UNIVERSIDAD POLITÉCNICA DE VALENCIA


DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN


# GENERACIÓN DE APLICACIONES WEB A PARTIR DE PROCESOS DE NEGOCIO

TESIS

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

Freddy Geovanny Vargas Lizcano

Supervisores:

Dr. Vicente Pelechano Ferragud

Dr. Victoria Torres Bosch

Septiembre 2012

# Agradecimientos

En primera medida agradezco a Dios por el regalo de la vida y por cuidar de mí durante cada uno de estos años, así como por todas aquellas cosas que me ha brindado durante el camino, familia, experiencias, amistades, conocimientos, alegrías y tristezas, siendo cada una de ellas fortalecedora para mí, tanto a nivel personal como espiritual.

A mis padres Reinaldo y Ofelia quienes desde el primer instante del que tengo recuerdo, han estado siempre ahí brindándome amor, cariño, apoyo, enseñanzas, siendo una guía para convertirme en la persona que soy, gracias infinitas por ser un verdadero modelo a seguir.

A mis hermanos Alexander, Edwin y Wilber quienes junto a mis padres siempre han estado brindándome lo mejor de sí mismos y creando el ambiente necesario para ser una familia donde se priorizan las enseñanzas de Dios y se vive su amor.

A mi novia Nicole y a mis amigos quienes han estado siempre ahí compartiendo conmigo cada una de sus vivencias, esperanzas y sueños, gracias por aportarme lo mejor de sí.

A Vicente y Vicky, mis tutores por confiar en mí, por sus valiosos aportes en el desarrollo de este trabajo y especialmente por su disposición a colaborarme en cualquier tema.

En últimas a todas aquellas personas que siempre han estado ahí,

¡Muchas Gracias!

# Tabla de Contenido

# Lista de Figuras

# Lista de Figuras

**No se encuentran elementos de tabla de ilustraciones.**

# Lista de Tablas

# Abstract

Bearing in mind that the link between business processes and value generation leads to a large majority of participants see business processes as workflows that take place within the tasks of an organization, we believe that business processes can be seen as a formula for running a business and achieve the goals set.

The work is a methodological contribution regarding building web apps developed for the Django framework, the main methodological contribution lies in the definition of a level of abstraction than existing technology for development. (The level of abstraction is focused on defining business processes and then use them to generate the respective applications).

As part of this development we define a metamodel for creating business processes and by means of transformations Xpand the respective web application is generated.

It can only demonstrated that by means of the definition of this top level and the application of the respective transformations,

It is shown that by defining the upper level and the implementation of the respective transformations, web applications are obtained for the Django framework of good quality and a faster, which represent business processes defined by metamodels for specifying processes and system structure respectively.

# Resumen

En la actualidad es muy importante establecer los procesos de negocio que determinan la manera en que funciona una empresa, esto con el fin de poder identificar las oportunidades que se tienen para mejorar y emprender una adecuada alineación hacia la manera de alcanzar los objetivos planteados.

Teniendo presente que el enlace entre procesos de negocio y generación de valor lleva a una gran mayoría de participantes a ver los procesos de negocio como los flujos de trabajo que se efectúan dentro de las tareas de una organización, podemos pensar que los procesos de negocio pueden ser vistos como una fórmula para hacer funcionar un negocio y alcanzar las metas definidas.

El trabajo desarrollado constituye un aporte metodológico en cuanto a la construcción de aplicaciones web desarrolladas para el framework Django, la principal aportación metodológica radica en la definición de un nivel de abstracción superior al tecnológico ya existente para el desarrollo. (El nivel de abstracción superior está enfocado a definir procesos de negocio y luego utilizarlos para generar las respectivas aplicaciones).

En el marco de este desarrollo se define un metamodelo que permite crear procesos de negocio y mediante transformaciones Xpand se genera la respectiva aplicación web.

Queda demostrado que mediante la definición de este nivel superior y la aplicación de las respectivas transformaciones, se obtienen aplicaciones web para el framework Django de buena calidad y de una forma más rápida, las cuales representan los procesos de negocio definidos mediante los metamodelos para la especificación de los procesos y la estructura del sistema respectivamente.

# 1. Introducción

Hoy en día, la tecnología es un factor crítico en el éxito o fracaso de un negocio y por tanto las empresas actuales han ido evolucionando al mismo tiempo que lo han hecho las nuevas tecnologías, hasta el punto de que ahora mismo los sistemas informáticos son esenciales para seguir manteniéndose competitivas. Con el objeto de que estas empresas puedan obtener los sistemas software que las soportan en un menor tiempo, los modelos de gestión y desarrollo del software han tenido que ir evolucionando para ser cada vez más eficientes y ágiles. Así, para conservar su competitividad, las compañías de desarrollo de software se están viendo forzadas a reducir sus costes y el tiempo de desarrollo, sin que ello signifique una penalización sobre la calidad de los productos que se generan. La consecución de este doble objetivo pasa por la utilización de herramientas muy perfeccionadas no solo de soporte al proceso de desarrollo sino también para la gestión de dicho proceso.

Durante el proceso de desarrollo, un aporte importante en las fases de entendimiento y modelado de los requisitos del sistema lo constituye el enfoque basado en el Modelado de los Procesos de Negocio (BPM). Mediante esta estrategia, se pretende que antes de empezar a desarrollar cualquier tipo de software, se deba realizar un análisis y un modelado exhaustivo de los procesos de negocio de la organización. De ésta forma, se conseguirá un mayor conocimiento de ellos y, además, mediante su análisis se podrá intentar mejorar el rendimiento de cada uno de ellos. La importancia de este enfoque se convierte en crucial desde el momento en que ayuda a las empresas a estar constantemente mejorando y adaptándose a los continuos cambios del mercado, con unos sistemas de información que se adecuan perfectamente a dichos cambios.
La importancia que está adquiriendo esta corriente es tal que ya se han desarrollado algunos lenguajes específicos para el Modelado de Procesos de Negocio, como es la Notación para el Modelado de Procesos de Negocio (BPMN) por el que el grupo de gestión de objetos (OMG) está apostando fuertemente. Sin embargo, la incorporación efectiva del modelado de procesos de negocio al proceso de desarrollo pasa por el diseño de un proceso de desarrollo dirigido por procesos de negocio y por la existencia

de herramientas de soporte al proceso que permitan trabajar de forma ágil y eficiente con los conceptos propuestos por dicha disciplina.

## 1.1 Motivación

Teniendo en cuenta que los modelos de desarrollo software están volcando su interés sobre los procesos de negocio y que el modelado de procesos de negocio (BPM), trae de forma conjunta los procesos, las personas y la tecnología de la información podemos considerar una gran idea empresarial la adopción de BPM. El modelo de desarrollo tradicional se centra en descubrir los requisitos del sistema para modelarlos como casos de uso. Se ha demostrado que siguiendo este modelo las relaciones entre las diferentes funcionalidades que forman parte de un mismo proceso de negocio no son tan visibles, por lo que posteriormente se hace necesario un esfuerzo extra para readaptar la implementación de dichas funcionalidades con el objeto de ajustar las relaciones entre ellas. En contraposición, el desarrollo dirigido por procesos de negocio se enfoca en una captación de requisitos basada en la identificación y modelado de los procesos de negocio.

Dichos procesos estarán presentes y guiarán todo el ciclo de desarrollo. Un proceso de negocio comprende un conjunto de actividades y decisiones del negocio interrelacionadas para conseguir o lograr un objetivo. Así, el modelado de los procesos de negocio se refiere a una teoría o estrategia para la administración y análisis del negocio de una organización, para que pueda ser rápidamente evolucionable y adaptable a los nuevos retos del mercado y a las nuevas soluciones tecnológicas, además de adaptarse a la perfección a los procesos definidos en ese momento. Adoptar esta estrategia significa tratar los procesos de negocio de una forma comprensiva y dinámica, analizándolos primero para comprenderlos a la perfección y, posteriormente, reconocer aquellas partes no deseadas o superfluas, y así mejorar su rendimiento.

Para la gestión de los procesos de negocio, de una organización, se proponen una serie de etapas y actividades, que establecen el ciclo de vida que se debe seguir para alcanzar, de una forma eficaz, todos los objetivos y beneficios perseguidos por BPM.
En la figura 1.1 se observa el ciclo de vida iterativo de los procesos de negocio.

Las principales fases son:

• **Descubrimiento:** El principal objetivo es descubrir y entender cada uno de los procesos de negocio que forman la organización. Especificando todos los detalles de cada uno de los requisitos y centrándose, principalmente, en las funcionalidades clave del sistema.

• **Análisis:** En esta fase se analizan cada uno de los procesos de negocio del sistema, modelándolos con las nuevas características y reglas que deben seguir para obtener una mayor productividad.

• **Desarrollo:** Aquí se desarrollan los procesos de negocio analizados y diseñados en la etapa anterior.

• **Monitorizar:** Cada proceso de negocio debe ser medible para saber su grado de éxito y calidad con el que ha sido llevado a cabo; de esta forma, se pueden analizar los resultados de cada uno de los procesos para que puedan ser redefinidos y optimizados.

• **Optimizar:** Aquellos procesos que no han cumplido las expectativas deseadas, bien porque no poseen un conjunto coherente de tareas, o bien porque las necesidades han cambiado, son optimizados para que puedan mejorar su rendimiento y así también el de la empresa. Si se necesita crear un nuevo software que soporte las optimizaciones, será Imprescindible que estos procesos pasen, de nuevo, a la fase de análisis.

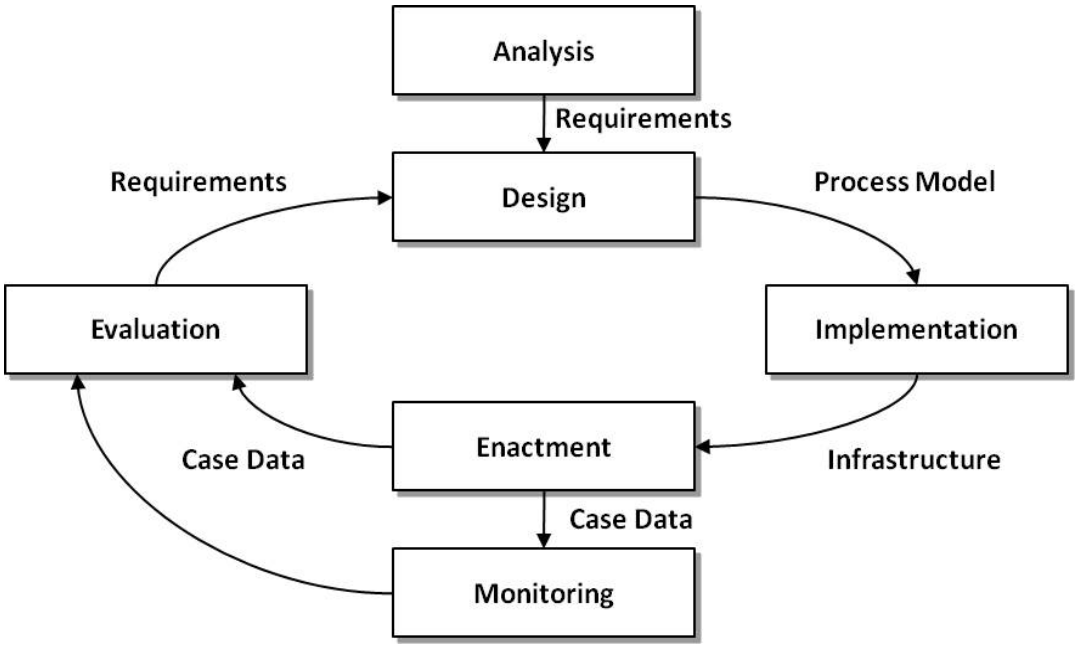


**Figura 1.1** Ciclo de vida iterativo de los procesos de negocio.

Actualmente no existen herramientas de soporte al proceso de desarrollo que ofrezcan un enfoque basado en procesos de negocio, siendo esta una de las grandes motivaciones de este trabajo. Dado que a partir de una estructura clara de procesos de negocio y basándose en la Arquitectura dirigida por modelos (MDA) se pretende dar soporte al desarrollo de aplicaciones.

## 1.2 Presentación del problema

En la actualidad no se cuenta con herramientas que nos permitan la generación de aplicaciones WEB partiendo de estructuras de procesos de negocios que sigan los lineamientos que plantea la OMG con el BPM y que a su vez utilicen la arquitectura dirigida por modelos (MDA) para tal fin.

Todo esto supone un alto coste en cuanto al desarrollo de aplicaciones independientes ya que debemos trabajar por separado los procesos de negocio y luego pasar a realizar la aplicación web rescatando las funcionalidades que podamos del paso anterior y agregando el resto funcionalidades de acuerdo a los requisitos captados.

Por tanto es muy importante evaluar primero la utilidad de cada una de las herramientas por separado y la complejidad tanto de conocimiento, como el factor económico que puede suponer para alcanzar el objetivo que se plantea.

Es por esto que aparece como planteamiento la siguiente pregunta: ¿Cómo podemos dar soporte al desarrollo de aplicaciones web, utilizando la Arquitectura dirigida por modelos, donde podamos partir de una estructura clara de Procesos de negocio según los lineamientos BPM estipulados por la OMG?

## 1.3 Solución Propuesta

Para responder a la pregunta planteada en la sección anterior, esta tesis tiene como objetivo principal obtener una aplicación web sobre un framework específico, a partir de una estructura clara de procesos de negocios y un motor de flujo de trabajo, al cual se aplicará el proceso de desarrollo de software dirigido por modelos (MDA).

Para alcanzar el objetivo se ha procedido de la siguiente manera:

1. **Definir** un metamodelo con la estructura general de los procesos de negocio y los lineamientos establecidos en la Business Process Management (BPM) por la OMG. Dicho metamodelo se ha construido a partir del lenguaje de procesos utilizado en el motor de procesos OpenWFE, así mismo se ha definido otro metamodelo que representa la estructura del sistema.

Posteriormente y partiendo de la ingeniería dirigida por modelos (MDA), se han llevado a cabo los siguientes pasos:

2. **Diseño de un modelo de datos**. Donde consideramos a un modelo de datos como un escenario que presenta un flujo de trabajo de un proceso de negocio definido.
3. **Generar de forma automática código** a partir del modelo validado mediante el fichero checks y aplicando las plantillas necesarias, obtenemos como resultado la implementación de código.
4. **Implantación de código sobre framework específico.** A partir del código obtenido en el paso anterior, se generará una aplicación web que da soporte al proceso de negocio que ha sido definido en la etapa de modelado.

El seguir una aproximación MDA nos garantiza el contar con un alto nivel de abstracción y además una independencia de la tecnología que se utilizará en la implementación final.

El aplicativo que se obtiene consta de tres partes fundamentales que son:

1. Modelos ó clases del sistema.
2. Vistas con la lógica del negocio, representando las funcionalidades del sistema.
3. Páginas HTML que tienen como fin la presentación de la información que se maneja en las vistas.

Mediante esta secuencia de pasos conseguimos de una forma rápida y más sencilla la simulación del sistema final, que puede ser ajustado posteriormente con herramientas estándar.

### 1.4 Estructura de la Tesis

Esta tesis se encuentra estructurada en 9 capítulos. El capítulo 2 muestra el contenido referente a los procesos de negocios parte base de este proyecto. En el capítulo 3 se explica con detalle la Arquitectura dirigida por modelos (MDA) otro de los pilares base de este trabajo. En el capítulo 4 encontramos el contexto tecnológico en el cual se desarrolla el presente trabajo. El capítulo 5 nos presenta algunos trabajos relacionados con el área del mismo. El capítulo 6 presenta la propuesta metodológica realizada y se explica con cierto nivel de detalle los metamodelos utilizados tanto para definir la estructura de los procesos de negocios como el metamodelo para el diagrama de clases. En el capítulo 7 se aplica la propuesta realizada a un caso de estudio seleccionado. El capítulo 8 presenta las conclusiones.

# 2. Procesos de Negocio

Hoy en día son cada vez más las empresas que para conseguir sus objetivos, organizan su actividad por medio de un conjunto de procesos de negocio. Cada uno de ellos se caracteriza por una colección de datos que son producidos y manipulados mediante un conjunto de tareas, en las que ciertos agentes (por ejemplo, trabajadores o departamentos) participan de acuerdo a un flujo de trabajo determinado. Además, estos procesos se hallan sujetos a un conjunto de reglas de negocio, que determinan las políticas y la estructura de la información de la empresa. Por tanto, la finalidad del modelado del negocio es describir cada proceso del negocio, especificando sus datos, actividades (o tareas), roles (o agentes) y reglas de negocio.

El primer paso del modelado del negocio consiste en capturar los procesos de negocio de la organización bajo estudio. La definición del conjunto de procesos del negocio es una tarea crucial, ya que define los límites del proceso de modelado posterior. De acuerdo con el concepto de objetivo estratégico de [Cockburn, 1997] capturamos los procesos del negocio a partir de los objetivos principales de la empresa. En primer lugar, consideramos los objetivos estratégicos de la organización.

Teniendo en cuenta que estos objetivos van a ser muy complejos y de un nivel de abstracción muy alto, serán descompuestos en un conjunto de subobjetivos más concretos, que deberán cumplirse para conseguir el objetivo estratégico. Estos subobjetivos pueden a su vez ser descompuestos en otros, de manera que se defina una jerarquía de objetivos.

## 2.1 Qué es un Proceso de Negocio

Un proceso de negocio es una colección de actividades diseñadas para producir una salida específica para un cliente o un mercado en particular. Esto implica un fuerte énfasis en **cómo** se realiza el trabajo dentro de una organización, en contraposición con un enfoque del producto en **qué** se produce. Por lo tanto, el proceso es una secuencia

específica de actividades de trabajo a través del tiempo y del espacio, con un inicio, un final y unas entradas y salidas claramente definidas: una estructura para la acción.

## 2.2 Entradas, Recursos e Información

Los procesos de negocio emplean información para adaptar o completar sus actividades. La información, a diferencia de los recursos, no se consume en los procesos, sino que se usa como parte del proceso de transformación. La información puede provenir de fuentes externas, de los clientes, de las unidades organizacionales internas e inclusive puede ser el producto de otros procesos.
Un recurso es una entrada para un proceso de negocio y, a diferencia de la información, típicamente se consume durante el procesamiento. Por ejemplo, a medida que cada servicio diario de tren sale y registran las novedades, el recurso servicio se usa tanto como concierna al proceso de registración de novedades de tiempos de los trenes.

## 2.3 Identificación de Roles del Entorno del Negocio

Una vez se han identificado los procesos de negocio, es preciso encontrar los agentes involucrados en su realización. Cada uno de estos agentes o actores del negocio desempeña cierto papel (juega un rol) cuando colabora con otros para llevar a cabo las actividades que conforman dicho caso de uso del negocio. De hecho, identificaremos los roles que son jugados por agentes de la propia empresa (que incluyen trabajadores, departamentos y dispositivos físicos) o agentes externos (como clientes u otros sistemas).

Para tener una visión general de los diferentes procesos de negocio de la organización, puede construirse un diagrama de casos de uso del negocio, en el cual aparece cada proceso del negocio como un caso de uso. Este diagrama permite mostrar los límites y el entorno de la organización bajo estudio. Sólo se mostrarán en este diagrama los actores del negocio correspondientes a los roles externos al sistema, de forma que los procesos de negocio en los que sólo tomen parte roles internos a la organización no estarán conectados a ningún actor (ver figura 2.1).

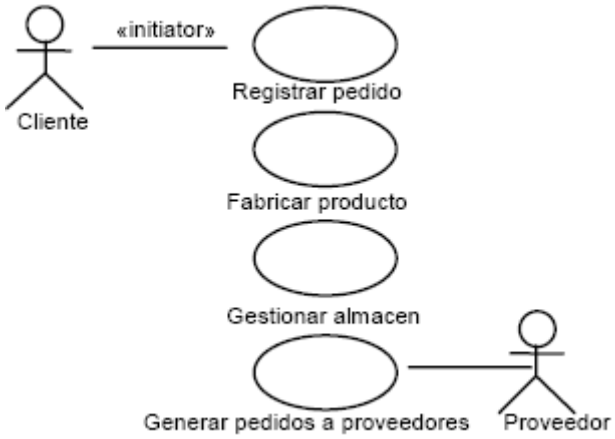


**Figura 2.1** Diagrama de casos de uso del negocio para el sistema de producción

## 2.4 Descripción de los Casos de Uso del Negocio

El siguiente paso dentro del modelado del negocio es introducirse en cada uno de los casos de uso del negocio identificados, para describirlo en detalle. Podemos considerar un caso de uso Registrar Pedido.

A continuación, hemos de determinar los agentes internos que juegan un rol en cada caso de uso del negocio. Hasta el momento hemos identificado los roles que pertenecen al entorno de la organización. Ahora es necesario estudiar la descripción de cada caso de uso del negocio, y observar el conjunto completo de roles involucrados, tanto externos como internos a la organización. Los roles del caso de uso del negocio Registrar pedido son: Cliente, Comercial, Jefe Técnico, y Jefe Producción (siendo los tres últimos internos al sistema). Se describe a continuación el caso de uso Registrar Pedido:

1. El cliente envía una orden de pedido, que debe incluir la fecha de solicitud, datos del cliente y productos solicitados. Es posible que sea un empleado del departamento comercial quien introduzca el pedido, a petición de un cliente que realizó su pedido por teléfono o lo envió por fax o correo ordinario al departamento comercial de la empresa.

2. El empleado revisa el pedido (completándolo, si es necesario), y comienza su procesamiento enviándolo al jefe técnico, que está encargado de su análisis.

3. El jefe técnico analiza la viabilidad de cada producto del pedido por separado:

· Si el producto pedido está en el catálogo, su fabricación es aceptada.

· En caso contrario, es considerado un producto especial, y el jefe técnico estudia su producción:

- Si es viable, la fabricación del producto especial es aceptada.

- Si no es viable, el producto especial no será fabricado.

4. Una vez estudiado el pedido completo, el jefe técnico Informa al departamento comercial de la aceptación o rechazo de cada producto pedido; Si todos los productos de un pedido han sido aceptados, se crea una orden de trabajo para cada producto, a partir de una plantilla de fabricación (estándar si el producto estaba catalogado, o una nueva, específicamente diseñada para el producto, si éste no estaba en el catálogo). Cada orden de trabajo es enviada al jefe de producción, y queda pendiente de su lanzamiento.

5. El comercial comunica al cliente el resultado final del análisis de su pedido.

La colaboración entre los roles para llevar a cabo un caso de uso del negocio, puede ser representado en un diagrama de roles, en el que cada rol, se presenta como una clase en el Lenguaje Unificado de Modelado (UML) estereotipada, donde aparece asociada con los roles con los que puede colaborar (ver figura 2.2). Por tanto, este diagrama permite expresar el conocimiento que unos roles tienen de otros, así como las características (multiplicidad) de cada relación entre roles.

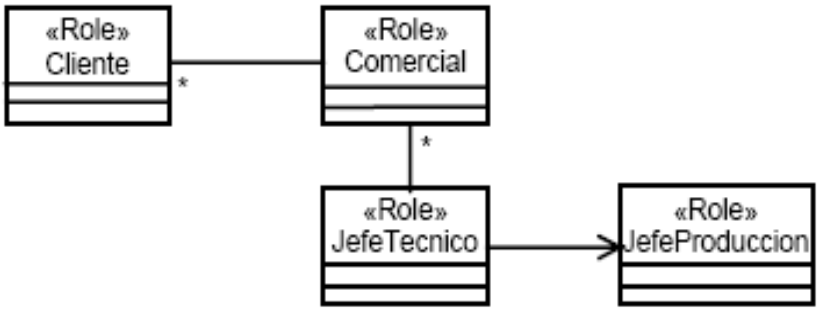


**Figura 2.2.** Diagrama de roles para el caso de uso del negocio Registrar Pedido

Ahora lo que deseamos es mostrar el aspecto de comportamiento de la colaboración. Para ello utilizaremos diagramas de secuencia UML (figura 2.3), en los que los objetos denotan las instancias de los roles que intervienen en la interacción.

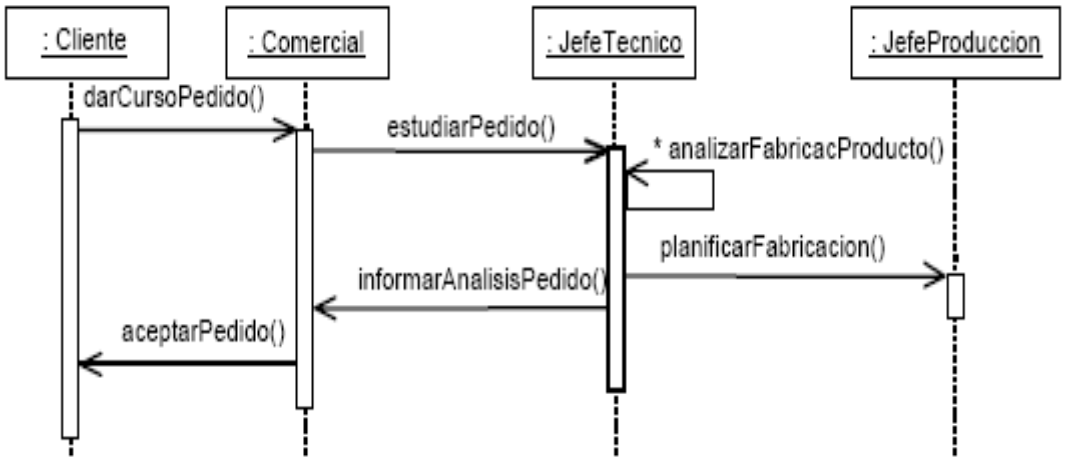


**Figura 2.3.** Diagrama de secuencia caso de uso del negocio Registrar Pedido

## 2.5 Modelado de procesos de negocio

Dentro de BPM existen diferentes notaciones para modelar los procesos de negocio:

- IDEF [Mayer el al., 1992]
- Diagramas de actividades UML [Dumas & ter Hofstede, 2001]
- ebXML BPSS [Hofreiter et al., 2002]
- Business Process Modeling Notation [OMG, 2006]

Todas ellas comparten la capacidad para modelar la secuencia de actividades, los participantes involucrados en el proceso y los datos y mensajes intercambiados entre ellos.

Notación para el modelado de procesos de negocio (BPMN) es la más extendida actualmente, fue desarrollada por el consorcio de Iniciativas de Gestión de de Procesos de Negocio (BPMI) con el objetivo de definir una notación que fuera fácilmente comprendida por los distintos participantes en el modelado de procesos (analistas, desarrolladores y clientes). La especificación fue adoptada como estándar en 2005 por el Object Management Group (OMG) para el modelado de procesos de negocio.

BPMN proporciona cuatro categorías de elementos de modelado:

1. Elementos de flujo: Eventos, Actividades y Bifurcaciones (Gateway).
2. Elementos de conexión: Secuencias, Mensajes y Asociaciones.
3. Franjas de responsabilidad (Swinlane): Pool y Lane.
4. Artefactos: Grupo, Objeto de datos y Asociaciones.

### 2.5.1 Elementos de flujo

Se usan para definir cómo trabaja el proceso de negocio (comportamiento). Existen tres tipos de elementos de flujo:

- **Evento**, un evento representa situaciones que afectan al flujo de ejecución de un proceso. Todo evento tiene una causa (disparador de alguna acción) y un efecto (un resultado). Con los eventos se consigue que un proceso se pueda iniciar, interrumpir, detener entre otras. Existen tres tipologías de eventos, según su representación en el proceso:

1. Eventos de inicio (Start), para representar flujos de entrada.
2. Eventos intermedios (Intermediate), para representar flujos tanto de entrada como de salida.
3. Eventos de fin (End), para representar flujos de salida.

A continuación se presentan los elementos de flujo (eventos) en la Notación para el Modelado de Procesos de Negocio BPMN (figura 2.4).

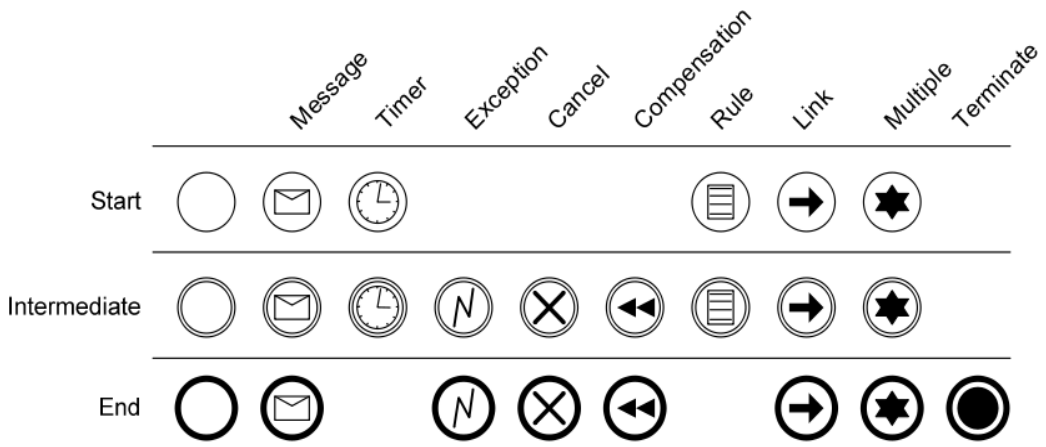


**Figura 2.4.** Elementos de flujo (Eventos) en notación BPMN

- **Actividad**, es un término genérico para denominar una acción hecha por un participante de un proceso. Una actividad puede ser en función de su ejecución (Manual ó Automática). Las Actividades pueden ser atómicas (task) o no atómicas (sub-process). Además, se incluyen atributos en la notación para indicar si la actividad se realiza una sola vez o se repite, detallando si las repeticiones se hacen de manera secuencial (looping) o en paralelo (instancias múltiples). En la figura 2.5 se muestran los tipos de Actividades definidas:

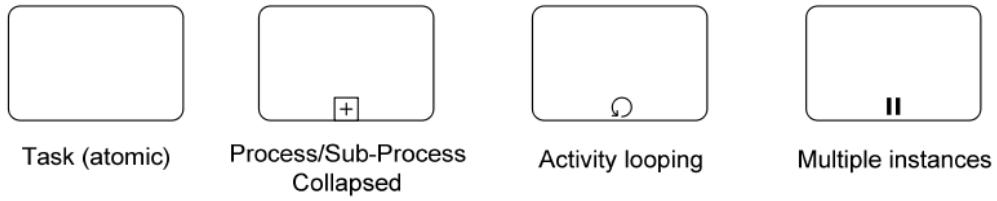


**Figura 2.5** Notación BPMN – Elementos de flujo: Actividades

- **Bifurcaciones (Gateways)**, es usado para controlar la convergencia y/o divergencia en la secuencia del flujo de proceso. Estas bifurcaciones determinan divisiones, uniones y combinaciones, algunos tipos son: decisión exclusiva, basada en un evento, inclusiva. En la figura 2.6 se presentan los tipos de Bifurcaciones definidas.


**Figura 2.6.** Notación BPMN – Elementos de flujo: Bifurcaciones

### 2.5.2 Elementos de conexión

Son usados para conectar entre sí, elementos de flujo, o elementos de flujo con otros elementos BPMN. En la notación BPMN, se definen los siguientes tipos de elementos de conexión:

- **Secuencias**, indican el orden en el cual las actividades son ejecutadas en el proceso. Este tipo de elemento se especializa en: Normal, Conditional y Default.
- **Mensajes**, indican el flujo de los mensajes que intervienen entre dos participantes.
- **Asociaciones**, son usados para indicar información de asociación (en texto o gráfica) a los elementos de flujo.

En la figura 2.7 se observan los diversos tipos de elementos de conexión.

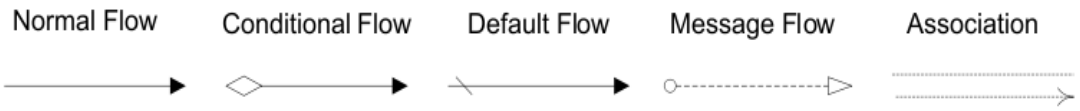


**Figura 2.7.** Notación BPMN – Elementos de flujo: Conexión

### 2.5.3 Swimlanes

Son los elementos gráficos utilizados para organizar las actividades del flujo en diferentes categorías visuales que representan áreas funcionales, roles o responsabilidades. Dentro de los Canales encontramos:

- Pool, actúa como contenedor de un proceso y representa un participante, entidad o rol dentro del proceso.
- Lane, subdivide los elementos Pool en entidades más lógicas. Representa los

diversos participantes al interior de una organización.

En la figura 2.8 se muestra los tipos de Swimlanes.

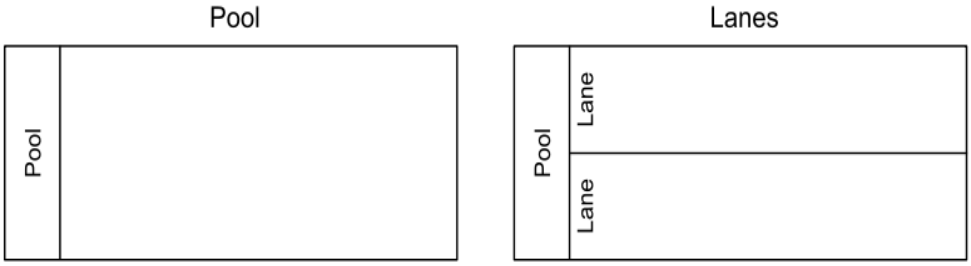


**Figura 2.8** Notación BPMN – Swimlanes

### 2.5.4 Artefactos

Son elementos que no influyen en el cambio del flujo de ejecución, pero su objetivo es proveer información adicional sobre el proceso. La notación BPMN define los siguientes artefactos:

- **Grupo**, Se utiliza para agrupar un conjunto de actividades, ya sea para efectos de documentación o análisis, no afecta la secuencia del flujo.
- **Objeto de datos**, Permite mostrar la información que una actividad necesita, como las entradas y las salidas.
- **Anotaciones**, permite adicionar información, para clarificar la lectura del diagrama.

Los artefactos que se consideran dentro de BPMN son presentados en la figura 2.9.

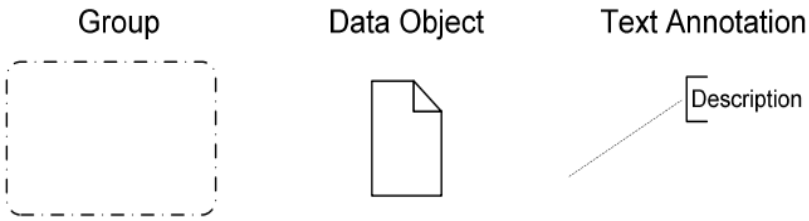


**Figura 2.9.** Notación BPMN – Artefactos

## 2.6 Ejemplo de un Proceso de negocio

Para observar con más detalle un proceso de negocio, se presenta un proceso de crédito el cual consta básicamente de un Registro de la solicitud, donde el cliente manifiesta su interés de adquirir un crédito, en esta etapa se incluye la presentación de la solicitud y documentación requerida a la entidad, luego se realiza una verificación de la información, posteriormente la etapa donde se realiza el Análisis o Estudio de la solicitud de crédito y por ultimo encontramos las actividades referentes a hacer efectivo el crédito o informar el rechazo al cliente.

A continuación se presenta un ejemplo de procesos de negocio (figura 2.10)

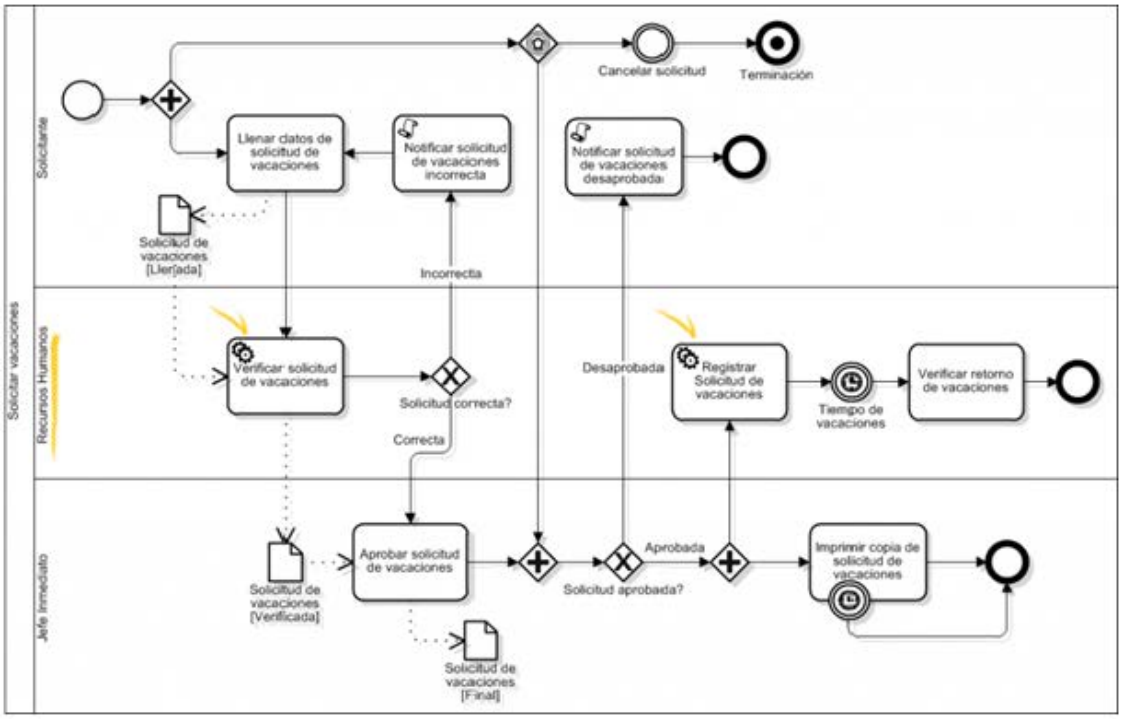


**Figura 2.10.** Ejemplo proceso de Negocio

Para el caso que se está modelando se considera que existe un sistema para el control de asistencia de los trabajadores, este sistema permite registrar todas las solicitudes de vacaciones, de igual forma debe ser capaz de chequear que un trabajador en específico tiene determinada cantidad de días acumulados de vacaciones, esto lo realiza basándose en su asistencia laboral y en los días de vacaciones que ha tomados. Ésta puede ser una funcionalidad de un servicio a invocar desde el proceso y sustituir la actividad humana

Verificar Solicitud de Vacaciones, donde el encargado de Recursos Humanos debe revisar desde un documento Excel la cantidad de días disponibles y de acuerdo a su revisión aceptar o rechazar la solicitud del empleado. Esta actividad se puede hacer de manera automática, lo que conlleva a que se realice de manera inmediata y sin la intervención humana.

# 3. MDA - MODEL DRIVEN ARCHITECTURE

## 3.1 Object Management Group (OMG)

El Object Management Group [OMG] es un consorcio internacional abierto y sin ánimo de lucro de la industria de la informática creada en 1989. En él se encuentran incluidas una gran cantidad de organizaciones dedicadas al desarrollo de software, enfocadas en diversos mercados de negocio.

Aproximadamente alrededor del año 2001 OMG adoptó un nuevo marco de trabajo llamado Arquitectura Dirigida Por Modelos (MDA).

Una ventaja que sin duda ofrece la OMG a la Arquitectura dirigida por modelos es ser tal vez el mejor camino para conservar la inversión y mantener estable las finanzas invertidas en el desarrollo de cualquier tipo de sistemas de información o software en general.

De igual forma con OMG encontramos diversos estándares de integración para un amplio rango de tecnologías: tiempo real, sistemas empotrados, entre otros y una amplia variedad de industrias diversas como son: finanzas, administración pública, etc.

Gracias a los estándares de modelado de la OMG podemos contar con un desarrollo y un mantenimiento de software mucho más potente. De igual forma otra ventaja es que la documentación y los estándares definidos por OMG se encuentran disponibles al público en general.

## 3.2 Model-Driven Architecture (MDA)

La arquitectura dirigida por modelos (MDA) [Mellor 2004] es una aproximación para el desarrollo de software definido por el OMG.

MDA define una aproximación para especificar sistemas de tecnologías de información que separa claramente lo que es la especificación de la funcionalidad del sistema, de la especificación de su implementación sobre una determinada plataforma.

En la figura 3.1 se observa el ciclo de vida del desarrollo de software MDA según [Kleppe 2005].

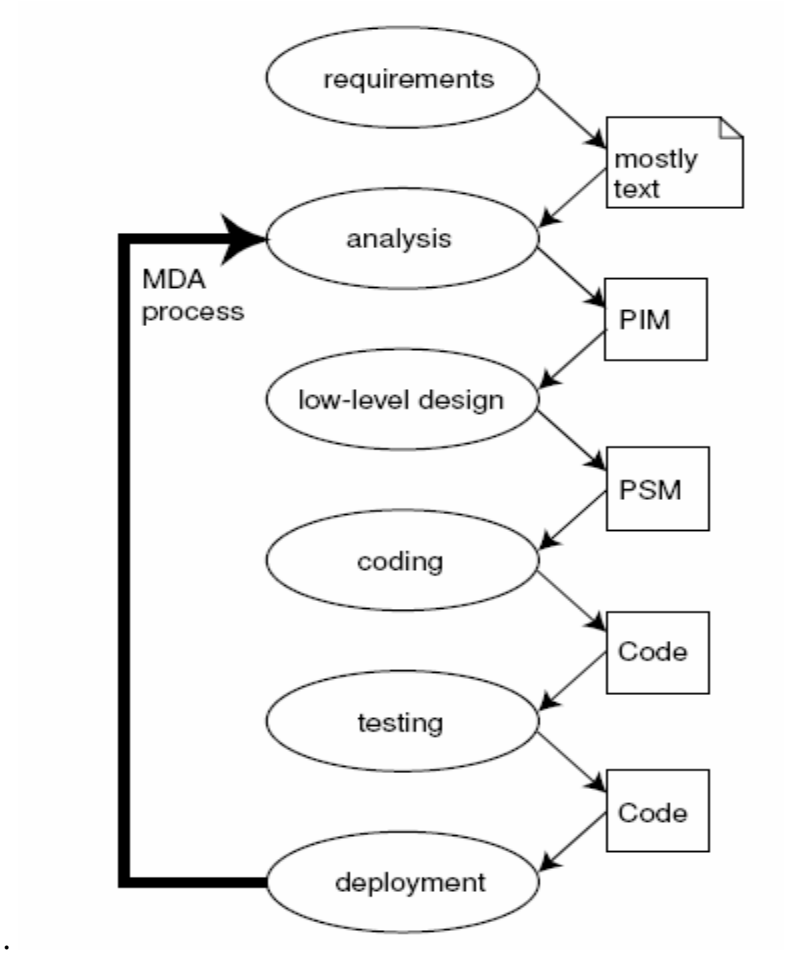


**Figura 3.1.** Ciclo de vida del desarrollo de software MDA

De acuerdo a [Klasse 2004], el proceso de MDA gira alrededor de tres pasos principales:

**1)** Primero se construye un modelo con el nivel más alto de abstracción, este modelo es independiente de cualquier tecnología de implementación y se conoce con el nombre de Modelo Independiente de la Plataforma (PIM).

**2)** A continuación este PIM es transformado en uno o varios Modelos Específicos de la plataforma (PSM).

**3)** Finalmente el PSM es transformado en el código. Teniendo en cuenta que el PSM ya lleva todo lo necesario los detalles que se requieren para la codificación, en este paso no suelen ser complejos.

En la figura 3.2 se presentan los pasos MDA según [Klasse 2004].

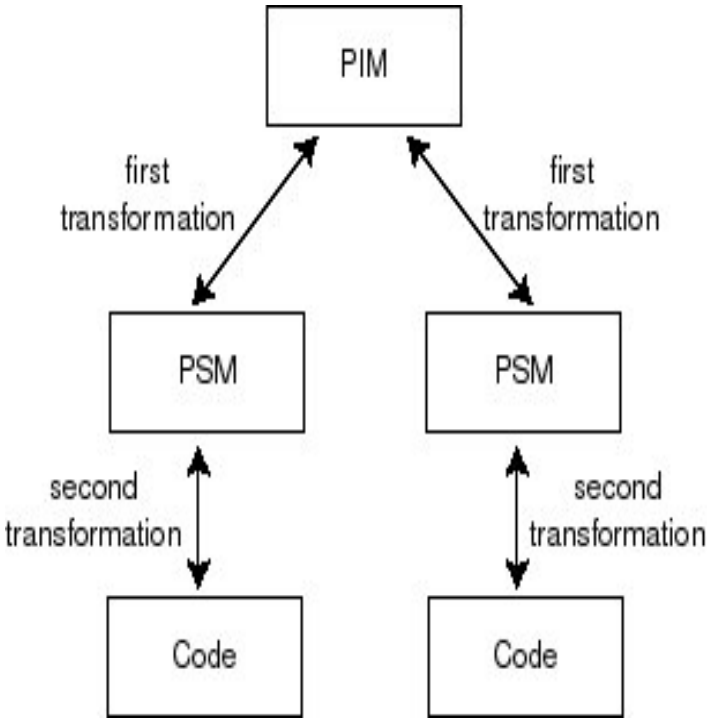


**Figura 3.2.** Pasos de MDA [Klasse, 2004]

A continuación se describen brevemente cada uno de los pasos de la figura:

**Paso 1) Modelo Independiente de la Plataforma (PIM**): El Modelo independiente de la Plataforma es un modelo con un alto nivel de abstracción que es independiente de cualquier tecnología de implementación. Un lenguaje de modelado capaz de generar todos los artefactos requeridos tal como el Lenguaje de Modelado Unificado (UML) es requerido en este nivel.

Un modelo de MDA por lo general tiene múltiples niveles de PIMS. Todos estos PIMS incluyen los aspectos independientes de la plataforma, excepto la base PIM, ya que esta expresa sólo la funcionalidad del negocio y comportamiento. Los diagramas producidos en esta etapa tal como las clases UML y los diagramas de objetos, incorporan la estructura; los diagramas de secuencia y de actividad incorporan el comportamiento.

**Paso 2) Modelo Específico de la Plataforma (PSM**): Cuando el PIM es completado, este se almacena en el Meta Object Facility (MOF) y esto sirve como entrada al paso de mapeo que producirá un Modelo Específico de la Plataforma (PSM). Un Modelo Específico de la plataforma es definido para especificar el sistema en términos de la implementación en una de las plataformas tecnológicas disponibles.

Para producir el PSMS, una o varias plataformas objetivo deben ser seleccionadas para cada módulo del software. En contraste con el primer paso, aquí en vez de expertos en una tecnología, se requieren instrumentos automatizados.

Existen dos maneras de mapear PIM a PSM, la primera de ellas es un mapeo de tipo de modelo que está basado sobre los tipos de elementos del modelo. La segunda es un mapeo instancia del modelo, que especifica cómo hacer un mapeo de los elementos del modelo específico usando marcas.

A continuación en la figura 3.3 se observa el esquema de mapeo de PIM a PSM.

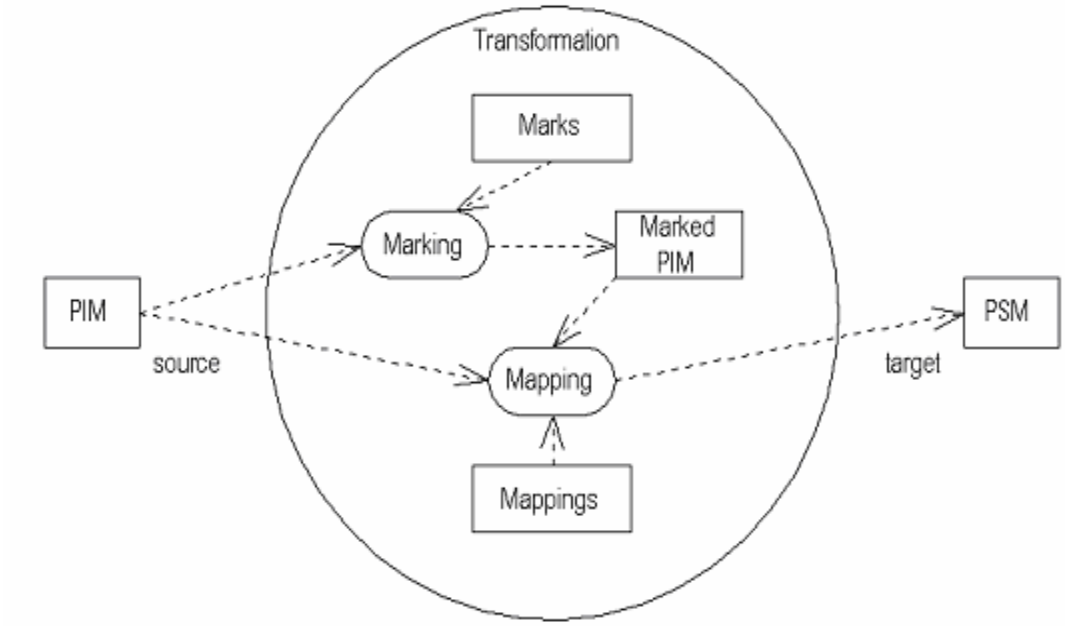


**Figura 3.3.** Mapeando de PIM a PSM (MDA)

Cuando un PIM es transformado en muchos PSMS, pueden existir algunas relaciones entre estas transformaciones. En MDA estas relaciones se denominan puentes y este detalle se puede apreciar en la siguiente figura 3.4.

Estos puentes pueden ser producidos por las herramientas. No importa cuántos PSMS sean, siempre habría un puente entre cada uno de ellos. De igual forma algunos puentes pueden producirse en el nivel del código.

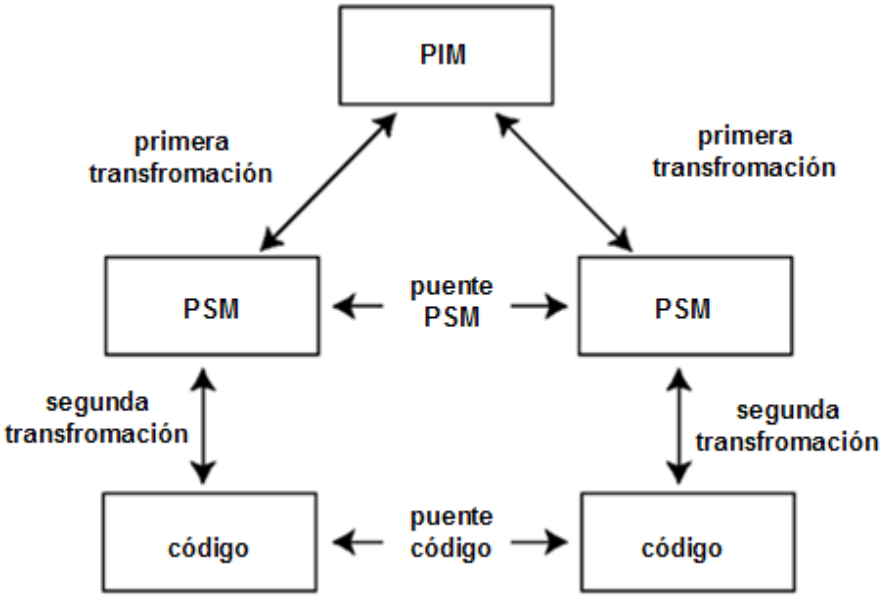


**Figura 3.4.** Pasos MDA detallado

**Paso 3) Generación de Código**: Este paso toma un PSMS de entrada y produce la implementación de este PSMS para la plataforma particular usando la herramienta de transformación especificada.

Los procesos y roles que se consideran dentro de MDA se presentan en la figura 3.5.

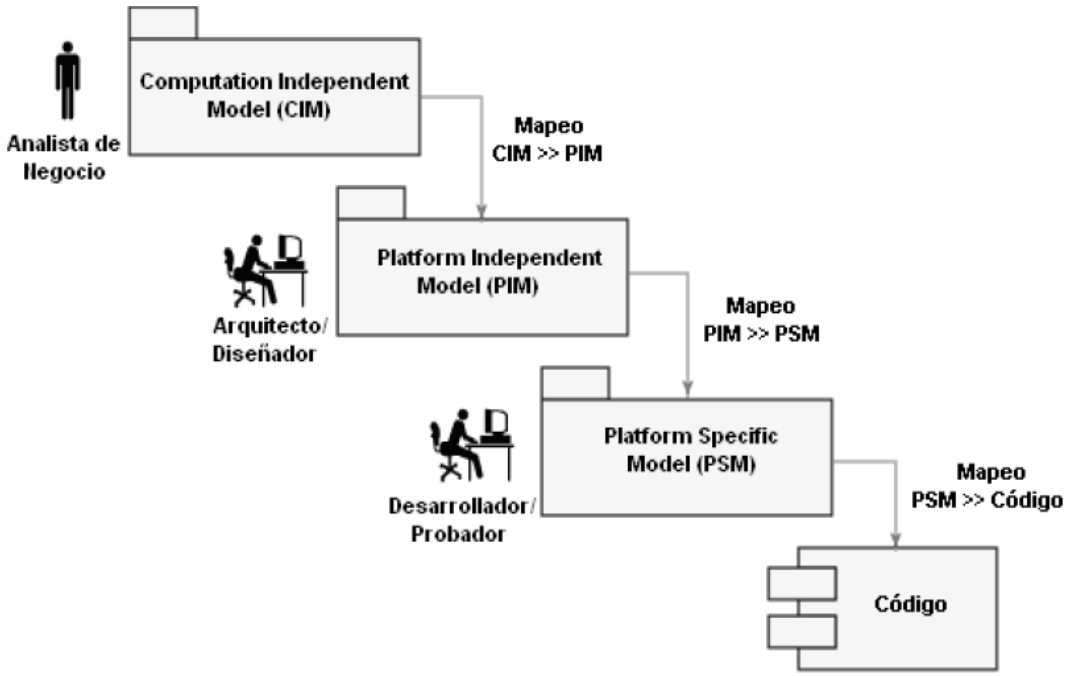


**Figura 3.5.** Procesos y Roles en MDA

En concreto, la mayor aportación de MDA y su mayor beneficio, es la transformación de modelos PIM a modelos PSM.

Uno de los aspectos importantes es que la portabilidad en MDA se consigue gracias a su propio planteamiento. Siempre se va a partir del mismo PIM y en el caso de tener que migrar el sistema a otra tecnología, sólo será necesario generar el PSM apropiado para la nueva plataforma.

### 3.2.1 ¿Por qué utilizar Modelos?

Un modelo es una abstracción de un sistema o una parte de ella. Dependiendo del tipo de modelo este puede proporcionar una vista simple del sistema o una más detallada.
En la ingeniería del software los modelos tienen una larga e importante historia, teniendo en cuenta aspectos como que los diseñadores suelen utilizar modelos para la planificación de sus proyectos o sencillamente hacen uso de eso ellos para explicarlos a otras personas.
Hoy en día una gran parte de los modelos utilizados son los UML, considerando que una gran parte de los desarrolladores utiliza dichos modelos UML en paralelo dentro del diseño de los sistemas software.

### 3.2.2 Principales ventajas que ofrece MDA

a) **Portabilidad:** Incrementando la reutilización de la aplicación y reduciendo el costo y la complejidad en el desarrollo de las aplicaciones y su gestión, ahora y en el futuro.
b) **Interoperabilidad:** Para esto se utilizan métodos rigurosos, buscando garantizar que basándose en los estándares para tecnologías de implementación múltiple todas las funciones del negocio sean implementadas de forma similar.
c) **Independencia de la Plataforma:** Reduciendo considerablemente el tiempo, costo y la complejidad asociada con el rediseño de la aplicación para diferentes plataformas.
d) **Productividad:** Permitiendo a los desarrolladores, diseñadores y administradores de sistemas utilizar lenguajes y conceptos con los que se sientan cómodos. Logrando así una buena comunicación e integración entre los equipos.

Las ventajas mencionadas anteriormente, llevan a reducir los costos durante el ciclo de vida de la aplicación, reduciendo el tiempo de desarrollo para nuevas aplicaciones y mejorando la calidad de las mismas.

### 3.3 Metamodelo MDA

Un metamodelo es un modelo que define el lenguaje para expresar un modelo, dicho de otra forma, también se puede decir que un metamodelo es un modelo de un modelo.

Otra forma de decirlo es que un metamodelo es una clase especial de modelo que especifica la sintaxis abstracta de un lenguaje de modelado. Esto puede ser entendido como la representación de la clase de todos los modelos expresados en este lenguaje. Los Metamodelos en el contexto de MDA son expresados usando MOF.
En la figura 3.6 se presentan los modelos, lenguajes, metamodelos y metalenguajes que se consideran en la arquitectura de metamodelado de cuatro capas MDA.

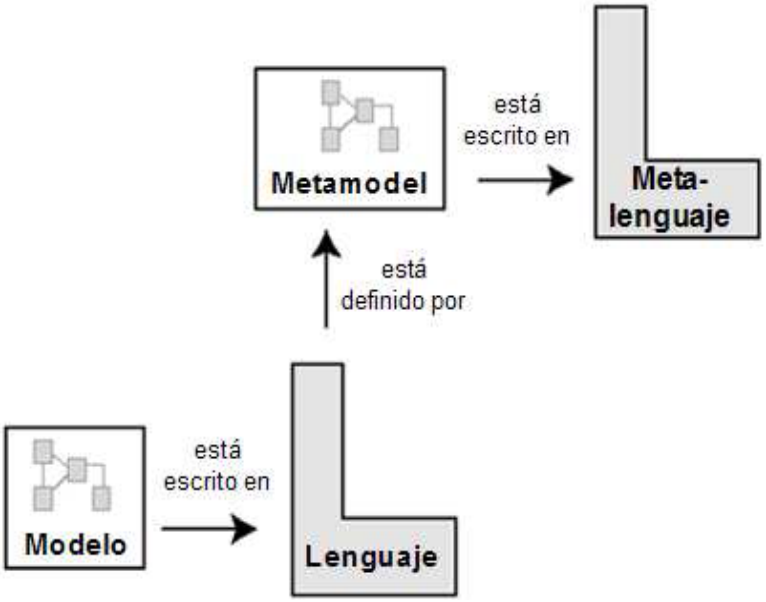


**Figura 3.6.** Modelos, lenguajes, metamodelos y metalenguajes.

### 3.3.1 Metamodelo de Cuatro capas

**Capa M0:** Las instancias. Es la capa en donde se ejecuta el sistema, donde están las instancias reales que se han creado durante la ejecución. En otras palabras es el sistema real.

**Capa M1:** El modelo del sistema. A este nivel se define el modelo del sistema o la aplicación propiamente dicha

**Capa M2:** Metamodelo. Este nivel contiene los elementos del lenguaje de modelado o metamodelo. El lenguaje tendrá elementos como Clases y Atributos que permitirán definir el modelo del sistema de la capa M1.

**Capa M3:** Meta-metamodelo. Siguiendo la misma línea que los anteriores, podemos definir los elementos existentes en la capa M2, mediante instancias de elementos existentes en esta capa.
A continuación se presenta un esquema representativo del Metamodelo de cuatro capas (ver figura 3.7).

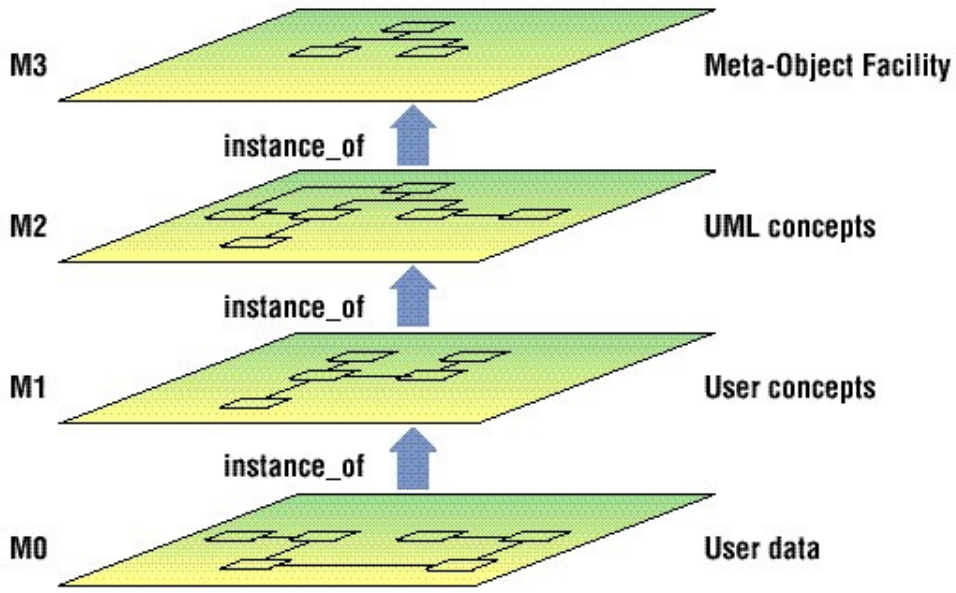


**Figura 3.7.** Metamodelado de cuatro capas

### 3.3.2 Transformaciones de Modelos

El concepto de transformación juega un papel muy significativo en el proceso de MDA. Fundamentalmente, la transformación es el proceso automático de convertir un artefacto en otro artefacto deseado, usando una herramienta, teniendo una regla que describe como transformarlo.

Las herramientas de transformación toman un artefacto como la entrada y transforman esto en un artefacto deseado según las definiciones de transformación y reglas de transformación y el proceso completo de conversión es conocido como transformación.

La figura 3.8 muestra el proceso de Transformación MDA.

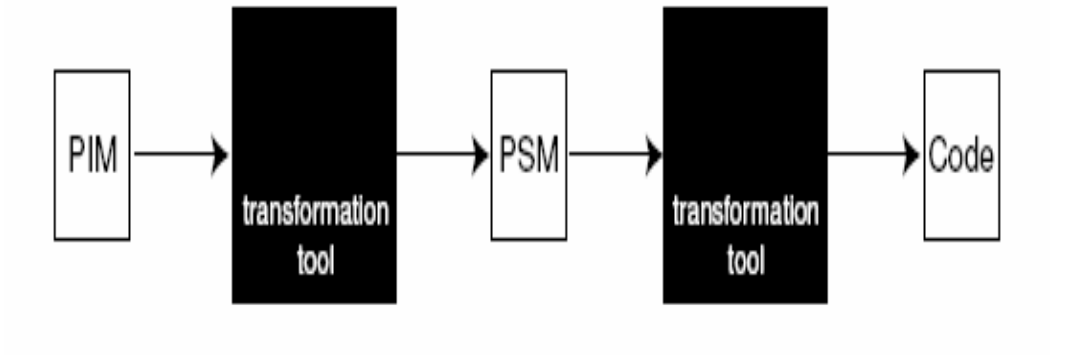


**Figura 3.8.** El Proceso de Transformación MDA

### 3.3.3 Conceptos importantes en las Transformaciones de Modelos

**- Definición de transformación:** En la herramienta de transformación, se requieren definiciones de transformación para el proceso de transformación de modelos.

Una definición de transformación, es de hecho, un conjunto de reglas de transformación o juego de transformación. Las herramientas de transformación usan definiciones de transformación para generar las transformaciones.

**- Reglas de Transformación:** Una transformación consiste de una colección de reglas de transformación, que son las directrices que definen como usar un modelo para generar otro.

**- Herramienta de Transformación:** La herramienta de transformación juega un papel vital en el proceso completo de MDA. Una herramienta de transformación realiza una

transformación para un modelo fuente específico de acuerdo a la definición de la transformación.

En un enfoque de transformación de modelos se observa como este se basa en el mapeo de tipos independientes de la plataforma sobre tipos específicos de la plataforma.

En la figura 3.9 que se presenta a continuación se observa la transformación de modelos.

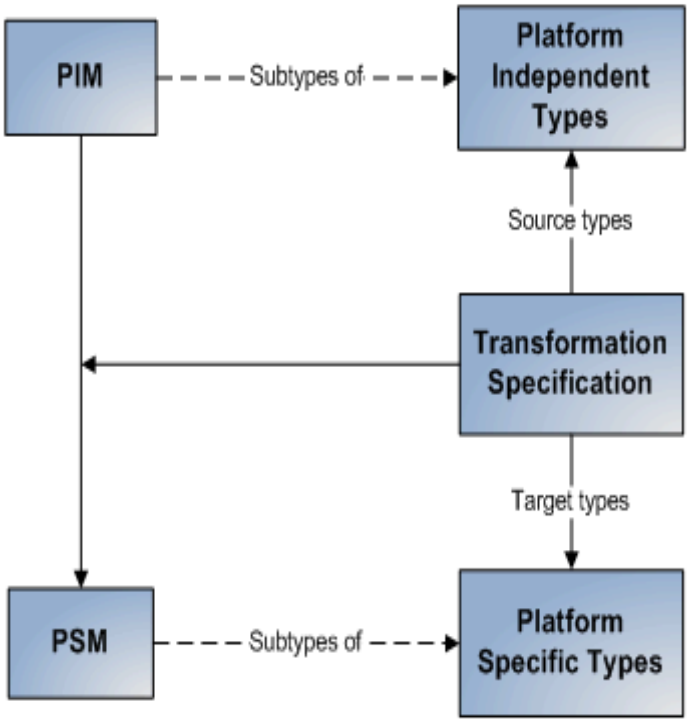


**Figura 3.9.** Transformación del Modelo.

Así mismo debemos tener muy presente que la especificación de la transformación define el mapeo entre los metamodelos. Como podemos observar en la figura 3.10 que se presenta a continuación.

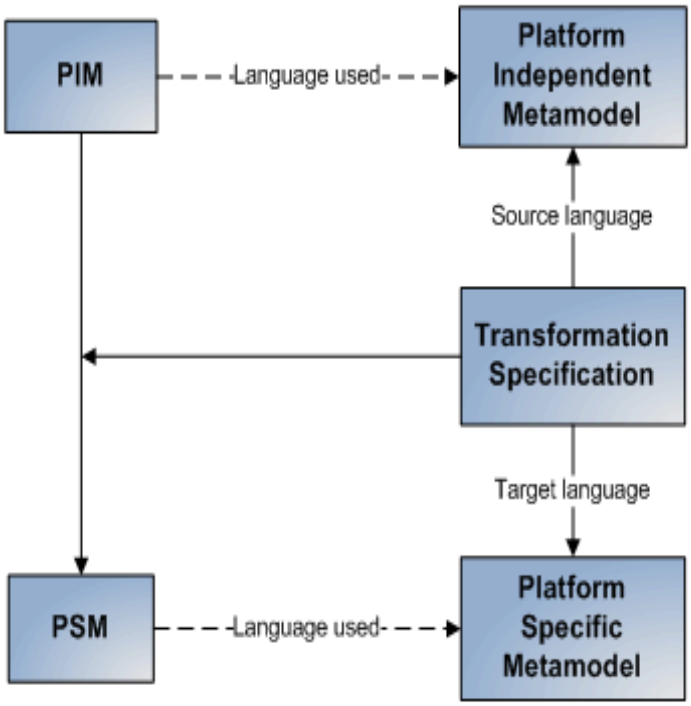


**Figura 3.10.** Transformación del Metamodelo.

### 3.4 Transformaciones MDA

Las transformaciones de modelos son realizadas de muchas formas, a continuación se describen brevemente cuatro aproximaciones de transformación, la transformación manual, transformando un PIM que está preparado usando un archivo, la transformación usando patrones y marcas y la transformación automática.

#### 3.4.1 Transformación manual

Este proceso de transformación manual no es muy diferente de como se ha realizado el de diseño de software durante todos estos años. La aproximación MDA adiciona valor en dos sentidos:

- La distinción explícita entre un modelo independiente de la plataforma y el modelo específico de la plataforma transformado.
- El registro de la transformación.

#### 3.4.2 Usando un Archivo

Un PIM puede estar preparado usando un archivo UML independiente de la plataforma. Este modelo puede ser transformado en un PSM usando un segundo un segundo archivo UML específico de la plataforma.

La transformación puede implicar la marca del PIM utilizando marcas provistas por el archivo específico de la plataforma.

### 3.4.3 Usando Patrones y Marcas

Los patrones pueden ser usados en la especificación del mapeo. El mapeo incluye un patrón y marcas correspondientes a algunos elementos del patrón.

En las transformaciones instancia las marcas especificadas son usadas para preparar un PIM marcado. Los elementos marcados del PIM son transformados de acuerdo a los patrones para producir el PSM.

En las reglas de transformación de tipo del modelo se especifica que todos los elementos en el PIM que juegan con un patrón en particular serán transformados en instancias de cualquier otro patrón en el PSM.

### 3.4.4 Automático

Hay contextos en los cuales un PIM puede proporcionar toda la información necesaria para la implementación, por tanto no existe ninguna necesidad de adicionar marcas o usar datos de archivos adicionales.

En este contexto, el desarrollador no necesita ver un PSM, tampoco es necesario adicionar información en el PIM. La herramienta en este caso interpreta directamente el modelo o transforma directamente el modelo a código del programa.

Algunas de las preocupaciones que se ha planteado con el enfoque MDA la OMG son las siguientes:

a) Estándares incompletos: El enfoque de MDA se basa en una serie de normas técnicas, algunas de las cuales aún no se han especificado (por ejemplo, un lenguaje de acción semántica para xtUML), o aún no se han aplicado de una manera estándar (por ejemplo, un motor de transformaciones QVT o un PIM con un entorno de ejecución virtual).
b) Dependencia de un proveedor: A pesar de que MDA fue concebido como un enfoque para lograr independencia de la plataforma técnica, los actuales proveedores de MDA han sido renuentes a diseñar su conjunto de herramientas MDA para brindar la interoperabilidad.
**c)** Idealista: MDA se concibe como un enfoque de ingeniería avanzada en la que

los modelos que se incorporan son transformados en artefactos de implementación tal como código ejecutable o esquemas de bases de datos. Sin embargo esta visión resulta un poco idealista en algunos despliegues del mundo real.

# 4. Marco Tecnológico

## 4.1 Eclipse

La plataforma de desarrollo que se ha utilizado para implementar el proyecto, es Eclipse. Este framework nos proporciona un completo entorno de trabajo con el que podemos desarrollar fácilmente nuestros proyectos, esta plataforma cuenta con una buena documentación.

El entorno de desarrollo Eclipse (también sus plugins) está desarrollado completamente en java, pero se base en la librería de widgets SWT que es equivalente a Swing pero se aprovechan los widgets nativos del sistema sobre el que se ejecuta, esto permite que la ejecución de interfaces de usuario sea mucho más rápida que si se utiliza Swing.

Las características principales del IDE son las siguientes:

- Permite trabajar con varios proyectos a la vez.
- El editor de código tiene colores para la sintaxis y también lo que es conocido como "code highlighting".
- Los errores de compilación a parte de darte una descripción de error te indica donde se ha producido el error.
- Posee un formateador de código.
- Permite encontrar código duplicado.
- Tiene lo que se conoce como "code folding".
- Permite personalizar el entorno.
- El editor permite buscar y reemplazar palabras.
- Integración con aplicaciones controladoras de versión como por ejemplo CVS.
- En la compilación se permite una compilación incremental.
- Permite el uso de herramientas externas como es ANT ó JUNIT.
- Permite codificar código java, c/c++, xml, jsp, entre otros.

## 4.2 Proyectos que forman eclipse

El trabajo de desarrollo en Eclipse se divide en tres proyectos principales:

### 4.2.1 El proyecto Eclipse

La plataforma Eclipse consiste en un Entorno de Desarrollo Integrado (IDE, Integrated Development Environment) abierto y extensible. Un IDE es un programa compuesto por un conjunto de herramientas útiles para un desarrollador de software. Como elementos básicos, un IDE cuenta con en un editor de código, un compilador/intérprete y un depurador. El framework Eclipse esta implementado utilizando Java pero es usado para implementar herramientas de desarrollo para otros lenguajes (por ejemplo, C++ y XML, entre otros).

El proyecto Eclipse se encuentra dividido en tres subproyectos:

- La Plataforma (Platform) es el componente central de Eclipse, y es a menudo considerado como el propio Eclipse. Se utiliza para definir frameworks y los servicios requeridos para soportar la conexión e integración de las herramientas.

- Herramientas de Desarrollo Java (Java Development Tools, JDT) las cuales proporcionan un completo entorno de desarrollo Java. Pueden ser utilizadas para desarrollar programas Java para Eclipse o para otras plataformas.

- Entorno de Desarrollo de Plug-ins (Plug-in Development Environment, PDE) proporciona vistas y editores para facilitar la creación de plug-ins para Eclipse. Además proporciona soporte para actividades propias del desarrollo de un plug-in, como el registro de extensiones.

A continuación se presenta la figura 4.1 con el fin de tener claro la Arquitectura Eclipse.

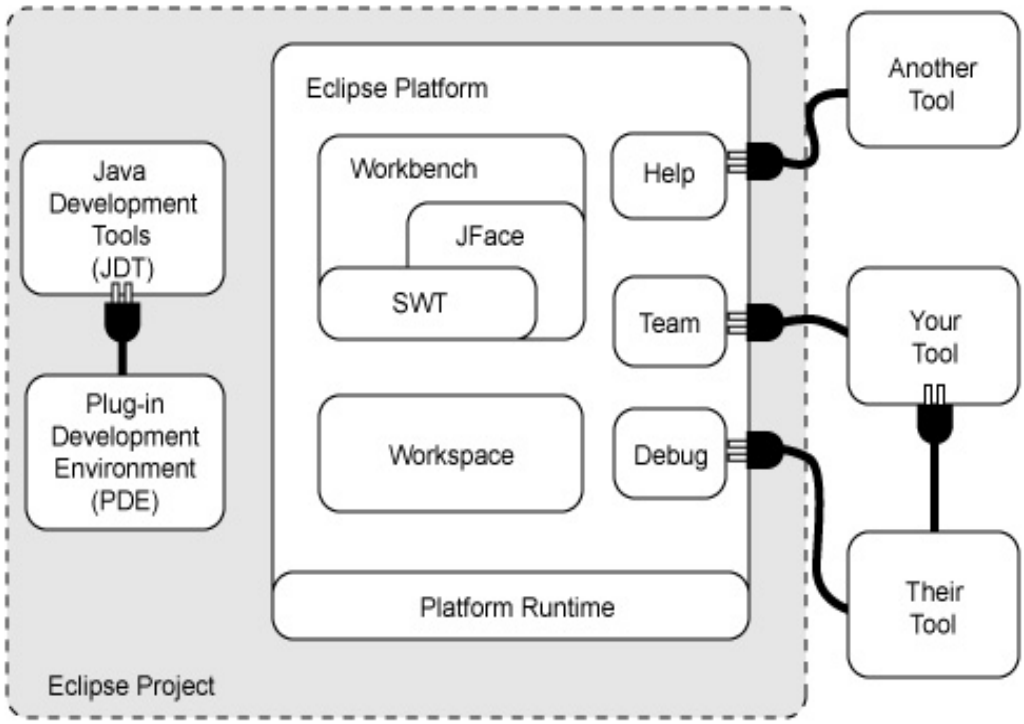


**Figura 4.1** Arquitectura Eclipse

### 4.2.2 El proyecto de Herramientas

Este proyecto define y coordina la integración de diferentes conjuntos o categorías de herramientas basadas en la plataforma Eclipse. El subproyecto de Herramientas de Desarrollo C/C++ (C/C++ Development Tools, CDT), por ejemplo, engloba un conjunto de herramientas que definen un IDE C++.

### 4.2.3 El proyecto de Tecnología

Este proporciona una oportunidad a los investigadores y educadores para formar parte de la continua evolución de la plataforma Eclipse.

## 4.3 Componentes de la plataforma Eclipse

La Plataforma Eclipse es un framework para construir entornos de desarrollo integrados (IDEs). Algunos de sus componentes principales son los siguientes:

### 4.3.1 La arquitectura de plug-ins

La unidad funcional básica, o componente, en Eclipse se conoce como plug-in. La misma Plataforma Eclipse, y las herramientas que la extienden, están compuestas por plug-ins.

Al igual que en cualquier otro software, eclipse permite la instalación de plug-ins destinados a mejorar las funcionalidades del propio IDE y a extenderse en cada vez mas tecnologías.

Desde una perspectiva de paquetes, un plug-in incluye todo lo que se necesita para que funcione un componente como código Java, imágenes, texto traducido, etc. También incluye un archivo de manifiesto, llamado plugin.xml, que declara las interconexiones con otros plug-ins. En este archivo se declaran, entre otras cosas, lo siguiente:

- Required plug-ins sus dependencias → con otros plug-ins.
- Exported plug-ins → la visibilidad de sus clases públicas con otros plug-ins.
- Extension points → declaración de la funcionalidad que se pone a la disposición de otros plug-ins.
- Extensions → implementación de puntos de extensión de otros plug-ins.

En el arranque, la Plataforma Eclipse (específicamente Platform Runtime) encuentra todos los plug-ins disponibles y asocia las extensiones con sus correspondientes puntos de extensión.

### 4.3.2 Workspace (Recursos espacio de trabajo)

El área de trabajo es la ubicación física (ruta de archivo) en que está trabajando. Sus proyectos, archivos de código fuente, imágenes y otros objetos pueden ser almacenados y guardados en el espacio de trabajo, pero también puede referirse a los recursos externos. Un proyecto es un tipo especial de recurso que se asocia a una carpeta del usuario en el sistema de ficheros. Las subcarpetas del proyecto son las mismas que las de la carpeta física, pero los proyectos son carpetas de alto nivel en un contenedor virtual de proyectos, llamado espacio de trabajo.

### 4.3.3 El Framework UI

El framework UI (User Interface) de Eclipse consiste en dos conjuntos de herramientas de propósito general, SWT y JFace, y el Escritorio de Eclipse (workbench UI).
SWT (Standard Widget Toolkit) es un conjunto de librerías gráficas independientes del sistema operativo utilizado.
JFace es un conjunto de herramientas de alto nivel, implementado usando SWT. Proporciona clases para soportar tareas comunes relativas a interfaces de usuario, como manejo de registros de imágenes y fuentes, diálogos, asistentes, monitores de progreso, etc.
Una parte importante de JFace son las clases utilizadas como visores para listas, árboles y tablas, que proporcionan un mayor nivel de conexión con los datos que SWT (por ejemplo, mecanismos de población a partir de un modelo de datos y sincronización con este).
Por último, el Escritorio de Eclipse (workbench) es la ventana principal que el usuario ve cuando arranca Eclipse. Esta implementado mediante SWT y JFace. Como interfaz principal de usuario, se considera a menudo que es la propia Plataforma.

### 4.3.4 Soporte para el trabajo en grupo

La Plataforma Eclipse permite asignar una versión a un proyecto en el workspace y gestionar sus versiones mediante un repositorio de trabajo en grupo.

## 4.4. Eclipse Modeling Framework (EMF)

EMF es un entorno de modelado y de igual forma una herramienta de generación de código para la plataforma Eclipse. Su objetivo es ayudar en la construcción de herramientas y otras aplicaciones basadas en modelos estructurados.
EMF ayuda además a generar código Java a partir de estos modelos de una manera fácil, correcta, personalizable y eficiente. De igual forma EMF utiliza XMI como una manera canónica de definir y persistir los modelos. Además incorpora un editor gráfico para poder definir de esta forma los modelos.

Cuando se ha especificado un MetaModelo EMF, el generador EMF puede crear una serie de clases Java que permiten crear instancias del MetaModelo y manipular sus elementos.

Una vez generado el código se pueden editar las clases generadas para añadir nuevos métodos y variables. Además, se proporcionan mecanismos para la notificación de cambios en el modelo.

En la figura 4.2 se muestra la Arquitectura Eclipse Modeling Framework (EMF).

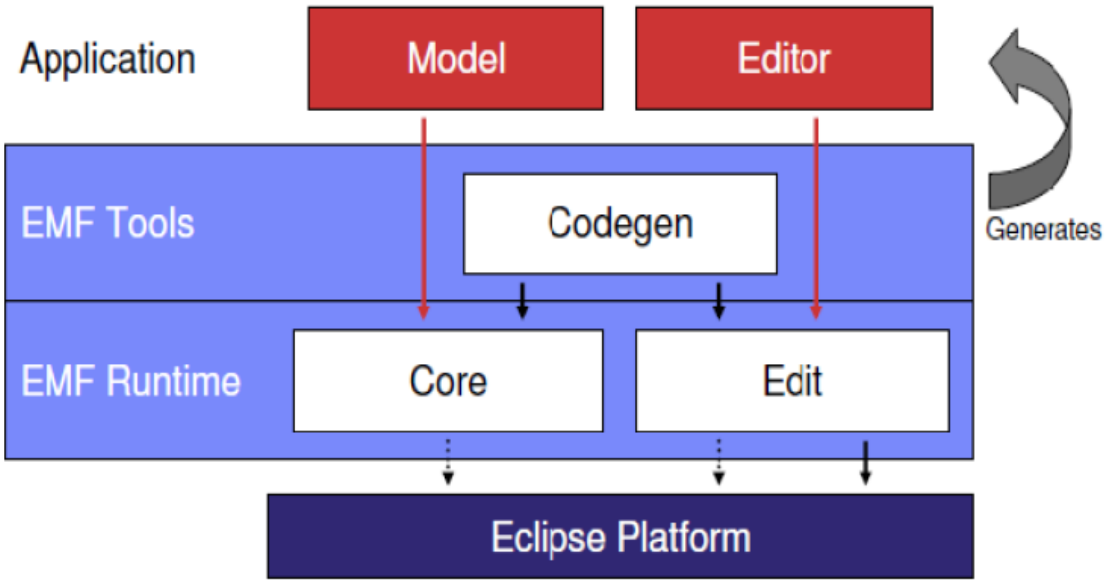


**Figura 4.2** Arquitectura EMF

### 4.4.1 Metamodelo en EMF (Ecore)

Ecore es un vocabulario diseñado para permitir la definición de cualquier tipo de MetaModelos. Para ello proporciona elementos útiles para describir conceptos y relaciones entre ellos. En definitiva, Ecore es un subconjunto de MOF, el cual está basado en el diagrama de clases de UML.

En la figura 4.3 se muestra la Jerarquía de clases del metamodelo ecore.

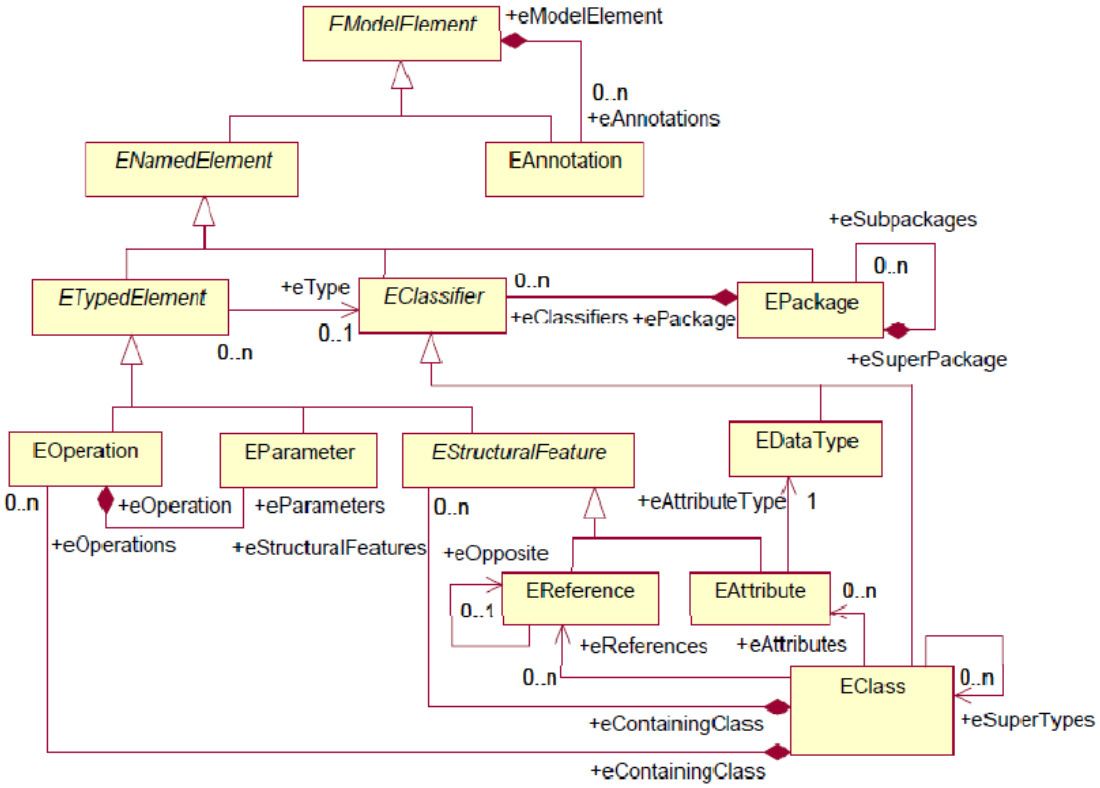


**Figura 4.3** Jerarquía de Clases del Metamodelo Ecore.

El elemento principal de un Metamodelo de Ecore es Eclass el cual modela el concepto de una clase y a su vez está compuesta por un conjunto de atributos y referencias, así como por un número de super clases (el símil con UML sigue siendo aplicable).

A continuación se comentan los elementos más importantes que aparecen en la figura anterior:

● **EClassifier** Es un tipo abstracto que agrupa a todos los elementos que describen conceptos.

● **EAttribute**. Tipo que permite definir los atributos de una clase. Estos tienen nombre y tipo. Como especialización de ETypedElement, EAttribute hereda un conjunto de propiedades como cardinalidad (lowerBound, upperBound), si es un atributo requerido o no, si es derivado, etc.

● **EDataType** se utiliza para representar el tipo de un atributo. Un tipo de datos puede ser un tipo básico como int o float o un objeto.

● **EPackage** Agrupa un conjunto de clases en forma de modulo, de forma similar a un paquete en UML.

Tiene como atributos más importantes el nombre, el prefijo y la URI. La URI es un identificador único gracias al cual el paquete puede ser identificado unívocamente.

### 4.4.2 Como crear el Ecore

Un modelo Ecore se puede crear de varias formas, unas de ellas son:

1) Importarlo de un modelo en uno de los siguientes formatos:

a. Diagrama de Rational case.
b. XML schema.
c. Modelo UML.
d. Java anotado.

2) Utilizando el editor gráfico de modelos Ecore proporcionado por GMF.

En la figura 4.4 se muestra el esquema de importación de modelos EMF.

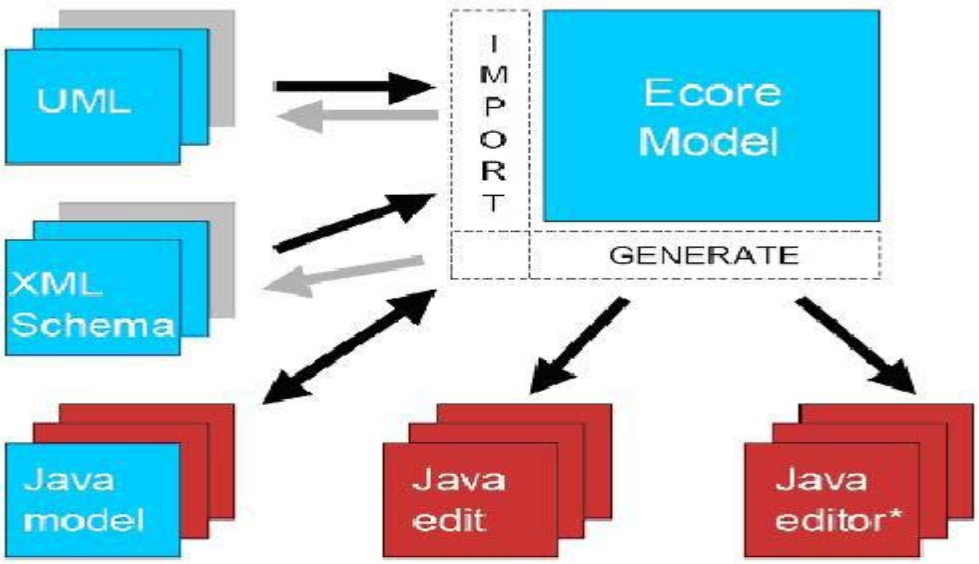


**Figura 4.4.** EMF: Importación de modelos

### 4.4.3 Generando código

Si tenemos un modelo Ecore, podemos generar con el Eclipse Modeling Framework de una forma simple y eficiente lo siguiente:

- Clases que implementan el MetaModelo.
- Clases para la edición independientes de la interfaz de usuario.

- Una vista de edición de modelos integradas en el IDE eclipse.
- Esqueletos para las clases de test de JUnit.

- Además de lo anterior también se generan los elementos necesarios para encapsular cada uno de los puntos anteriores en plug-ins de eclipse:
- Manifiestos.
- Archivos properties.
- Iconos.
- Clases relacionadas con los plug-in.

### 4.5 OpenArchitecture Ware (OAW)

El oAW es un motor de flujos de trabajo, que proporciona un lenguaje de configuración sencillo basado en XML, con el cual todos los pasos del generador de flujo de trabajo pueden ser descritos, un generador de flujos de trabajo consiste de un número de componentes del flujo que son ejecutados de forma secuencial en una simple JVM, oAW es una solución de código abierto para el desarrollo de software dirigido por modelos. La Arquitectura OAW se visualiza en la figura 4.5.

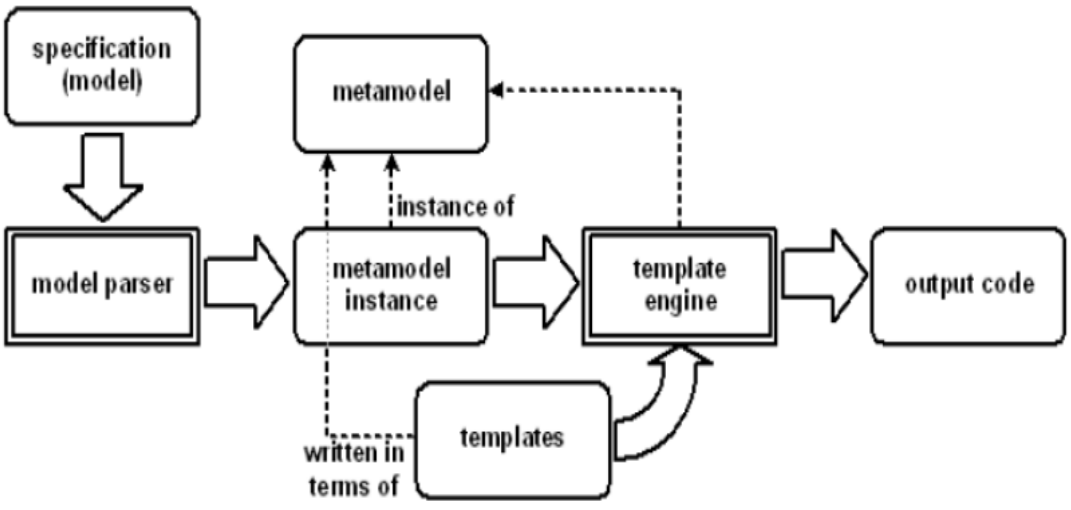


**Figura 4.5** Arquitectura OAW

## 4.6 Xpand2

El framework openArchitectureWare contiene un lenguaje especial llamado Xpand que es usado en plantillas para controlar la generación de salida.

### 4.6.1 Templates

Las plantillas son almacenadas en archivos con la extensión .xpt. Los archivos de plantilla deben residir en el classpath de Java del proceso generador.

Casi todos los carácteres usados en la sintaxis estándar son la parte de ASCII y por lo tanto deberían estar disponibles en cualquier codificación**.** Los nombres de propiedades, plantillas, namespaces etc. sólo deben contener letras y números.

### 4.6.2 Estructura general de las Plantillas

Un archivo de plantillas consta de cualquier número de declaraciones de IMPORTACIÓN, seguidas de cualquier número de declaraciones de EXTENSIÓN, seguidas de uno o varios bloques DEFINE (llamados definiciones). A continuación se presenta la estructura general de una plantilla:

```
«IMPORT meta::model»
«EXTENSION my::ExtensionFile»

«DEFINE javaClass FOR Entity»
«FILE fileName()»
package «javaPackage()»;

public class «name» {
// implementation
}
«ENDFILE»
«ENDDEFINE»
```

**Tabla 4.1** Estructura general plantilla Xpand

### 4.6.3 Principales declaraciones

#### 4.6.3.1 IMPORT

Se utiliza para importar el modelo. Si la plantilla contiene tal declaración, se pueden usar los nombres cualificados de todos los tipos y archivos de plantilla contenidos en su namespace. Esto es similar a una declaración de importación en Java.

#### 4.6.3.2 EXTENSION

Los metamodelos son típicamente descritos de un modo estructural (gráfico, o jerárquico, etc.). Un defecto de esto es que es difícil de especificar el comportamiento adicional (operaciones, propiedades derivadas, etc.). También es una idea buena ya que de esta forma no se contamina el metamodelo con información específica de la plataforma. Las extensiones proporcionan un modo flexible y conveniente de definir los rasgos adicionales de metaclases.

#### 4.6.3.3 DEFINE

El concepto central de Xpand es el bloque Define, también llamado una plantilla. Esto es la unidad identificable más pequeña en un archivo de plantilla. La etiqueta consiste en un nombre, una lista de parámetro opcional separado por coma, así como el nombre de la clase metamodel para la cual la plantilla es definida.

#### 4.6.3.4 FILE

La declaración de File remite la salida generada de sus declaraciones de cuerpo al objetivo especificado. El objetivo es un archivo en el sistema de archivos cuyo nombre es especificado por la expresión (relativo al directorio objetivo especificado para ejecutar el generador).

#### 4.6.3.5 EXPAND

La declaración Expand amplia otro bloque define y amplía el otro bloque (en un contexto variable separado), inserta su salida en la posición correspondiente y continua con la próxima declaración.

#### 4.6.3.6 FOREACH

Esta declaración expande el cuerpo del bloque Foreach para cada elemento de la expresión que resulta de la expresión.

#### 4.6.3.7 IF

La declaración apoya la extensión condicional. Permite cualquier número de declaraciones ELSEIF.

### 4.7 Django

Es un Framework para el desarrollo de aplicaciones Web basado en el lenguaje de programación Python que sigue el patrón de diseño MVC. Sabiendo que en los últimos tiempos la palabra "framework" parece que se ha convertido en el "Santo Grial" del desarrollo de aplicaciones Web.

Como ya hemos mencionado al estar hecho en Python y por tanto también será Python (con todas sus bondades) el lenguaje que se utiliza para crear nuestros sitios. Es software libre, con lo que tenemos acceso a su código fuente para aprender, entender, ayudar a mejorarlo, etc. Y además goza de una comunidad muy grande y activa, lo que ayuda a que se mantenga actualizado, se detecten y corrijan sus errores, tenga documentación actualizada y detallada, y algunas otras ventajas.

#### 4.7.1 Estructura de proyectos web en Django

Django distingue entre proyectos y aplicaciones. Un proyecto es un sitio web completo que consta de una o varias aplicaciones. Estas aplicaciones las proporciona Django o las escribe el desarrollador. El comando que crea un proyecto es django-admin.py. El cual contiene la siguiente estructura:

- **_init_.py:** Define nuestro directorio como un módulo Python válido.
- **manage.py:** Utilidad para gestionar nuestro proyecto: arrancar servidor de pruebas, sincronizar modelos, etc.
- **Settings.py:** Configuración del proyecto.
- **Urls.py:** Gestión de las urls. Este fichero sería el controlador de la aplicación. Mapea las urls entrantes a funciones Python definidas en módulos.

En la figura 4.6 se muestra la estructura de proyectos y aplicaciones en Django.

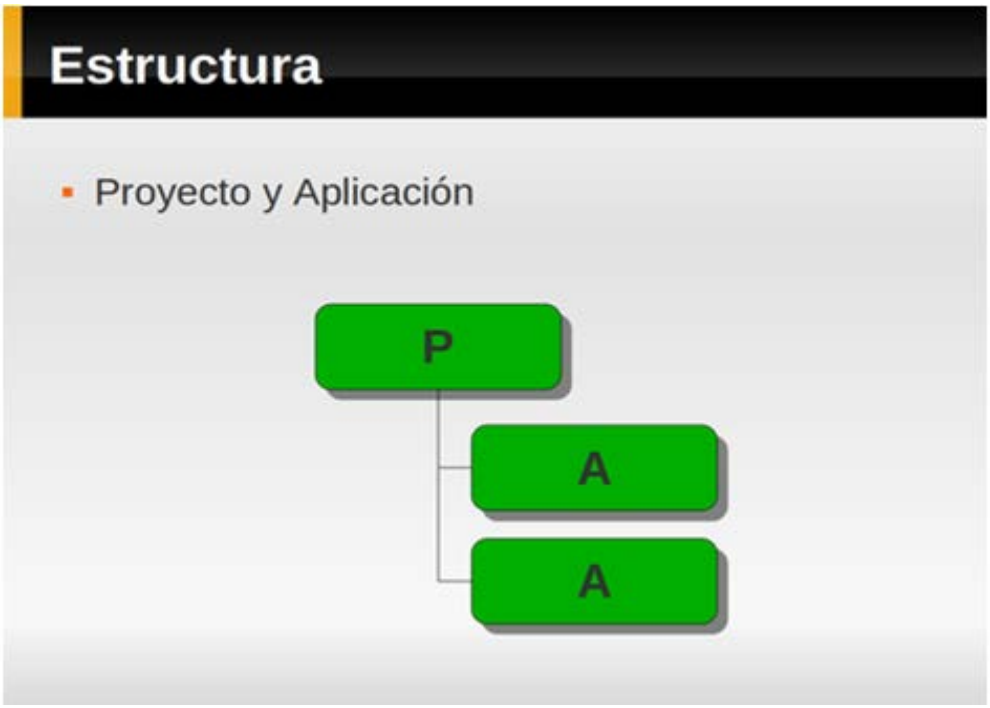


**Figura 4.6** Estructura de proyectos y aplicaciones en Django

### 4.7.2 Estructura de aplicaciones en Django

Para crear una aplicación nueva dentro del proyecto ejecutamos python manage.py startapp Libreria. Este comando crea el directorio Librería (aplicación) con la siguiente estructura:

- **_init_.py:** Define nuestro directorio como un módulo Python válido.
- **models.py:** Aquí se definen los modelos u objetos que serán mapeados a una base de datos relacional.
- **views.py:** Define las funciones que van a responder a las urls entrantes.

Esto es un diseño MVC: donde el modelo es (**models.py**), vista (**views.py**), controlador (**urls.py**).

## 4.8 Definiendo los modelos

Para crear los modelos, editamos el fichero **models.py** que nos ha generado la aplicación por ejemplo Liberia/models.py para definir nuestros objetos. Las clases que representan modelos deben heredar de la clase Model y siguen una sintaxis muy sencilla

e intuitiva. Django incorpora en el paquete django.contrib.auth todo un sistema de autentificación y gestión de usuarios.

Después de haber modificado dicho fichero, hacemos que Django sincronice la información que tiene de los modelos con el sistema relacional, es decir que cree las tablas necesarias por medio del siguiente comando: python manage.py syncdb, este comando también creará las tablas necesarias para la aplicación administrativa y el sistema de gestión de usuarios (de hecho nos pedirá los datos necesarios para crear un "superusuario").

### 4.9 Vistas genéricas

Las vistas genéricas de Django recogen ciertos estilos y patrones comunes del desarrollo de vistas y los abstraen, de manera que se puedan escribir vistas rápidamente sobre los datos sin tener que escribir mucho código.

Django contiene vistas genéricas que hacen lo siguiente:

- Realizan tareas “sencillas” y comues: redirigir a una página distinta, renderizar una plantilla dada.
- Muestran listados y páginas de detalle para un objeto.
- Permiten a los usuarios: crear, actualizar y borrar objetos.

# 5. TRABAJOS RELACIONADOS

## 5.1 Introducción

Hoy en día es más notoria la nueva tendencia en las organizaciones a hacer uso del Workflow como una herramienta clave para lograr mayor agilidad y aumentar la descentralización de las actividades administrativas y comerciales dentro de la misma. Los sistemas de Workflow o de flujo de trabajo, también conocidos como Business Process Management Systems / Sistemas de Gestión de Procesos de Negocio (BPMS) tienen el objetivo de acercar personas, procesos y máquinas, ahorrando tiempo y facilitando también la automatización de los flujos de trabajo entre procesos, pudiendo integrar estos en la empresa de acuerdo a unas estrategias concretas.

La evolución de Workflow consiste en buscar la máxima automatización de los procesos de trabajo y el control total de las diferentes etapas, durante las cuales los documentos, la información o las tareas pasan de un participante a otro, según unas normas o procedimientos previamente definidos por aquellos que dirigen la empresa. Las aplicaciones Workflow automatizan la secuencia de acciones, actividades o tareas en la ejecución del proceso, permiten realizar un seguimiento de cada etapa del mismo y aportan las herramientas necesarias para su control o gestión del flujo de trabajo.

**Beneficios del Workflow o flujo de trabajo**

Según los procesos de negocio que implantemos en la empresa los beneficios de los flujos de trabajo pueden ser:

1. Ahorro de tiempo y mejora de la productividad y eficiencia de la empresa, debido a la automatización de muchos procesos de negocio.
2. Mejora del control de procesos a través de la normalización de los métodos de trabajo.
3. Mejor atención y servicio al cliente; un incremento en la coherencia de los procesos da lugar a una mayor previsibilidad en los niveles de respuesta a los clientes.
4. Mejora en los procesos; mayor flexibilidad de acuerdo con las necesidades empresariales.

5. Optimización de la circulación de información interna con clientes y proveedores.
6. Integración de procesos empresariales.

En el mercado existen diversos tipos de herramientas Workflow, las principales son: Workflow Corporativo, Workflow de Aplicación, Workflow Documental y Workflow de Producción**.** Algunos de ellas se limitan a su área en particular y otras permiten la comunicación con aplicaciones externas de manera síncrona (esperando la respuesta antes de proseguir) y/o asíncrona (solamente deja un "mensaje" y recupera la respuesta más adelante).
A continuación mencionamos algunos de los workflow que se suelen utilizar.

## 5.2 Goflow

Es un sistema de gestión de flujo de trabajo basado en actividades, siendo la actividad la base de los procesos, los flujos de trabajo son secuencias de actividades que deben ser completadas para finalizar el proceso, esto difiere de los flujos de trabajo basados en entidades donde el foco es identificado sobre un documento dado y los estados se deben recorrer en orden para ser completados.

Por ejemplo, el proceso que describe una investigación en una biblioteca del campus puede ser manejado como un proceso basado en actividades: primero usted tiene que esperar se le asigne una determinada investigación por parte de su profesor, luego va a buscar los libres de su interés utilizando los catálogos de la biblioteca o un ordenador o la ayuda del bibliotecario, por último realiza una comprobación de que son los libros que usted necesita. Esto es una lista de actividades que deben ser realizadas para conseguir algunos libros de la biblioteca. No hay ningún documento principal en este proceso.

### 5.2.1 Publicación principal

La publicación principal para un sistema de flujo de trabajo debe contestar a las preguntas " quien debe hacer qué, cuándo y cómo”, ver figura 5.1. En un sistema de gestión de flujo basado en actividades como Goflow la pregunta tiene una respuesta.

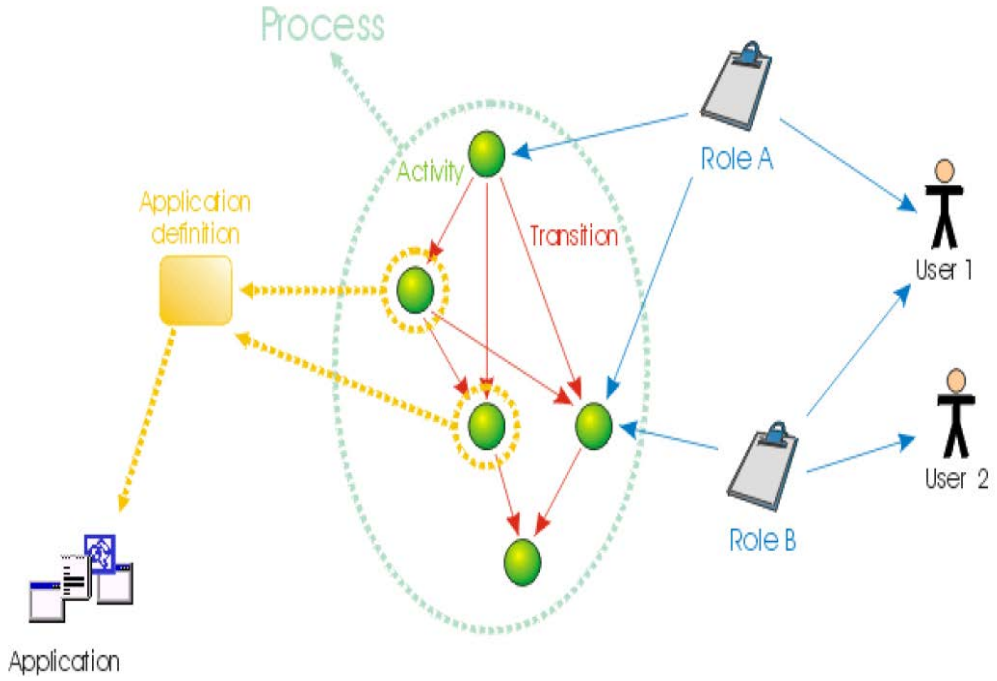


**Figura 5.1** "Quien debe hacer qué, cuándo y cómo?

El proceso que define la secuencia de actividades para ser realizadas se dice que debería estar dado por la definición de actividades y transiciones. Una actividad (el qué de la publicación principal representa algo que debe ser realizado: dar una autorización, actualización de una base de datos, enviar un correo electrónico, carga de un camión, relleno de una forma, impresión de un documento etc. Las transiciones definen la secuencia apropiada de actividades para un proceso (el cómo de la publicación principal). El quien es la parte asociada a la persona que tiene a su cargo el desarrollo de la actividad.

### 5.2.2 Razones para usar Goflow

Una razón es que la utilización de un sistema de gestión de flujo de trabajo permite mejorar la eficiencia y el rendimiento de los procesos que se manejan, muchas de las actividades que se realizan dentro de la organización pueden ser desarrolladas por un sistema automático.

La segunda razón es tener siempre de forma clara la respuesta a las preguntas quien debe hacer qué, cómo y cuándo. Formalizando su proceso de trabajo y no gastando tiempo en decisiones sobre qué debe hacer cada persona y de igual forma garantizar que su proceso será completado de forma correcta.

### 5.2.3 Desventaja

El soporte con el que cuenta para desarrollo de aplicaciones web no es muy amplio lo cual no le hace conveniente en el marco de este trabajo.

## 5.3 FlowMind

Es una solución de flujo de trabajo para la plataforma Java, aunque fue diseñado especialmente para proveedores de software, también se encuentra disponible para organizaciones públicas y privadas. Son más de 1000 organizaciones las que utilizan FlowMind de forma cotidiana, especialmente los gobiernos centrales y locales, instituciones financieras, etc.

### 5.3.1 Aspectos destacados

Sus principales ventajas son: Calidad de la arquitectura y las interfaces de programación de las aplicaciones. Otra de sus ventajas es que se puede ejecutar sobre la mayoría de los sistemas operativos y trabaja con la mayoría de las bases de datos (Oracle, PostgreSQL, DB2).

Uno de sus componentes se ve a continuación:

**FlowPoint:** Es una aplicación gráfica diseñada para los expertos de negocios que puede asignar sus procesos de negocio sin ningún tipo de habilidad técnica. Los modelos de proceso se guardan en formato BPDL y están basados en el lenguaje de marcas extensible (XML).

Es de resaltar que su principal desventaja es la escasa documentación al respecto.

En la figura 5.2 se presenta un esquema de procesos de negocio utilizando Flowpoint.

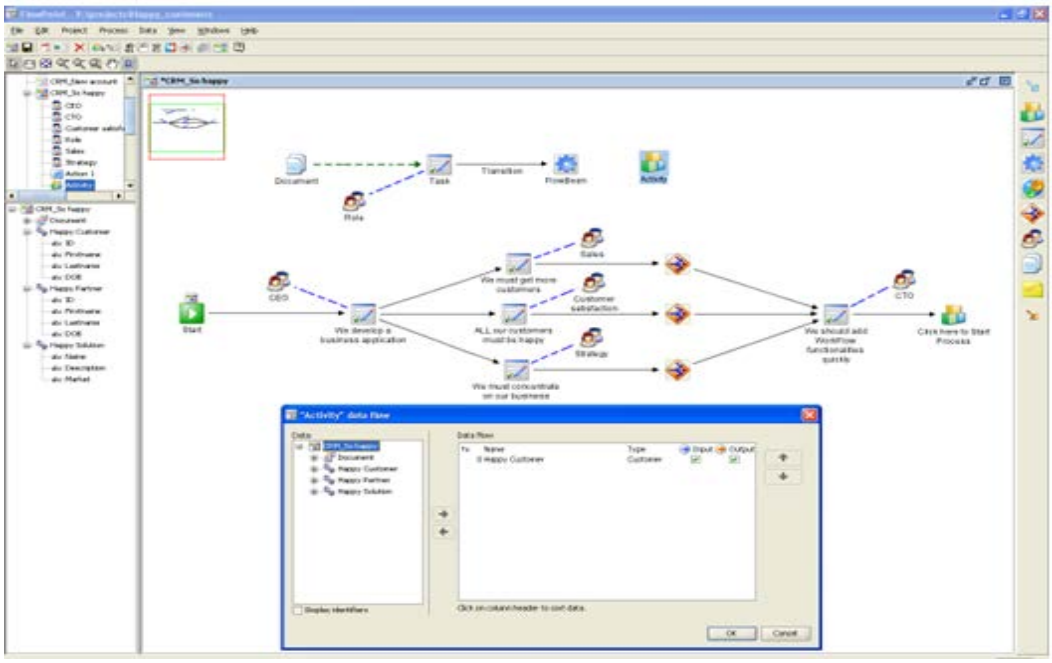

**Figura 5.2** Esquema con Flowpoint

## 5.4 Aqualogic BPM Suite

Es una suite BPM que ofrece un conjunto de gestión de procesos empresariales. Combina la tecnología de flujo de trabajo y procesos con funcionalidad de integración de aplicaciones en las organizaciones. La suite consta de herramientas destinadas a ser utilizadas por el personal de línea de negocio para la creación de modelos de procesos de negocio (BPM AquaLogic Designer), así como herramientas para el personal de TI para crear aplicaciones reales de procesos de negocio directamente de dichos modelos (AquaLogic BPM Studio). Esta suite permite la colaboración entre las áreas de negocio, y el área de TI (Tecnologías de la Información), para automatizar y optimizar los procesos del negocio, impulsando eficiencia y agilidad, mientras se reducen los costos, y se mejora la calidad.

Los analistas de negocio pueden diseñar y correr simulaciones de un proceso completo sin necesidad de apoyo de TI, y solo cuando el proceso abarca las especificaciones de negocio, es traspasado a TI para que implemente e integre los componentes necesarios para implementar el proceso.

Las interfaces de usuario para integrar a los participantes en el proceso (WorkFlow) son generadas automáticamente y además se provee Portlets (componentes de presentación)

estándar para el ambiente de ejecución. Datos en tiempo real, e históricos son recolectados por el servidor y están disponibles en dashboards (paneles de control), existen reportes que permiten realizar una optimización continua de los procesos de negocio, debido a que se puede realizar un seguimiento de las actividades del negocio.

A continuación se exponen algunos componentes de la Suite.

### 5.4.1 BEA Aqualogic Designer

Es el ambiente de diseño para los analistas de negocio, permite la creación de cualquier tipo de proceso realizando el arrastre de los elementos a las áreas de rol (swimlanes) correspondientes.

Este módulo soporta estándares como el lenguaje de ejecución de procesos de negocio (BPEL), y lenguaje de Modelado Unificado (UML).

Se puede modelar y simular un proceso sin necesidad de programar, ni del departamento de Tecnologías de información (TI), ni tampoco necesita que el proceso este implementado en producción, ni de estar conectado al servidor de procesos.

En la figura 5.3 se muestra un ejemplo de diseño con AquaLogic Designer.

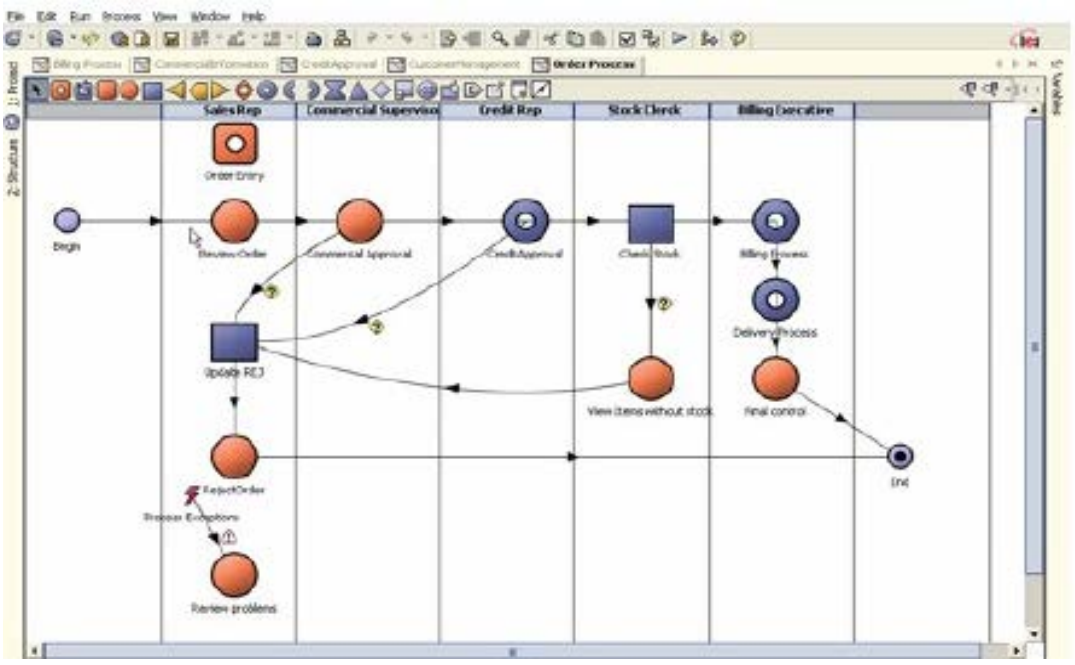


**Figura 5.3** Diseño en AquaLogic Designer

### 5.4.2 BEA Aqualogic BPM Studio

Es el ambiente de trabajo de los desarrolladores de procesos de negocio, incluye todo lo del BPM Designer, sumando las herramientas necesarias para que el desarrollador cree los componentes de negocio, se integre a los sistemas existentes, y ensamble la interfaz de usuario para la interacción de las personas participantes. Esta herramienta soporta interfaces estándares como J2EE, .NET, WebServices, XML, entre otras.

### 5.4.3 BEA Aqualogic BPM Enterprise Server

Orquesta todos los procesos y sus recursos; personas, organizaciones, sistemas, gestionando la secuencia y reglas de negocio definidas, de igual forma audita cada paso asegurando la fluidez en la ejecución del proceso. El servidor ejecuta los procesos diseñados en el BPM Studio, así como cualquier proceso escrito en BPEL.

### 5.4.4 BEA Aqualogic BPM Workspace

Es parte del BPM Enterprise Server y es el ambiente de trabajo para los participantes del proceso de negocio. Las actividades de negocio que requieren de interacción de personas son automáticamente publicadas con una interfaz Web, sin necesidad de diseño Web manual, o programación. Los participantes tienen una vista de sus tareas pendientes y se les ofrece un medio para ejecutar dichas tareas.

## 5.5 WfMOpen

WfMOpen es una implementación basada en J2EE de una instalación de flujo de trabajo (motor de flujo de trabajo), propuesto por la Coalición de gestión de flujos de trabajo (WfMC) y el Grupo de Gestión de Objetos (OMG).

El componente de flujo de trabajo está basado en un conjunto de interfaces Java que definen un API para la gestión de los flujos de trabajo, las interfaces “omgcore” siguen la gestión de flujos de trabajo definida por la OMG.

### 5.5.1 Ventaja

Proporciona una excelente escalabilidad y soporta con comodidad la integración de tecnologías incluyendo SOAP.

En la figura 5.4 se observa la estructura de WfMOpen.

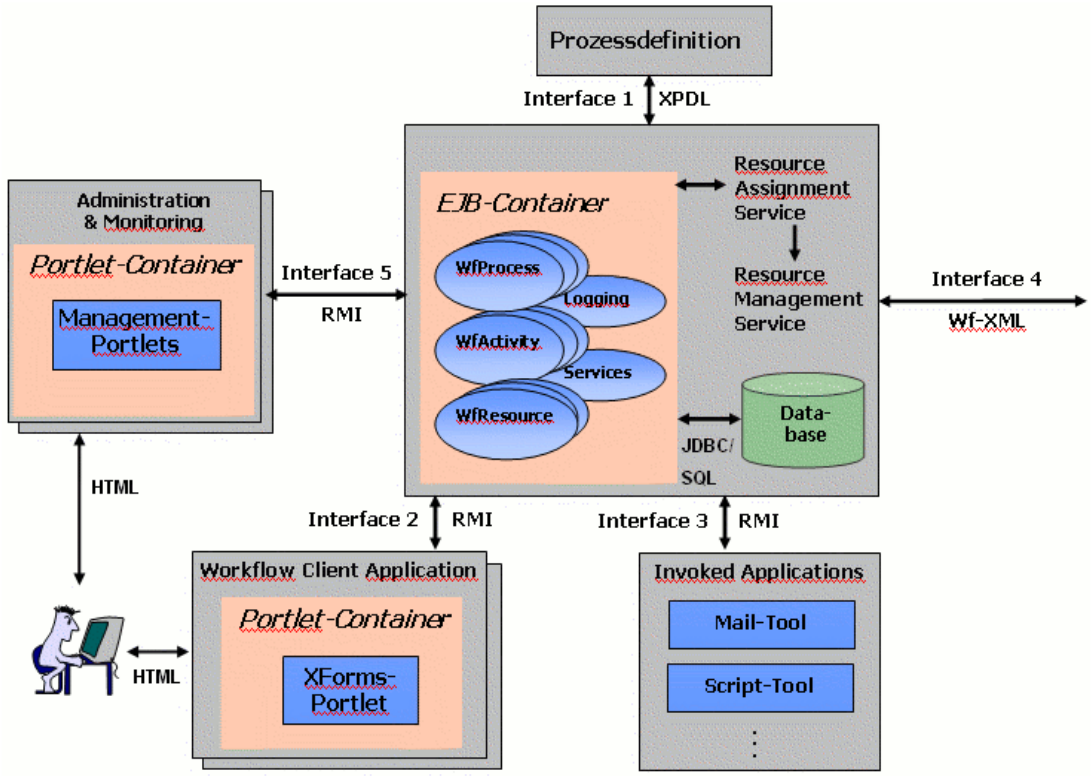


**Figura 5.4** Estructura de WfMOpen

## 5.6 OFBIz

El motor para flujos de trabajo está basado en la Wfmc y las especificaciones de la OMG, el motor de flujo de trabajo es un miembro del Marco de Servicios y está fuertemente integrado con el motor de entidad. Es decir utiliza las entidades encontradas en el flujo de trabajo del modelo de entidades, utilizando el lenguaje de marcas extensible (XML) para la definición de la información del modelo de entidades y su almacenamiento en tiempo de ejecución. Todos los cambios a un proceso o actividad son reflejados en tiempo real. Por lo tanto, el motor no corre en un hilo, es más bien un grupo de APIS y objetos comunes que manejan el flujo. Cuando se realiza un cambio al

flujo de trabajo el motor procesa dicho cambio y cuando ha finalizado el motor retorna, de tal forma que el flujo de trabajo siempre continua donde se reinició la última vez.

### 5.6.1 Atributos extendidos a actividades

- **AcceptAllAssignments:** Este atributo extendido le dice al motor de procesos de trabajo que todas las asignaciones deben ser aceptadas antes de que una actividad empiece. Esto es útil cuando hay múltiples participantes en una actividad y cada uno debe aceptar su prioridad de asignación al comienzo de la actividad.
- **CompleteAllAssignments**: Es similar al anterior y le dice al motor de flujo de trabajo que todas las asignaciones deben ser completadas antes de que la actividad se termine.
- **LimitService:** Cuando se define una actividad un límite puede ser establecido, esta es la cantidad de tiempo en la cual la actividad debe ser realizada, un servicio se puede poner en ejecución si la actividad no se ejecutó en la cantidad de tiempo asignada.
- **LimitAfterStart:** Esto le dice al motor de flujo de trabajo que la comprobación de límite debería ocurrir después de que la actividad ha comenzado.
- **CanStart:** Este atributo le dice al motor de flujos que permita a la actividad conectada a su ExtendedAttribute iniciar el flujo de trabajo.

### 5.6.2 Actividad de inicio por defecto

Por defecto cuando un flujo de trabajo es iniciado la primera actividad en la lista es puesta como actividad de inicio. Esto quiere decir que cuando un proceso o flujo es comenzado normalmente, se encontrará ejecutando la primera actividad, luego seguirá con las transiciones. Este es el modo más común de invocar un flujo de trabajo. Sin embargo, cuando un flujo de trabajo debe ser invocado desde un punto o puntos diferentes de inicio, el cliente API estará preparado para llamar a la actividad apropiada para comenzar y las transiciones continúan desde aquel punto.

### 5.6.3 Actividades del flujo de trabajo

La combinación de actividades manuales y automáticas es lo que hace a un flujo de trabajo tan poderoso.

- **NO:** Describe actividades manuales
- **ROUTE**: Una actividad de ruta es usada para simplemente encaminar a otras actividades vía las transiciones.
- **SUB-FLOW**: Un subproceso es creado y se ejecuta, pueden ser definidos como síncronos o asíncronos.

## 5.7 jBPM

Es una Suite de gestión de procesos de negocio (BPM) flexible. Esta realiza el puente entre los analistas del negocio y los desarrolladores. Los motores tradicionales de BPM tienen un enfoque que se limita a personas sin conocimientos técnicos solamente, jBPM tiene un doble enfoque ofreciendo características del proceso de gestión de una manera que tanto los usuarios del negocio como los desarrolladores se sientan a gusto.

### 5.7.1 Qué puede hacer jBPM

Un proceso de negocio le permite modelar sus objetivos de negocio mediante la descripción de los pasos que se deben ejecutar, para lograr ese objetivo y el orden en que se deben ejecutar se utiliza un diagrama de flujo ver figura 5.5. Esto mejora la visibilidad y la agilidad de su lógica de negocio, los resultados en el nivel superior y las representaciones específicas del dominio pueden ser entendidos por los usuarios del negocio y son más fáciles de controlar.

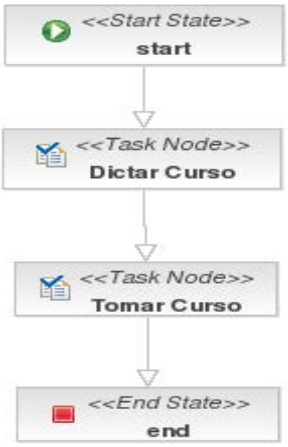


**Figura 5.5** Esquema jBPM.

El núcleo de jBPM es un peso ligero, el motor de flujo de trabajo extensible se encuentra escrito en Java puro lo que le permite ejecutar los procesos de negocio utilizando la última Notación del modelado de Procesos de Negocio (BPMN 2.0). Por esta razón puede funcionar en cualquier ambiente Java.

En la parte superior del motor central, se encuentran una gran cantidad de funciones y herramientas que se ofrecen para apoyar los procesos de negocio a lo largo de su ciclo de vida:

- Editor basado en Eclipse y basados en la web para apoyar la creación gráfica de los procesos de su negocio.
- Persistencia y transacciones basadas en JPA / JTA.
- Servicio de tareas humanas basado en Servicios web para representar las tareas que deben llevarse a cabo por los actores humanos.
- Consola de administración y apoyo a la gestión de procesos, listas de tareas, gestión de formas de trabajo y generación de informes.
- Repositorio de procesos opcionales para implementar el proceso (y los conocimientos relacionados con otros)
- Un registro de la historia (para realizar consultas / monitoreo / análisis).
- La integración con Seam, Spring, OSGi, entre otras.

jBPM soporta a los procesos adaptativos y dinámicos que requieren flexibilidad para modelados complejos, situaciones reales que no se pueden describir fácilmente mediante un proceso rígido. Se lleva el control a los usuarios finales por lo que les permite controlar qué partes del proceso debe ser ejecutado o desviarse de forma dinámica.

### 5.7.2 Lenguajes de proceso

JBPM se basa en un motor de procesos genéricos, que es la base para soportar múltiples idiomas de forma nativa. jBPM5 se centra en **BPMN 2.0** como el lenguaje para expresar los procesos de negocio. BPMN 2.0 es una especificación estándar que define la visualización y la serialización del lenguaje de marcas extensible (XML) de procesos de negocio, se puede ampliar si es necesario para incluir funciones más avanzadas.

### 5.7.3 JBPM 5

JBPM5 es la última versión de la comunidad del proyecto jBPM. Se basa en la especificación de BPMN 2.0 y soporta todo el ciclo vital del proceso de negocio (desde la creación hasta la ejecución, supervisión y gestión ) El actual jBPM5 de código abierto ofrece la ejecución de procesos de negocio y gestión, incluyendo:

- Es integrable, ligero motor de procesos Java, el apoyo a la ejecución nativa BPMN 2.0.
- BPMN 2.0 modelado de procesos en Eclipse (los desarrolladores) y la web (los usuarios de negocios).
- Colaboración de procesos, seguimiento y gestión a través del repositorio Guvnor y la consola web.
- La interacción humana con una organización independiente WS-HT.
- Poderosa integración con las reglas de negocio y el procesamiento de eventos.

## 5.8 Bonita Open Solution

Esta herramienta destaca sobre todo por la facilidad en su utilización debido al diseño intuitivo de los diferentes elementos que lo componen y por el bajo coste de su implantación (no requiere más inversión que el tiempo de aprendizaje). Además, la modelización de los procesos es compatible con la especificación BPMN 2.0.

Las características principales de esta herramienta son:

- Open Source
- Es ligera
- Compatibilidad con BPMN 2.0
- Interfaz Intuitiva
- Potente
- Fácil importación de procesos desde otras herramientas (Tibco, Lobardi, etc.)
- Personalización de las herramientas
- Integración de los procesos con reglas de negocio
- Conectores nativos (+ de 100) : SAP , Microsoft Exchange , etc
- Conectores propios (realizados por la comunidad de BonitaSoft)
- Integración en Talend MDM Enterprise Edition

A continuación se describen brevemente los módulos que la conforman

### 5.8.1 Bonita Execution Engine

Es el motor de BPM de Bonita y se encarga de la conexión de los procesos que existen en el sistema, así como el despliegue y ejecución de los procesos. El módulo de Bonita Studio está conectado directamente a este otro módulo para funcionar. Por suerte, este motor es genérico y extensible por lo que siempre seremos capaces de añadir con mayor o menor dificultad nuevos estándares o bien servicios que puedan aparecer en el mundo de BPM con posterioridad.

### 5.8.2 Bonita Studio

Es la aplicación gráfica cuya función es diseñar los procesos BPM usando la Notación de Gestión de Procesos de Negocio (BPMN) sobre un área de diseño (pizarra) de forma muy intuitiva basada en "arrastrar" los elementos y en su configuración específica mediante una o varias pestañas habilitadas para ello.

### 5.8.3 Bonita Form Builder

Es la aplicación encargada de mostrar los formularios a los usuarios de la aplicación. Recordar que muchos de los pasos que se producen en un proceso BPM requieren de la entrada de datos por parte del usuario implicado.

### 5.8.4 Bonita User Experience (User XP)

Es la aplicación encarga de la gestión de todo lo relacionado con los procesos BPM desplegados. Por suerte es muy intuitiva ya que su interfaz se "parece" a una aplicación de gestión de correo.

Ahora es el momento de comentar el motor del Flujos de trabajo seleccionado (**OpenWfe)**. Su selección radica en dos grandes ventajas:

1) Cuenta con una buena documentación.
2) Ofrece un buen soporte al desarrollo de aplicaciones Web.

## 5.9 OpenWfe

OpenWFE es un entorno de flujo de trabajo libre, de igual forma es un sistema de gestión de flujo de trabajo completo y eficiente para todo tipo de entornos. Cuenta con cuatro componentes:

- Un motor de flujo de trabajo.
- Un manejador de listas de trabajo.
- Un entorno de ejecución de participantes automático en tiempo de ejecución.
- Una interfaz Web.

La interfaz web, como se presenta es sólo una "prueba de concepto". OpenWFE está publicado sólo en su conjunto completo. Los cuatro componentes son empaquetados para poder interactuar con cada uno de los otros componentes que forman parte del entorno de ejecución del motor de trabajo de OpenWfe.

Esta herramienta nos es muy útil cuando una aplicación tiene etapas o pasos a seguir, acciones concretas para cada paso y diferentes acciones para distintos perfiles de usuario. Por ejemplo, supongamos que tenemos una aplicación editorial, para controlar el proceso de la publicación de un libro habrá una serie de acciones a seguir: el autor entrega un primer capítulo, el editor lo recibe y lo admite como publicable o no, si se pretende publicar, avisará al escritor de que siga escribiendo el libro. En ese caso, el autor irá enviando versiones sucesivas, que irán recibiendo el editor, correctores y personas que aporten ideas. Cuando el libro esté finalizado, lo recibirá el corrector y luego se mandará a publicar.

A continuación se presentan algunos de los principales elementos de un flujo de trabajo en OpenWfe:

### 5.9.1 Process definitions

El nodo raíz es llamado "proceso de definición y tiene dos atributos obligatorios:" nombre " y “revisión”.
A continuación se presenta un ejemplo de esquema de Process definition:

```
<process-definition
name="flow"
revision="1.13">
   <description>
A recursive flow subdefinition
   </description>
   <subprocess ref="review" />
   <process-definition name="review">
     <sequence>
       <participant ref="role-alpha" />
          <if>
            <not>
             <equals field-value="reviewed" other-value="true"/>
            </not>
            <subprocess ref="review" />
            <!-- loop until 'reviewed' is set to 'true' -->
          </if>
    </sequence>
  </process-definition>
</process-definition>
```

**Tabla 5.1** Process Definition en OpenWfe

### 5.9.2 Sequence

La expresión llamada "secuencia" cubre este patrón como se muestra a manera de ejemplo a continuación:

```
<sequence>
       <participant ref="task-a" />
       <participant ref="task-b" />
</sequence>
```

**Tabla 5.2** Sequence en OpenWfe

### 5.9.3 Concurrence

La Concurrencia proporciona la ejecución en paralelo de expresiones "hijos". Se entiende una cierta sintaxis de sincronización para conciliar ciertas ramas una vez que termina la ejecución.

La Sincronización ve múltiples ramas que convergen en un solo hilo de control.

Las expresiones de concurrencia dividen el flujo en ramas en paralelo, los hijos son las ramas. Los atributos de la expresión de concurrencia se utilizan para definir cómo debe realizarse la sincronización como "respuestas" de cada rama.
A continuación se presenta un ejemplo de un esquema de concurrencia.

```
<concurrence
        sync="generic"
        count="x|*"
        over-if="x == y"
        merge="first|last|highest|lowest"
        merge-type="mix|override"
        remaining="cancel|forget" >

        (...)
</concurrence>
```

**Tabla 5.3** Concurrence en OpenWfe

### 5.9.4 Participant

El flujo más sencillo es tal vez el que consta de sólo un participante. Un participante es para el motor (y para una definición del flujo) sólo un nombre, una referencia que se busca en el mapa de los participantes. tal vez llamarle ‘directorio' habría sido más adecuado para esta abstracción.

La expresión de los participantes tiene un atributo principal, llamado 'ref' (abreviatura de "referencia").
A continuación observamos la estructura del elemento participante.

```
<participant ref="role-alpha" />
<participant ref="role-alpha" timeout="2h" />
```

**Tabla 5.4** Participant en OpenWfe

### 5.9.5 If

La' expresión ‘If’ puede contener dos o tres cláusula directas. La primera es la cláusula condicional, la segunda es la cláusula ‘Then’, la tercera y opcional es la cláusula ‘Else’, cualquier otro hijo directo después de la tercera cláusula será ignorado.

A continuación se presenta un esquema de ejemplo para dicha expresión.

```
<if>
        <equals field-value="accepted" other-value="true" />
        <participant ref="employee1" />
</if>
<if>
      <equals variable-value="can_deliver" other-value="true" />
      <participant ref="logistics" />
      <participant ref="support" />
</if>

<if>
     <greater-than variable-value="amount" other-value="1000" />
     <participant ref="logistics" />
</if>
```

**Tabla 5.5** Expresión If en OpenWfe

# 6. Propuesta Metodológica

Hasta este momentos hemos visto la tecnología y las herramientas que se van a utilizar, en este capítulo se presenta el desarrollo de la propuesta planteada.

En vista de que este trabajo se enmarca en el desarrollo de aplicaciones en el framework Django utilizando el Modelado de Procesos de Negocios (BPM) y la Arquitectura dirigida por modelos (MDA), iniciaremos el análisis de la propuesta en la sección 6.1 viendo los pasos para desarrollar aplicaciones en Django, a continuación se menciona en la sección 6.2 la Especificación del Sistema, en la sección 6.2.1 se presenta la especificación de los procesos de negocio y los conceptos de OpenWFE modelados y formalizados en el metamodelo de procesos de negocios que vamos a utilizar, luego veremos en la sección 6.2.2 el metamodelo de diagrama de clases planteado para realizar la especificación de la estructura del sistema y por ultimo en la sección 6.3 se observan las transformaciones implementadas para la generación de código.

## 6.1 Desarrollo de aplicaciones en Django

El primer paso es la creación de un proyecto: Un proyecto en Django es una estructura que contiene un archivo de configuración (settings), urls, y una colección de aplicaciones Django.

El comando que crea un proyecto es django-admin.py startproject biblioteca. El cual genera un proyecto de nombre biblioteca para este ejemplo, el cual contiene la siguiente estructura:

- **_init_.py:** Define nuestro directorio como un módulo Python válido.
- **manage.py:** Utilidad para gestionar nuestro proyecto: arrancar servidor de pruebas, sincronizar modelos, etc.
- **Settings.py:** Configuración del proyecto.
- **Urls.py:** Gestión de las urls. Este fichero sería el controlador de la aplicación. Mapea las urls entrantes a funciones Python definidas en módulos.

Tanto este paso como los otros necesarios para el desarrollo de aplicaciones con Django se presenta en la figura 6.1.

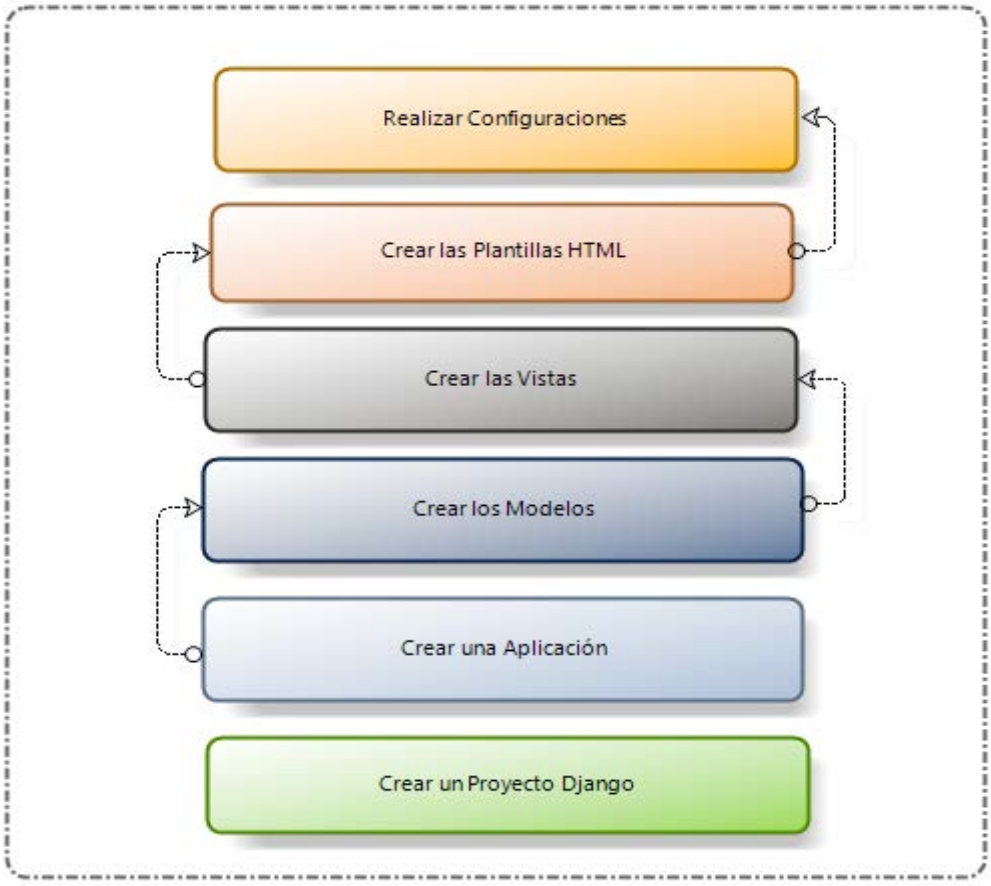


**Figura 6.1** Pasos para desarrollar aplicaciones web en Django.

El segundo paso hace referencia a la creación de una aplicación, para esto utilizamos el comando python manage.py startapp miaplicacion el cual genera una aplicación con el nombre miaplicacion y con la siguiente estructura:

- **_init_.py:** Define nuestro directorio como un módulo Python válido.
- **models.py:** Contiene una descripción de las tablas de la base de datos, representadas por una clase la cual se conoce con el nombre de Modelo.
- **views.py:** Contiene la lógica del negocio, define las funciones que van a responder a las urls entrantes.

El tercer paso es la creación de los modelos los cuales contienen la descripción de las tablas de la base de datos, se encuentran representados por una clase la cual se conoce como modelo.

Siguiendo con el proceso el próximo paso es la generación de las templates HTML, dichas plantillas describen el diseño de la página usando un lenguaje con sentencias lógicas y básicas.

Por último se deben realizar las configuraciones necesarias tanto del proyecto como de la aplicación, de esta forma se configura el motor de base de datos a utilizar, se crean las urls para hacer referencia a las vistas, entre otras configuraciones.

**6.2 Especificación del Sistema:** La idea es presentar los metamodelos definidos tanto para la especificación de procesos de negocios basándose en el motor OpenWfe, asi como el metamodelo definido para la especificación de la estructura del sistema.

**6.2.1 Especificación de los Procesos de Negocio:** Para realizar esta especificación se ha definido un metamodelo, pero dada la complejidad visual que presenta el diagrama de clases de dicho metamodelo, se ha decidido presentarlo por medio de la descomposición de sus partes, es decir las respectivas relaciones de cada una de las clases presentes en el metamodelo.

En la figura 6.2 se presenta el metamodelo openwfe.ecore definido para especificar los procesos de negocio basados en el motor OpenWFE.

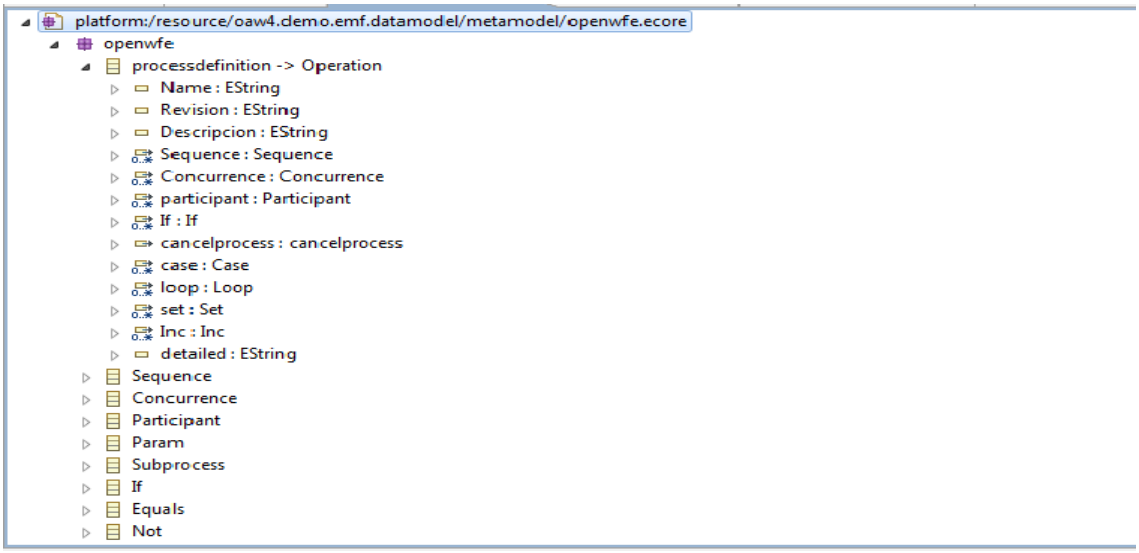


**Figura 6.2** Metamodelo para la especificación de procesos de negocio.

A continuación se describe la clase principal del metamodelo la cual permite definir los procesos de negocios cuyo nombre es "**processdefinition**", esta clase se considera el nodo raíz y posee dos atributos obligatorios: Name y Revision, así como una Descripción de tipo opcional. En la figura 6.3 se observan las relaciones de esta clase.

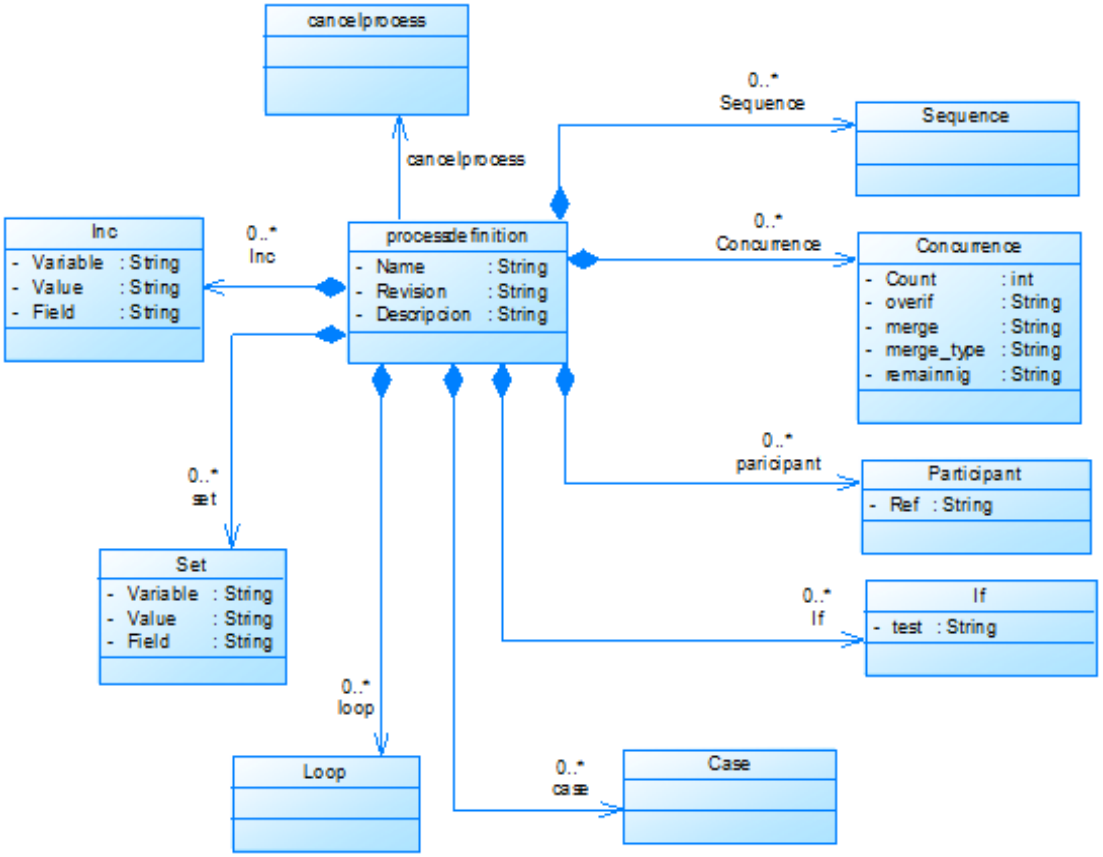


**Figura 6.3** Diagrama de Clases para la clase processdefinition.

Continuando con el diagrama de clases del metamodelo planteado nos encontramos con la clase "**Sequence**", esta clase dirige la ejecución de una serie de expresiones que componen un segmento de los procesos de negocio.

En la figura 6.4 se observan las relaciones de esta clase.

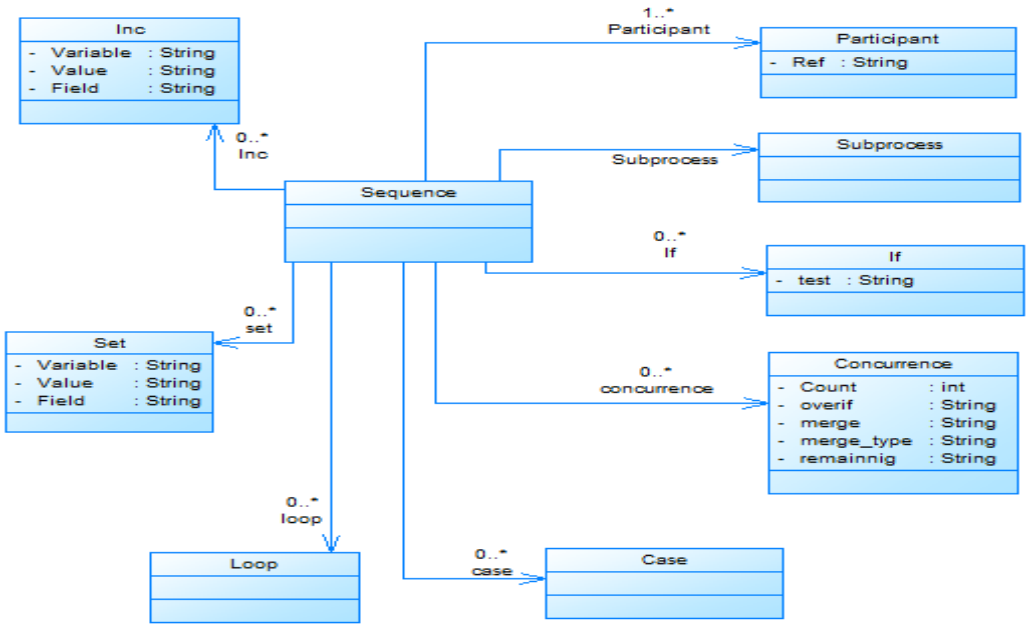


**Figura 6.4** Diagrama de Clases para la clase Sequence.

Otro de los conceptos importantes y que puede presentarse en cualquier proceso de negocio es el que describe la clase "**Concurrence**", esta clase establece la ejecución en paralelo de expresiones hijas, posee una sintaxis de sincronización segura que permite una conciliación entre las ramas una vez termina el proceso. En la figura 6.5 podemos observar las relaciones que posee esta clase.

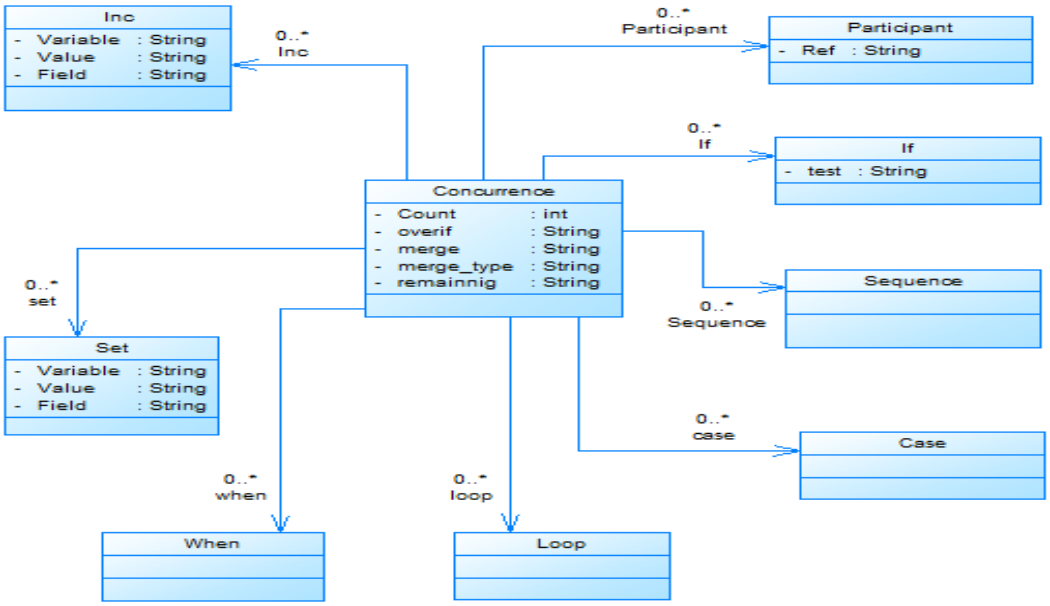


**Figura 6.5** Diagrama de Clases para la clase Concurrence.

Es el momento de presentar el concepto descrito por la clase "**Participant**", uno de los conceptos importantes ya que son aquellos los que se encargan de realizar determinada tarea. Tal como se observa en la figura 6.6 se encuentra relacionada con clase: Param.

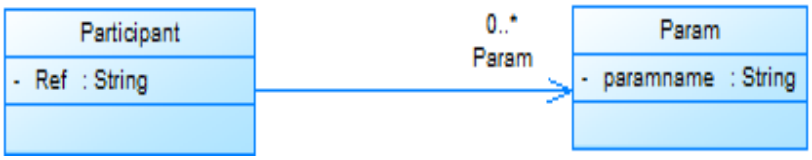


**Figura 6.6** Diagrama de Clases para la clase Participant

Dando continuidad a la descripción del metamodelo definido nos encontramos con otro de los conceptos importantes y es la clase "**If**", esta expresión tiene dos o tres hijos directos, el primero es la cláusula condicional, el segundo la cláusula Then y el tercero o Else es de tipo opcional. En la figura 6.7 podemos observar las relaciones que posee esta clase.

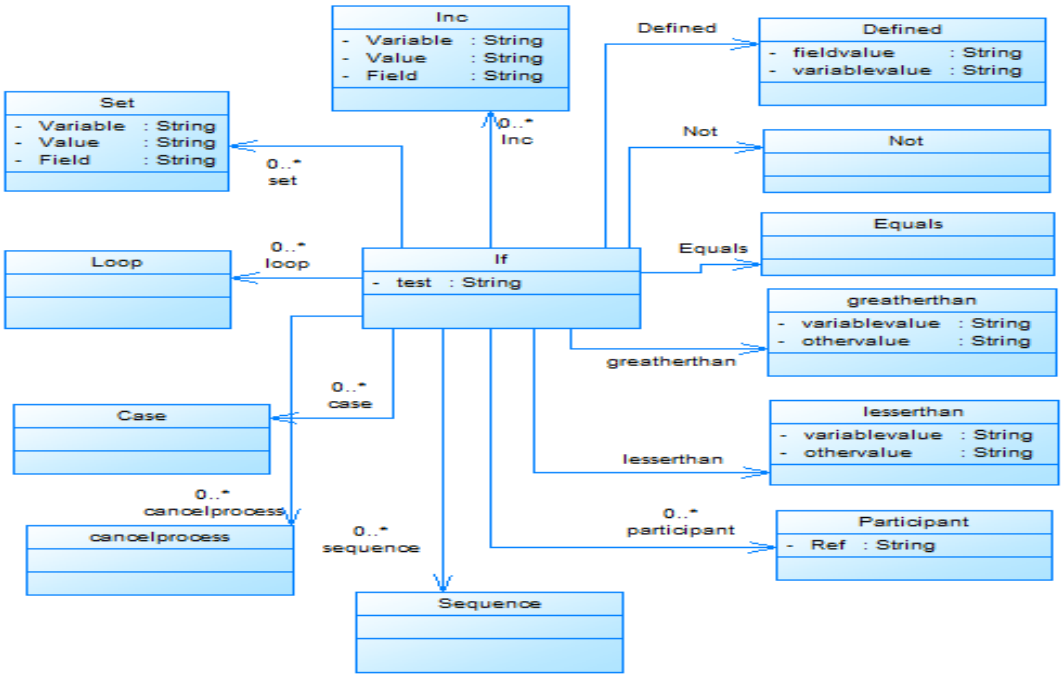


**Figura 6.7** Diagrama de Clases para la clase If

Otra de las clases del metamodelo es la clase "**Case**", esta clase permite evaluar una serie de expresiones booleanas seguidas por unas expresiones clásicas. En la figura 6.8 se observan las relaciones de esta clase.

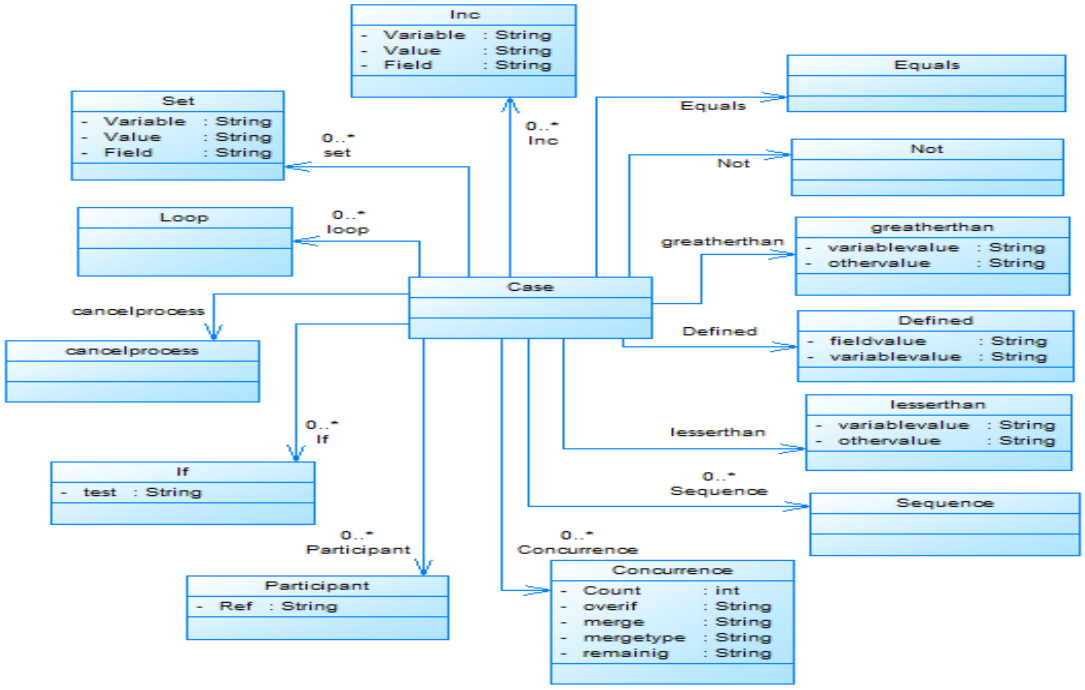


**Figura 6.8** Diagrama de Clases para la clase Case

Otro de los conceptos importantes y que puede presentarse en cualquier proceso de negocio es el que describe la clase "**And**", esta clase se evalua a sí misma como verdadero si todos sus hijos han sido evaluados a verdadero. En la figura 6.9 se observan las relaciones de esta clase.

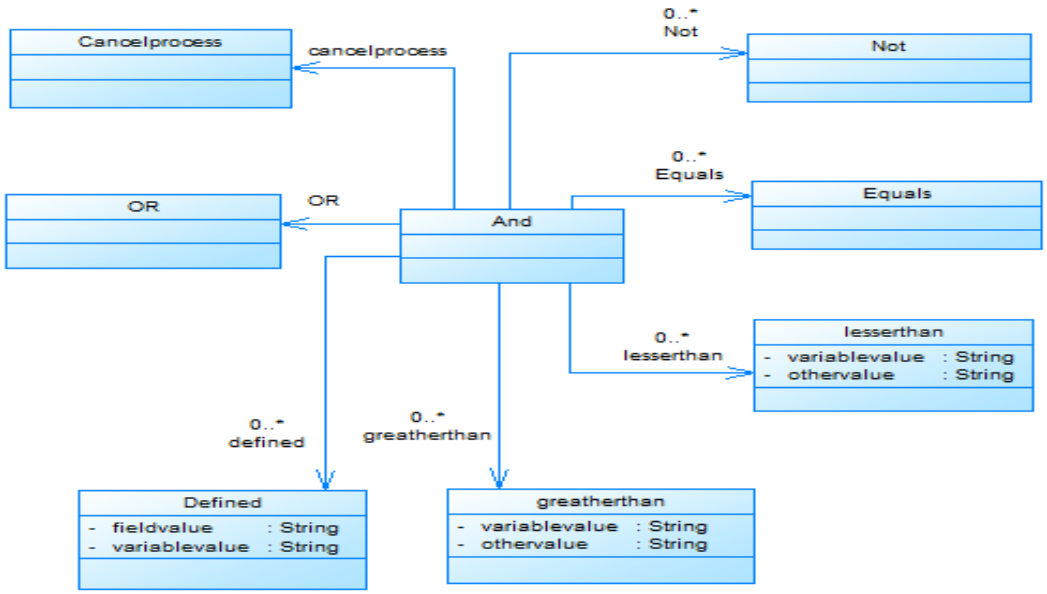


**Figura 6.9** Diagrama de Clases para la clase And

De la misma forma existe otro concepto que puede estar presente en cualquier proceso de negocio y en el metamodelo que hemos definido, el nombre para dicha clase es

"**Cursor**", esta clase es una extensión de la expresión secuencia y permite un control completo del flujo. En la figura 6.10 podemos observar las relaciones de esta clase.

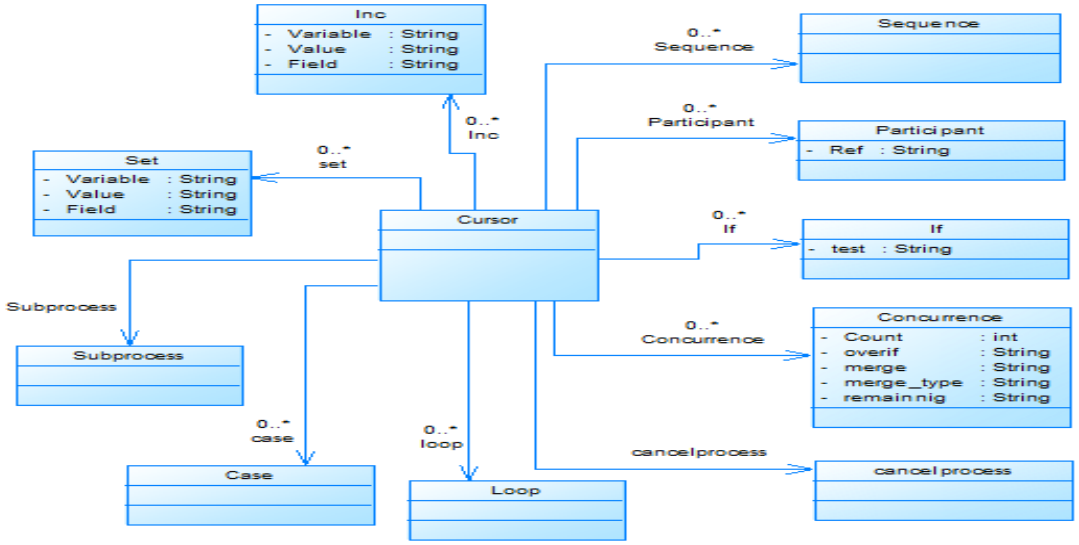


**Figura 6.10** Diagrama de Clases para la clase Cursor

Ahora mencionaremos el concepto definido en nuestro metamodelo en la clase "**OR**", esta clase indica otra expresión condicional y evalua la totalidad de sus hijos. En la figura 6.11 se observan las relaciones de esta clase.

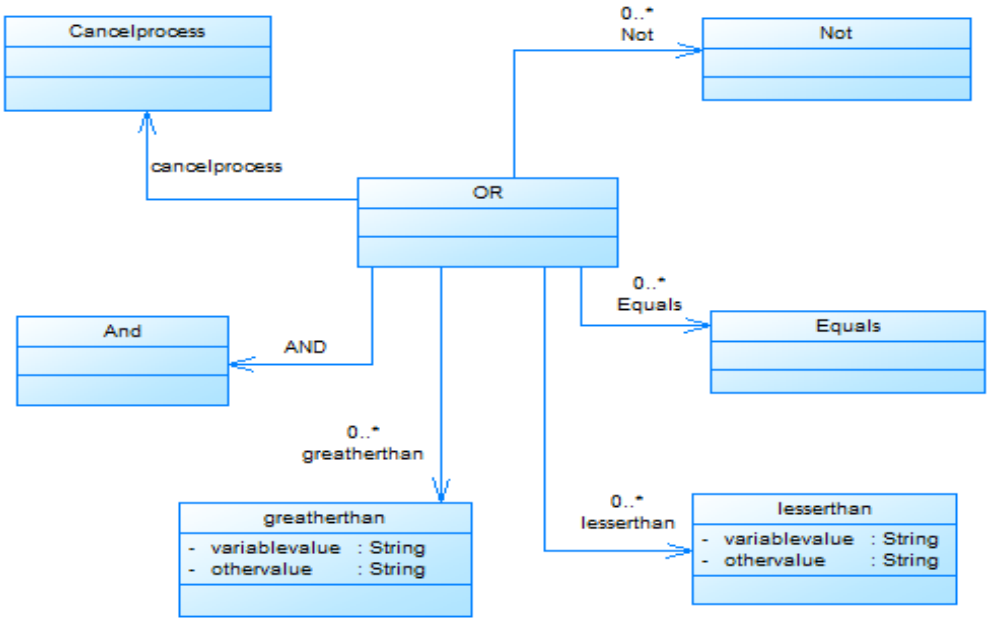


**Figura 6.11** Diagrama de Clases para la clase OR

Otro de los conceptos que se encuentra dentro del metamodelo definido para los procesos de negocio se encuentra en la clase "**Loop**". . En la figura 6.12 podemos observar las relaciones de esta clase.

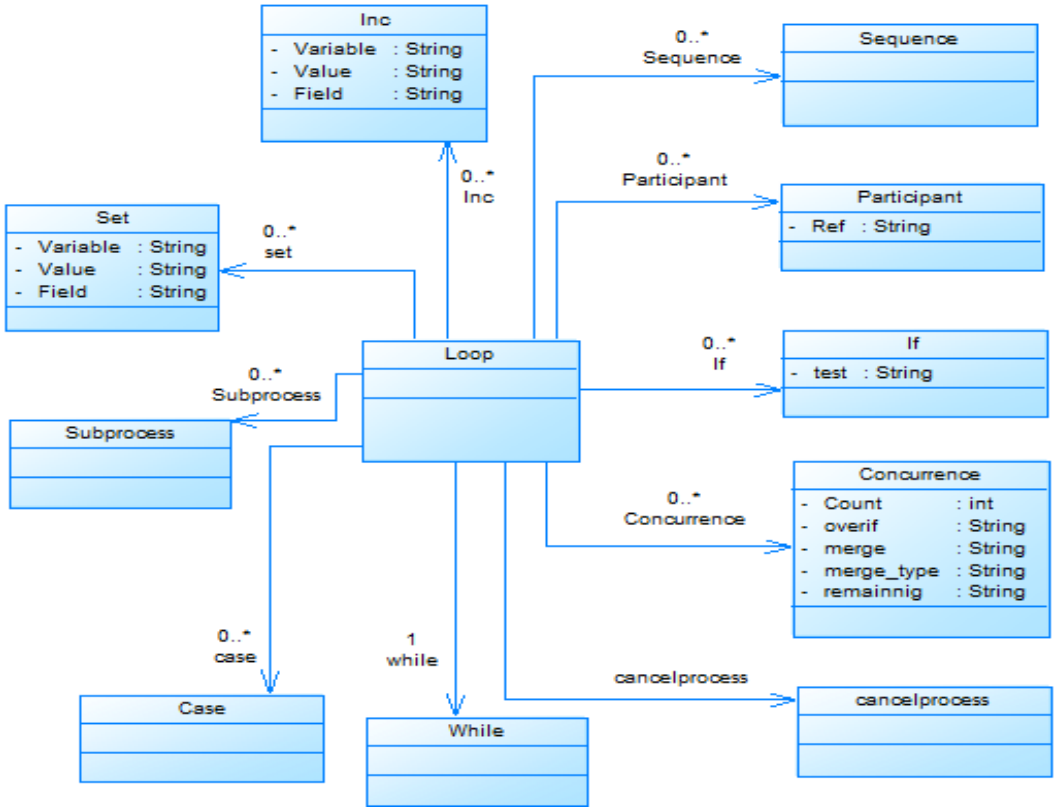


**Figura 6.12** Diagrama de Clases para la clase Loop

Es momento de mencionar otro concepto importante en el metamodelo y se representa con el nombre de la clase "**When**", cuando esta clase se aplica el motor comprueba si la cláusula condicional es verdadera.

En la figura 6.13 se observan las relaciones de esta clase.

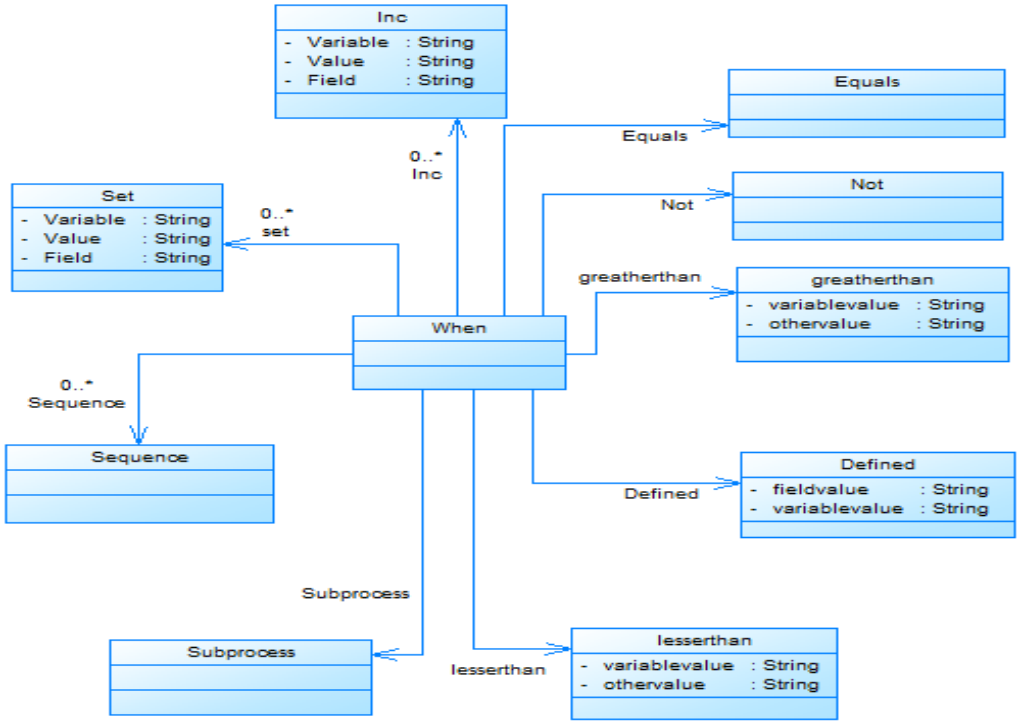


**Figura 6.13** Diagrama de Clases para la clase When

Ahora podemos mencionar otro concepto que puede estar presente en cualquier proceso de negocio, es el definido en nuestro metamodelo en la clase "**While**", esta clase hace que el circuito se realice siempre y cuando la condición dentro del <while> sea evaluada a verdadero. En la figura 6.14 se observan las relaciones de esta clase.

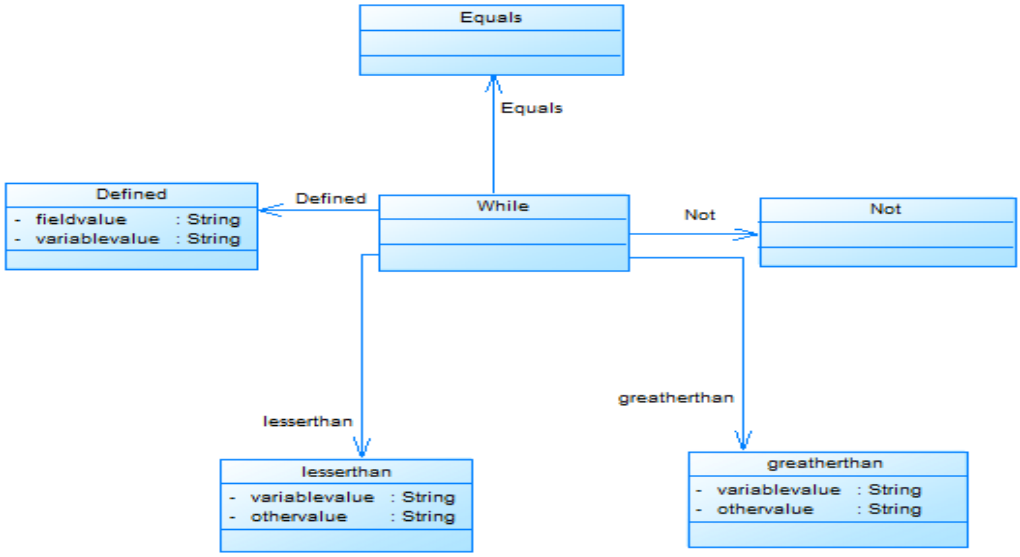


**Figura 6.14** Diagrama de Clases para la clase While

Como ha sido mencionado en capítulos anteriores uno de los conceptos importantes dentro de los procesos de negocio son los subprocesos, donde básicamente se especifica que un proceso puede estar fraccionado en uno o más subprocesos. En el metamodelo que hemos definido el nombre que recibe dicha clase es "**Subprocess**", En la figura 6.15 se observan las relaciones de esta clase.

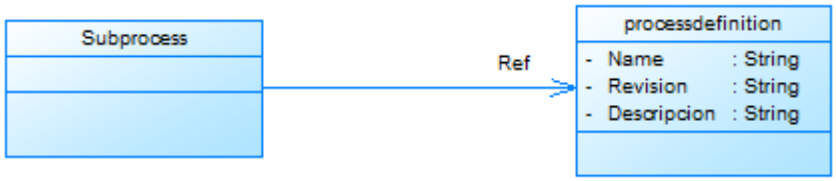


**Figura 6.15** Diagrama de Clases para la clase Subprocess

**6.2.2 Especificación de la estructura del Sistema:** En la figura 6.16 se observa el metamodelo data.ecore definido para realizar la especificación de la estructura del sistema.

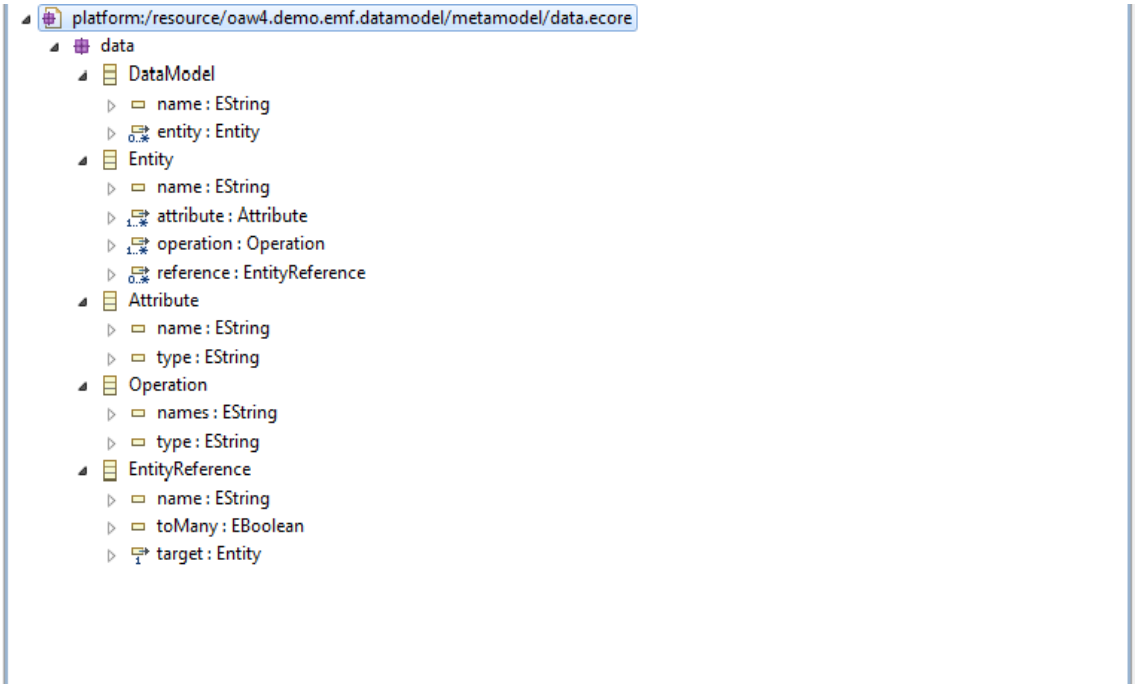


**Figura 6.16** Metamodelo para la especificación de la estructura del sistema.

De acuerdo al metamodelo que se utiliza para la especificación de la estructura tenemos el diagrama de clases que se presenta en la figura 6.17 se presenta el metamodelo de diagrama de clases utilizado en el desarrollo de la propuesta.

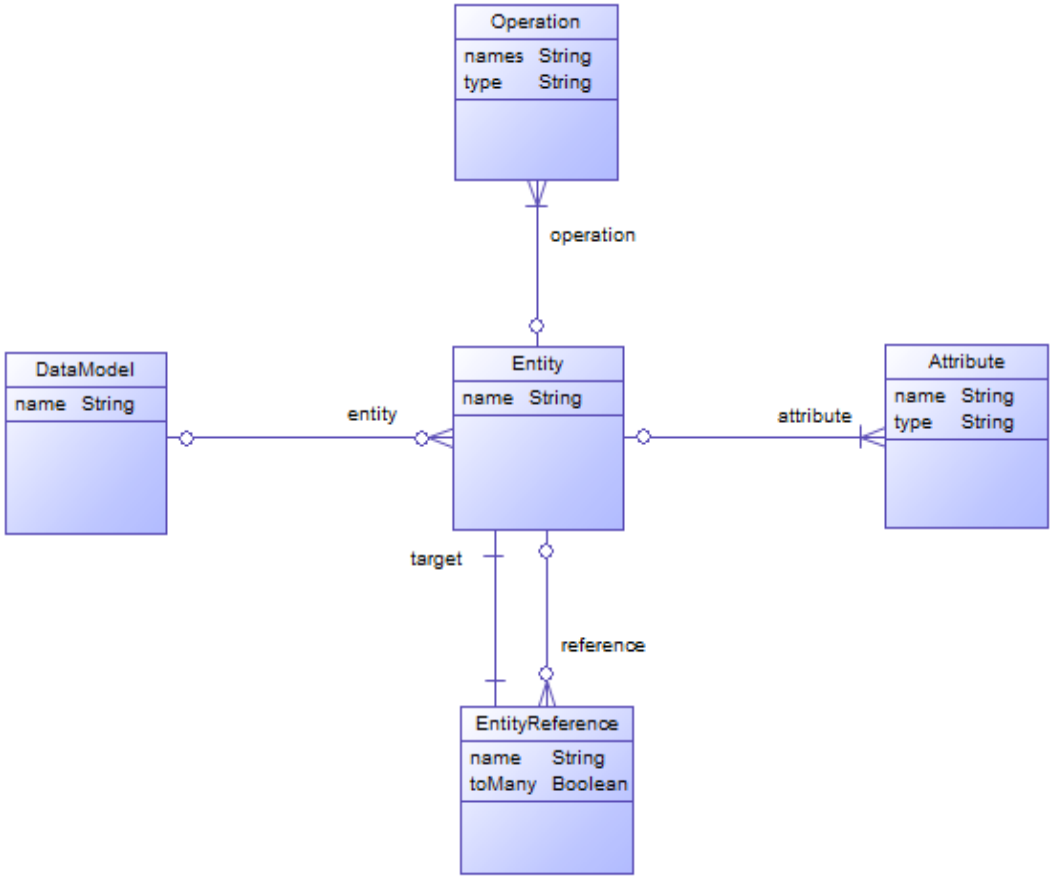


**Figura 6.17** Metamodelo Diagrama de Clases

El metamodelo definido consta de una clase *DataModel* la cual representa el modelo de datos y tiene un atributo name, para poder definir la cantidad de entidades necesarias a contemplar dentro del modelo con el que se desee trabajar se ha definido la clase *Entity* que tiene definido el atributo name, otra de las clases definidas dentro del metamodelo es *Atrribute* la cual nos permite definir cada uno de los respectivos atributos que poseen las clases Entity definidas dentro del modelo, esta clase tiene los atributos name y type, para definir las operaciones a modelar que en este caso estarán relacionadas con los procesos de negocios se ha definido la clase *Operation* que tiene como atributos un names y un type. Por último dentro del metamodelo se encuentra la clase *EntityReference* la cual permite establecer las relaciones entre las distintas clases del modelo.

### 6.3 Proceso de Generación de Código:

En las secciones anteriores hemos definido los metamodelos tanto para la especificación de los procesos de negocio como para especificar la estructura del sistema, el siguiente paso es establecer unas reglas que nos permitan realizar una validación del modelo con el que vamos a trabajar.

#### 6.3.1 Validación del Modelo:

Para esta parte debemos utilizar openArchitectureWare (OAW), en concreto debemos utilizar un fichero llamado chechks.chk el cual incluimos dentro del generador ó proyecto OAW que hemos definido.

Se debe tener presente que el fichero encargado de gestionar el flujo de validación y la generación del código es workflow.oaw, de tal forma que una vez definido el modelo de datos se puede proceder a realizar la validación de las reglas definidas en el archivo chechks.chk.

A continuación se presentan algunas de las reglas definidas en dicho archivo.

| **Regla:** La clase padre debe tener alguna clase hija |
|---|
| **Clases:** DataModel, Entity |
| **context** DataModel **ERROR**<br>"DataModel: entity can't be null":<br>entity.size !=0; |

**Tabla 6.1** Checks.chk La clase padre debe tener alguna clase hija.

| **Regla:** Se deben evitar los nombres de clases sin valor |
|---|
| **Clases:** Entity |
| **context** Entity **ERROR**<br>"Entity: must have a name!":<br>**this**.name!=**null**; |

**Tabla 6.2** Checks.chk Se deben evitar los nombres de clases sin valor.

| **Regla:** Se debe evitar que los nombres de las clases se repitan |
|---|
| **Clases:** DataModel, Entity |
| **context** DataModel **ERROR**<br>"DataModel: Names of Entity must be unique":<br>entity.forAll(b1\|entity.notExists(b2\|b1!=b2&&b1.name.toUpperCase()==<br>b2.name.toUpperCase())); |

**Tabla 6.3** Checks.chk Se debe evitar que los nombres de las clases se repitan.

| **Regla:** Todo nombre de atributo debe tener una longitud mayor a 1 |
|---|
| **Clases:** Attribute |
| **context** Attribute **ERROR**<br>"Names must be more than one char long":<br>**this**.name.length > 1; |

**Tabla 6.4** Checks.chk Todo nombre de atributo debe tener una longitud mayor a 1.

| **Regla:** Toda operación debe tener un nombre |
|---|
| **Clases:** Operation |
| **context** Operation **ERROR**<br>"Operation: must have a names!":<br>**this**.names!=**null**; |

**Tabla 6.5** Checks.chk Toda operación debe tener un nombre.

### 6.3.2 Implementación de Transformaciones:

Para la generación de código utilizamos técnicas de transformación de modelo a texto (M2T), esto se realiza dentro del marco de openArchitectureWare y con el desarrollo de plantillas xpand, dichas plantillas nos permiten recorrer el modelo de datos planteado y generar el respectivo código asociado a la aplicación que deseamos generar.

Las plantillas xpand contienen partes estáticas y dinámicas. Cuando nos referimos a partes estáticas estamos haciendo mención a partes de código que son pasadas automáticamente al código generado, mientras que para las partes dinámicas se suelen recorrer los elementos del modelo de datos obteniendo de esta forma segmentos de código con valores que pertenecen al modelo.

Como resultado final obtenemos los ficheros models.py, views.py y los Html asociados a las vistas que se han definido. Como vimos en la sección 6.1 los ficheros generados se enmarcan dentro de la estructura necesaria para el desarrollo de aplicaciones web con el framework Django.

Los diferentes modelos ó clases del modelo de datos generados estarán en el archivo models.py, la definición de la lógica del negocio generada que viene dada por los procesos del negocio definidos en el modelo de datos estará definida en el archivo

views.py y la presentación de las mismas se encuentra definida en los archivos Html generados.

Veamos a continuación de forma general las transformaciones realizadas:

#### 6.3.2.1 Generación de models.py

La finalidad es obtener la totalidad de las clases definidas en el modelo de datos, las cuales vienen a representar las tablas que se crean en el sistema gestor de bases de datos. Para lograr esto realizamos una inclusión de código estático, el cual tiene que ver con los import que son necesarios dentro del archivo models.py.
Posteriormente procedemos a recorrer cada uno de los atributos definidos en las entidades que se presentan en el modelo y vamos generando el código python correspondiente.

En la tabla 6.6 que se presenta a continuación observamos la respectiva transformación.

```
«DEFINE Entity FOR data::Entity»
«FILE name + ".py"»
from django.db.models  import *
from datetime import datetime
class «name» (models.Model):
«FOREACH attribute AS a»
«REM»
CODIGO PARA CADA GENERAR CADA ATRIBUTO DEFINIDO EN EL MODELO
«ENDREM»
«IF a.type=="String"»
«a.name» = models.CharField(max_length=128, null = True)
«ENDIF»
«IF a.type=="Integer"»
«a.name» = models.IntegerField()
«ENDIF»
«IF a.type=="Date"»
«a.name» = models.DateField()
«ENDIF»
«IF a.type=="Decimal"»
«a.name» = models.DecimalField(max_digits=9, decimal_places=3, null=True)
«ENDIF»
def get_«a.name»(self):
      return self.«a.name»
«ENDFOREACH»
«ENDFILE»
«ENDDEFINE»
```

**Tabla 6.6** Plantilla xpand – Generación de models.py

### 6.3.2.2 Generación de views.py

En esta sección se pretende generar el código referente a la lógica del negocio especificada por los procesos definidos en el modelo de datos. Para lograr esto en primera medida realizamos una inclusión de código estático, el cual tiene que ver con los import que serán necesarios dentro del archivo views.py.
A continuación procedemos a recorrer los procesos definidos en el modelo y vamos generando el código python correspondiente a dichas definiciones.

En la tabla 6.7 que se presenta a continuación observamos las transformaciones xpand que se han implementado.

```
«DEFINE Entity FOR data::Entity»
«FILE name + ".py"»
from django.core.context_processors import csrf
from linkdump.linktracker.models import Link
from django.template import Context, loader
from django.shortcuts import render_to_response
from models import «name»
«FOREACH operation AS a»
«IF a.names=="Consulta"»
«REM»
CODIGO QUE GENERA LAS VISTAS DE TAL FORMA QUE SE IDENTIFICA EL TIPO DE
OPERACIÓN A REALIZAR.
«ENDREM»

def «a.Name»(request, message = ""):
   lista«name» =  «name».objects.all()
   return render_to_response(
      'lista«name».html',
      {'lista«name»': lista«name»,
      'message': message
      }
   )

«ENDIF»
«ENDFOREACH»
«REM»
CODIGO PARA GENERAR LAS VISTAS DE ALTA.
«ENDREM»
«FOREACH operation AS b»
«IF b.names=="Alta"»

def new(request):
   return render_to_response(
      'Alta«name».html',
      {'action': 'add',
      'button': 'Add'}
   )
«ENDIF»
«ENDFOREACH»

def add(request):
«FOREACH attribute AS b»
```

```
«IF b.name!=""»
   «b.name» = request.POST["«b.name»"]
«ENDIF»
«ENDFOREACH»
«name»s = «name»(
«FOREACH attribute AS b»
      «b.name» = «b.name»,
«ENDFOREACH»
   )
   link.save()
   return list(request, message="«name» Agregado!")

«REM»
CODIGO PARA GENERAR LAS VISTAS DE MODIFICACIÓN.
«ENDREM»
«FOREACH operation AS c»
«IF c.names=="Modifica"»

def edit(request, id):
   «name»s= «name».objects.get(id=id)
   return render_to_response(
      'Modifica«name».html',
      {'action': 'update/' + id,
      'button': 'Update',
«FOREACH attribute AS b»
      «b.name» = «name»s.«b.name»,
«ENDFOREACH»
      }
   )
«ENDIF»
«ENDFOREACH»

«REM»
CODIGO PARA GENERAR LAS VISTAS DE ELIMINAR.
«ENDREM»
«FOREACH operation AS d»
«IF d.names=="Elimina"»
def delete(request, id):
   «name».objects.get(id=id).delete()
   return list(request, message="«name» borrado!")
«ENDIF»
«ENDFOREACH»
«ENDFILE»
«ENDDEFINE»
```

**Tabla 6.7** Plantilla xpand – Generación de views.py

#### 6.3.2.3 Generación de plantillas HTML para las vistas definidas

Es el momento de generar las plantillas html encargas de la presentación de los datos en las vistas que se han generado anteriormente.

Para alcanzar este objetivo vamos recorriendo los procesos definidos en el modelo y a su vez generando el código python correspondiente a dichas definiciones.

Se presentan las transformaciones definidas en la plantilla xpand en la tabla 6.8.

```
«REM»
CODIGO PARA GENERAR EL HTML DE LAS VISTAS
«ENDREM»
«FOREACH operation AS d»
«IF d.names=="Consulta"»
{% if message %}
 <b>{{ message }}</b>
 <p>
 {% endif %}
 {% if lista«name» %}
    <table>
    {% for «name» in lista«name» %}
       <tr bgcolor='{% cycle FFFFFF,EEEEEE as rowcolor %}'>
          «FOREACH attribute AS b»
      <td><a href='{{ «name».«b.name» }}'></a></td>

«ENDFOREACH»
       <td><a href='/«name»s/edit/{{ «name».id }}'>Elija «name» a Modificar</a></td>
       <td><a href='/«name»s/delete/{{ «name».id }}'>Eliminar «name» Seleccionado</a></td>
       </tr>
    {% endfor %}
    </table>
    <p>
 {% else %}
    <p>No se han encontrado «name»s.</p>
 {% endif %}
 <p>
 <a href='/«name»s/new'>Adicione un «name»</a>
«ENDIF»
«ENDFOREACH»
«REM»
CODIGO PARA GENERAR EL HTML DE LAS VISTAS
«ENDREM»
«FOREACH operation AS d»
«IF d.names=="Alta"»
<form action="/«name»s/{{ action }}/" method="post">
   «FOREACH attribute AS b»
   «b.name»:
   <input name=«b.name» value="{{ «b.name» }}"><br />
   «ENDFOREACH»
   <input type=submit value="{{ button }}">
</form>
«ENDIF»
«ENDFOREACH»
```

**Tabla 6.8** Plantilla xpand – Generación de plantillas HTML

En este capítulo se ha presentado el desarrollo de la propuesta planteada utilizando las tecnologías y herramientas definidas anteriormente.

# 7. Caso de estudio

En este capitulo se presenta el desarrollo de un caso de estudio referente a la Administración de un negocio de ventas de productos. En primer lugar describiremos brevemente el caso de estudio, luego veremos los conceptos que deseamos modelar expresados en notación de modelado de procesos de negocio (BPMN), a continuación analizaremos el resultado de aplicar las transformaciones al modelo y por último veremos la aplicación que se ha generado.

## 7.1 Descripción del negocio de ventas

El negocio de ventas es de productos electrónicos y ofrece las funcionalidades referentes a la "administración del negocio", realizando funciones como consultar la existencia de un producto, agregar productos al stock, modificar características de productos existentes, realizar una compra de inventario, gestionar la venta de productos y la respectiva factura asociada, creación y gestión de usuarios del sistema.

## 7.2 Modelado del caso de estudio

La idea es identificar los conceptos a modelar mediante el uso de BPMN, esto se presenta en las figuras 7.1 y 7.2.

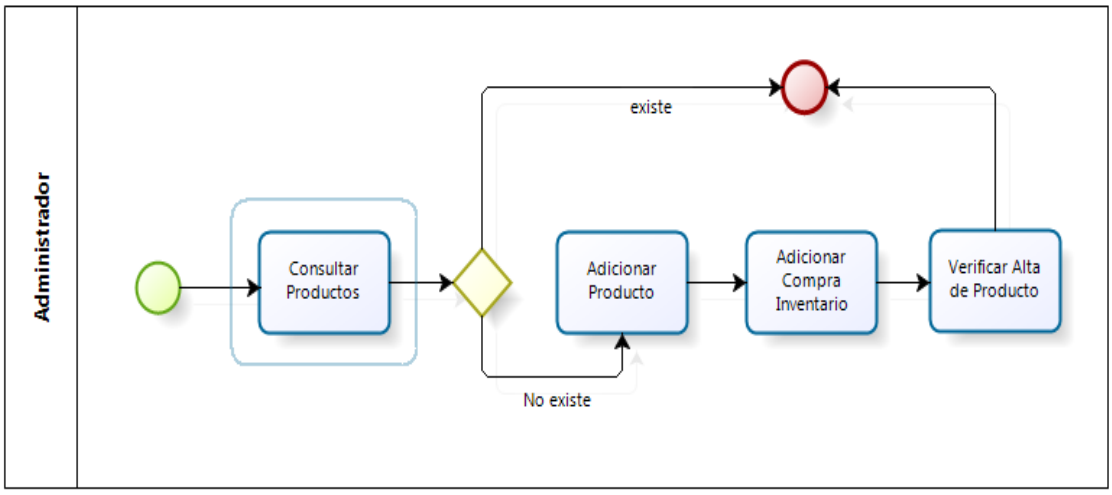


**Figura 7.1** Diagrama BPMN para la gestión de alta de productos.

Tal como se ha presentado en el diagrama anterior referente a la gestión de alta de productos, el administrador como persona encargada de dicho proceso consulta en el sistema la existencia de un producto determinado, si dicho producto no se encuentra dentro de los ofrecidos por el almacén procede a adicionar dicho producto con las características asociadas al mismo, luego procede a adicionar la compra de inventario de dicho producto y por último verifica que en el sistema se encuentran unidades disponibles en el stock del almacén de dicho artículo (disponible para la venta).

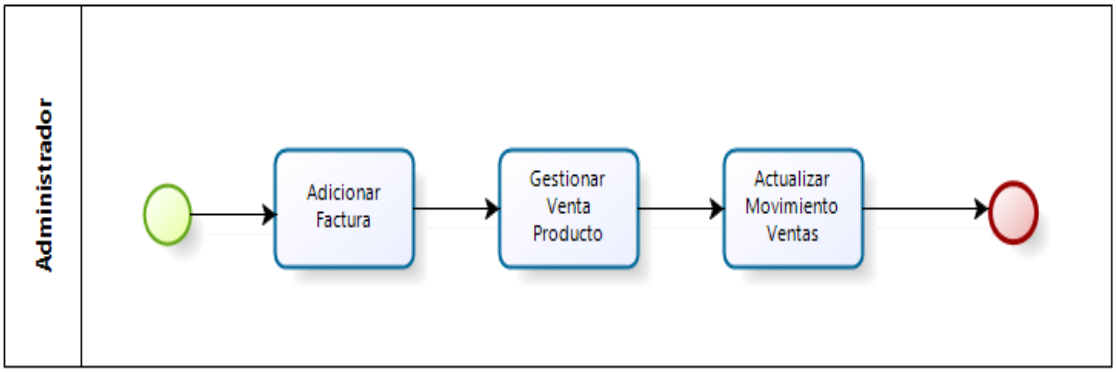


**Figura 7.2** Diagrama BPMN para la gestión de facturación.

En la figura anterior el proceso se inicia cuando se ha realizado una venta de forma presencial, el cliente va a la tienda y retira el producto, luego el administrador del sistema procede a adicionar una factura, gestionar la venta del producto y realizar una actualización en el movimiento de ventas de productos.

### 7.3 Creación del modelo de datos

En la sección anterior hemos visto los conceptos a modelar e identificado los procesos u operaciones que se desean realizar.

En la figura 7.3 se presenta el modelo de datos definido, analizando la entidad producto y los procesos que pueden ser realizados dentro del marco de la misma observamos que: es posible consultar existencia de un producto, dar de alta un producto, realizar una modificación del mismo ó eliminar el producto en caso de que el almacén decida no continuar ofreciéndolo a los clientes.

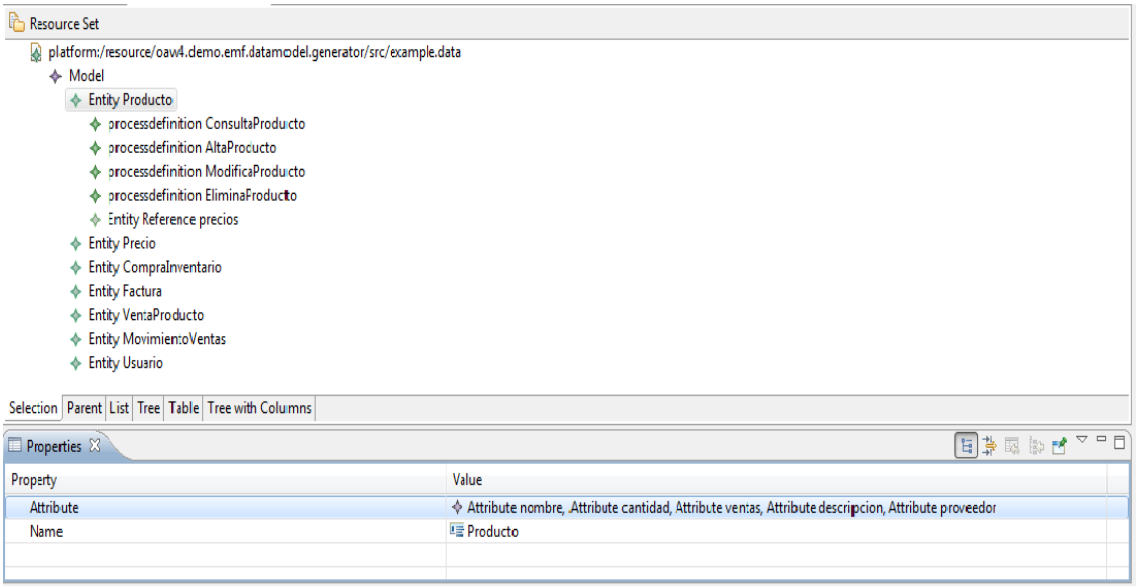


**Figura 7.3** Modelo de datos productos

## 7.4 Transformaciones realizadas

Una vez se ha realizado el modelo de datos con el nombre example.data y la respectiva validación del modelo con el fichero checks.chk, se aplican las plantillas xpand y se obtienen los ficheros models.py, views.py y los respectivos HTML.

### 7.4.1 Archivo models.py

Tras aplicar la plantilla xpand se han obtenido los modelos de datos que como se ha explicado en la sección 6.3.2.1 se corresponden con las tablas que se crean en el sistema gestor de bases de datos que hemos especificado en el fichero settings.py dentro de la aplicación con el framework django, en este caso se ha trabajado con MySQL.

En la tabla 7.1 se presenta el archivo models.py para algunos de los modelos generados, como es el caso de los Productos.

```
class Producto(models.Model):
    """Productos a la venta"""
    nombre   = models.CharField(max_length=128, null = True)
    cantidad = models.IntegerField()
    ventas   = models.IntegerField()
    descripcion = models.CharField(max_length=128, null = True)
    proveedor = models.CharField(max_length=128, null = True)
```

```
    def get_nombre(self):
        return self.nombre

    def get_descripcion(self):
        return self.descripcion

    def get_proveedor(self):
        return self.proveedor

class Precio(models.Model):
    """Precio de un producto en un determinado momento"""
    producto = models.ForeignKey(Producto)
    valor    = models.DecimalField(max_digits=9, decimal_places=3, null=true)
    fecha    = models.DateField('fecha de validez', unique = True)
    def get_producto(self):
        return self.producto

    def get_valor(self):
        return self.valor

    def get_fecha(self):
        return self.fecha

class CompraInventario(models.Model):
    """Representa la compra de inventario de un producto en especifico"""
    producto = models.ForeignKey(Producto)
    cantidad = models.IntegerField()
    fecha    = models.DateField('fecha de compra', unique = True)
    costo    = models.DecimalField(max_digits=9, decimal_places=3, null=True)

    def get_producto(self):
        return self.producto

    def get_cantidad(self):
        return self.cantidad

    def get_fecha(self):
        return self.fecha
```

```
    def get_costo(self):
        return self.costo
```

**Tabla 7.1** Sección del fichero models.py

### 7.4.2 Archivo views.py

En la sección 6.3.2.2 se ha explicado que las vistas en el framework django corresponden a la lógica del negocio, aplicando las plantillas xpand se ha obtenido el código para dichas vistas.

En la tabla 7.2 se presenta el archivo views.py para las vistas producto.

```
from django.core.context_processors import csrf
from linkdump.linktracker.models import Link
from django.template import Context, loader
from django.shortcuts import render_to_response
from models import Producto


def ConsultaProducto(request, message = ""):
    listaProducto =  Producto.objects.all()
    return render_to_response(
        'listaProducto.html',
        {'listaProducto': listaProducto,
        'message': message
        }
    )
def new(request):
    return render_to_response(
        'AltaProducto.html',
        {'action': 'add',
        'button': 'Add'}
    )

def add(request):

    nombre = request.POST["nombre"]
    cantidad = request.POST["cantidad"]
    ventas = request.POST["ventas"]
    descripcion = request.POST["descripcion"]
    proveedor = request.POST["proveedor"]
Productos = Producto(
        nombre = nombre,
        cantidad = cantidad,
        ventas = ventas,
        descripcion = descripcion,
        proveedor = proveedor,
    )
    link.save()
    return list(request, message="Producto Agregado!")
```

**Tabla 7.2** Sección del fichero views.py (Producto)

### 7.4.3 Plantillas HTML

Como se ha mencionado en la sección 6.3.2.3 estas plantillas se encargan de la presentación de los datos.

En la tabla 7.3 se presenta el código generado para el html referente la Producto.

```
{% if message %}
 <b>{{ message }}</b>
 <p>
 {% endif %}
 {% if listaProducto %}
    <table>
    {% for Producto in listaProducto %}
       <tr bgcolor='{% cycle FFFFFF,EEEEEE as rowcolor %}'>
      <td><a href='{{ Producto.nombre }}'></a></td>
      <td><a href='{{ Producto.cantidad }}'></a></td>
      <td><a href='{{ Producto.ventas }}'></a></td>
      <td><a href='{{ Producto.descripcion }}'></a></td>
      <td><a href='{{ Producto.proveedor }}'></a></td>
       <td><a href='/Productos/edit/{{ Producto.id }}'>Escoja Producto a Modificar</a></td>
       <td><a href='/Productos/delete/{{ Producto.id }}'>Eliminar Producto Seleccionado</a></td>
       </tr>
    {% endfor %}
    </table>
    <p>
 {% else %}
    <p>No se han encontrado Productos.</p>
 {% endif %}
 <p>
 <a href='/Productos/new'>Añadir Producto</a>
```

**Tabla 7.3** HTML para Producto.

## 7.5 Vistas de la aplicación generada

A continuación se presentan las vistas que el administrador del sistema de ventas manipula para llevar a cabo el proceso de negocios definido utilizando BPMN en la figura 7.1.

En primera medida el administrador desea saber si el producto Cámara Canon se encuentra disponible en el stock del almacén, esto se presenta en la figura 7.4.


**Figura 7.4** Consulta de Existencia del producto

En el momento en que el administrador realiza la consulta de productos y observa que el artículo Cámara Canon no se encuentra en stock debe añadir el producto en la vista correspondiente, esta funcionalidad se presenta en la figura 7.5.

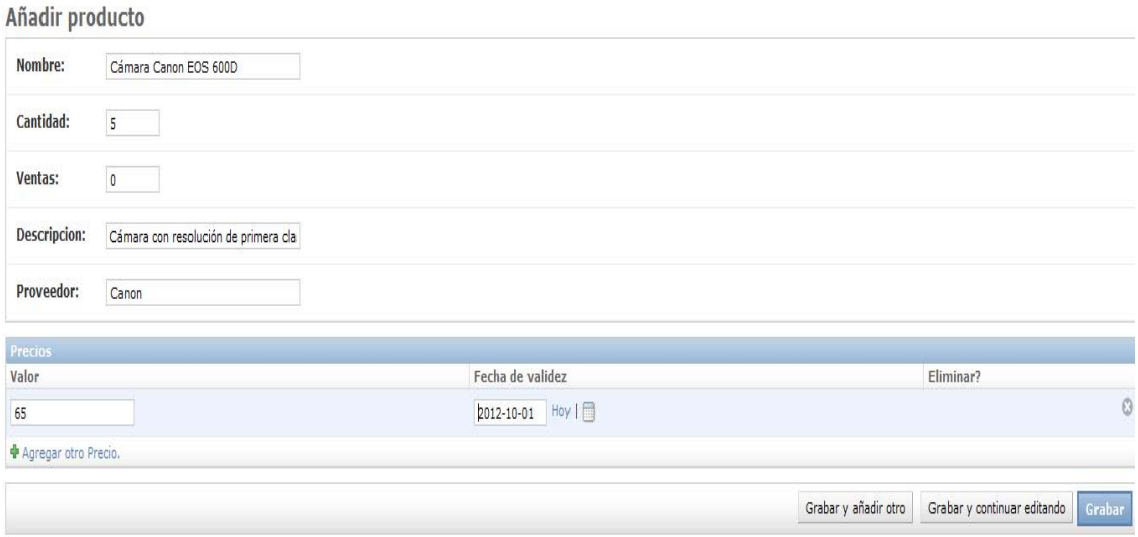


**Figura 7.5** Alta de Producto.

A continuación el administrador adiciona una compra de inventario referente a cámaras canon, seleccionando una cantidad a comprar, la fecha en que se realiza la compra y el costo del movimiento, esta vista se puede observar en la figura 7.6.

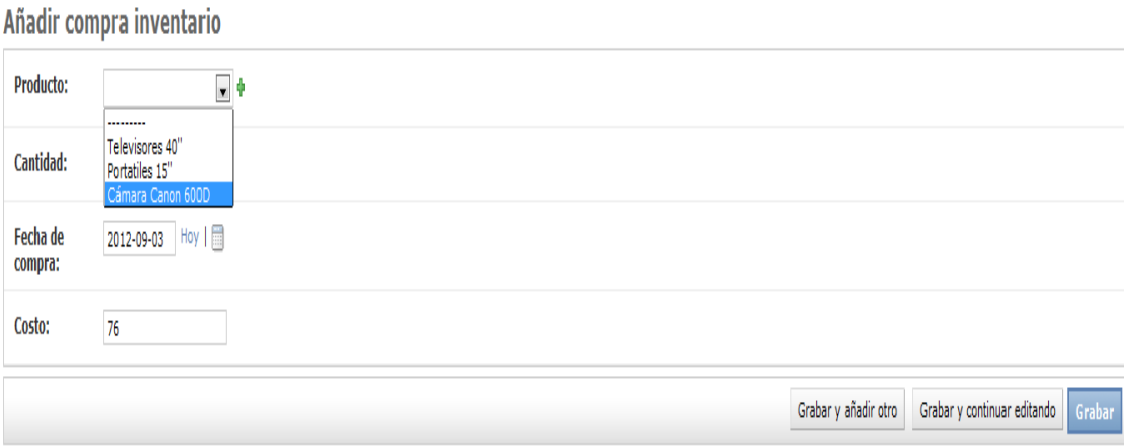


**Figura 7.6** Alta compra de inventario.

Para finalizar el proceso gestión de alta de productos presentado en la figura 7.1 el administrador verifica que el producto cámara canon 600D se ha adicionado de forma correcta al stock de artículos en el almacén, lo cual da por finalizado el proceso. Esta comprobación se presenta en la figura 7.7.


**Figura 7.7** Verificación del producto adicionado.

# 8. Conclusiones

Definitivamente las tecnologías de información juegan un papel cada vez más importante en facilitar la introducción de productos nuevos o de servicios, en mejorar procesos operacionales, y en la guía de la toma de decisiones directivas.

Todos los negocios se transforman por el uso de la tecnología, y la competitividad que pueda generar dicho ajuste dependerá del nivel de conexión o alineación entre la tecnología y los requerimientos reales del negocio; es decir, del nivel de asociación de la tecnología con un planteamiento de cambio.

De ahí la importancia que tiene el establecer los procesos de negocio que determinarán la manera en que la empresa opera, de otro modo, sólo se hace más complejo el panorama al tratar de implementar primeramente una tecnología de información, y después definir los procesos de la empresa. Esto solamente derivará en altos costos de inversión en diversas tecnologías de información dentro del marco de la compañía.

Como sabemos, las tecnologías de información no determinan los procesos de negocio de una compañía, pero son habilitadores para hacer mucho más eficiente la manera en que estos se llevarán a cabo dentro de la misma organización y sus diferentes áreas. Las organizaciones se apoyan en sus procesos de negocios para ser guiados en este complejo escenario.

Teniendo presente que el enlace entre procesos de negocio y generación de valor lleva a una gran mayoría de participantes a ver los procesos de negocio como los flujos de trabajo que se efectúan dentro de las tareas de una organización, podemos pensar que los procesos de negocio pueden ser vistos como una fórmula para hacer funcionar un negocio y alcanzar las metas definidas en la estrategia de negocio de la empresa.

Es muy importante tener claro que para que el análisis de procesos de negocios tenga éxito, este debe contener toda una estructura de lo que sucede realmente en los procesos, con el fin de poder determinar las oportunidades que tiene la empresa para mejorar, y emprender la mejor alineación hacia la cadena de valor posible.

Cuando logremos alinear estos procesos de negocio hacia la meta planteada, es cuando podemos hablar de la capacidad de una compañía de rediseñar continuamente su cadena de valor y modificar sus activos humanos, estructurales, financieros y tecnológicos, para lograr una ventaja competitiva respecto a sus competidores, siendo ésta la mayor razón por la que un proceso de negocio se vuelve la parte más fundamental en la que se basa una compañía.

De nada nos servirá la administración computarizada de una empresa mediante las tecnologías de información, si antes no tenemos bien definidos correctamente los procesos de negocio, en los cuales la compañía sustenta su correcto funcionamiento.

La Arquitectura Dirigida por Modelos (**MDA**) rompe con el concepto de producción de software artesanal, automatizando las tareas de diseño, desarrollo y en buena parte despliegue de aplicaciones. Gracias a esta automatización, la ingeniería del software puede participar de las ventajas de que gozan otras ingenierías al usar herramientas de soporte a su disciplina.
Centrando la atención sobre la aplicación de MDA a los procesos de negocio, en el momento de desarrollo de este documento, no existe ningún soporte específico para los mismos, pero mediante la propuesta planteada, podemos observar como a partir de metamodelos de procesos de negocio que definimos, se pueden utilizar técnicas de MDA propias en forma de plantillas y reglas de transformación.

La transformación de modelos es el mecanismo central que promueve MDA, sin embargo, no es exclusivo de MDA. Teniendo presente que se abarcan temas que van más allá de MDA como son los lenguajes de dominio específico o lenguajes de transformación.

Actualmente los modelos son costosos de construir y una vez construido el modelo este debe ser transformado manualmente en código. Esta tarea es tediosa, propensa a errores y repetitiva en muchos casos, por ello la arquitectura dirigida por modelos (MDA) contribuye a disminuir las cargas ó el trabajo que se presenta en el desarrollo de estas tareas.

Podemos asegurar que MDA es el resultado de reconocer que la interoperabilidad es algo bueno y que el modelado también lo es, aunque MDA no ofrezca la solución a todas las problemáticas inherentes al desarrollo de software.

# 9. Referencias


[Adrian 2008] Adrian, Holovaty y Jacob, Kaplan-Moss. Django Book. Autoedición, 2008.

[Atkinson 2003] Atkinson, Colin and Kuhne, Thomas. Model-driven development: A metamodeling foundation. En: IEEE Software, vol. 20, no 5, 2003.

[Bézivin 2004] Bézivin, Jean. In search of a basic principle for model driven engineering. En: Upgrade, vol. V, no. 2, ISSN 1684-5285, 2004.

[Booch 2004] Booch, G. Brown, A. and Rumbaugh, J. An MDA manifesto. IBM Rational Software, 2004.

[Budinsky 2003] Budinsky, Frank. Steinberg, David. Merks, Ed. Ellersick, Raymond. J. Grose, Timothy. Eclipse Modeling Framework: A Developer's Guide. Addison-Wesley, ISBN: 0-13-142542-0, 2003.

Catalog of Business Modeling and Management Specifications http://www.omg.org/technology/documents/br_pm_spec_catalog.htm

[Cockburn 1997] Cockburn, A.: Using Goal-Based Use Cases. JOOP, Vol. 10, No. 7, 1997.

Eclipse Platform Technical Overview. Object Technology International, Inc. http://www.eclipse.org/whitepapers/eclipse-overview.pdf. EclipseOverview (2003).

[Jacobson 1997] Jacobson, Ivar. Griss, Martin and Jonsson, Patrik. Software reuse: Arquitecture, process and organization for business success. Addison Wesley, 1997.

[Klasse, 2004] Klasse Objecten MDA, 2004

[Kleppe 2005] Anneke Kleppe, Jos Warmer y Win Bast. MDA Explained – The Model Driven Architecture: Practice And Promise. Addison-Wesley, 2005

MDA. Model Driven Architecture http://www.omg.org/mda/

[Mellor 2003] Mellor, Stephen J. Clark, Anthony N. and Futagami, Takao. Model-driven development. En: IEEE Software, vol. 20, no 5, 2003.

[Mellor 2004] Stephen J. Mellor, Kendall Scott, Axel Uhl y Dirk Weise. MDA Distilled – Principles of Model-Driven Architecture. Addison-Wesley, 2004.

[Mettraux 2006] John Mettraux. OpenWfe Open source WorkFlow Environment, 2006.

[OMG 2006] OMG. Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification, 2006.

[OMG 2011] OMG. Business Process Model and Notation (BPMN) version 2.0. OMG Final Adopted Specification. Formal, 2011.

[P, Wohed 2006] P, Wohed. W.M.P, van der Aalst. M. Dumas, A.H.M, ter Hofstede and N, Russell. On the Suitability of BPMN for Business Process Modelling. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, Business Process Management, volume 4102 of Lecture Notes in Computer Science, 2006.

[S, Jablonski 1996] S, Jablonski and C, Bussler. Workflow Management: Modeling Concepts, Architecture and Implementation. Thomson Computer Press, London, 1996.

[S, Mellor 2004] S, Mellor. S, Kendall. A, Uhl. D, Weise. MDA Distilled, Addison-Wesley, 2004.

[Stuart 2002] Kent Stuart. Model Driven Engineering. Springer, 2002.

[Sven, Efftinge 2008] Sven, Efftinge. Peter, Friese. Arno, Haase. Dennis, Hübner. Clemens, Kadura. Bernd, Kolb. Jan, Köhnlein. Dieter, Moroff. Karsten, Thoms.

Markus, Völter. Patrick, Schönbach and Moritz, Eysholdt. OpenArchitectureWare User Guide Versión 4.3, 2008.

[Völter 2005] Völter, M. Software architecture patterns - a pattern language for building sustainable software architectures, 2005.