />
          </mjs:ActivitySourceType>
        </xpd:Activity>
        <xpd:Activity Id="conand_6" Name="nodo_5">
          <xpd:Route GatewayType="AND" />
          <mjs:ActivitySourceType>
            <mjs:ActivityBasic />
          </mjs:ActivitySourceType>
        </xpd:Activity>
        <xpd:Activity Id="end_15" Name="nodo_8">
          <xpd:Event><xpd:EndEvent /></xpd:Event>
          <mjs:ActivitySourceType>
            <mjs:ActivityBasic />
          </mjs:ActivitySourceType>
        </xpd:Activity>
        <xpd:Activity Id="act_esa" Name="Estudio de la situación actual" IsATransaction="false"
          StartActivity="false">
          <xpd:Implementation><xpd:No /></xpd:Implementation>
          <mjs:ActivitySourceType>
            <mjs:ActivityBasic />
          </mjs:ActivitySourceType>
        </xpd:Activity>
        <xpd:Activity Id="act_drs" Name="Definición de los requisitos del sistema"
          IsATransaction="false" StartActivity="false">
          <xpd:Implementation><xpd:No /></xpd:Implementation>
          <mjs:ActivitySourceType>
            <mjs:ActivityBasic />
          </mjs:ActivitySourceType>
```

```
</xpd:Activity>
<xpd:Activity Id="act_easol" Name="Estudio de alternativas de solución"
  IsATransaction="false" StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_val" Name="Valoración de las alternativas" IsATransaction="false"
  StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_ssol" Name="Selección de la solución" IsATransaction="false"
  StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="conand_10" Name="nodo_6">
  <xpd:Route GatewayType="AND" />
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
  <xpd:TransitionRestrictions>
    <xpd:TransitionRestriction>
      <xpd:Join Type="AND" />
    </xpd:TransitionRestriction>
    <xpd:TransitionRestriction>
      <xpd:Split Type="AND">
        <xpd:TransitionRefs>
          <xpd:TransitionRef Id="tran7" Name="tran7" />
        </xpd:TransitionRefs>
      </xpd:Split>
    </xpd:TransitionRestriction>
  </xpd:TransitionRestrictions>
</xpd:Activity>
<xpd:Activity Id="act_eas" Name="Establecimiento del alcance del sistema"
  IsATransaction="false" StartActivity="false">
  <xpd:Description>Establecimiento del alcance del sistema</xpd:Description>
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType >
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
  <xpd:InputSets>
    <xpd:InputSet>
      <xpd:Input ArtifactId="art_1">
        <mjs:IOIdentifier OwnerId="0" PersonalId="1">
      </xpd:Input>
    </xpd:InputSet>
  </xpd:InputSets>
  <xpd:OutputSets>
    <xpd:OutputSet>
      <xpd:Output ArtifactId="art_2">
        <mjs:IOIdentifier OwnerId="0" PersonalId="2" />
      </xpd:Output>
    </xpd:OutputSet>
```

```
        </xpd:OutputSets>
      </xpd:Activity>
    </xpd:Activities>
    <xpd:Transitions>
      <xpd:Transition Id="tran1" From="sta_1" To="act_eas" Name="transicion1">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran3" From="conand_6" To="act_esa" Name="tran3">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran4" From="conand_6" To="act_drs" Name="tran4">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran5" From="act_esa" To="conand_10" Name="tran5">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran6" From="act_drs" To="conand_10" Name="tran6">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran8" From="act_easol" To="act_val" Name="tran8">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran9" From="act_val" To="act_ssol" Name="tran9">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran10" From="act_ssol" To="end_15" Name="tran10">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      </xpd:Transition>
      <xpd:Transition Id="tran7" From="conand_10" To="act_easol" Name="tran7">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran2" From="act_eas" To="conand_6" Name="tran2">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
    </xpd:Transitions>
  </xpd:WorkflowProcess>
  <xpd:WorkflowProcess Id="EVS" Name="Evaluacion del Sistema de Informacion"
    ProcessType="Abstract" AccessLevel="PUBLIC">
    <mjs:ProcessCategory>
      <mjs:ProcessBasic>
        <mjs:VersionInformation IdVersion="1" CurrentVersion="1" />
      </mjs:ProcessBasic>
    </mjs:ProcessCategory>
    <xpd:ProcessHeader>
      <xpd:Created>19/05/2008</xpd:Created>
    </xpd:ProcessHeader>
    <xpd:RedefinableHeader PublicationStatus="UNDER_REVISION" />
    <xpd:Activities>
      <xpd:Activity Id="sta_1" Name="nodo_1">
        <xpd:Event><xpd:StartEvent /></xpd:Event>
        <mjs:ActivitySourceType>
          <mjs:ActivityBasic />
        </mjs:ActivitySourceType>
      </xpd:Activity>
      <xpd:Activity Id="conand_6" Name="nodo_5">
        <xpd:Route GatewayType="AND" />
        <mjs:ActivitySourceType>
          <mjs:ActivityBasic />
        </mjs:ActivitySourceType>
```

```
</xpd:Activity>
<xpd:Activity Id="end_15" Name="nodo_8">
  <xpd:Event><xpd:EndEvent /></xpd:Event>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_esa" Name="Estudio de la situación actual" IsATransaction="false"
  StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_drs" Name="Definición de los requisitos del sistema"
  IsATransaction="false" StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_easol" Name="Estudio de alternativas de solución"
  IsATransaction="false" StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_val" Name="Valoración de las alternativas" IsATransaction="false"
  StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="act_ssol" Name="Selección de la solución" IsATransaction="false"
  StartActivity="false">
  <xpd:Implementation><xpd:No /></xpd:Implementation>
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
</xpd:Activity>
<xpd:Activity Id="conand_10" Name="nodo_6">
  <xpd:Route GatewayType="AND" />
  <mjs:ActivitySourceType>
    <mjs:ActivityBasic />
  </mjs:ActivitySourceType>
  <xpd:TransitionRestrictions>
    <xpd:TransitionRestriction>
      <xpd:Join Type="AND" />
    </xpd:TransitionRestriction>
    <xpd:TransitionRestriction>
      <xpd:Split Type="AND">
        <xpd:TransitionRefs>
          <xpd:TransitionRef Id="tran7" Name="tran7" />
        </xpd:TransitionRefs>
      </xpd:Split>
    </xpd:TransitionRestriction>
  </xpd:TransitionRestrictions>
</xpd:Activity>
```

```
      <xpd:Activity Id="act_eas" Name="Establecimiento del alcance del sistema"
        IsATransaction="false" StartActivity="false">
        <xpd:Description>Establecimiento del alcance del sistema</xpd:Description>
        <xpd:Implementation><xpd:No /></xpd:Implementation>
        <mjs:ActivitySourceType>
          <mjs:ActivityBasic />
        </mjs:ActivitySourceType>
        <xpd:InputSets>
          <xpd:InputSet>
            <xpd:Input ArtifactId="art_1">
              <mjs:IOIdentifier OwnerId="0" PersonalId="7" />
            </xpd:Input>
          </xpd:InputSet>
        </xpd:InputSets>
        <xpd:OutputSets>
          <xpd:OutputSet>
            <xpd:Output ArtifactId="art_2">
              <mjs:IOIdentifier OwnerId="0" PersonalId="8" />
            </xpd:Output>
          </xpd:OutputSet>
        </xpd:OutputSets>
      </xpd:Activity>
    </xpd:Activities>
    <xpd:Transitions>
      <xpd:Transition Id="tran1" From="sta_1" To="act_eas" Name="transicion1">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran3" From="conand_6" To="act_esa" Name="tran3">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran4" From="conand_6" To="act_drs" Name="tran4">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran5" From="act_esa" To="conand_10" Name="tran5">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran6" From="act_drs" To="conand_10" Name="tran6">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran8" From="act_easol" To="act_val" Name="tran8">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran9" From="act_val" To="act_ssol" Name="tran9">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran10" From="act_ssol" To="end_15" Name="tran10">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran7" From="conand_10" To="act_easol" Name="tran7">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
      <xpd:Transition Id="tran2" From="act_eas" To="conand_6" Name="tran2">
        <xpd:Condition Type="CONDITION" />
      </xpd:Transition>
    </xpd:Transitions>
  </xpd:WorkflowProcess>
  <xpd:WorkflowProcess Id="met1" Name="Metrica 3 Simple">
    <xpd:ProcessHeader>
      <xpd:Description>Adaptación de Métrica 3 para una PYME</xpd:Description>
    </xpd:ProcessHeader>
```

```
    <xpd:RedefinableHeader PublicationStatus="UNDER_REVISION" />
    <mjs:ProcessCategory>
      <mjs:ProcessMethodology>
        <mjs:VersionInformation IdVersion="0" CurrentVersion="1" />
        <mjs:ProcessesReference>
          <mjs:ProcessReference IdProcess="EVS" IdProcessVersion="0" />
        </mjs:ProcessesReference>
      </mjs:ProcessMethodology>
    </mjs:ProcessCategory>
    <xpd:Activities>
      <xpd:Activity Id="2" Name="Especificación del alcance del EVS">
        <mjs:ActivitySourceType>
          <mjs:ActivityMethodology>
            <mjs:ActivityReference IdActivityReference="act_eaevs" Level="1" />
            <mjs:ProcessReference IdProcess="Establecimiento del alcance del sistema0"
              IdProcessVersion="0" />
          </mjs:ActivityMethodology>
        </mjs:ActivitySourceType>
        <xpd:OutputSets>
          <xpd:OutputSet>
            <xpd:Output ArtifactId="art_2">
              <mjs:IOIdentifier OwnerId="6" PersonalId="1" />
            </xpd:Output>
          </xpd:OutputSet>
        </xpd:OutputSets>
      </xpd:Activity>
      <xpd:Activity Id="3" Name="Definición de los requisitos del sistema">
        <mjs:ActivitySourceType>
          <mjs:ActivityMethodology>
            <mjs:ActivityReference IdActivityReference="act_drs" Level="0" />
            <mjs:ProcessReference IdProcess="EVS" IdProcessVersion="0" />
          </mjs:ActivityMethodology>
        </mjs:ActivitySourceType>
      </xpd:Activity>
      <xpd:Activity Id="4" Name="Estudio de alternativas de solución">
        <mjs:ActivitySourceType>
          <mjs:ActivityMethodology>
            <mjs:ActivityReference IdActivityReference="act_easol" Level="0" />
            <mjs:ProcessReference IdProcess="EVS" IdProcessVersion="0" />
          </mjs:ActivityMethodology>
        </mjs:ActivitySourceType>
      </xpd:Activity>
      <xpd:Activity Id="5" Name="Selección de la solución">
        <mjs:ActivitySourceType>
          <mjs:ActivityMethodology>
            <mjs:ActivityReference IdActivityReference="act_ssol" Level="0" />
            <mjs:ProcessReference IdProcess="EVS" IdProcessVersion="0" />
          </mjs:ActivityMethodology>
        </mjs:ActivitySourceType>
      </xpd:Activity>
    </xpd:Activities>
    <xpd:Transitions>
      <xpd:Transition Id="con_1" From="sta_1" To="act_eaevs" Name="transicion_27" />
      <xpd:Transition Id="con_2" From="act_eaevs" To="act_drs" Name="transicion_28" />
      <xpd:Transition Id="con_3" From="act_drs" To="act_easol" Name="transicion_29" />
      <xpd:Transition Id="con_4" From="act_easol" To="act_ssol" Name="transicion_30" />
      <xpd:Transition Id="con_5" From="act_ssol" To="end_6" Name="transicion_31" />
    </xpd:Transitions>
  </xpd:WorkflowProcess>
  <xpd:WorkflowProcess Id="Establecimiento del alcance del sistema0"
```

```
Name="Establecimiento del alcance del sistema0">
<mjs:ProcessCategory>
  <mjs:ProcessExplosion>
    <mjs:ProcessParent IdProcess="EVS" IdProcessVersion="1" />
    <mjs:ActivityParent IdActivity="act_eas" Level="0" />
  </mjs:ProcessExplosion>
</mjs:ProcessCategory>
<xpd:Activities>
  <xpd:Activity Id="sta_1" Name="nodo_11">
    <xpd:Event><xpd:StartEvent /></xpd:Event>
    <mjs:ActivitySourceType>
      <mjs:ActivityExplosion LevelNumber="1" />
    </mjs:ActivitySourceType>
  </xpd:Activity>
  <xpd:Activity Id="end_5" Name="nodo_15">
    <xpd:Event><xpd:EndEvent /></xpd:Event>
    <mjs:ActivitySourceType>
      <mjs:ActivityExplosion LevelNumber="1" />
    </mjs:ActivitySourceType>
  </xpd:Activity>
  <xpd:Activity Id="act_esol" Name="Estudio de la solicitud" IsATransaction="false"
    StartActivity="false">
    <xpd:Implementation><xpd:No /></xpd:Implementation>
    <mjs:ActivitySourceType>
      <mjs:ActivityExplosion LevelNumber="1" />
    </mjs:ActivitySourceType>
    <xpd:InputSets>
      <xpd:InputSet>
        <xpd:Input ArtifactId="art_1">
          <mjs:IOIdentifier OwnerId="0" PersonalId="9" />
        </xpd:Input>
      </xpd:InputSet>
    </xpd:InputSets>
  </xpd:Activity>
  <xpd:Activity Id="act_ias" Name="Identificación del alcance del sistema"
    IsATransaction="false" StartActivity="false">
    <xpd:Implementation><xpd:No /></xpd:Implementation>
    <mjs:ActivitySourceType>
      <mjs:ActivityExplosion LevelNumber="1" />
    </mjs:ActivitySourceType>
  </xpd:Activity>
  <xpd:Activity Id="act_eaevs" Name="Especificación del alcance del EVS"
    IsATransaction="false" StartActivity="false">
    <xpd:Implementation><xpd:No /></xpd:Implementation>
    <mjs:ActivitySourceType>
      <mjs:ActivityExplosion LevelNumber="1" />
    </mjs:ActivitySourceType>
    <xpd:OutputSets>
      <xpd:OutputSet>
        <xpd:Output ArtifactId="art_2">
          <mjs:IOIdentifier OwnerId="0" PersonalId="10" />
        </xpd:Output>
      </xpd:OutputSet>
    </xpd:OutputSets>
  </xpd:Activity>
</xpd:Activities>
<xpd:Transitions>
  <xpd:Transition Id="transicion_23" From="sta_1" To="act_esol" Name="transicion_23" >
    <xpd:Condition Type="CONDITION" />
  </xpd:Transition>
```

```
        <xpd:Transition Id="transicion_24" From="act_esol" To="act_ias" Name="transici_24" >
          <xpd:Condition Type="CONDITION" />
        </xpd:Transition>
        <xpd:Transition Id="transicion_25" From="act_ias" To="act_eaevs" Name="transi_25" >
          <xpd:Condition Type="CONDITION" />
        </xpd:Transition>
        <xpd:Transition Id="transicion_26" From="act_eaevs" To="end_5" Name="transi_26" >
          <xpd:Condition Type="CONDITION" />
        </xpd:Transition>
      </xpd:Transitions>
    </xpd:WorkflowProcess>
  </xpd:WorkflowProcesses>
</xpd:Package>
```

**Archivo Fuente 4-12: Archivo XPDL generado por la herramienta MJS Process Designer**

# 5. Observaciones, Conclusiones, Recomendaciones

En este capitulo se presentan las observaciones realizadas durante el desarrollo de la tesis, las conclusiones obtenidas y las recomendaciones que se han considerado pertinentes.

## 5.1. Observaciones

- Las investigaciones y el análisis realizados previos al desarrollo de la herramienta permiten identificar los conceptos, las técnicas y estrategias utilizadas en la actualidad para la definición de procesos. Esto permite centrar esfuerzos en las deficiencias o vacíos aun no cubiertos en esta área con el fin de evitar reinventar conceptos o herramientas previamente desarrollados.
- La popularización de conceptos tales como: explosión de niveles, definición de metodologías, gestión de versiones de procesos, entre otros; ha traído como consecuencia que dichos conceptos sean incluidos en algunas herramientas de administración de procesos que se encuentran disponibles en el mercado. Sin embargo, dichas herramientas incluyen estos conceptos mediante la implementación de un lenguaje de definición propio, lo que dificulta su posible interoperabilidad con otras herramientas.
- El modelado y definición de procesos de negocio, en especial en el área de Ingeniería de Software, se desarrolla de forma vertiginosa con la continua aparición de notaciones, lenguajes de definición de procesos, herramientas, entre otros. Cada una de ellas son implementadas por distintos grupos de investigación, quienes abordan el tema de administración de procesos desde diversos enfoques pero siempre con el objetivo de asegurar la adecuada gestión de procesos pertenecientes a una organización.

- La interoperabilidad requerida entre las distintas herramientas utilizadas en el modelado de procesos tiene como consecuencia la creación de lenguajes de definición de procesos formales tales como: XPDL, BPEL y YAWL. Con ello se asegura que los modelos de procesos definidos puedan ser interpretados por otras herramientas afines que utilicen el mismo LDP.
- XPDL es un lenguaje para definir e intercambiar modelos de procesos, el cual a su vez puede ser considerado como la notación textual de BPMN. En capítulos anteriores se mencionó que XPDL en su versión 2.0 se modificó precisamente para reflejar todos y cada uno de los elementos de BPMN, por lo tanto XPDL y BPMN son un binomio a tener muy en cuenta dentro del campo del modelado de procesos de software.
- Dentro de la herramienta solamente se incluye la definición de las principales entidades que ofrece el lenguaje XPDL debido al alto costo (en tiempo y recursos) que implicaba incluirlas en su totalidad dentro de la misma. Asimismo, dado que este costo sobrepasaba el alcance establecido para el presente trabajo de tesis, se optó por dejar pendiente la implementación completa del lenguaje como parte del trabajo futuro a realizar.
- El modelado de procesos y metodologías presupone ciertos conocimientos básicos por parte del usuario modelador, entre los cuales se encuentran: Conceptos de modelado de proceso, conocimiento de notaciones gráficas de modelado de proceso, validación de flujos de trabajo mediante patrones de workflow, entre otros.
- La herramienta ha sido desarrollada usando un análisis, diseño e implementación orientados a objetos. Dicho enfoque hace posible que la herramienta posea un alto grado de estabilidad, con la finalidad de que en un futuro se pueda añadir más funcionalidades u otras características sin mayor dificultad.
- Para la etapa de análisis se utilizó la notación gráfica UML como notación estándar para la construcción de los artefactos de software.
- El desarrollo de un prototipo funcional antes de comenzar la etapa de implementación de la herramienta permite validar si las definiciones y consideraciones inicialmente planteadas durante la etapa de análisis son realmente correctas. En el caso del presente trabajo la elaboración de dicho prototipo permitió identificar las diversas dificultades asociadas al primer diagrama de clases de análisis, esto hizo posible la reingeniería de dicho diagrama dando como resultado la versión utilizada finalmente en el desarrollo del producto.
- Para la elaboración de los diagramas correspondientes a la etapa de análisis y diseño de la herramienta se utilizó el software Rational Rose y el plugin del IDE Eclipse que integra la notación gráfica UML. Este último, facilitó la elaboración final de los diagramas de diseño y secuencia debido a que los atributos y métodos asociados a las clases eran extraídos directamente del código fuente.
- La creciente popularidad de aplicaciones desarrolladas en plataforma Web, ha tenido como consecuencia la aparición de una gran variedad de tecnologías, frameworks, patrones de

diseño y herramientas de desarrollo, muchas de ellas diseñadas para cumplir un mismo objetivo. Es de vital importancia tener en claro cuáles son aquellas tecnologías que se encuentran más orientadas a los objetivos y limitaciones del producto, debido a que este conocimiento sirve de base para una correcta toma de decisiones al momento de elegir las tecnologías que se utilizarán en la elaboración del mismo.

- El uso del framework Spring brinda grandes beneficios para la construcción de aplicaciones en base a componentes, como lo es la herramienta MJS Process Designer. Cada componente ha sido construido independientemente de los otros componentes por cada uno de los integrantes de la tesis en mención. Del mismo modo, la integración de estos componentes se tornó sencilla debido al uso de interfaces, ya que en vez de referenciar directamente una clase dentro del código se hace referencia a interfaces y se puede decidir con qué clases se implementarán dichas interfaces sin tener que modificar el código fuente gracias al principio de Inversión de Control que ofrece el framework.
- La arquitectura propuesta para la herramienta permite cumplir con cinco de las características principales del modelo de calidad planteado en la norma ISO 9126, entre las cuales se tiene: (i) Funcionalidad, ya que permite interactuar con uno o más sistemas debido a la capacidad de modularización que posee y el encapsulamiento de la lógica del negocio; (ii) Fiabilidad, gracias al manejo de errores a través de clases genéricas que brinda el framework Spring; (iii) Usabilidad, la tecnología AJAX permite que la aplicación sea más interactiva y atractiva para el usuario; (iv) Mantenimiento, el cual se hace más rápido y sencillo debido a los principios de Inversión de Control (framework Spring), la separación de capas y modularización, (v) Movilidad, ya que al estar basado en una plataforma Web, puede ser utilizado desde cualquier punto a través de un navegador.
- El cambio de una o más de las tecnologías preseleccionadas para la implementación de una herramienta tiene como consecuencia un retraso en el cronograma del proyecto.

  Durante el desarrollo del presente trabajo de tesis se reemplazó el uso de JDom por XMLBeans como parser de lectura y escritura de archivos en formato XML. Esto trajo como consecuencia la inclusión de horas extras debido a la curva de aprendizaje; sin embargo, este cambio de tecnología disminuyó considerablemente el número de horas destinadas a la implementación de la funcionalidad que permita la generación del archivo XML. Adicionalmente, este cambio trajo como consecuencia la mejora del tiempo de respuesta de la herramienta en el proceso de generación del archivo XML.

  Es así, que en algunos casos se justifica la inclusión de horas adicionales destinadas a superar la curva de aprendizaje, siempre y cuando la evaluación previa del costo beneficio lo amerite.
- Si el desarrollo de una aplicación es realizado por un equipo de personas, es de vital importancia establecer una adecuada gestión de la configuración del sistema, con la finalidad de asegurar la disponibilidad constante de una versión estable de cada artefacto

generado para toda persona involucrada en el citado desarrollo. Entre las acciones más importantes a realizar se tienen: la actualización permanente del diccionario de datos y el establecimiento de un repositorio de versiones para la documentación y módulos desarrollados.

En el caso de no establecer una adecuada gestión de la configuración se podría presentar problemas durante el proceso de desarrollo del sistema, como por ejemplo: Pérdida de versiones desarrolladas, retrasos al no conocer los avances realizados por otros miembros del equipo, entre otros.

- La herramienta MJS Process Designer está enmarcada dentro del proyecto COMPETISOFT (Mejora de Procesos para Fomentar la Competitividad de la Pequeña y Mediana Industria de Software de Ibero América) del programa CYTED (Ciencia y tecnología para el desarrollo) y forma parte de un conjunto de herramientas, PS-PUCP, cuyo objetivo es la gestión de procesos. Cabe mencionar que las demás herramientas recibirán como input las definiciones de procesos generadas en la herramienta MJS Process Designer.
- La extensión desarrollada al lenguaje XPDL se presentó en el 8vo. Simposio de Ingeniería de Software desarrollado por la Sociedad Argentina de Informática (SADIO) – Mar del Plata, obteniendo un alto grado de aprobación por parte del jurado calificador especialista en la materia. (Ver Méritos).
- El desarrollo de la herramienta modeladora de procesos MJS Process Designer se presentó en el IV Congreso Regional de Estudiantes de Ingeniería de Sistemas - Trujillo. (Ver Méritos).

## 5.2. Conclusiones

- Luego de haber ejecutado las pruebas de cada uno de los componentes de la herramienta, las cuales incluían la definición de procesos de marcos referenciales como por ejemplo Métrica v3; se puede concluir que se logró construir una herramienta que brinda los elementos necesarios para la definición y el almacenamiento de procesos.
- Se puede concluir que MJS Process Designer es una herramienta que congrega los conceptos de explosión de niveles, definición de metodologías y gestión de versiones de procesos y metodologías en una sola herramienta. Además, gracias a la extensión realizada sobre el lenguaje XPDL para incluir dichos conceptos, se elimina la necesidad de crear un LDP propio que obstaculice el posible intercambio de dichas definiciones con otras herramientas.
- Se comprobó las bondades que brinda el lenguaje XPDL para la definición de procesos debido a la gran variedad de elementos que incorpora, los cuales a su vez pueden ser representados gráficamente utilizando la notación BPMN.

- Se comprobó que XPDL es un lenguaje flexible para su extensión, ya que se logró extenderlo para incorporar en él los conceptos de explosión de una actividad en niveles, la definición de metodologías basadas en definiciones de procesos pre-existentes y la administración de versiones de procesos y metodologías. (Anexo H: Modelador y Gestor de Versiones de Procesos basado en XPDL).
- Se logró construir una herramienta de carácter multiempresarial ya que permite que usuarios de diferentes organizaciones puedan almacenar las definiciones de sus procesos en un repositorio común y acceder a ellas a través de un esquema de perfiles.
- Se puede afirmar que la herramienta MJS Process Designer cumple con los objetivos de acceso descentralizado por parte de los usuarios que hacen uso de ella. La plataforma Web sobre la cual ha sido construida permite que cualquier usuario registrado en el sistema pueda acceder a ella desde cualquier ubicación geográfica.
- Se desarrolló la herramienta MJS Process Designer de forma que sea amigable y de fácil uso para el usuario. Esto se logró mediante el uso de las tecnologías AJAX y DHTML, las cuales permiten que el funcionamiento de la herramienta sea más interactivo y dinámico. Adicionalmente, el modelado gráfico de procesos se realiza a través operaciones simples e intuitivas para el usuario.
- Se logró construir una herramienta modeladora de procesos con una sólida arquitectura basada en el framework Spring y los patrones de diseño MVC, Business Delegate y Data Access Object, los cuales facilitan su mantenimiento y su posible ampliación en el futuro. El uso de patrones de diseño y buenas prácticas permitió cumplir con las características principales del modelo de calidad planteado en la norma ISO 9126.
- Se logró construir una herramienta con la capacidad de interactuar con otras herramientas que utilicen el mismo lenguaje de definición de procesos formal, salvaguardando así su interoperabilidad.

### 5.3. Recomendaciones

- Si bien es cierto que la presente herramienta está concebida como un proyecto de fin de carrera (pre-grado), ésta ha sido desarrollada de tal forma que sirva de base para que en un futuro, con las correcciones y mejoras necesarias, se convierta en un producto de categoría comercial satisfaciendo las necesidades existentes en el mercado.
- Como trabajo futuro se recomienda culminar con la implementación de las entidades del lenguaje XPDL que no están incluidas dentro de la herramienta, esto con la finalidad de ampliar la gama de entidades disponibles para el modelado de procesos.
- Finalizado el presente trabajo de tesis se recomienda presentar la extensión realizada al lenguaje XPDL ante la WfMC con la finalidad de que sea validada y posteriormente incorporada en un siguiente release del lenguaje.

- Debido a los buenos resultados obtenidos en la extensión realizada al lenguaje XPDL se recomienda continuar enriqueciéndolo, a fin de que pueda incorporar otros conceptos básicos en la administración de procesos, como por ejemplo la definición de métricas, debido a la importancia que presenta para la constante evaluación de los procesos.
- Entre las posibles extensiones de la herramienta se podrían considerar: (i) Manejar un sistema de métricas basado en los tiempos de ejecución de las actividades. (ii) Importar archivos XML que contengan definiciones de procesos basados en el lenguaje XPDL. (iii) Elaborar plantillas que permitan el registro de una definición de proceso siguiendo los lineamientos establecidos por las diversas metodologías o modelos de procesos que existen en la actualidad.
- Actualmente la herramienta sólo es soportada por el navegador Web Internet Explorer. Se recomienda realizar una investigación acerca de las nuevas tecnologías que permitan diagramados vectoriales y que a su vez sean soportados por diferentes tipos de navegadores Web como Mozilla Firefox.

# Bibliografía

[1] ISO/IEC 12207.1995/2002 Information Technology. Software Life Cycle Process.

[2] Norma Técnica Peruana NTP-ISO/IEC12207. Visitado 2007, de http://www.bvindecopi.gob.pe/normas/isoiec12207.pdf

[3] Workflow Management Coalition. Workflow Standard. “Process Definition Interface – XML Process Definition Language" Document Number WFMC-TC-1025 Versión2.0. (2005).

[4] Curtis, B.; Kellner, M.; Over, J.: Processs Modeling. Communications of The ACM, Vol. 35. New York, NY, USA (1992)

[5] Booch, G.; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley, Massachusetts (2005).

[6] Jacobson, I.: Applying UML in The Unified Process. (1998)

[7] Fuggetta, A. Software process: a roadmap. 2000. International Conference on Software Engineering (ICSE). ACM Press.

[8] Derniame, J.C.; Kaba, A.; Warboys, B.: The Software Process: Modelling and Technology, in Software process: principles, methodology, and Technology. C. Montenegro, Editor. (1999).

[9] Manual de Procesos Documentación de Procesos USBI-VER Unidad de Servicios Bibliotecarios y de Información Universidad Veracruzana Región Veracruz-Boca del Río México Febrero 16 de 2003

[10] Gestión de Proyectos: Modelado de procesos. Visitado 2007, de http://www.gestionempresarial.info/VerItemProducto.asp?Id_Prod_Serv=30&Id_Sec=8

[11] BPM: The Promise and the challenge. Visitado 2007, de http://delivery.acm.org/10.1145/990000/984503/verner.pdf?key1=984503&key2=7331696811&coll=GUIDE&dl=GUIDE&CFID=15151515&CFTOKEN=6184618

[12] Anexo I - Análisis, Diseño y Construcción de una herramienta para Modelado de Procesos: MJS Process Designer

[13] Ould, M.: Business Processes: Modelling and Analysis for re–engineering and Improvement. John Wiley & Sons (1995).

[14] Anexo H – Modelador y Gestor de Versiones de Procesos basado en XPDL

[15] Proceso Software y Gestión del Conocimiento. Visitado 2007, de http://alarcos.inf-cr.uclm.es/doc/psgc/doc/psgc-3.pdf

[16] Oasis. "Web Services Business Process Execution Language" WSBPEL Specification Versión2.0. (2005).

[17] YAWL, Yet Another Workflow Language. Visitado 2007, de http://yawlfoundation.org/

[18] SEPM. Software Engineering Process Management. Visitado 2007, de http://www.sei.cmu.edu/programs/sepm/

[19] Cass, A.G.; Lerner, A.; McCall, E. : Little JIL/Juliette: A Process Definition Language and Interpreter. (2000)

[20] jBPM Process Definition Language (JPDL). Visitado 2007, de http://docs.jboss.com/jbpm/v3/userguide/jpdl.html

[21] Joost, R. : Feasibility Study on a Graphical Workflow Editor based on the Workflow Management System "AlphaFlow". (2005)

[22] OMG: The Object Management Group. Visitado 2007, de http://www.omg.org.

[23] The BPMN-XPDL-BPEL value chain. Visitado 2007, de http://kswenson.wordpress.com/2006/05/26/bpmn-xpdl-and-bpel/

[24] Together Workflow: Frequently Asked Questions. Visitado 2007, de http://www.together.at/together/prod/twe/twefaq/index.html.

[25] Shapiro, R.: XPDL 2.0 Integrating Process Interchange and BPMN. (2007)

[26] Pérez, J.; Durán, A.; Ruiz, A.: ¿Por qué OMG ha elegido BPMN para modelar de Procesos de Negocio si ya existe UML? (2007)

[27] Patrón de Diseño. Visitado 2007, de http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

[28] Análisis y Diseño Orientado a Objetos, Patrones de Análisis. Visitado 2007, de http://www.dcc.uchile.cl/~luguerre/cc40b/clase6.html

[29] Buschmann, F.; Henney, K.; Schmidt, D. : Pattern-oriented software architecture. John Wiley & Sons, Ltd. (2007)

[30] Patterns and Software: Essential Concepts and terminology. Visitado 2007, de http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

[31] Microsoft patterns & practices developer center: Model-View-Controller. Visitado 2007, de http://msdn2.microsoft.com/en-us/library/ms978748.aspx

[32] Servlets y Patrón MVC. Visitado 2007, de http://www.dei.inf.uc3m.es/docencia/p_s_ciclo/pa4/practicas/mvc.pdf

[33] Modelo, Vista y Controlador. Visitado 2007, de http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

[34] Framework. Visitado 2007, de http://es.wikipedia.org/wiki/Framework

[35] Nash, M: Java Frameworks and Components: Accelerate Your Web Application Development. Cambridge University Press (2003)

[36] Presentation Frameworks. Visitado 2007, de http://www.sun.com/software/sunone/docs/arch/chapter8.pdf

[37] Struts. Visitado 2007, de http://struts.apache.org/

[38] Spielman, S. : The Struts Framework Practical Guide for Java Programmers. Morgan Kaufmann Publishers (2003)

[39] Spring Framework. Visitado 2007, de http://www.springframework.org/

[40] Jacobi, J.; Fallows, J. : Pro JSF and Ajax Building Rich Internet Components. Apress (2006)

[41] JSF. Visitado 2007, de http://www.itprofessionals.com.pe/j2ee/docs/01-JSF.pdf

[42] Mapping Objects to Relational Databases: O/R Mapping in detail. Visitado 2007, de http://www.agiledata.org/essays/mappingObjects.html

[43] Frameworks de persistencia en Java. Visitado 2007, de http://www.jtech.ua.es/jornadas/06/charlas/Persistencia.pdf

[44] JDBC: Conexión a base de datos en lenguaje Java. Visitado 2007, de http://camoralesm.googlepages.com/jdbc

[45] Van Haecke, B.: JDBC: Java Database Connectivity. IDG Books (1997)

[46] Hibernate: Relational persistence for Java and .Net. Visitado 2007, de http://www.hibernate.org

[47] Hibernate, um Robusto Framework de Persistência Objeto-Relacional. Visitado 2007, de http://gppd.inf.ufrgs.br/wiki/uploads/PODWebSis2004.WEB04Hibernate/WEB04Hibernate.pdf

[48] Iverson, W.: Hibernate: A J2EE™ Developer's Guide. Addison Wesley Professional (2004)

[49] Begin, C.; Goodin, B.; Meadors, L. : iBATIS in Action. Manning Publications Co., Greenwich (2007).

[50] XML Parser. Visitado 2007, de http://www.w3schools.com/xml/xml_parser.asp

[51] JDOM: Frequently Asked Questions. Visitado 2007, de http://www.jdom.org/docs/faq.html

[52] Processing XML with Java. Visitado 2007, de http://www.cafeconleche.org/books/xmljava/chapters/

[53] SAX: Features and Properties. Visitado 2007, de http://sax.sourceforge.net/get-set.html

[54] XMLBeans. Visitado 2007, de http://xmlbeans.apache.org/

[55] JSP: Java Server Pages Standard Tag Library. Visitado 2007, de http://java.sun.com/products/jsp/jstl/

[56] Bayern, S: JSTL in Action. Manning Publications Co., Greenwich (2003).

[57] AJAX. Visitado 2007, de http://es.wikipedia.org/wiki/AJAX

[58] AJAX: Asynchronous JavaScript and XML. Visitado 2007, de http://paginaspersonales.deusto.es/dipina/ajax/CursoAJAX.pdf

[59] Servidor de Aplicaciones. Visitado 2007, de http://es.wikipedia.org/wiki/Servidor_de_aplicaciones#Otros_servidores_de_aplicaci.C3.B3n

[60] Web Server. Visitado 2007, de http://en.wikipedia.org/wiki/Web_server

[61] JBoss Application Server. Visitado 2007, de http://labs.jboss.com/jbossas/

[62] Proyecto 'El libro de Tomcat'. Visitado 2007, de http://tomcatbook.sourceforge.net/es/proj_intro.shtml

[63] Database Management System. Visitado 2007, de http://en.wikipedia.org/wiki/Database_management_system#External_references

[64] PostgreSQL. Visitado 2007, de www.postgresql.org/

[65] PostgreSQL: Documentation. Visitado 2007, de http://www.postgresql.org/docs/8.0/interactive/index.html

[66] MySQL Hispano. Visitado 2007, de www.mysql-hispano.org/

[67] MySQL 5.0 Reference Manual. Visitado 2007, de http://dev.mysql.com/doc/refman/5.0/en/index.html

[68] Together Workflow Editor. Visitado 2007, de http://www.together.at/together/prod/twe/index.html

[69] JOpera for Eclipse. Visitado 2007, de http://www.jopera.ethz.ch/

[70] Bonita: Open Source Workflow. Visitado 2007, de http://bonita.objectweb.org/

[71] Workflow Gen: Product Features. Visitado 2007, de http://www.workflowgen.com/workflow/workflow_software_features.htm

[72] AllFusion Process Modeler 7 (BPwin). Visitado 2007, de http://www.bpwin.ru

[73] XFLOW Process Management System. Visitado 2007, de http://xflow.sourceforge.net/

[74] Java User Group Perú. Visitado 2007, de http://www.jugperu.com/portal/modules.php?name=Surveys&op=results&pollID=10&mode=&order=&thold=

[75] Pino, F.; García, F.; Piattini, M.: Contribución de los Estándares Internacionales a la Gestión de Procesos de Software, Vol. 4. Ciudad Real, España (2007)

**Méritos**