UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

DEFINIDOR VISUAL BAJO ECLIPSE EUROPA

Autora: Mónica Burcio Sánchez
Tutora: Pilar Aránzazu Herráez López

Julio 2009

AGRADECIMIENTOS

A Oscar, por TODO.

A mi famila, por su apoyo incondicional, y porque sin ellos no estaría donde estoy ni sería lo que soy.

A mis amigos, por formar parte de mi vida, y poder contar con ellos siempre.

A mi tutora, Pilar Aránzazu Herráez López, por la oportunidad que me ha brindado para realizar este proyecto, cuya importancia ha sido vital en el desarrollo del mismo.

A todos vosotros, GRACIAS.

INDICE

INDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. PROPÓSITO DEL DOCUMENTO

El propósito de este documento, es que de la lectura de la memoria se obtenga claramente sin necesidad de consultar otros documentos una idea concreta de lo que el proyecto representa. Para ello se expone la información suficiente para proporcionar un conocimiento completo del proyecto desarrollado, la forma en que se ha llevado a cabo, la cuantía de la inversión y todo lo relacionado con su realización.

La extensión del documento de la memoria debe ser tal que su lectura sea clara, concisa, directa y completa.

1.2. DEFINICIONES Y ACRÓNIMOS

1.2.1. Definiciones

TÉRMINO	SIGNIFICADO
Applets	Un applet Java es un applet escrito en el lenguaje de programación Java. Los applets de Java pueden correr en un navegador web utilizando la Java Virtual Machine (JVM), o en el AppletViewer de Sun.
Byte	Unidad fundamental de datos en los ordenadores personales, un byte son ocho bits contiguos. El byte es también la unidad de medida básica para memoria, almacenando el equivalente a un carácter.
Bytecode	Código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina .
Drag and drop	Técnica basada en arrastrar con el ratón del ordenador.
Eclipse	Entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido".
Framework	Estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.
Interfaz	Conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario.
Java Beans	Modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.
JDK	El paquete JDK contiene el entorno de desarrollo de Java de Sun. Sirve para desarrollar programas Java y proporciona el entorno de ejecución necesario para ejecutar dichos programas. También incluye un módulo (plugin) para que los navegadores puedan ejecutar programas Java.
Jface	Juego de herramientas construida sobre SWT.

Metamodelo	El metamodelo del lenguaje es un "mapa". Ayuda al comunicador a reconocer y explicitar la manera en que distintas personas utilizan su lenguaje verbal para representar el mundo en el que viven.
Open Source	Término con el que se conoce al software distribuido y desarrollado libremente.
Plug-in	Aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.
Servicio Web	Conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
Servlets	Objeto que se ejecuta en un servidor o contenedor JEE, especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML.
Sun Microsystems	Empresa informática de Silicon Valley, fabricante de semiconductores y software.
.Net	Proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permite un rápido desarrollo de aplicaciones.

1.2.2. Acrónimos

ACRÓNIMO	SIGNIFICADO
ATL	ATLAS Transformation Language
BPD	Business Process Diagram
BPEL4WS	Business Process Execution Language for Web Services
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
BPMS	Business Process Management System
CASE	Ingeniería de Software Asistida por Ordenador
CGI	Common Gateway Interface
CIM	Modelo independiente de la computación
CORBA	Common Object Request Broker Architecture
CORBA IDL	CORBA Interface Definition Language
CVS	Concurrent Versions System

CWM	Common Warehouse Metamodel
DSL	Lenguaje específico de dominio
DTD	Document Type Definitions
EJB	Enterprise JavaBeans
EMF	Eclipse Modelling Framework
EMFT-OCL	EMF Technologies OCL
GEF	Graphical Eclipse Framework
GMF	Graphical Modelling Framework
GNU GPL	Licencia Pública General de GNU
GUI	Interfaz gráfica para el usuario
HTML	HyperText Markup Language
IDE	Entorno de desarrollo integrado
IDL	Interface description language
J2EE	Java 2 Platform
JCP	Java Community Process
JDK	Java Development Kit
JIV	Java Image Viewer
JLS	Java Language Specification
JMI	Interface Metadata de JAVA
JRE	Java Runtime Environment
JSP	Java Server Pages
JSRs	Java Specification Requests
JVM	Java Virtual Machine
MDA	Model Driven Architecture
MDD	Model-driven development
MDE	Model-Driven Engineering
MIC	Model Integration Computing
MOF	Meta-Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
OSGi	Open Services Gateway Initiative
OTI	Object Technology International
PDE	Plug-in Development Environment
PGML	Precision Graphics Markup Language
PIM	Modelo independiente de la plataforma
PSM	Modelo específico de la plataforma
QVT	Query-View-Transformation
RBS	Resource Breakdown Structure
RCP	Rich Client Platform
SVG	Scalable Vector Graphics
SWT	Standard Widget Toolkit
UML	Unifique Model Lenguage
XMI	Metadata Interchange
XML	Extensive Markup Language
XSD	XML Schema Definition

1.3. REFERENCIAS

- [1] A lo largo del documento se hace referencia a la norma *BPMN* en su versión 1.0. Para ampliar información al respecto visite la página oficial de la norma, alojada en la siguiente dirección: http://www.bpmn.org/

- [2] OMG, Business Process Modeling Notation Specification, OMG Final Version Adopted, Febrero 2006 disponible en: http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf

2. DESCRIPCIÓN GENERAL

2.1. MOTIVACIÓN DEL PROYECTO

La idea de la realización de este proyecto, surge con el propósito de ampliar conocimientos sobre el concepto de la ingeniería dirigida por modelos, ya que me parecía un tema interesante y que está muy de moda en la actualidad. Además, durante la carrera había oído hablar muy por encima de ello, y me apetecía investigar y conocer más cosas sobre dicho concepto.

En cuanto al entorno de desarrollo utilizado, también había oído hablar bastante, puesto que está muy presente en el entorno de programación en java. Pero nunca había tenido la oportunidad de utilizarla, y mucho menos en el ámbito que dicha tecnología ofrece para el desarrollo de proyectos basados en la ingeniería dirigida por modelos, a través de los pulg-ins EMF y GMF.

También me pareció interesante, la utilización de la norma BPMN, ya que es un estándar que no conocía.

En cuanto a la motivación personal debo decir que desde el principio me pareció un tema interesante y que me iba a aportar muchos conocimientos nuevos, puesto que iba a realizar un proyecto en un entorno completamente desconocido para mí.

2.2. OBJETIVOS DEL PROYECTO

El objetivo principal del proyecto es desarrollar una herramienta basada en la norma BPMN para el modelado gráfico de procedimientos de negocio.

La implementación de la herramienta se realiza bajo el entorno de desarrollo Eclipse Europa, y la arquitectura dirigida por modelos.

Para llevar a cabo el modelado gráfico de los procedimientos de negocio, se dispone de una paleta que contiene todos los elementos a modelar. Desde dicha paleta y mediante la técnica "drag and drop", se pueden situar dichos elementos en una pizarra, y acceder a la información de los mismos mediante una ventana de propiedades, al hacer clic sobre ellos.

La herramienta, ofrece la posibilidad de generar un fichero en formato XML compatible con la norma BPMN, con la información de los elementos modelados.

2.3. CONTEXTO DEL PROYECTO

Para llevar a cabo el desarrollo de la aplicación cumpliendo con todos los objetivos expuestos en el apartado anterior, se ha considerado que lo más acertado es utilizar los siguientes componentes:

- Ingeniería dirigida por modelos: Porque fortalece el desarrollo orientado a objetos al incorporar herramientas que generan código a partir de modelos de plataformas específicas. Además incentiva el uso de modelos en el proceso de desarrollo del software, mejorando así la calidad del software, la portabilidad, la interoperabilidad y la reusabilidad.

- Entorno de desarrollo ECLIPSE EUROPA: Porque permite desarrollar proyectos en el entorno de la ingeniería dirigida por modelos, y en la actualidad es una de las plataformas más utilizadas en este ámbito. Es un entorno fácil de extender con distintos plug-ins.

- Librería de EMF como plataforma de Modelado Eclipse: Porque es el plug-in de Eclipse que permite definir meta-modelos fácilmente.

- Librería de GMF como herramienta de Modelado Gráfico: Porque es el plug-in de eclipse que permite crear editores gráficos de modelos a partir de meta-modelos EMF.

- Lenguaje OCL: Porque es el lenguaje que se utiliza en el entorno Eclipse para especificar las restricciones de los modelos generados.

- Lenguaje de programación JAVA: es el lenguaje de programación que se utiliza en el entorno Eclipse para el desarrollo de aplicaciones.

- Notación para el Modelado de Procesos de Negocio BPMN 1.0: Porque es un estándar de modelado de procesos de negocio, formado por elementos gráficos que habilitan el fácil desarrollo de diagramas simples, que son familiares para la mayoría de analistas de negocio.

3. ESTADO DEL ARTE

3.1. INTRODUCCIÓN

En este capítulo se van a exponer los conceptos teóricos que se han tenido en cuenta para la realización del proyecto.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Ingeniería dirigida por modelos.
- Notación para el modelado de procesos de negocio (BPMN 1.0).
- Lenguaje de programación Java.
- Entorno de desarrollo Eclipse Europa.
- Eclipse Modelling Framework (EMF).
- Graphical Modelling Framework (GMF).
- Restricciones OCL.
- Comparación con otras herramientas.

3.2. INGENIERÍA DIRIGIDA POR MODELOS

Según el diccionario de la Real Academia de la Lengua Española, un modelo es un "esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento".

El término Ingeniería Dirigida por Modelos (MDE) hace referencia a un conjunto de técnicas que hacen un uso sistemático y reiterado de modelos a lo largo de todo el ciclo de vida de desarrollo del software. El uso de modelos permite aumentar el nivel de abstracción con que se realizan los diseños, así como el nivel de reutilización de los mismos. Además, la utilización del enfoque MDE facilita la comunicación de ideas, ya que éstas se pueden expresar de manera explícita (por lo general utilizando una notación gráfica asociada a los conceptos que se modelan), y no diluidas entre interminables líneas de código.

Si bien es cierto que el desarrollo dirigido por modelos data ya de hace algunos años, su empleo no ha sido posible hasta que se han desarrollado las primeras herramientas que proporcionan el soporte necesario para su aplicación. En este punto, ha desempeñado un papel destacado el OMG (Object Management Group) , desarrollando un amplio conjunto de herramientas y estándares para su MDA (Model Driven Architecture). Entre ellos, cabe mencionar: MOF (Meta-Object Facility), XML (Extensive Markup Language), OCL (Object Constraint Language).

Actualmente, la plataforma Eclipse es una de las más utilizadas por la comunidad MDE. Dicha plataforma para llevar a cabo cada una de las fases del desarrollo de software dirigido por modelos, dispone de una serie de herramientas (plug-ins) relacionadas con MDE. Entre otras, destacan las siguientes herramientas:

- Eclipse Modelling Framework (EMF), implementación de MOF.
- Graphical Modelling Framework (GMF), plug-in para el desarrollo de editores gráficos de modelos.

- EMF Technologies OCL (EMFT-OCL), herramienta que permite definir y validar restricciones OCL sobre los modelos definidos.

3.2.1. Introducción a MDA

La arquitectura dirigida por modelos es una especificación detallada por el OMG que integra diferentes especificaciones y estándares definidos por la misma organización con la finalidad de ofrecer una solución a los problemas relacionados con los cambios en los modelos de negocio, la tecnología y la adaptación de los sistemas de información a los mismos.

MDA nos permite el despliegue de aplicaciones empresariales, diseñadas sin dependencias de plataforma de despliegue y expresado su diseño mediante el uso de UML (Unifique Model Lenguaje) y otros estándares, potencialmente en cualquier plataforma existente, abierta o propietaria, como servicios web, .Net, Corba, J2EE, u otras.

La especificación de las aplicaciones y la funcionalidad de las mismas se expresa en un modelo independiente de la plataforma que permite una abstracción de las características técnicas específicas de las plataformas de despliegue. Mediante transformaciones y trazas aplicadas sobre el modelo independiente de la plataforma se consigue la generación automática de código específico para la plataforma de despliegue elegida, lo que proporciona finalmente una independencia entre la capa de negocio, y la tecnología empleada. De esta manera es mucho más simple la incorporación de nuevas funcionalidades, o cambios en los procedimientos de negocio sin tener que llevar a cabo los cambios en todos los niveles del proyecto. Simplemente se desarrollan los cambios en el modelo independiente de la plataforma, y éstos se propagarán a la aplicación, consiguiendo por tanto una considerable reducción del esfuerzo en el equipo de desarrollo, en los errores que tienden a producirse en los cambios introducidos en las aplicaciones mediante otros métodos de desarrollo, y por consiguiente, la reducción de costes y aumento de productividad que conlleva, tan demandados tanto la industria de desarrollo de software como el resto de las empresas.

MDA se apoya sobre los siguientes estándares para llevar a cabo su función:

- UML: empleado para la definición de los modelos independientes de la plataforma y los modelos específicos de las plataformas de destino. Es un estándar para el modelado introducido por el OMG.

- MOF: establece un marco común de trabajo para las especificaciones del OMG, a la vez que provee de un repositorio de modelos y metamodelos.

- XMI (Metadata Interchange): define una traza que permite transformar modelos UML en XML para poder ser tratados automáticamente por otras aplicaciones.

- CWM (Common Warehouse Metamodel): define la transformación de los modelos de datos en el modelo de negocio a los esquemas de base de datos.

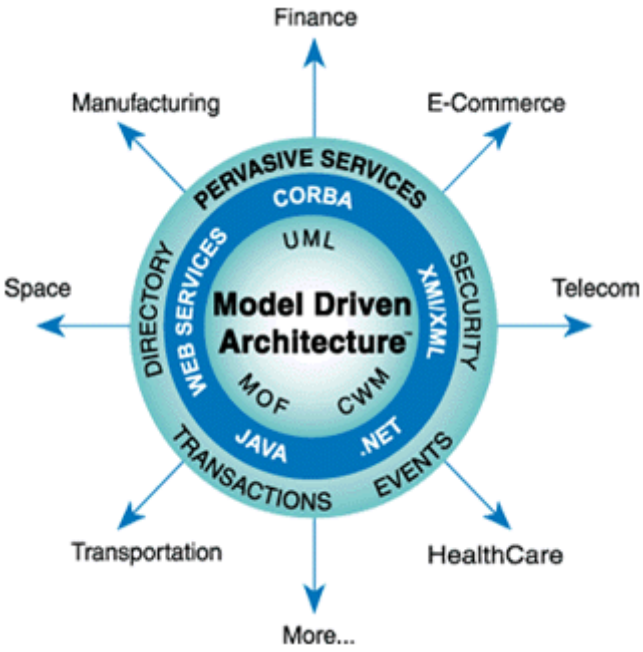


Figura 3.1. Arquitectura dirigida por modelos (MDA).

3.2.2. Conceptos de MDA

De cara a entender MDA y sus características, su funcionamiento y su aplicación al proceso de desarrollo, a continuación se exponen los conceptos básicos de MDA y su forma de uso.

3.2.2.1. Sistema

Los conceptos de MDA se definen centrados en la existencia o planteamiento de un sistema, que puede contener un simple sistema informático, o combinaciones de componentes en diferentes sistemas informáticos, o diferentes sistemas en diferentes organizaciones, etc.

3.2.2.2. Modelo

Un modelo de un sistema, es una descripción o especificación de ese sistema y su entorno para desempeñar un determinado objetivo. Los modelos se presentan normalmente como una combinación de texto y dibujos. El texto se puede presentar en lenguaje de modelado, o en lenguaje natural.

3.2.2.3. Dirigido por modelos

MDA es un acercamiento al desarrollo de sistemas, que potencia el uso de modelos en el desarrollo. Se dice que MDA es dirigido por modelos porque usa los modelos para dirigir el ámbito del desarrollo, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación de los sistemas.

3.2.2.4. Arquitectura

La arquitectura de un sistema es la especificación de las partes del mismo, las conexiones entre ellas, y las normas de interacción entre las partes del sistema haciendo uso de las conexiones especificadas.

MDA determina los tipos de modelos que deben ser usados, cómo preparar dichos modelos y las relaciones que existen entre los mismos.

3.2.2.5. Punto de vista

Un punto de vista es una abstracción que hace uso de un conjunto de conceptos de arquitectura y reglas estructurales para centrarse en aspectos particulares del sistema, obteniendo un modelo simplificado.

3.2.2.6. Vista

Una vista es una representación del sistema desde un determinado punto de vista.

3.2.2.7. Plataforma

Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades a través de interfaces y determinados patrones de uso, que cualquier aplicación que se construya para esa plataforma puede usar sin preocuparse por los detalles de la implementación o como se lleva a cabo la misma dentro de la plataforma.

3.2.2.8. Aplicación

En MDA se define el término aplicación como una funcionalidad que tiene que ser desarrollada. Por tanto podemos definir un sistema en términos de la implementación de una o más aplicaciones, soportadas por una o más plataformas.

3.2.2.9. Independencia de la plataforma

La independencia de la plataforma es una cualidad que tienen que presentar los modelos. Lo que significa que un modelo es independiente de las facilidades o características que implementan las plataformas, de cualquier tipo.

3.2.2.10. Puntos de vista MDA

MDA establece tres puntos de vista que se emplearán a lo largo del proceso de ingeniería:

- Punto de vista independiente de la computación: se centra en el entorno del sistema y los requisitos para el mismo. Los detalles de la estructura y procesamiento del sistema no se muestran, o aún no están especificados.

- Punto de vista independiente de la plataforma: se centra en las operaciones del sistema, mientras oculta los detalles necesarios para una determinada plataforma. Muestra aquellas partes de la especificación del sistema que no cambian de una plataforma a otra. En este punto de vista debe emplearse

lenguaje de modelado de propósito general, o bien algún lenguaje específico del área en que se empleará el sistema, pero en ningún caso se emplearán lenguajes específicos de plataformas.

- Punto de vista de plataforma específica: combina el punto de vista independiente de la plataforma con un enfoque específico para su uso en una plataforma específica en un sistema.

3.2.2.11. Modelo independiente de la computación (CIM)

Un modelo independiente de la computación es una vista del un sistema desde el punto de vista independiente de la computación. En algunos casos, nos referimos al modelo independiente de la computación como el modelo del dominio, y se usa vocabulario propio de los expertos en el dominio para la especificación.

3.2.2.12. Modelo independiente de la plataforma (PIM)

Un modelo independiente de la plataforma es una vista del sistema desde el punto de vista independiente de la plataforma. Expone un carácter independiente de la plataforma sobre la que se desplegará, de modo que pudiera ser empleado en diferentes plataformas de carácter similar.

Una técnica común para alcanzar el grado de independencia de la plataforma necesario es definir un sistema basado en una máquina virtual que abstraiga los modelos particulares de las plataformas existentes y sea neutral respecto a las mismas.

3.2.2.13. Modelo específico de la plataforma (PSM)

Un modelo específico de la plataforma es una vista de un sistema desde el punto de vista dependiente de la plataforma. Combina las especificaciones del modelo independiente de la plataforma, con los detalles que especifican el uso de una plataforma específica por parte del sistema.

3.2.2.14. Modelo de plataforma

Un modelo de plataforma expone un conjunto de conceptos técnicos que representan las diferentes formas o partes que conforman un sistema, y los servicios que provee. También expone, para su uso en los modelos específicos de la plataforma, conceptos que explican los diferentes elementos que provee una plataforma para la implementación de una aplicación en un sistema.

Un modelo de plataforma incluye también la especificación de requisitos en la conexión y uso de las partes que integran la plataforma, y la conexión de aplicaciones a la plataforma.

3.2.2.15. Transformación de modelos

La transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema.

La transformación de un modelo independiente de la plataforma en un modelo dependiente de la plataforma no es necesaria para PIM basados en una máquina virtual. En este caso hay que transformar el PIM correspondiente a la máquina virtual en un modelo de plataforma específico.

3.2.2.16. Servicios penetrantes

Los servicios penetrantes son servicios que están disponibles un amplio catálogo de plataformas. MDA proveerá servicios penetrantes comunes e independientes de la plataforma y dispondrá de trazas para la transformación de los modelos, que permitirá la transformación de los servicios expuestos en los PIMs a las plataformas de destino.

3.2.3. Evolución de MDA

A finales del año 2000, OMG presentó su iniciativa MDA que planteaba una nueva forma de entender el desarrollo y mantenimiento de sistemas de información con el uso de modelos como principales artefactos del proceso de desarrollo de software.

En MDA, los modelos son utilizados para dirigir las tareas de comprensión, diseño, construcción, pruebas, espliegue, operación, administración, mantenimiento y modificación de los sistemas. MDA también permite elevar el nivel de abstracción, lo que posibilita la creación de especificaciones que usan diferentes modelos para representar cada uno de los diferentes aspectos del sistema, al mismo tiempo que se automatiza la producción de estas especificaciones y la generación del código que las implementa.

Con el propósito de enfrentarse a los continuos cambios que sufren los sistemas software y las tecnologías relacionadas con las plataformas de implementación, MDA establece una separación clara entre la funcionalidad básica del sistema y su posterior implementación, esto es, separa la lógica del negocio, de la tecnología de la plataforma sobre la que se implementa, de modo que los cambios de plataforma no afecten al código de la lógica del negocio. Así, MDA distingue entre modelos independientes de la plataforma (PIM) y modelos específicos de la plataforma (PSM).

Por otro lado, las transformaciones de modelos (en sus variantes modelo-a-modelo y modelo-a-texto) permiten generar de forma automática parte del código final de implementación de un sistema a partir de los diferentes modelos creados. Los lenguajes de transformación de modelos permiten describir tanto las correspondencias entre los modelos, como las especificaciones que dirigen a las herramientas que generan código a partir de modelos.

MDA no determina ningún lenguaje concreto para expresar los modelos, que pueden ser descritos en cualquier lenguaje de modelado o lenguaje específico de dominio (DSL) que permita definir modelos que representen algún aspecto específico del sistema, al nivel de abstracción apropiado. En los primeros años de MDA, el lenguaje más utilizado para crear modelos PIM y PSM fue UML (y sus perfiles), pero conforme maduran las herramientas de definición de DSL, UML está cediendo el protagonismo. En cuanto a

las transformaciones modelo-a-modelo, OMG elaboró la especificación QVT que define tres lenguajes de transformación, y en la actualidad hay disponibles herramientas que lo soportan, aunque a lo largo de estos años han surgido numerosos lenguajes de transformación de modelos, algunos de los cuáles, como es el caso de ATL, han tenido un amplio reconocimiento.

Además de UML y QVT, OMG propuso otros estándares para conseguir la interoperabilidad entre fabricantes y herramientas de modelado. Algunos ejemplos de esos estándares son OCL, para especificar restricciones en los modelos, XMI como formato de intercambio de modelos y MOF como lenguaje de metamodelado. Sin duda, la apuesta por el metamodelado como base teórica para la creación de los DSL ha sido uno los puntos fuertes de MDA y a lo largo de estos años, MOF ha sido la principal referencia como paradigma de metamodelado.

El nacimiento, evolución, y progresiva adopción de MDA por parte de la comunidad científica y empresas, han tenido una gran influencia en los sistemas software modernos. Así, a lo largo de estos años hemos visto surgir la Ingeniería o Desarrollo Dirigido por Modelos (MDE, MDD) como una nueva área dentro del campo de la ingeniería del software. Con anterioridad a MDA ya habían aparecido algunas aproximaciones a la construcción de software dirigida por modelos, como el desarrollo basado en DSL. Pero es justo reconocer que MDA fue la propuesta que consiguió centrar el interés de la industria y la academia en el desarrollo dirigido por modelos. Además, la aparición de herramientas open source que soportan la visión MDA, sobre todo en el entorno Eclipse, ha contribuido de forma muy significativa al éxito de MDD como nuevo paradigma de desarrollo.

3.2.4. Tecnologías MDA

UML, MOF y XMI son tres tecnologías clave para el desarrollo de software bajo el enfoque de MDA. Usadas de forma conjunta nos proporcionan grandes ventajas que hacen que nuestros modelados sean más claros y fácilmente mantenibles. Realmente lo que definen estas tecnologías es una forma estándar de almacenar e intercambiar modelos, bien sean de negocio o de diseño. Esto permite a los constructores de herramientas CASE establecer un lenguaje común que se transformará en grandes beneficios para el desarrollador. Una vez que las herramientas implementen estos estándares veremos cómo podemos automatizar y estandarizar numerosos procesos del desarrollo que simplificarán muchas tareas, que antes eran manuales o que realizábamos de forma automática por medio de alguna característica propietaria de la herramienta, y que en muchos casos hacia imposible el intercambio con otras herramientas del mercado.

Actualmente la mayoría de las herramientas CASE más populares ya implementan estas tecnologías por lo que ya sólo tenemos que hacer uso de ellas para obtener sus beneficios, y poder así crear modelos, almacenarlos e intercambiarlos. Estas tecnologías ponen en nuestras manos una nueva forma de desarrollar software que no debemos desaprovechar.

3.2.4.1. UML

El lenguaje unificado de modelado es un lenguaje gráfico para el modelado de sistemas y nos permite modelar la arquitectura, los objetos, las interacciones entre objetos, datos y aspectos del ciclo de vida de una aplicación, así como otros aspectos más relacionados con el diseño de componentes incluyendo su construcción y despliegue. Los elementos del modelado de UML, es decir, clases, interfaces, casos de uso, diagramas de actividad, etc. además pueden ser intercambiados entre herramientas del ciclo de vida utilizando XMI. Actualmente nos encontramos con algunos perfiles UML para diferentes tecnologías como pueden ser CORBA, EJB, etc. y otros muchas que están en desarrollo. Estos perfiles son muy importantes dentro de la comunidad UML dentro de los modelos de diseño y también proporcionan un enlace con la comunidad de desarrolladores y de integradores.

Los diferentes elementos del modelado UML permiten especificaciones estáticas y dinámicas de un sistema orientado a objetos. Los modelos estáticos incluyen la definición de clases, atributos, operaciones, interfaces y relaciones entre clases, como pueden ser la herencia, asociación, dependencia, etc. La semántica de comportamiento de un sistema puede ser representada por medio del lenguaje UML gracias a sus diagramas de secuencia y colaboración. Para modelados más complejos, UML también proporciona mecanismos para la representación de máquinas de estado. Por último UML también proporciona una notación para representar el agrupamiento del diseño lógico por medio de componentes y el des pliegue y ubicación de esos componentes en nodos dentro de una arquitectura distribuida.

El lenguaje UML está formalmente definido por un metamodelo, que está a su vez definido de forma recursiva en UML. Este mecanismo circular permite definir UML en un número reducido de elementos.

3.2.4.2. MOF

La OMG ha creado el estándar MOF, que extiende UML para que este sea aplicado en el modelado de diferentes sistemas de información. El estándar MOF define diversos metamodelos, esencialmente abstrayendo la forma y la estructura que describe los metamodelos. MOF es un ejemplo de un meta meta modelo o un modelo del metamodelo orientado a objetos por naturaleza. Define los elementos esenciales, sintaxis y estructuras de metamodelos que son utilizados para construir modelos orientados a objetos de sistemas. Además, proporciona un modelo común para los metamodelos de CWM y UML. Más en concreto, el estándar MOF proporciona:

- Un modelo abstracto de los objetos genéricos MOF y sus asociaciones.

- Un conjunto de reglas para el mapeo de metamodelos basados en MOF a interfaces independientes del lenguaje de programación definidos por medio del estándar CORBA IDL. Una implementación de estos interfaces para un modelo concreto podría ser utilizado para acceder y modificar cualquier modelo basado en ese metamodelo.

- Reglas que definen el ciclo de vida, composición y semántica de los elementos basados en metamodelos MOF.

- Una jerarquía de interfaces reflexivos. Estos definen un conjunto de operaciones para localizar y manipular modelos basados en MOF pero de los cuales no conocemos sus interfaces.

El gran potencial de MOF reside en que permite interoperar entre metamodelos muy diferentes que representan dominios diversos. Las aplicaciones que utilizan MOF no tienen por qué conocer los interfaces del dominio específico de algunos de sus modelos, pero pueden leer y actualizar los modelos utilizando las operaciones genéricas y reflexivas ofrecidas en los interfaces. La semántica de MOF define generalmente servicios para el repositorio de metadatos, para así permitir la construcción, la localización, la actualización, etc. En concreto, una implementación MOF debe proporcionar herramientas para la autorización y la publicación de metadatos contra un repositorio MOF.

En el framework de cuatro capas que se muestra en la siguiente imagen, podemos observar que en la capa de información se encuentran bases de datos relacionales, orientadas a objetos, ficheros e información en XML. Cada fuente de datos tiene un metamodelo, definido en la capa del modelo, que determina su estructura y las relaciones que existen entre los diferentes tipos de entidades del modelo. El meta-metamodelo, definido en la capa del metamodelo, es un metamodelo que describe el contenido de los metamodelos, es decir, los tipos de entidades que son compartidas a través de los diferentes sistemas de información. El estándar MOF proporciona los mecanismos para poder describir todos los tipos de sistemas de información y puede ser ampliado para incluir más sistemas de los cuatro predefinidos.

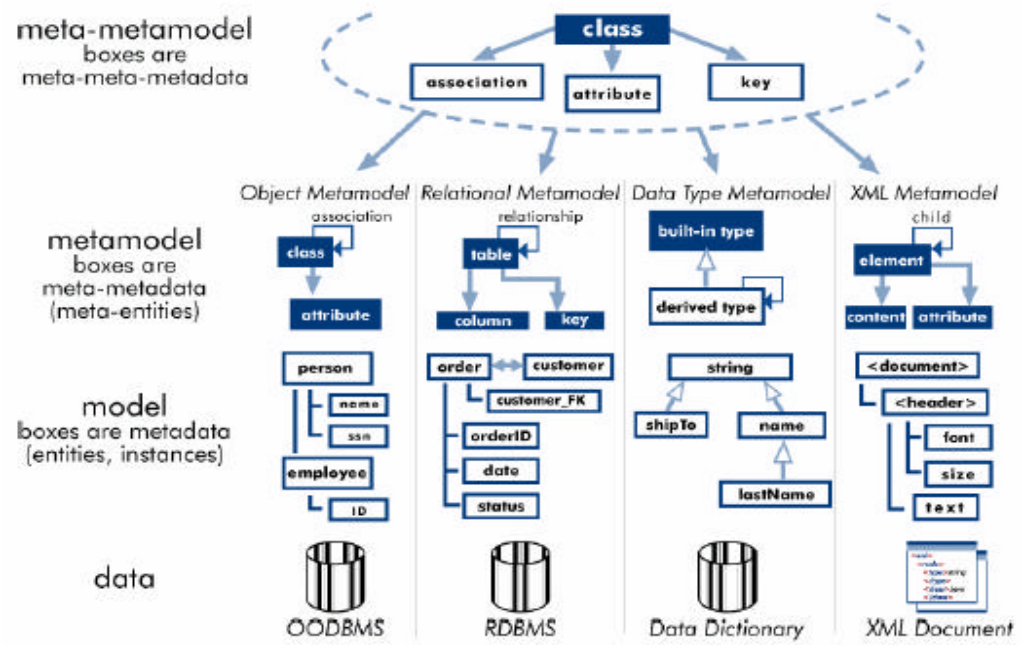


Figura 3.2. Framework MOF. Organización en capas del framework.

El framework MOF tiene bastantes ventajas sobre los sistemas de modelado simples y permite:

- Soportar cualquier tipo de modelo y paradigma de modelado imaginable.
- Permite relacionar diferentes tipos de metadatos.
- Permite añadir de forma incremental metamodelos y nuevos tipos de metadatos.
- Permite intercambiar modelos y metamodelos entre elementos que usen el mismo metametamodelo.

Esta arquitectura también tiene un conjunto de características que la diferencian de otras arquitecturas: El modelo MOF está orientado a objetos y formado por elementos totalmente compatibles con el lenguaje de modelado UML; Las capas de la arquitectura MOF no son fijas y aunque normalmente utilizaremos cuatro niveles estos podrían ser más o menos en función de cómo se quiera utilizar MOF. Estas capas son simplemente un mecanismo para entender las relaciones entre los diferentes tipos de datos y metadatos; Un modelo no está limitado únicamente a un único meta-nivel; El modelo

MOF se autodescribe a así mismo y normalmente es definido usando sus propios elementos del metamodelo, y además este modelo está representado por medio del dibujo del paquete UML. Esta característica tiene unas consecuencias importantes:

- Indica que el modelo MOF es lo suficientemente expresivo para realizar metamodelos.

- Permite a los interfaces MOF ser definidos por medio de mapeo IDL (Interface description language).

- Proporciona las bases de una arquitectura para su extensión y modificación.

- Permite nuevas implementaciones del repositorio del metamodelo y herramientas asociadas.

3.2.4.3. XMI

XML Metada Interchange (XMI) es un estándar de la OMG que mapea MOF a XML. XMI define cómo deben ser usadas las etiquetas XML que son usadas para representar modelos MOF. Los metamodelos MOF son convertidos en DTD (Document Type Definitions) y los modelos son convertidos en documentos XML que son consistentes con su DTD correspondiente. XMI resuelve muchos de los problemas encontrados cuando intentamos utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones, y además el hecho de que XMI esté basado en XML significa que tanto los metadatos (etiquetas) y las instancias que describen (elementos) pueden ser agrupados en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos.

Muchas de la herramientas CASE como Rational Rose, Together, Omondo, etc. soportan XMI y es el estándar para importar y exportar el formato. XMI no sólo puede ser usado como un formato de intercambio UML, sino que puede ser utilizado para cualquier formato descrito por medio de un metamodelo MOF. Las herramientas compatibles con MOF permiten definir metamodelos o importarlos, por ejemplo de repositorios y herramientas CASE, y empezar a editar o modificar la información del

modelo, por ejemplo instancias concretas de los servicios de negocios. Una vez hecho esto, la información del modelo podría ser intercambiada con otra herramienta compatible con MOF por medio de XMI.

Un ejemplo existente hoy en día de un metamodelo compatible con MOF aparte del conocido UML es el CWM (Commom Warehouse Metamodel), definido por la OMG.

XMI define muchos de los aspectos importantes involucrados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- Cómo los objetos están normalmente interconectados, XMI incluye un mecanismo estándar para enlazar objetos dentro del mismo fichero o entre ficheros diferentes.
- El versionando de objetos y sus definiciones son gestionadas por el modelo XMI.
- La validación de documentos XMI se realiza por medio de sus DTD o XMI Schemas.

XMI proporciona reglas para la creación de DTD, XML Schemas y documentos XML, y para ello define varios tipos de reglas de producción:

- Producción de DTD desde un modelo de objetos.
- Producción de XML Schemas desde un modelo de objetos.
- Producción de documentos XML desde un modelo de objetos.

3.3. NOTACIÓN PARA EL MODELADO DE PROCESOS DE NEGOCIO (BPMN)

3.3.1. ¿Qué es BPMN?

El Business Process Management Initiative (BPMI) ha desarrollado una notación estándar llamada Business Process Modeling Notation (BPMN). La especificación de la versión 1.0 salió al público en mayo del 2004. El objetivo principal de los esfuerzos de BPMN era dar una notación rápidamente comprensible para toda la gente de negocios, desde el analista de negocio que hace el borrador inicial de los procesos, pasando por los desarrolladores técnicos responsables de implementar la tecnología que llevarán a cabo dichos procesos, llegando finalmente a la gente de negocio que gestionará y monitorizará esos procesos. Además, BPMN está apoyado en un modelo interno que genera el ejecutable BPEL4WS. **Así, BPMN crea un puente estandarizado para el hueco entre el diseño de los procesos de negocio y la implementación de procesos.**

BPMN define un Business Process Diagram (BPD), que se basa en una técnica de grafos de flujo para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos, que son actividades (trabajo) y controles de flujo que definen su orden de rendimiento.

3.3.2. Fundamentos de BPMN

Un BPD está formado por un conjunto de elementos gráficos. Estos elementos habilitan el fácil desarrollo de diagramas simples que serán familiares para la mayoría de analistas de negocio (diagrama de flujo). Los elementos fueron elegidos para ser distinguibles los unos de los otros y para usar formas familiares para la mayoría de modeladores. Por ejemplo, las actividades son rectángulos y las decisiones son diamantes. Debe notarse que uno de los objetivos del desarrollo de BPMN es crear un mecanismo simple para crear modelos de procesos de negocio, y al mismo tiempo que sea posible gestionar la complejidad inherente en dichos procesos. El método elegido para manejar estos dos conflictivos requisitos fue organizar los aspectos gráficos de la notación en categorías específicas. Esto da un pequeño grupo categorías que alguien que lea un BPD pueda reconocer fácilmente los tipos básicos de elementos y pueda entender el diagrama. Dentro de las categorías básicas de elementos, se puede añadir información y variaciones adicionales para dar soporte a los requerimientos complejos sin cambiar

dramáticamente el *look-and-feel* básico del diagrama. Las cuatro categorías básicas de elementos son:

- Objetos de flujo.
- Objetos conectores.
- *Swimlanes*.
- Artefactos.

3.3.2.1. Objetos de flujo

Un BPD es un pequeño conjunto de elementos básicos, que son los Objetos de Flujo, de modo que los modeladores no tienen que aprender y reconocer un gran número de formas diferentes. Los tres objetos de flujo son:

- Evento: un evento se representa con un círculo. Es algo que "pasa" durante el curso del proceso de negocio. Estos eventos afectan al flujo del proceso y suelen tener una causa o un impacto. Los eventos representados con un círculo con centro abierto permiten a los marcadores internos diferenciar diferentes triggers y resultados. Hay tres tipos de eventos, basados en cuando afectan al flujo: Start , Intermediate, y End.

		
Start Event	Intermediate Event	End Event

Figura 3.3. Objetos de flujo BPMN. Eventos.

- Actividad: una actividad se representa con un rectángulo redondeado y es un término genérico para el trabajo que hace una compañía. Una actividad puede

ser atómica o compuesta. Los tipos que hay son: Task y Sub-Process. El Sub-Process se distingue por una pequeña marca de suma en la parte central inferior de la figura.

Figura 3.4. Objetos de flujo BPMN. Actividad.

- Gateway (compuerta): una gateway se representa por la típica figura de diamante y se usa para controlar la divergencia o convergencia de la secuencia de flujo. Así, ésto determina las tradicionales decisiones, así como la creación de nuevos caminos, la fusión de estos o la unión. Los marcadores internos indicarán el tipo de control de comportamiento.

Figura 3.5. Objetos de flujo BPMN. Gateway.

3.3.2.2. Objetos conectores

Los objetos de flujo se conectan entre ellos en un diagrama para crear el esqueleto básico de la estructura de un proceso de negocio. Hay tres objetos conectores que hacen esta función. Estos conectores son:

- Sequence Flow: el flujo de secuencia se representa por una línea sólida con una cabeza de flecha sólida y se usa para mostrar el orden (la secuencia) en el que las diferentes actividades se ejecutarán en el Proceso. El término "control flow" normalmente no se usa en BPMN.

- Message Flow: el flujo de mensaje se representa por un línea discontinua con una punta de flecha hueca y se usa para mostrar el flujo de mensajes entre dos participantes del proceso separados (entidades de negocio o roles de negocio).

En BPMN, dos pools separadas en el diagrama representan los dos participantes.

- Association: una asociación se representa por una línea de puntos con una punta de flecha de líneas y se usa para asociar datos, texto, y otros artefactos con los objetos de flujo. Las asociaciones se usan para mostrar entradas y salidas de las actividades.

Figura 3.6. Conectores BPMN. Sequence Flow, Message Flow y Association.

Para los modeladores que requieren o desean más precisión para crear modelos de proceso por motivos de documentación y comunicación, los elementos básicos más los conectores dan la posibilidad de crear fácilmente diagramas comprensibles.

Para los diseñadores que necesiten un nivel más alto de precisión, para análisis detallado o que sean manejados por un BPMS, existen detalles adicionales que se pueden añadir a los elementos básicos.

3.3.2.3. Swimlanes (Rol de Proceso)

Muchas metodologías de modelado de procesos usan el concepto de swimlanes como un mecanismo para organizar actividades en categorías separadas visualmente para ilustrar diferentes capacidades funcionales o responsabilidades. BPMN soporta los swimlanes con dos constructores principales. Los dos tipos de objetos swimlanes son:

- Pool: una pool representa un Participante de un Proceso. Además actúa como un contenedor gráfico para particionar un conjunto de actividades desde otros pools.

- Lane: una lane es una sub-partición dentro de un pool y extiende la longitud del pool, verticalmente u horizontalmente. Las lanes se usan para organizar y categorizar actividades.

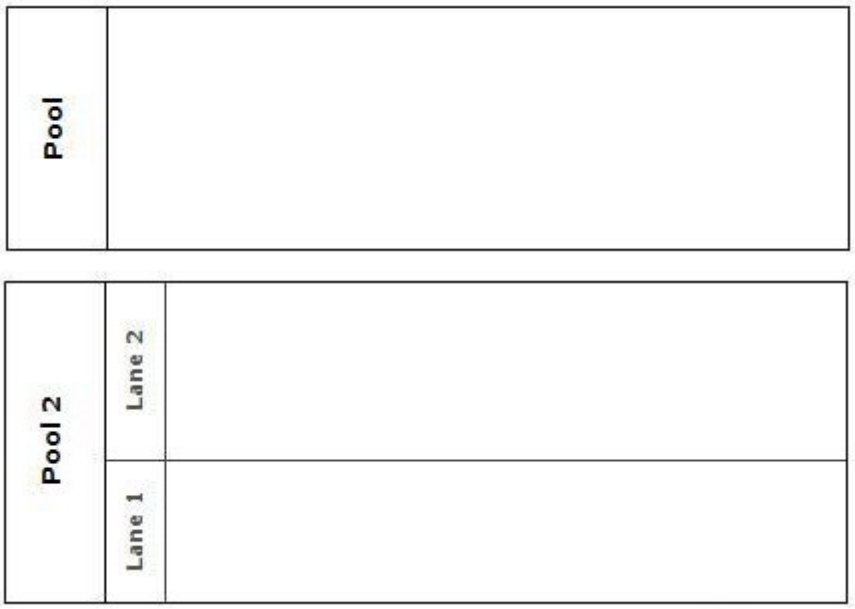


Figura 3.7. Swimlanes BPMN. Pool y Lane.

Las pools se usan cuando un diagrama implica dos entidades de negocio o participantes separados y están físicamente separados en el diagrama. Las actividades dentro de pools separadas se consideran procesos autocontenidos. Así, el flujo de secuencia no debe cruzar el límite de un pool. El flujo de mensajes se define como el mecanismo para mostrar las comunicaciones entre dos participantes, y, de este modo debe conectar dos pools (o los objetos dentro de las pools).

Las pistas (lanes) están más estrechamente relacionadas con las metodologías tradicionales de las swimlanes. Las pistas se suelen usar para separar las actividades asociadas con la función o rol de una compañía específica. El flujo de secuencia puede cruzar los límites de las pistas dentro de un pool, pero el flujo de mensajes no puede ser usado entre objetos de flujo en pistas de mismo pool.

3.3.2.4. Artefactos

BPMN fue diseñado para permitir a los modeladores y las herramientas de modelado un poco de flexibilidad a la hora de extender la notación básica y a la hora de habilitar un contexto apropiado adicional según una situación específica, como para un mercado vertical (por ejemplo, seguros o banca). Se puede añadir cualquier número de artefactos a un diagrama como sea apropiado para un contexto de proceso de negocio específico. La versión actual de la especificación de BPMN sólo tiene tres tipos de artefactos BPD predefinidos, los cuales son:

- Data Object: los objetos de datos son un mecanismo para mostrar como los datos son requeridos o producidos por las actividades. Están conectados a las actividades a través de asociaciones.

- Group: un grupo es representado por un rectángulo redondeado con línea discontinua. El agrupamiento se puede usar documentación o análisis, pero no afecta al flujo de secuencia.

- Annotation: las anotaciones son mecanismos para que un modelador pueda dar información textual adicional.

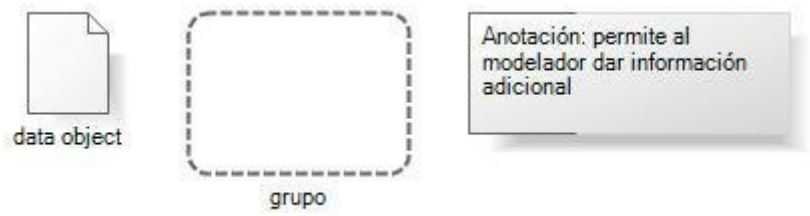


Figura 3.8. Artefactos BPMN. Data Object, Group y Annotation.

Los modeladores pueden crear sus propios tipos de artefactos, que añaden más detalle sobre cómo se ejecuta el proceso (bastante a menudo para mostrar las entradas y las salidas de las actividades del Proceso). Sin embargo, la estructura básica del proceso, determinada por las actividades, gateways, y flujos de secuencia, no se cambia por añadir artefactos al diagrama.

3.3.3. ¿Cuál es el valor de modelar en BPMN?

Los miembros de BPMI representan un gran segmento de la comunidad de modelado de procesos de negocio y han llegado a un consenso y presentan BPMN como la notación de modelado de procesos de negocio estándar. El desarrollo de BPMN es un paso importante para reducir la fragmentación que existe con la gran cantidad de herramientas de modelado de procesos y notaciones. El BPMI porta una gran experiencia con muchas de las notaciones existentes y trabajan para consolidar las mejores ideas de todas estas notaciones para crear una sola notación estándar. Ejemplos de otras notaciones o metodologías que fueron revisadas son: diagramas de actividades de UML, UML EDOC.

Una única notación bien definida reduce la confusión entre los usuarios IT y de negocios. Otro factor del desarrollo de BPMN es que, históricamente, los modelos de procesos de negocio desarrollados por la gente de negocios han estado técnicamente separados de las representaciones de procesos requeridas por los sistemas diseñados para implementar y ejecutar dichos procesos. Así, era necesario traducir manualmente los modelos de procesos de negocio originales a los modelos de ejecución. Esas traducciones están sujetas a errores y dificultan a los dueños de los procesos entender la evolución y el rendimiento de los procesos desarrollados.

3.4. LENGUAJE DE PROGRAMACIÓN JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

3.4.1. Historia

La tecnología Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en el año 1991. El equipo (Green Team), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente Oak (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse Green tras descubrir que Oak era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a Java.

El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: *James Gosling, Arthur Van Hoff, y Andy Bechtolsheim*. Otros abogan por el siguiente acrónimo, *Just Another Vague Acronym* ("sólo otro acrónimo ambiguo más"). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana, de ahí que el icono de java sea una taza de cafe caliente. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el número mágico) de los archivos .class que genera el compilador, son en hexadecimal, 0xCAFEBABE. Otros simplemente dicen que el nombre fue sacado al parecer de una lista aleatoria de palabras.

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era Write Once, Run Anywhere (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.

Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa Java Specification Requests (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la Java Language Specification (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

Además de los cambios en el lenguaje, con el paso de los años se han efectuado muchos más cambios dramáticos en la biblioteca de clases de Java que ha crecido de unos pocos cientos de clases en JDK 1.0 hasta más de tres mil en J2SE 5.0. APIs completamente nuevas, como Swing y Java2D, han sido introducidas y muchos de los métodos y clases originales de JDK 1.0 están obsoletas.

3.4.2. Filosofía

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.

5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA, Internet Communications Engine o OSGi respectivamente.

3.4.3. Orientado a objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

3.4.4. Independiente de la plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run everywhere".

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste

último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la máquina virtual es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran

número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en cualquier parte” por “Escríbelo una vez, depúralo en todas partes”).

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

3.4.5. El recolector de basura

Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria solicitada dinámicamente de forma manual:

En C++, el desarrollador puede asignar memoria en una zona conocida como heap (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada puede llevar a una fuga de memoria, ya que el sistema operativo seguirá pensando que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual cuelgue. No obstante, se debe señalar que C++ también permite crear objetos en la pila de llamadas de una función o bloque, de forma que se libere la memoria (y se ejecute el destructor del objeto) de forma automática al finalizar la ejecución de la función o bloque.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java

borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

3.4.6. Sintaxis

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase (recordemos que una clase es un molde a partir del cual pueden crearse varios objetos).

3.4.6.1. Applets

Las applets de Java son programas incrustados en otras aplicaciones, normalmente una página Web que se muestra en un navegador.

3.4.6.2. Servlets

Los servlets son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.

3.4.7. Entornos de funcionamiento

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

3.4.7.1. En dispositivos móviles y sistemas empotrados

Desde la creación de la especificación J2ME, una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de consumo, se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores específicamente diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes, teléfonos móviles, buscapersonas, sintonizadores de TV y otros pequeños electrodomésticos.

El modelo de desarrollo de estas aplicaciones es muy semejante a las applets de los navegadores salvo que en este caso se denominan MIDlets.

3.4.7.2. En el navegador Web

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (plug-in) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

El éxito de este tipo de aplicaciones no fue realmente el esperado debido a diversos factores, siendo quizás el más importante la lentitud y el reducido ancho de banda de las comunicaciones en aquel entonces que limitaba el tamaño de las applets que se incrustaban en el navegador. La aparición posterior de otras alternativas (aplicaciones web dinámicas de servidor) dejó un reducido ámbito de uso para esta tecnología, quedando hoy relegada fundamentalmente a componentes específicos para la

intermediación desde una aplicación web dinámica de servidor con dispositivos ubicados en la máquina cliente donde se ejecuta el navegador.

Las applets Java no son las únicas tecnologías (aunque sí las primeras) de componentes complejos incrustados en el navegador. Otras tecnologías similares pueden ser: ActiveX de Microsoft, Flash, Java Web Start, etc.

3.4.7.3. En sistemas de servidor

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica).

Los servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es muy sencilla, flexible y extensible.

- Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.

- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy en día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks que se sobreponen sobre los servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la especialización de roles sea posible (desarrolladores, diseñadores gráficos, ...) y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas.

Este modelo de trabajo se ha convertido en un estándar de-facto para el desarrollo de aplicaciones web dinámicas de servidor y otras tecnologías se han basado en él.

3.4.7.4. En aplicaciones de escritorio

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico. Desde la aparición de la biblioteca “Swing”, la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT (Standard Widget Toolkit) hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

3.4.8. Plataformas soportadas

Una versión del entorno de ejecución Java JRE está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de

applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

3.4.9. Industria relacionada

Sun Microsystem, como creador del lenguaje de programación Java y de la plataforma JDK (Java Development Kit), mantiene fuertes políticas para mantener una especificación del lenguaje1 así como de la máquina virtual2 a través del JCP (Java Community Process). Es debido a este esfuerzo que se mantiene un estándar de facto.

Son innumerables las compañías que desarrollan aplicaciones para Java y/o están volcadas con esta tecnología:

- La industria de la telefonía móvil está fuertemente influenciada por la tecnología Java.

- El entorno de desarrollo Eclipse ha tomado un lugar importante entre la comunidad de desarrolladores Java.

- La fundación Apache tiene también una presencia importante en el desarrollo de bibliotecas y componentes de servidor basados en Java.

- IBM, BEA, IONA, Oracle,... son empresas con grandes intereses y productos creados en y para Java.

3.5. ENTORNO DE DESARROLLO ECLIPSE EUROPA

Eclipse es un proyecto de desarrollo software de código abierto, cuyo propósito es proporcionar una plataforma de herramientas altamente integradas. El trabajo en Eclipse consiste en un proyecto central que incluye un framework genérico para la integración de herramientas, y un entorno de desarrollo Java construido usando el framework anterior.

Otros proyectos extienden el framework núcleo para soportar tipos de herramientas y entornos de desarrollo específicos, entre los que encontramos PDE (Plug-in Development Environment), EMF (Eclipse Modeling Framework) o GMF (Graphical Modeling Framework). Los proyectos en Eclipse se implementan en Java y se ejecutan en diversos sistemas operativos, incluyendo Windows y Linux.

Eclipse.org es un consorcio de diversas compañías (como IBM, Borland, Intel, AMD, Oracle, Nokia, entre otras) que se han comprometido en proporcionar soporte al proyecto Eclipse en términos de tiempo, experiencia, tecnología o conocimiento. Los proyectos que conforman Eclipse operan bajo un organigrama bien definido que marca los roles y responsabilidades de los diversos participantes, incluyendo el consejo, los usuarios de Eclipse, los desarrolladores y los comités de gestión de proyectos.

El amplio soporte empresarial de la plataforma Eclipse, su estructura bien definida y la naturaleza abierta del proyecto han contribuido al éxito de esta plataforma tanto como entorno de desarrollo de Java, como plataforma base para el desarrollo de cualquier tipo de aplicación (Aplicaciones RCP, Rich Client Platform). Ejemplo de estas aplicaciones son IBM Rational Software Architect, Borland JBuilder 2007, IBM Lotus Expeditor o Azureus.

3.5.1. Historia

Eclipse comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International) como reemplazo de VisualAge también desarrollado por OTI. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, la fundación independiente de IBM fue creada.

Eclipse 3.0 (2003) seleccionó las especificaciones de la plataforma OSGi como la arquitectura de tiempo de ejecución.

- Callisto: En 2006 la fundación Eclipse coordinó sus 10 proyectos de código abierto, incluyendo la Platafoma 3.2, para que fueran liberados el mismo día. Esta liberación simultánea fue conocida como la liberación Callisto.5

- Europa: La versión consecutiva a Callisto es Europa, que corresponde a la versión 3.3 de Eclipse, salió el 29 de Junio del 2007.

- Ganymede: La versión consecutiva a Europa es Ganymede, que corresponde a la versión 3.4 de Eclipse, salió el 25 de Junio del 2008.

3.5.2. Arquitectura

El Proyecto Eclipse es un proyecto de desarrollo de software de código abierto, dedicado a proporcionar una plataforma industrial robusta, con amplias características y con calidad comercial para el desarrollo de herramientas altamente integradas.

Está compuesto de tres subproyectos: la Plataforma Eclipse, la Java Development Tool y el Plug-in Development Environment. El éxito de la Plataforma Eclipse depende de cómo sea capaz de admitir una amplia gama de herramientas de desarrollo para reproducir lo mejor posible las herramientas existentes en la actualidad.

Eclipse lo forman el núcleo, el entorno de trabajo (Workspace), el área de desarrollo (Workbench), la ayuda al equipo (Team support) y la ayuda o documentación (Help).

- Núcleo: su tarea es determinar cuáles son los plug-ins disponibles en el directorio de plug-ins de Eclipse. Cada plug-in tiene un fichero XML manifest que lista los elementos que necesita de otros plug-ins así como los puntos de extensión que ofrece. Como la cantidad de plug-ins puede ser muy grande, sólo se cargan los necesarios en el momento de ser utilizados con el objeto de minimizar el tiempo de arranque de Eclipse y recursos.

- Entorno de trabajo: maneja los recursos del usuario, organizados en uno o más proyectos. Cada proyecto corresponde a un directorio en el directorio de trabajo de Eclipse, y contienen archivos y carpetas.

- Interfaz de usuario: muestra los menús y herramientas, y se organiza en perspectivas que configuran los editores de código y las vistas.
  A diferencia de muchas aplicaciones escritas en Java, Eclipse tiene el aspecto y se comporta como una aplicación nativa. No está programada en Swing, sino en SWT (Standard Widget Toolkit) y Jface (juego de herramientas construida sobre SWT), que emula los gráficos nativos de cada sistema operativo. Este ha sido un aspecto discutido sobre Eclipse, porque SWT debe ser portada a cada sistema operativo para interactuar con el sistema gráfico. En los proyectos de Java puede usarse AWT y Swing salvo cuando se desarrolle un plug-in para Eclipse.

- Ayuda al grupo: este plug-in facilita el uso de un sistema de control de versiones para manejar los recursos en un proyecto del usuario, y define el proceso necesario para guardar y recuperar de un repositorio.
  Eclipse incluye un cliente para CVS.

- Documentación: al igual que el propio Eclipse, el componente de ayuda es un sistema de documentación extensible. Los proveedores de herramientas pueden añadir documentación en formato HTML y, usando XML, definir una estructura de navegación.

3.5.3. Plug-ins

Un plug-in es la mínima unidad de la plataforma que puede ser desarrollado por Separado y que la aporta una nueva funcionalidad. Los hay freeware y de pago; incluso puedes programar uno tú mismo. Se instalan descomprimiendo el zip del plug-in en el directorio plugins de Eclipse.

La carpeta que aloja un plug-in tiene por nombre el del plug-in seguido de un guión bajo (_) seguido del número de versión. Ante plug-ins con mismo nombre, Eclipse selecciona la última versión.

3.6. ECLIPSE MODELLING FRAMEWORK (EMF)

3.6.1. ¿Qué es EMF?

Eclipse Modeling Framework (Framework de modelado Eclipse, EMF) es un framework de modelado y de facilidad de generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Desde una especificación del modelo descrita en XMI, EMF suministra herramientas y soporte runtime para producir un conjunto de clases Java para el modelo, un conjunto de clases Adapter que permiten visualización y edición basándose en comandos del modelo, y un editor básico. Los Modelos pueden ser especificados usando Anotación Java, documentos XML, o herramientas de modelado como Rational Rose, y después ser importados a EMF. Lo más importante de todo, EMF suministra las bases para la interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

3.6.2. ¿Qué nos permite EMF?

- Describir el modelo de datos a alto nivel. P.e. podemos usar un diagrama de Rational Rose, un diagrama UML, XSD, interfaces Java anotadas... etc. para construir un fichero ecore con el modelo de datos de nuestro sistema.

- Con el fichero ecore, podemos generar un fichero genmodel que a su vez puede ser usado para generar el código Java que describe nuestro modelo. Crea además un editor en línea de comandos y otro gráfico como plugins de Eclipse. Todo eso sin tener que escribir una línea de código.

- Con el código que hemos generado, podemos serializar/deserializar nuestro modelo a XMI y XML, tenemos relaciones entre objetos, notificaciones, notificaciones inversas, adaptadores... etc, todo ello como he comentado out-

of-the-box, sólo con la descripción a alto nivel de nuestro modelo y sin escribir (casi) una sola línea de código.

- Podemos manipular el código generado, y si en un futuro hay cambios en el modelo, se puede recargar y regenerar el código, respetándose nuestros cambios.

- Podemos usar los ficheros ecore y genmodel anteriores para crear un editor gráfico (en plan pinchaiconos, copiar y arrastrar) con GMF.

3.7. GRAPHICAL MODELLING FRAMEWORK (GMF)

3.7.1. ¿Qué es GMF?

GMF (Graphical Modelling Framework) proporciona un generador de modelos. GMF está embebido en Eclipse, plataforma de desarrollo que permite la ejecución de los editores gráficos generados por GMF a partir de un modelo. Eclipse está construido y trabaja en Java. GMF esta basado en EMF (Eclipse Modelling Framework) y en GEF (Graphical Eclipse Framework), ambos proporcionados por la plataforma Eclipse. En GMF se ha adoptado el término "toolsmith" para referirse a los desarrolladores que utilizan la herramienta para construir extensiones que posteriormente se integran como parte de la misma. Estas extensiones reciben el nombre de plug-ins. Otro término, "practitioner", se utiliza para referirse a aquellos que utilizan los plug-ins como medio para el desarrollo. Durante la utilización de GMF para la generación de un modelo, la descripción del modelo se realiza una sola vez, al comienzo. Una vez realizada la especificación del dominio, la herramienta se encarga de interpretar las correspondencias con el modelo durante el resto de proceso.

3.8. LENGUAJE OCL

El éxito de los lenguajes gráficos de modelado, tales como el UML, se basa principalmente en el uso de diagramas que transmiten un significado intuitivo. Estos lenguajes resultan atractivos para los usuarios ya que aparentemente son fáciles de entender y aplicar. Sin embargo, la falta de precisión en la definición de su semántica puede originar diversos problemas.

OCL es un nuevo lenguaje notacional un subconjunto del UML estándar industrial, que permite a los desarrolladores de software escribir restricciones sobre modelos de objetos (pre y postcondiciones, guardas, invariantes, valores derivados, restricciones sobre operaciones, etc.). Estas restricciones son particularmente útiles en la medida en que permiten a los desarrolladores crear un amplio conjunto de reglas que rigen el aspecto de un objeto individual.

3.8.1. Características del Lenguaje OCL

OCL tiene las características de un lenguaje de expresiones, un lenguaje de modelos y un lenguaje formal:

- Lenguaje de expresiones: OCL es un lenguaje de expresiones puro. Una expresión OCL garantiza que quedará sin efecto. Esto no puede cambiar nada en el modelo. Esto significa que un estado del sistema nunca cambiará debido a una expresión OCL, incluso una expresión OCL podría usarse para describir tal cambio de estado (p.e. en una postcondición).Todos los valores de todos los objetos, incluidos los enlaces, no cambiarán.

- Lenguaje de modelos: OCL es un lenguaje de modelos y no un lenguaje de programación. No se puede escribir un programa lógico o un flujo de control en OCL. Especialmente, no se puede invocar procesos o activar operaciones no de consulta en OCL. Debido a que OCL es en primer lugar un lenguaje de modelos, no se puede asegurar que todo sea directamente ejecutable. Como lenguaje de modelos, todos los aspectos de implementación están fuera de alcance y no pueden expresarse en OCL. Cada expresión OCL es

conceptualmente atómica. El estado de los objetos en el sistema no pueden cambiar durante la evaluación.

- Lenguaje formal: OCL es un lenguaje formal donde todos los constructores tienen un significado formal definido. La especificación de OCL es parte de la especificación de UML. OCL no tiene la intención de reemplazar los lenguajes formales existentes.

Los lenguajes formales tradicionales se usaban por personas con conocimientos matemáticos, pero dificulta su uso para la mitad de empresas y modeladores de sistemas. OCL ha sido desarrollado para cubrir esta necesidad.

Puesto que en un proyecto hay mucha gente involucrada (usuario, expertos, personas que después se deberán encargar de su mantenimiento, etc.) los modelos deben ser entendidos por una amplia y variada audiencia. OCL es fácil de aprender y usar por los desarrolladores sin amplios conocimientos matemáticos. OCL tiene ciertas características que permiten a los desarrolladores adoptarlo a su ritmo y sólo donde lo necesiten. Hace accesibles las especificaciones formales con un trasfondo matemático limitado.

Otro aspecto importante es que OCL no es un lenguaje completo en sí mismo. Muchos lenguajes formales mandan (o al menos se supone) que la especificación completa se escriba en el mismo lenguaje. Con OCL, no se necesita, incluso se tiene la posibilidad de escribir las especificaciones completas en OCL. La intención de OCL es la de utilizarlo en combinación con los modelos visuales UML. Muchos aspectos de modelaje pueden expresarse mucho mejor usando diagramas visuales y OCL no intenta reemplazarlos por sus propios mecanismos. Por el contrario, toma la información expresada en los modelos visuales y permite al desarrollador acceder a esta información en las expresiones OCL.

OCL es un lenguaje tipado, por lo que cada expresión OCL tiene un tipo. Para ser bien formada, una expresión debe concordar con los tipos de las reglas del lenguaje. Por ejemplo, no puede compararse un Integer con un String. Cada clasificador definido en

un modelo UML representa un tipo distinto en OCL. Además, OCL incluye un conjunto de tipos adicionales predefinidos.

3.8.2. Conexión con el metamodelo UML

3.8.2.1. SELF

Cada expresión OCL se escribe en el contexto de una instancia de un tipo específico. En una expresión OCL, la palabra reservada "self" se usa para referirse a la instancia contextual.

3.8.2.2. Especificar el contexto UML

El contexto de una expresión OCL en un modelo UML puede especificarse mediante una declaración de pseudo-contexto al inicio de una expresión OCL.

Si la restricción se muestra en un diagrama, con el estereotipo correspondiente y las líneas de guiones largos para conectarlos con su elemento contextual, no hay necesidad de una declaración de contexto explícita en el test de las restricciones. La declaración de contexto es opcional.

3.8.2.3. Invariantes

Una expresión OCL puede ser parte de un invariante el cual es una restricción estereotipada como <<invariante>>. Una expresión OCL es un invariante del tipo y debe ser cierta para todas las instancias de ese tipo en cualquier momento. (Hay que destacar que todas las expresiones OCL que expresan invariantes son de tipo Boolean).

3.8.2.4. Pre- y Postcondiciones

La expresión OCL puede ser parte de una pre- o postcondición, que se corresponden con los estereotipos <<precondicion>> y <<postcondicion>>, respectivamente, de la restricción asociada con una operación o método. La instancia contextual self es entonces una instancia del tipo que pertenece a la operación o método como una característica. La declaración de contexto en OCL utiliza la palabra clave context,

seguido del tipo y la declaración de la operación. El estereotipo de restricción se muestra mediante las etiquetas 'pre:' y 'post:' antes de las pre- y postcondiciones reales.

```
context Typename::operationName(param1:Type1, ...):ReturnType
pre parameterOk: param1>...
post resultOk: result=...
```

Puede usarse la palabra “self” en la expresión refiriéndose al objeto sobre el que la operación fue llamada. Opcionalmente, el nombre de la pre- o postcondición puede escribirse tras las palabras claves pre o post.

3.8.3. Operadores

OCL incluye las siguientes funciones aritméticas y agregadas:

Tipo de operandos	Operaciones
Real	=, +, -, *, /, abs, floor, max, min, <, >, <=, >=
Integer	=, +, -, *, /, abs, div, mod, max, min
Boolean	=, or, xor, and, not, implies, if-then-else
String	=, size, concat, toUpper, toLower, substring
Enumeration	=, <>

Figura 3.9. Tabla operadores OCL.

3.8.3.1. La expresión LET

A veces una subexpresión se usa varias veces en una restricción. La expresión “let” permite definir una variable que puede usarse en la restricción.

3.8.3.2. Concordancia de tipos

Una expresión OCL en la que los tipos no concuerdan es una expresión inválida. Ésta contiene un tipo de error de concordancia. Un tipo type1 concuerda con un tipo type2 cuando una instancia de type1 puede ser sustituida en cualquier lugar donde se espera una instancia de type2. Las reglas de concordancia de tipos en los diagramas de clases son simples:

- Cada tipo concuerda con cada uno de sus supertipos.

- La concordancia de tipos es transitiva: si type1 concuerda con type2, y type2 concuerda con type3, entonces type1 concuerda con type3.

Tipo	Concuerda con / Es un subtipo de
Set(T)	Collection(T)
Sequence(T)	Collection(T)
Bag(T)	Collection(T)
Integer	Real

Figura 3.10. Tabla concordancia de tipos OCL.

3.8.3.3. Retipado o casting

Cuando es cierto que el tipo real del objeto es el subtipo, el objeto puede ser retipado usando la operación oclAsType (OclType). Esta operación tiene efecto en el mismo objeto, pero el tipo conocido es el argumento OclType. Cuando se tiene un objeto object de tipo Type1 y Type2 es otro tipo, se puede escribir:

Object.oclAsType(Type2) --- Evalua al objeto con el tipo Type2

Un objeto puede ser sólo retipado a uno de sus subtipos; por tanto, en el ejemplo, Type2 debe ser un subtipo de Type1.

3.8.3.4. Reglas de precedencia

El orden de precedencia de las operaciones, de mayor a menor prioridad, en OCL es:

1. @pre
2. punto y operaciones flecha: '.' y '->'
3. operadores unitarios 'not' y menos '-'
4. '*' y '/'
5. '+' y el operador binario '-'
6. 'if-then-else-endif'

7. '<','>','<=','>='
8. '=','<>'
9. 'and','or', y 'xor'
10. 'implies'

Los paréntesis '(' y ')' pueden usarse para variar la precedencia.

3.8.4. Propiedades

3.8.4.1. Atributos

Por ejemplo, la edad de una persona se escribe como:

```
context  Person inv:
    self.age>0
```

El valor de la subexpresión self.age es el del atributo age en una instancia, en concreto, de Person identificada por self. El tipo de esta subexpresión es el tipo del atributo age.

3.8.4.2. Operaciones

Las operaciones pueden tener parámetros.

```
context Person::income (d:Date):Integer
    post: result = age*1000
```

3.8.4.3. Combinación de propiedades

Las propiedades se pueden combinar para obtener expresiones más complicadas. Una regla importante es que una expresión OCL siempre evalúa a un objeto específico de un tipo en concreto. Tras obtener un resultado, siempre se puede aplicar otra propiedad al resultado para obtener un nuevo valor resultante. Por tanto, cada expresión OCL puede leerse y evaluarse de izquierda a derecha.

3.8.4.4. Acceso a las propiedades que sobrescriben supertipos

Las propiedades siempre se redefinen en un tipo, la propiedad de los supertipos puede accederse usando la operación "oclAsType()". Siempre que tengamos una clase B como un subtipo de la clase A, y una propiedad p1 tanto de A como de B, podemos escribir:

```
context B inv:
    self.oclAsType(A).p1 -- accede a la propiedad p1 definida en A
    self.p1              -- accede a la propiedad p1 definida en B
```

3.8.4.5. Propiedades predefinidas en todos los objetos

Hay varias propiedades que se aplican a todos los objetos y que están predefinidas en OCL. Éstas son:

```
OclIsTypeOf (t : OclType) : Boolean
OclIsKingOf (t : OclType) : Boolean
OclInState (s : OclState) : Boolean
OclisNew : Boolean
OclAsType (t : OclType) : instance of OclType
```

El resultado de la operación oclTypeOf es verdad si el tipo de self y t son el mismo. La propiedad oclIsKindOf además determina si t es el tipo directo de uno de los supertipos de un objeto.

La operación oclInState devuelve verdad si el objeto está en el estado s. Los valores para s son los nombres de los estados de la máquina de estado s vinculada al clasificador del objeto. Si tenemos un conjunto de estados donde unos están contenidos en otros, los nombres de los estados pueden combinarse usando la notación ::.

La operación oclIsNew se evalúa a verdad si, usado en una postcondición, el objeto se crea durante la ejecución de la operación.

3.8.4.6. Características de las propias clases

Una característica predefinida en cada tipo es allInstance, cuyo resultado es el conjunto de todas las instancias del tipo existente en un momento determinado a la hora de evaluar una expresión.

El uso de allInstances tiene algunos problemas y su uso está desaconsejado en muchos casos. Para los tipos integer, real y string el significado de allInstances está indefinido. Además, la existencia de objetos debe ser considerada en algunos contextos globales, como un sistema o un modelo. Este contexto global debe ser definido, lo que no se hace con OCL.

3.8.4.7. Características de las propias clases

El tipo Collection define un gran número de operaciones predefinidas que permiten al modelador de expresiones OCL manipular colecciones. Las operaciones de colecciones nunca cambian las colecciones originales, sino que guardan el resultado en otra colección.

Colección es un supertipo abstracto para todos los tipos de colección en OCL. Cada ocurrencia de un objeto en una colección se llama elemento. Si un objeto aparece por segunda vez en una colección, hay dos elementos. Algunas propiedades se pueden definir con el subtipo, el cual significa que hay una post condición adicional o un valor de retorno especial. La definición de algunas propiedades comunes, es diferente para cada subtipo.

OCL distingue tres tipos diferentes de colecciones: Set, Sequence y Bag. Set es un conjunto matemático. No contiene elementos duplicados. Bag es como un conjunto, que puede contener duplicados. Sequence es como un Bag en el que los elementos están ordenados. Para definir una colección, los elementos se encierran entre llaves y están separados por comas.

3.8.4.8. Valores previos en las postcondiciones

El valor de una propiedad en una postcondición es el valor tras completar la operación. Para referirse al valor de la propiedad al comienzo de la operación, se tiene que añadir tras el nombre de la propiedad la palabra clave '@pre':

3.9. COMPARACIÓN CON OTRAS HERRAMIENTAS (PROS Y CONTRAS)

En la actualidad existe una amplia gama de herramientas que permiten dar soporte a MDA. Se han seleccionado alguna de ellas para realizar una evaluación de las mismas.

A continuación, se explica brevemente el comportamiento de las herramientas seleccionadas, y cuál ofrece un mejor rendimiento.

3.9.1. Eclipse

Eclipse nace como un entorno de desarrollo para JAVA y se ha convertido en una plataforma que permite el desarrollo y la integración de herramientas de desarrollo. Si bien no puede ser considerada como una herramienta MDA, es una herramienta con la que se pueden generar herramientas MDA. Fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas VisualAge.

En el año 2001 se forma un consorcio para el desarrollo futuro de Eclipse como código abierto, y finalmente, en el año 2003, este consorcio se independiza de IBM convirtiéndose en la fundación Eclipse.

Se ha incluido Eclipse en este estudio, porque puede ser utilizado, no sólo en el desarrollo de productos software, sino también en el desarrollo de herramientas que permitan construir productos software. Se puede decir también que Eclipse es un marco de trabajo para el modelado y la integración de datos permitiendo almacenar metamodelos y meta-datos.

La principal ventaja de Eclipse radica en que su arquitectura está diseñada de forma que la mayoría de la funcionalidad proporcionada está localizada en plug-ins o en un conjunto de plugins relacionados. Esto hace a Eclipse una herramienta extensible.

Del conjunto de plug-ins de Eclipse, vamos a centrarnos en Eclipse Modeling Framework (EMF). EMF es una herramienta para la generación de código a partir de modelos definidos por el usuario. A través de ella se pueden generar plug-ins que proporcionen editores de modelos. Cabe destacar dos plug-ins en concreto, el plug-in UML2 permite definir estereotipos de UML y el plug-in Graphical Modeling Framework (GMF) basado en EMF, que permite generar editores gráficos de modelos.

3.9.1.1. Evaluación de características

- Niveles que cubre: Utilizando EMF se pueden definir niveles de abstracción, ya que Ecore es una implementación de eMOF (Essential MOF), por lo que nos permite definir CIMs (Modelo independiente de la computación), PIMs (Modelo independiente de la plataforma) y PSMs (Modelo específico de la plataforma).

- Grado de generación de código: El lenguaje en el que se genera código por defecto con Eclipse, es JAVA, aunque también existen otros plug-ins que permiten generar código en otros lenguajes. El código generado puede ser verificado y, en caso necesario, se pueden redefinir los modelos y generar el código de nuevo.

- Transformaciones: Se pueden obtener diferentes PSMs a partir del mismo PIM de forma automática. Estos PSMs se pueden pasar al generador de código y se puede generar el código automáticamente. Para realizar las transformaciones entre modelos se deben utilizar plug-ins que implementen dichas transformaciones. Para definirlas se puede utilizar cualquier lenguaje que entienda Ecore.

- Grado de interacción con el usuario: el usuario participa activamente en las etapas de definición de los meta-modelos y los meta-datos. En las etapas de

definición de modelos y datos disminuye la participación del usuario ya que permite tomar como base los meta-modelos definidos con anterioridad, trasladándose de esta manera las características definidas.

- Tipo de Transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y de PSM a código). Para realizar las transformaciones horizontales se podrían definir nuevos plug-ins que las implementen.

- Lenguaje de almacenamiento y gestión de modelos: se almacenan en formato XMI. Esto permite definir los diferentes modelos en cualquier herramienta, por ejemplo Rational Rose, almacenarlo con formato XMI y exportarlo a EMF.

- Plataformas y tecnologías soportadas: soporta la plataforma de JAVA.

- Ámbito de aplicación: es una herramienta orientada al desarrollo de sistemas de propósito general.

- Uso de estándares: utiliza XMI, UML y MOF.

- Extensibilidad: se puede extender fácilmente por medio de plug-ins, ya que la ventaja principal de Eclipse reside en que cada plug-in puede definir puntos de extensión a los que otros plug-ins se pueden conectar, permitiendo de esta manera incrementar la funcionalidad. Así el usuario podría crear el plug-in necesario para cubrir nuevos requerimientos.

- Usabilidad: es una herramienta amigable. Permite definir los meta-modelos y meta-datos utilizando otras herramientas, por lo que se debe tener en cuenta que el usuario deberá tener conocimiento de las mismas.

- Interoperabilidad entre herramientas: se puede adaptar a otras herramientas, por medio de plug-ins existentes o desarrollar nuevos plug-ins.

3.9.2. AndroMDA

AndroMDA es una herramienta de generación de código que toma modelos UML en formato XMI como entrada y, genera código como salida en cualquier lenguaje de programación. Nace en el año 2002 como una iniciativa de Matthias Bohlen y en el año 2003 adquiere el nombre de AndroMDA debido a que toma las bases del paradigma MDA. AndroMDA está compuesta por "cartridges".

Los cartridges son un tipo especial de plug-ins, donde se definen los meta-modelos y reglas de transformación para transformar elementos del modelo de acuerdo al meta-modelo. En muchos casos, un cartridge puede contener solamente los meta-modelos, ya que las reglas de transformación pueden ser manejadas por muchos cartridges y pueden ser contenidas en un cartridge común.

Se ha seleccionado AndroMDA para este estudio, porque permite la generación de código en diferentes lenguajes de programación.

3.9.2.1. Evaluación de características

- Niveles que cubre: implementa los niveles PIM y PSM, permitiendo realizar transformaciones desde un PIM a varios PSMs. Las transformaciones entre los diferentes modelos se realizan de forma automática.

- Grado de generación de código: utiliza la tecnología de cartridge, que le permite obtener modelos en diferentes plataformas. A partir de los PSMs se puede generar código en el lenguaje de programación que se desee siempre que se tenga el cartridge correspondiente. Además se debe tener en cuenta que AndroMDA también permite generar código desde un PIM directamente.

- Transformaciones: una vez que se tienen definidos los cartridges correctamente, las transformaciones entre los modelos de diferentes niveles son automáticas. El lenguaje utilizado para definir los cartridges en AndroMDA es JAVA.

- Grado de interacción con el usuario: se puede decir que la interacción del usuario es alta, ya que debe participar activamente en todas las etapas del desarrollo.

- Tipos de transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y de PSM a código). Para realizar las transformaciones horizontales se podría, o bien modificar un cartridge existente, o definir nuevos cartridges que las implementen.

- Lenguaje de almacenamiento y gestión de modelos: los modelos generados son almacenados en XMI, permitiendo realizar el intercambio de modelos entre los diferentes niveles.

- Plataformas y tecnologías soportadas: utiliza herramientas adicionales como Apache's Maven, que simplifica su uso, ya que permite a los cartridges de AndroMDA para los diferentes lenguajes. Además se pueden utilizar herramientas CASE que permitan generar los diagramas en formato XMI. Utiliza Nhibernate para asegurar la persistencia de los objetos. También posee cartridges que permiten generar código en plataformas como Java, .Net o PHP.

- Ámbito de aplicación: está pensada para el desarrollo de software orientado a servicios y SI Web.

- Uso de estándares: es MOF-complaint, permitiendo así soportar estándares como UML y XMI.

- Extensibilidad: se extiende creando nuevos cartridges que soporten nuevos requisitos.

- Usabilidad: en cuanto a la facilidad de uso, se debe considerar que AndroMDA se implementa a través de cartridges que se pueden integrar con otras herramientas de desarrollo, por ejemplo Visual .Net o JAVA, por ello es

necesario que el usuario posea los conocimientos necesarios en el lenguaje seleccionado.

- Interoperabilidad entre herramientas: acepta como entrada los PIMs generados con otras herramientas, como por ejemplo ArgoUML. Además, como se ha dicho anteriormente, tiene cartridges definidos para la mayoría de los lenguajes de programación.

3.9.3. ArgoUML

ArgoUML es una herramienta para el desarrollo de SI. Nace a finales de la década de los 90 como una iniciativa del grupo liderado por Nora Koch, fue concebido como una herramienta CASE para realizar el análisis y el diseño en el desarrollo de sistemas orientados a objeto, y evolucionó hacia los sistemas informáticos Web.

Se ha seleccionado ArgoUML para este estudio, ya que es una herramienta que ha nacido como propuesta de la comunidad de desarrolladores Open Source. Está desarrollado utilizando el lenguaje JAVA.

3.9.3.1. Evaluación de características

- Niveles que cubre: cubre todas las etapas del ciclo de vida de un sistema, por lo que se pueden definir modelos para el nivel CIM, PIM y PSM y posteriormente su paso a código.

- Grado de generación de código: genera código en el lenguaje JAVA por defecto, aunque existen módulos auxiliares que permiten generar código en lenguajes como C# y C++.

- Transformaciones: brinda soporte para realizar transformaciones de PIM a PSM y de PSM a código. Las transformaciones entre los diferentes modelos se realizan de forma automática de acuerdo a lo definido por el usuario. Como se ha dicho anteriormente, ArgoUML está definido en JAVA, por lo que las transformaciones deben definirse utilizando ese lenguaje.

- Grado de interacción con el usuario: el usuario debe participar activamente en la definición de los diferentes modelos.

- Tipo de transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y PSM a código). Para realizar transformaciones horizontales se deberían definir las transformaciones, agregándolas como funcionalidad a la herramienta.

- Lenguaje de almacenamiento y gestión de modelos: la información de los diferentes modelos se almacena en formato XMI y los gráficos en PGML (Precision Graphics Markup Language) basado en XML.

- Plataformas y tecnologías soportadas: para su funcionamiento no necesita ninguna herramienta adicional, utiliza la plataforma de JAVA.

- Ámbito de aplicación: ArgoUML está pensada para el desarrollo de SI, concretamente en los SI Web. Posteriormente, con la evolución de los SI Web se adaptó al desarrollo de software orientado a servicios.

- Uso de estándares: el repositorio de la herramienta está implementado con JMI (Interface Metadata de JAVA) y basado en MOF, tomando la especificación de UML 1.4. También tiene soporte para OCL y XMI.

- Extensibilidad: Es un proyecto Open Source, al que se le puede seguir agregando funcionalidad.

- Usabilidad: Tiene una interfaz gráfica muy fácil de entender y manejar, es muy intuitiva. A través de la herramienta se pueden realizar todas las etapas del desarrollo.

- Interoperabilidad entre herramientas: los PIMs realizados con ArgoUML pueden ser utilizados como entrada para AndroMDA.

3.9.4. Conclusiones

Tras haber evaluado las características definidas para las herramientas seleccionadas, se ha llegado a las siguientes conclusiones:

- Al utilizar Eclipse con el plug-in EMF se pueden generar editores de meta-modelos de forma fácil si éstos se basan en UML. Sin embargo, no soporta la definición de estereotipos para definir extensiones UML. Además, con el uso del plug-in GMF se pueden definir los metamodelos y los modelos correspondientes facilitando así el modelado al usuario.

- AndroMDA tiene definidos los cartridges para la mayoría de los lenguajes de desarrollo, por lo que la implementación en distintas plataformas es muy fácil de realizar. Los cartridges se basan en el meta-modelo de UML. Si se desea realizar modelos basados en perfiles de UML que utilicen estereotipos, se deben modificar los cartridges o crear nuevos. Hay que tener en cuenta que se deberían crear cartridges para cada uno de los lenguajes de desarrollo con los que se desee trabajar y para cada meta-modelo que se desee introducir. La modificación o creación de nuevos cartridges no es trivial, se debe tener conocimiento del lenguaje para el cual se está desarrollando el cartridge y de la manera de definir las transformaciones en el mismo.

- ArgoUML se basa en el meta-modelo de UML. Por tanto, al igual que ocurre con AndroMDA, si los modelos que se desean generar son acordes a UML, éstos son fáciles de implementar. Pero si se desea realizar modelos basados en perfiles UML, se debe codificar la funcionalidad necesaria que permita recoger los estereotipos. Si bien ArgoUML únicamente genera código JAVA, los modelos generados con esta herramienta pueden servir de entrada a otras herramientas, como por ejemplo, AndroMDA.

Este estudio ha permitido concluir, que de las herramientas evaluadas, Eclipse es la que mejor se adapta a nuestras necesidades. Eclipse complementado con los plug-ins de EMF, GMF y UML2 permite de manera sencilla y gráfica generar editores de modelos en base a los meta-modelos definidos.

4. GESTIÓN DEL PROYECTO

4.1. INTRODUCCIÓN

En este capítulo se va a explicar como se ha llevado a cabo la gestión del proyecto del Definidor Visual.

El objetivo de la gestión del proyecto es organizar y administrar los recursos de tal manera que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y coste definidos.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Ciclo de vida.
- Organización del proyecto.
- Planificación del sistema.
- Estimación de costes del sistema.

4.2. CICLO DE VIDA ITERATIVO POR PROTOTIPOS

En este apartado, se va a explicar el ciclo de vida del desarrollo del software que se ha elegido para el proyecto, pero antes se va a definir de manera formal dicho concepto.

El ciclo de desarrollo del software es: *"El período de tiempo que comienza con la decisión de desarrollar un producto software y finaliza cuándo se ha entregado éste. Este ciclo incluye, en general, una fase de requisitos, fase de diseño, fase de implantación, fase de prueba, y a veces, fase de instalación y aceptación [AECC86]."* **[DE AMESCUA 95]**

Para el desarrollo del proyecto se ha llevado a cabo un ciclo de vida iterativo e incremental por prototipos, que se basa en la evolución de prototipos ejecutables.

En el ciclo de vida iterativo a cada iteración se reproduce el ciclo de vida en cascada a menor escala. Los objetivos de una iteración, se establecen en función de la evaluación de las iteraciones precedentes. En la siguiente figura se muestra de manera gráfica, el ciclo de vida iterativo llevado a cabo en el desarrollo del proyecto.

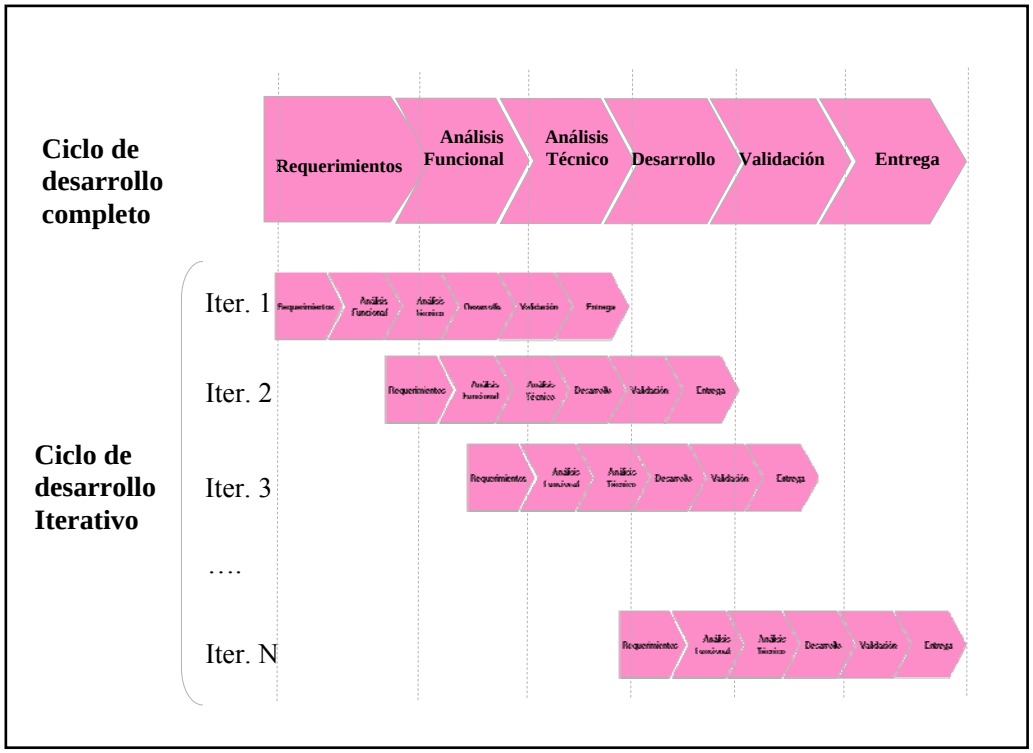


Figura 4.1. Ciclo de vida iterativo por prototipos.

4.3. ORGANIZACIÓN DEL PROYECTO

A la hora de llevar a cabo cualquier proyecto, se hace necesaria la tarea de realizar una planificación, detallando el tiempo estimado necesario para cumplir las distintas actividades y tareas que forman parte del proyecto, así como los recursos asignados a cada una de ellas. Por ello, se ha desarrollado la planificación en función del ciclo de vida seleccionado anteriormente, ciclo de vida iterativo por prototipos.

Los recursos humanos (personas y roles) que han intervenido en el desarrollo del proyecto, se muestran a continuación, así como las funciones de las que se ha encargado cada uno de ellos:

- **Jefe de Proyecto:** Se encarga de la gestión del proyecto, su organización, planificación y supervisión a lo largo de todo el desarrollo del mismo.

- **Analista:** Se encarga de obtener y redactar los requisitos, además de modelar los procesos y tareas a codificar.

- **Diseñador:** Su tarea es el diseño de la arquitectura del sistema y del plan de verificación y validación.

- **Programador:** Se encarga de la codificación del sistema, así como de la ejecución de las pruebas necesarias del mismo.

El RBS es una lista jerárquica de requerimientos de recursos (tanto humanos como materiales) necesaria para la planificación de un proyecto. Esta técnica de organización de proyectos software tiene por objeto representar la organización humana del proyecto, su estructura, responsabilidades, etc., así como la estructura de recursos tecnológicos y materiales.

A continuación se muestra el RBS de este proyecto:

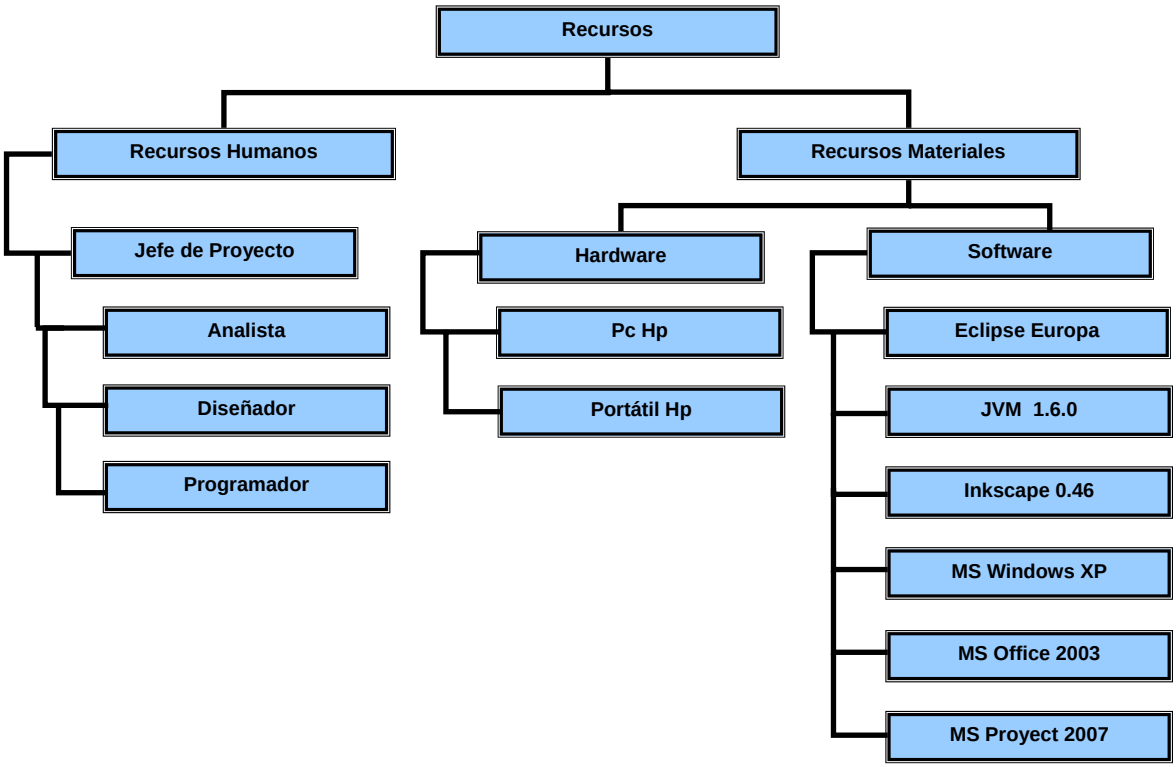


Figura 4.2. RBS.

4.4. PLANIFICACIÓN DEL SISTEMA

La duración total de este proyecto han sido 200 días laborables, desde el 15 de Septiembre de 2008, fecha en la que se inicia el desarrollo de la aplicación, hasta el 19 de Junio de 2009, fecha en la que la aplicación se da por finalizada y es entregada al cliente.

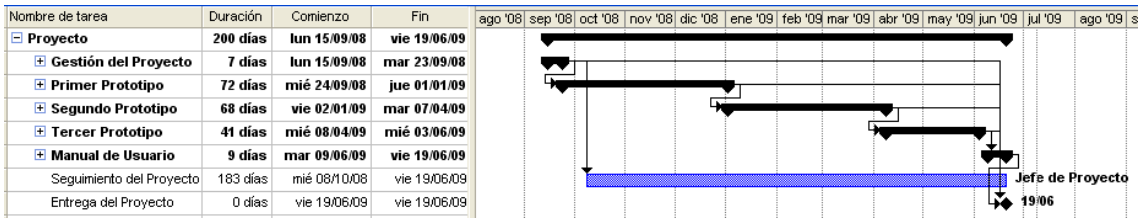


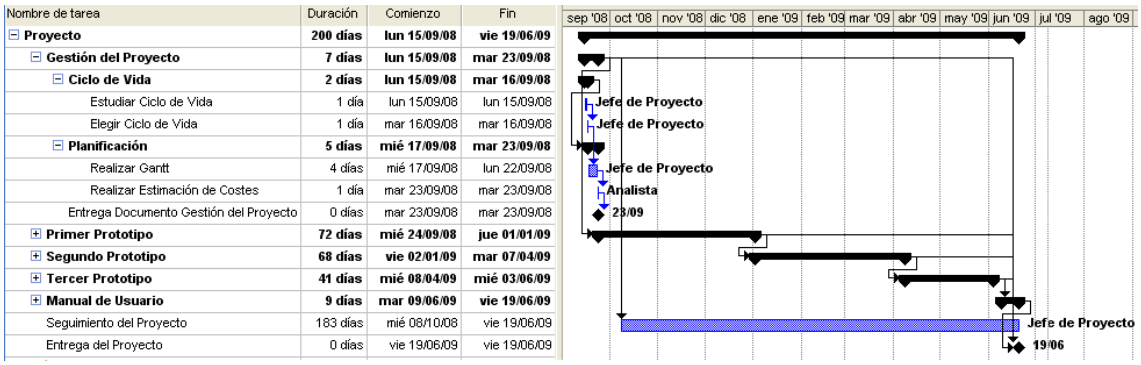
Figura 4.3. Diagrama de Gantt.

El proyecto comienza con la fase de gestión del proyecto. Ésta es llevada a cabo por el jefe de proyecto, el cual realiza un estudio sobre los ciclos de vida existentes, selecciona el más idóneo para el desarrollo de la aplicación, documentando claramente las razones y motivos de su elección.

A continuación, se realiza el diagrama Gantt en el cual figurarán todas las tareas y actividades necesarias para realizar el desarrollo de la aplicación, así como la duración estimada de cada una de ellas y los recursos necesarios que se utilizaran en cada una de las tareas. En este diagrama las tareas ya aparecen como finalizadas, puesto que ya se ha llegado al final del proyecto.

Por último, el jefe de proyecto realiza el documento de Gestión de Proyecto, y comienza el seguimiento de éste, el seguimiento se llevará a cabo en paralelo a todas las actividades a realizar hasta la entrega final de la aplicación.

En la siguiente figura se muestran las tareas que conforman la Gestión de Proyecto:



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Ciclo de Vida	**2 días**	**lun 15/09/08**	**mar 16/09/08**
Estudiar Ciclo de Vida	1 día	lun 15/09/08	lun 15/09/08
Elegir Ciclo de Vida	1 día	mar 16/09/08	mar 16/09/08
Planificación	**5 días**	**mié 17/09/08**	**mar 23/09/08**
Realizar Gantt	4 días	mié 17/09/08	lun 22/09/08
Realizar Estimación de Costes	1 día	mar 23/09/08	mar 23/09/08
Entrega Documento Gestión del Proyecto	0 días	mar 23/09/08	mar 23/09/08
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.4. Diagrama de Gantt. Gestión del Proyecto.

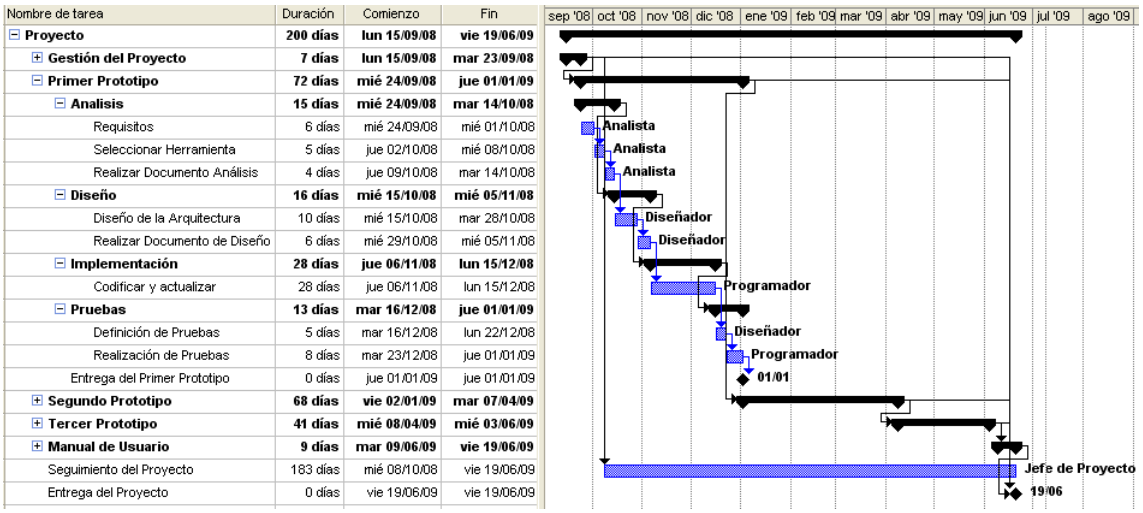
Tras los pasos anteriores, toma el control el analista y comienza a desarrollar el primer prototipo. Lo primero que debe hacer es recopilar los requisitos del usuario para obtener los requisitos software y seleccionar las herramientas necesarias para satisfacerlos. Es muy importante que antes de crear un prototipo, los analistas y el cliente o tutor deben trabajar juntos para identificar los requerimientos que tienen que satisfacer.

El diseñador lee la documentación generada por el analista y diseña la arquitectura de la aplicación, definiendo para ello todo lo necesario para la implementación.

El siguiente paso es realizado por el programador, que implementará la aplicación para satisfacer los requisitos del usuario lo más fielmente posible al diseño realizado.

El primer prototipo se concluye con una batería de pruebas (validación), para verificar su correcto funcionamiento y la evaluación por parte del cliente o tutor. Es responsabilidad del cliente trabajar con el prototipo y evaluar sus características y operación. La experiencia del sistema, bajo condiciones reales, permite obtener la familiaridad indispensable para determinar los cambios o mejoras que sean necesarios, así como las características inadecuadas.

Las pruebas serán redactadas por el diseñador y ejecutadas por el programador para comprobar el correcto funcionamiento de la aplicación.



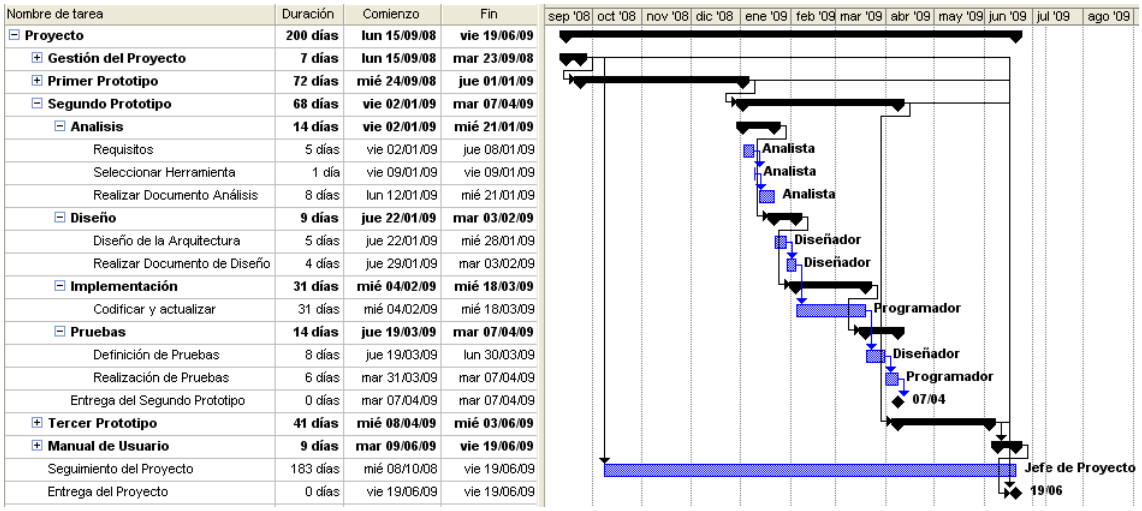
Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Analisis	**15 días**	**mié 24/09/08**	**mar 14/10/08**
Requisitos	6 días	mié 24/09/08	mié 01/10/08
Seleccionar Herramienta	5 días	jue 02/10/08	mié 08/10/08
Realizar Documento Análisis	4 días	jue 09/10/08	mar 14/10/08
Diseño	**16 días**	**mié 15/10/08**	**mié 05/11/08**
Diseño de la Arquitectura	10 días	mié 15/10/08	mar 28/10/08
Realizar Documento de Diseño	6 días	mié 29/10/08	mié 05/11/08
Implementación	**28 días**	**jue 06/11/08**	**lun 15/12/08**
Codificar y actualizar	28 días	jue 06/11/08	lun 15/12/08
Pruebas	**13 días**	**mar 16/12/08**	**jue 01/01/09**
Definición de Pruebas	5 días	mar 16/12/08	lun 22/12/08
Realización de Pruebas	8 días	mar 23/12/08	jue 01/01/09
Entrega del Primer Prototipo	0 días	jue 01/01/09	jue 01/01/09
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.5. Diagrama de Gantt. Primer Prototipo.

El analista comienza la segunda iteración recogiendo los comentarios del usuario en la evaluación del primer prototipo. Los cambios al prototipo son planificados con el cliente antes de llevarlos a cabo, sin embargo es el analista el responsable de tales modificaciones.

Como en el primer prototipo, el analista documenta los nuevos requisitos y selecciona las herramientas necesarias, el diseñador lee la documentación aportada por el analista y diseña las nuevas funcionalidades, y el programador codifica los nuevos procedimientos de acuerdo a las especificaciones anteriores.

Se realizan las pruebas para verificar que todo funciona correctamente y se le muestra al cliente para que evalúe el nuevo prototipo.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Analisis	**14 días**	**vie 02/01/09**	**mié 21/01/09**
Requisitos	5 días	vie 02/01/09	jue 08/01/09
Seleccionar Herramienta	1 día	vie 09/01/09	vie 09/01/09
Realizar Documento Análisis	8 días	lun 12/01/09	mié 21/01/09
Diseño	**9 días**	**jue 22/01/09**	**mar 03/02/09**
Diseño de la Arquitectura	5 días	jue 22/01/09	mié 28/01/09
Realizar Documento de Diseño	4 días	jue 29/01/09	mar 03/02/09
Implementación	**31 días**	**mié 04/02/09**	**mié 18/03/09**
Codificar y actualizar	31 días	mié 04/02/09	mié 18/03/09
Pruebas	**14 días**	**jue 19/03/09**	**mar 07/04/09**
Definición de Pruebas	8 días	jue 19/03/09	lun 30/03/09
Realización de Pruebas	6 días	mar 31/03/09	mar 07/04/09
Entrega del Segundo Prototipo	0 días	mar 07/04/09	mar 07/04/09
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.6. Diagrama de Gantt. Segundo Prototipo.

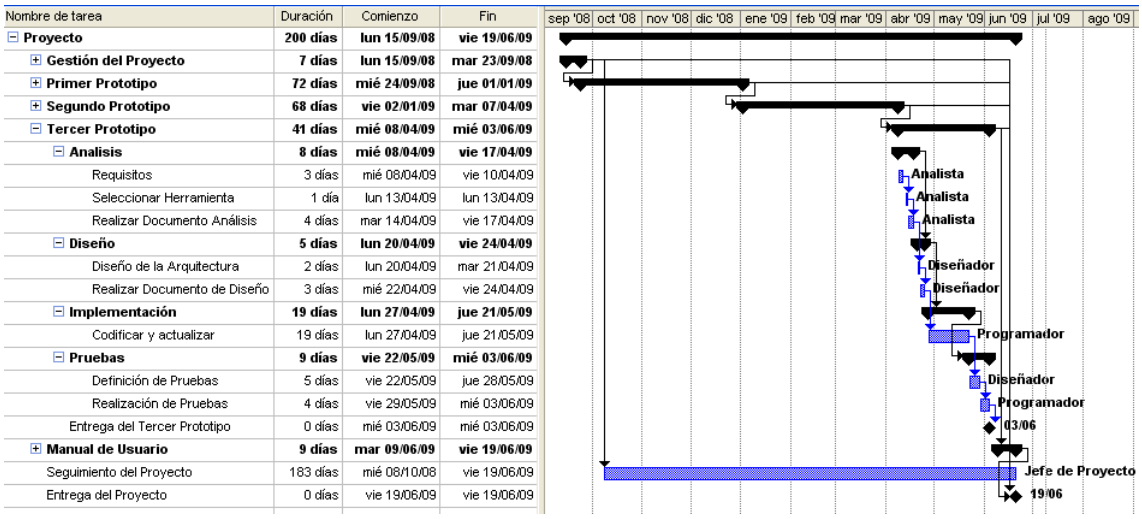
Finalmente se comienza con el tercer y último prototipo, en el que se intentarán satisfacer todos los requisitos expuestos por el cliente a lo largo del proyecto.

El analista recoge todos los requisitos y comentarios del usuario y elabora la documentación definitiva, si es necesario utilizar una nueva herramienta de desarrollo la buscará y seleccionará.

El diseñador plantea las interfaces y funcionalidades que faltan y el programador implementa la aplicación definitiva.

La fase de pruebas es crucial para determinar que los requerimientos han sido satisfechos y que el sistema se comporta como se esperaba.

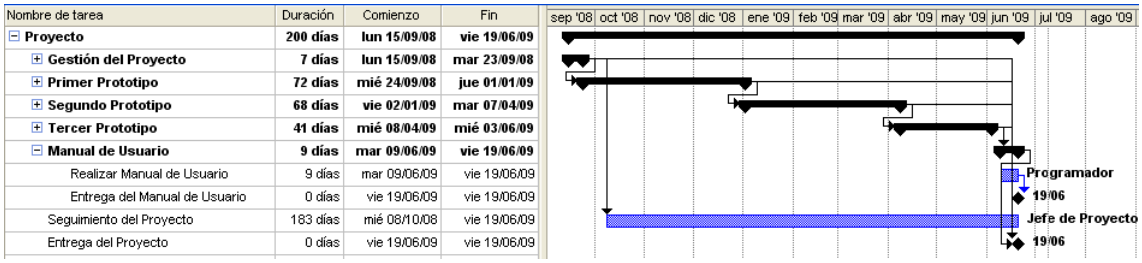
El prototipo debe pasar las pruebas de funcionalidad para validar su correcto funcionamiento y la evaluación por parte del cliente, siendo éste el que certifique la aceptación del sistema que utiliza.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Analisis	**8 días**	**mié 08/04/09**	**vie 17/04/09**
Requisitos	3 días	mié 08/04/09	vie 10/04/09
Seleccionar Herramienta	1 día	lun 13/04/09	lun 13/04/09
Realizar Documento Análisis	4 días	mar 14/04/09	vie 17/04/09
Diseño	**5 días**	**lun 20/04/09**	**vie 24/04/09**
Diseño de la Arquitectura	2 días	lun 20/04/09	mar 21/04/09
Realizar Documento de Diseño	3 días	mié 22/04/09	vie 24/04/09
Implementación	**19 días**	**lun 27/04/09**	**jue 21/05/09**
Codificar y actualizar	19 días	lun 27/04/09	jue 21/05/09
Pruebas	**9 días**	**vie 22/05/09**	**mié 03/06/09**
Definición de Pruebas	5 días	vie 22/05/09	jue 28/05/09
Realización de Pruebas	4 días	vie 29/05/09	mié 03/06/09
Entrega del Tercer Prototipo	0 días	mié 03/06/09	mié 03/06/09
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.7. Diagrama de Gantt. Tercer Prototipo.

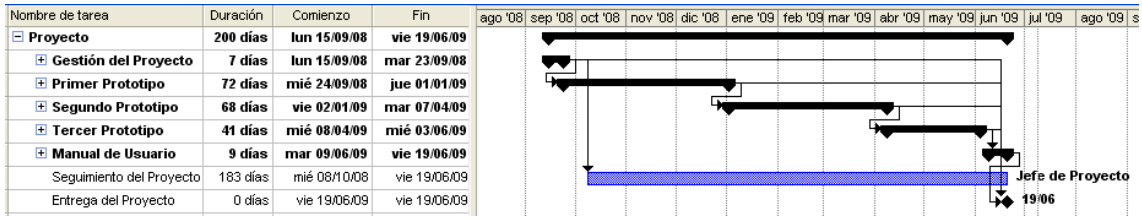
Finalmente se genera el manual de usuario indicando todos los aspectos de la aplicación, tanto sus funciones como la manera de ejecutarlas o seleccionarlas, para que el usuario final u otra persona sea capaz de utilizar la nueva aplicación sin problemas, y no tener que recurrir a los desarrolladores.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Realizar Manual de Usuario	9 días	mar 09/06/09	vie 19/06/09
Entrega del Manual de Usuario	0 días	vie 19/06/09	vie 19/06/09
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.8. Diagrama de Gantt. Manual de Usuario.

El último paso consiste en la entrega de la documentación y el software desarrollado al cliente y de esta manera se da por concluido el proyecto.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**200 días**	**lun 15/09/08**	**vie 19/06/09**
Gestión del Proyecto	**7 días**	**lun 15/09/08**	**mar 23/09/08**
Primer Prototipo	**72 días**	**mié 24/09/08**	**jue 01/01/09**
Segundo Prototipo	**68 días**	**vie 02/01/09**	**mar 07/04/09**
Tercer Prototipo	**41 días**	**mié 08/04/09**	**mié 03/06/09**
Manual de Usuario	**9 días**	**mar 09/06/09**	**vie 19/06/09**
Seguimiento del Proyecto	183 días	mié 08/10/08	vie 19/06/09
Entrega del Proyecto	0 días	vie 19/06/09	vie 19/06/09

Figura 4.9. Diagrama de Gantt. Entrega.

4.5. ESTIMACIÓN DE COSTES DEL SISTEMA

Para calcular el coste estimado del proyecto se deben tener en cuenta tanto los gastos materiales como los costes humanos del desarrollo. En las cantidades utilizadas se ha tenido en cuenta el IVA, que ya está añadido en las mismas.

A continuación se muestra el presupuesto en forma de tabla de las herramientas utilizadas para el desarrollo del proyecto:

PROGRAMAS	LICENCIA
Eclipse Europa	Gratuita
JVM 1.6.0	Gratuita
Inkscape 0.46	Gratuita
MS Windows XP	123,50 €
MS Office 2003	415 €
MS Proyect 2007	779 €
TOTAL	1.317,5€

Figura 4.10. Tabla Herramientas / Coste del proyecto.

Para calcular el coste humano se ha tomado una jornada laboral de 8 horas, distribuidas en horario de 9:00 a 14:00 y 16:00 a 19:00. El proyecto ha sido desarrollado por un sólo ingeniero, pero ha realizado las distintas funciones anteriormente especificadas. A continuación se muestra la relación entre las funciones que se han adoptado, el tiempo invertido en cada una de ellas y su coste:

RECURSOS HUMANOS	HORAS	PRECIO/HORA	TOTAL
Jefe de Proyecto	488	35	17.080€
Analista	304	30	9.120€
Diseñador	384	30	11.520€
Programador	840	25	21.000€
SUBTOTAL			58.720€
I.V.A.		16%	9.395,2€
TOTAL			68.115,2€

Figura 4.11. Tabla Recursos Humanos / Coste del proyecto.

El coste total de los recursos humanos del proyecto se obtiene sumando lo que recibe la persona que se ha encargado de su desarrollo. Como se puede observar en la anterior tabla la cantidad asciende a 68.115,2 €.

El coste total del proyecto se calcula mediante la suma de los costes materiales, más los costes humanos:

RECURSOS	TOTAL
Recursos Materiales	1.317,5€
Recursos Humanos	68.115,2€
TOTAL	69.432,7€

Figura 4.12. Tabla Recursos / Coste del proyecto.

Como se puede observar en la tabla anterior, el coste total del proyecto asciende a la cantidad de **69.432,7 €**, IVA incluido.

5. ANÁLISIS DEL SISTEMA

5.1. INTRODUCCIÓN

En este capítulo se va a explicar cómo se ha llevado a cabo el análisis de la aplicación Definidor Visual.

Con el análisis de la aplicación se pretende obtener una visión inicial de las necesidades de modelado que servirán de base para el diseño y posterior construcción del Definidor Visual.

A partir de la documentación recopilada en la etapa de Análisis, se establece el modelado de la arquitectura del Definidor Visual, se genera el catálogo de requisitos a satisfacer y en base a ellos, se crea una descripción de cómo aplicar el estándar de modelado de procesos de negocio BPMN para obtener la herramienta gráfica.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Identificación de las necesidades del sistema.
- Consideraciones de entorno.
- Usuarios del definidor visual.
- Especificación de requisitos software.
- Aplicación del estándar de especificación BPMN.
- Modelado de la arquitectura.

5.2. IDENTIFICACIÓN DE LAS NECESIDADES DEL SISTEMA

Con la herramienta gráfica de modelado de procedimientos se pretende que determinados usuarios puedan modelizar cualquier procedimiento, mediante la técnica "drag and drop" para situar los diferentes elementos sobre una pizarra.

Más concretamente, la herramienta de definición visual debe cubrir las siguientes necesidades:

- Incorporación de una pizarra con una paleta de objetos que se puedan dibujar en la pizarra mediante la técnica "drag and drop".
- Definición de procedimientos en distintos idiomas.
- Definición de fases.
- Definición de tareas y bloques.
- Definición de eventos, condiciones y conexiones entre objetos.
- Definición de documentos, componentes y notas.
- Definición y modificación de las propiedades de los elementos.
- Asociación de iconos diferentes a cada elemento de la paletta.
- Definición de relaciones entre elementos.
- Generación de un fichero XML con información de todos los objetos de la pizarra.

5.3. CONSIDERACIONES DEL ENTORNO

Para el desarrollo de la herramienta se usará JAVA J2EE como tecnología principal. Este se orientará a minimizar en la medida de lo posible su dependencia con herramientas externas y obtener una mayor autonomía en el funcionamiento de la aplicación.

Para el desarrollo del definidor visual utilizaremos como entorno de desarrollo integrado (IDE) Eclipse Europa con las librerías de EMF, plataforma de modelado Eclipse, y GMF, herramienta de modelado gráfico.

La definición de procedimientos respetará el estándar de especificación BPMN 1.0 y los datos relativos a los mismos serán almacenados en un fichero XML compatible con dicha norma.

5.4. USUARIOS DEL DEFINIDOR VISUAL

Los usuarios que harán uso de esta aplicación son los siguientes:

- **Definidor:** Usuario que realiza tareas de modelización y mantenimiento de los procedimientos.

- **Administrador:** Usuario especial que se hará cargo de la gestión y administración en general de la aplicación.

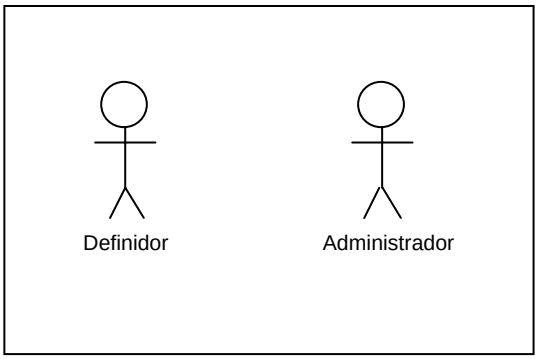


Figura 5.1. Usuarios de la Aplicación.

5.5. ESPECIFICACIÓN DE REQUISITOS SOFTWARE (FUNCIONALES Y NO FUNCIONALES)

En el presente apartado se lleva a cabo la definición de los requisitos concernientes al definidor visual, a partir de la información recopilada en los anteriores apartados referentes al Análisis de la aplicación.

El objetivo es obtener un listado lo más preciso y detallado posible de los requisitos que sirva como punto de referencia para el diseño y posterior desarrollo de la herramienta gráfica.

5.5.1. Requisitos funcionales

Identificador	Descripción
RF-001	**Multi-idioma**: Definición de procedimientos en diversos idiomas.

Identificador	Descripción
RF-002	**Pizarra** para situar los diferentes elementos que conforman un procedimiento mediante la **técnica "drag and drop"**.

Identificador	Descripción
RF-003	**Paleta** con una serie de gavetas que contendrán los elementos disponibles que podrán ser dibujados mediante la técnica "drag and drop" en la pizarra.

Identificador	Descripción
RF-004	**Edición de las propiedades de un elemento** mediante un cuadro de diálogo que se mostrará cuando se haga doble clic sobre él o bien cuando se incorpore como nuevo elemento en el procedimiento.

Identificador	Descripción
RF-005	**Distinto icono** para la representación gráfica de los objetos de la paleta.

Identificador	Descripción
RF-006	**Incorporar gaveta** para poder dibujar el **elemento pool**.

Identificador	Descripción
RF-007	**Incorporar gaveta** para poder dibujar los elementos de tramitación: **fase, bloque, tarea simple y documento.**

Identificador	Descripción
RF-008	**Incorporar gaveta** para poder dibujar eventos: **evento inicio, evento fin y evento intermedio.**

Identificador	Descripción
RF-009	**Incorporar gaveta** para poder dibujar los elementos de conexión: **flujo de secuencia, asociación y asociación de notas.**

Identificador	Descripción
RF-010	**Incorporar gaveta** para poder dibujar los elementos de condición: **condición exclusiva, condición no exclusiva, condición and y condición or.**

Identificador	Descripción
RF-011	**Incorporar gaveta** para poder dibujar el **elemento componente**.

Identificador	Descripción
RF-012	**Incorporar gaveta** para poder dibujar el **elemento texto anotación**.

Identificador	Descripción
RF-013	**Fases representadas gráficamente por un recuadro** en el que se ubicarán las tareas.

Identificador	Descripción
RF-014	**Reglar gráficamente tareas mediante arcos entre flechas**. El arco puede tener asociada una condición que se podrá editar al hacer doble clic sobre él.

Identificador	Descripción
RF-015	**Existirá un pool** que contendrá a todos los elementos del procedimineto.

Identificador	Descripción
RF-016	**Asignar una categoría** a un objeto de tramitación pinchando sobre el elemento definido en la pizarra, mediante la ventana de propiedades.

Identificador	Descripción
RF-017	**Asignar autorizaciones** al procedimiento y a las tareas pinchando sobre el elemento definido en la pizarra, mediante la ventana de propiedades.

Identificador	Descripción
RF-018	**Grabación de los datos** de un procedimiento **en un fichero XML compatible con la norma BPMN.**

5.5.2. Requisitos no funcionales

Identificador	Descripción
RNF-001	Se establecerá el estado en el que se encuentra un procedimiento: - En definición - Publicado o vigente. El procedimiento está accesible a los tramitadores para dar de alta expedientes. - No vigente. Los tramitadores no pueden crear nuevos expedientes pero si seguir con la tramitación de aquellos que se iniciaron previamente.

Identificador	Descripción
RNF-002	Una fase sólo tiene sentido en el ámbito del proyecto y tiene carácter informativo. No existirá un catálogo de fases.

Identificador	Descripción
RNF-003	Dentro de un procedimiento no se permitirá crear fases que no contengan tareas dentro de ellas.

Identificador	Descripción
RNF-004	Para eliminar una fase previamente ha de eliminarse las tareas que se ubican dentro de ella.

Identificador	Descripción
RNF-005	Los diferentes campos que definen una fase son: - Su orden dentro del procedimiento - Su obligatoriedad - Su nombre

Identificador	Descripción
RNF-006	Se puede crear una nueva tarea simple a partir de: - una tarea simple en blanco - una tarea ya definida en el procedimiento

Identificador	Descripción
RNF-007	Los atributos que definen una tarea son: - fase a la que pertenece - orden dentro de la fase a la que pertenece - obligatoriedad de la misma - prioridad como valor numérico - fase de definición en la que se encuentra: vigente o no vigente - si es accesible externamente por el ciudadano. Si lo es, indicar si es susceptible de consulta y/o edición.

Identificador	Descripción
RNF-008	Sobre una tarea se gestionarán las siguientes autorizaciones: - Ejecución - Consulta de datos - Opcionalmente, responsable de la misma

Identificador	Descripción
RNF-009	Se podrán asociar documentos a una tarea tanto si son generados desde la tarea o bien se aportan a la misma.

Identificador	Descripción
RNF-010	**Crear un número ilimitado de secciones para una tarea.** Por cada una de ellas se especificará: - número de orden en el que aparece - tipo de sección: cabecera, detalle o pie - modo de apertura: expandido o contraído - descripción

Identificador	Descripción
RNF-011	En el momento de incorporación de un componente a una tarea se podrán especificar: - condiciones de creación - condiciones de visibilidad - condiciones de acceso: modificable ó consulta. - y para cada campo: visibilidad, obligatoriedad y etiqueta

Identificador	Descripción
RNF-012	Un plazo se identifica por una descripción, una categoría, una fecha de inicio y una fecha de vencimiento que puede establecerse bien indicando su unidad temporal o bien en base a una fecha del procedimiento.

Identificador	Descripción
RNF-013	Se especificarán los perfiles de actuación de un procedimiento estableciendo para cada uno de ellos el conjunto de usuarios y acciones asignados al mismo.

Identificador	Descripción
RNF-014	Las condiciones pertenecen al procedimiento y pueden utilizarse en los siguientes casos: • Condicionar la ejecución de acciones asociadas a eventos propios de tareas, procedimientos y plazos. • Condicionar la visibilidad de componenetes en tareas o procedimientos. • Condicionar la asignación de expedientes o tareas a un grupo de usuarios.

Identificador	Descripción
RNF-015	No existirá limitaciones en cuanto al número de campos que puede contener un componente ni en cuanto al número de campos de un determinado tipo.

Identificador	Descripción
RNF-016	Los componentes incluirán lógica de validación en sus campos: - longitud máxima - valor por defecto - máscara de entrada - tipo de dato - obligatoriedad del campo

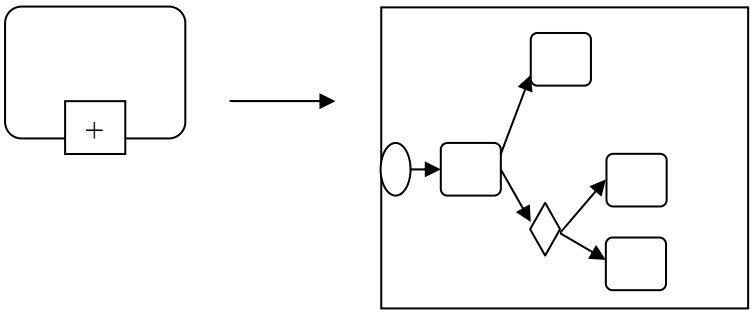
Identificador	Descripción
RNF-017	Los componentes que se incorporen a una tarea no estarán disponibles para el procedimiento.

Identificador	Descripción
RNF-018	Las categorías son específicas de cada entidad de tal manera que aquellos valores definidos para un procedimiento no pueden ser reutilizados en la entidad tarea.

5.6. APLICACIÓN DEL ESTÁNDAR DE ESPECIFICACIÓN BPMN

5.6.1. Correspondencia Definidor Visual – BPMN y su representación gráfica

5.6.1.1. Entidades de Tramitación

Elemento Definidor	Elemento BPMN
Fase	Artefacto "Group": Usado para propósitos de documentación o análisis y no afecta a la secuencia del flujo. 
Tarea simple	Objeto de flujo "Activity Task": Usado para definir actividades atómicas que realiza la organización. 
Bloque	Objeto de flujo "Activity Sub-Process": Usado para definir actividades no atómicas que realiza la organización. Existen distintos subprocesos pero en nuestro proyecto utilizaremos únicamente el tipo "subproceso embebido". Colapsado Expandido 

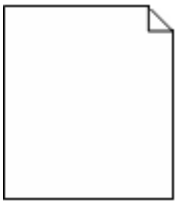
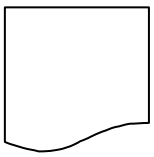
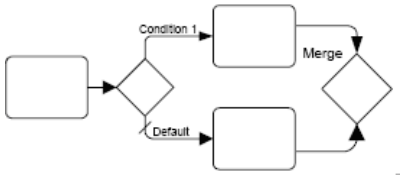
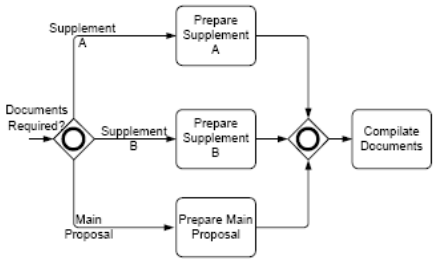
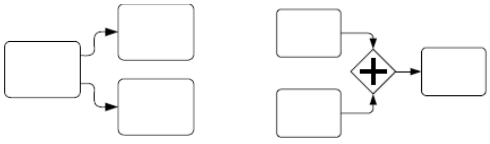
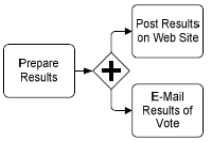
Documentos	Artefacto "Data-Object": Mecanismo para mostrar como las actividades requieren o producen objetos. 
Componente	Nuevo artefacto como extensión al BPD "Componente": Su objetivo es el mismo que se persigue con el artefacto "Data-Object". Su creación permitirá diferenciar los componentes y los documentos a través de respresntaciones gráficas distintas. 

Figura 5.2. Entidades de Tramitación Definidor - BPMN.

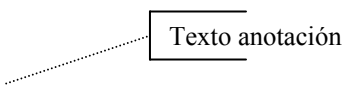
5.6.1.2. Entidades de Control

Elemento Definidor	Elemento BPMN
Condición para la divergencia o convergencia del flujo del proceso	Objeto de flujo "Gateway": Mecanismo para controlar la divergencia y convergencia de la secuencia de un flujo. - Exclusivas  - No exclusivas  - Divergencia y convergencia de caminos paralelos  

Evento	Objeto de flujo "Event": Mecanismo para expresar sucesos que ocurren durante el curso de un proceso de negocio y que afectan al flujo del proceso. - Start (de comienzo del expediente)  - Intermediate "Timer" (especificación y vencimiento de plazos)  - End (expediente archivado) 
Conexión entre objetos	"Connecting Objects": Mecanismo de conexión de los objetos entre sí para definir cómo progresa el flujo dentro de un proceso. - Flujo de secuencia: Especificación del orden de ejecución de las actividades.  - Asociación: Asociación de datos, textos u otros artefactos con flujos de objetos. 

Figura 5.3. Entidades de Control Definidor - BPMN.

5.6.1.3. Entidades de Apoyo

Elemento Definidor	Elemento BPMN
Nota	Artefacto "Annotation": Mecanismo para agregar información textual adicional para el lector del diagrama BPMN. 
Categoría	Todo objeto gráfico BPMN dispone de un atributo "Categories (0-n): String" para añadir categorías que pueden ser utilizadas para propósitos tales como el análisis y como

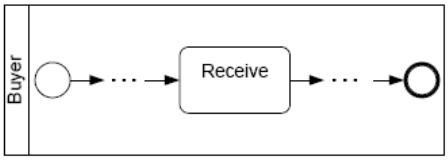
	fuente de información.
Grupo organizativo o rol	Swinlanes "Pool": Usado para indicar los participantes en el proceso. 

Figura 5.4. Entidades de Apoyo Definidor - BPMN.

5.6.2. Especificación de atributos para la representación gráfica

Acorde a las necesidades del proyecto, se especifican en color gris claro aquellos atributos adicionales a los especificados por el estándar BPMN que serán añadidos a las distintas entidades de tramitación.

5.6.2.1. Procedimiento

Atributo	Descripción
Código: Object	Identificador unívoco del procedimiento.
Version (0-1): Property	Número de versión del procedimiento.
Autor (0-1): String	Nombre del autor del procedimiento.
CreationDate (0-1): Date	Fecha de creación del procedimiento.
ModificationDate (0-1): Date	Fecha de la última modificación del procedimiento.
PublicationDate (0-1): Date	Fecha de la publicación del procedimiento.
Pools (1-n): Pool	Especificación de los distintos roles o entidades que participan en el trámite de un expediente de este procedimiento. En nuestro proyecto definiremos un Pool genérico donde se englobará la definición del procedimiento.
Estado: String	Especificación del estado en el que se encuentra el procedimiento: en definición, publicado o no vigente.

Name (1-n): String	Nombre del procedimiento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Category (0-n): String	Categoría/s asignadas al procedimiento desde un catálogo predefinido de categorías. Category[0] = categoría 1 ⋮ Category[n] = categoría n
Descripción adicional (0-n): String	Especificación de los cambios realizados si se trata de una nueva versión de un procedimiento anterior. Puede escribirse manualmente o bien vincular un documento especificando su ruta de ubicación. Modelado en forma de array de Property para permitir describir los cambios en los distintos idiomas cuando sean especificados manualmente. DA[0] = código idioma + descripción de los cambios realizados ⋮ DA[n] = código idioma + descripción de los cambios realizados
Documentación asociada (0-n): String	Especificación de documentación asociada al procedimiento en distintos formatos: página Web, pdf, etc. Cada elemento de la colección almacenará la ruta del documento o nombre de página Web asociado o bien se establecerá una relación con el gestor documental.
Referencia: String	Especificación del formato del valor Referencia para los expedientes del procedimiento. Se seleccionará desde un catálogo predefinido en el que se muestran los formatos aplicables a distintos objetos del procedimiento.
Situación administrativa externa (0-n): Property **Situación administrativa interna (0-n)**: Property	Especificación de los posibles valores que pueden tomar ambos atributos para el procedimiento. La descripción de una situación administrativa será multilingüe seleccionando los idiomas de un catálogo predefinido. <property name=”SAE/SAI”> <property name=”Cod_SAE/SAI”> //Código de la situación <property name=”Descripción> //Descripción multilingüe

	<property name=”Situación Inicial” idioma=”código idioma”/> …… <property name=”Initial Location” idioma=”código idioma”/> </property> </property> </property>
Fechas (0-n): Property	Especificación de aquellas fechas que tienen incidencia en el procedimiento (catálogo de fechas). Una fecha se identifica por: - Una denominación de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - Si se trata de un hito interno, externo o ninguno - Categoría/s asignadas <property name=”Fechas”> <property name=”FINICIO”> //codigo de la fecha <property name=”Denominación_Fecha”> //Denominación multilingüe <property name=”Fecha de inicio” idioma=”código idioma”/> …… <property name=”Start Date” idioma=”código idioma”/> </property> <property name=”HitoInterno”> ---- </property> <property name=”HitoExterno”> ---- </property> <property name=”Categorías”> <property name=”Tributos”/> ….. <property name=”Ciudadano”/> </property> </property> </property>

Secciones (0-n): Property	Especificación de las secciones en las que se divide la interfaz de usuario para determinar la ubicación de las entidades en ella. Una sección se identificar por: - Una descripción de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - Número de orden en el que aparece - Tipo de sección - Modo de apertura: expandido o contraído. <property name=”Secciones”> <property name=”Sección”> <property name=”Descripción”> //Descripción multilingüe <property name=”Datos administrativos” idioma=”código idioma”/> …… <property name=”Administrative Details” idioma=”código idioma”/> </property> <property name=”Orden”> ---- </property> <property name=”Tipo”> {detalle,cabecera,pie} </property> <property name=”Apertura”> {expandido,contraído} </property> </property> </property>
Condiciones (0-n): Property	Especificación de las condiciones que van a modelar el comportamiento de las entidades del procedimiento. Una condición se identifica por: - Una descripción de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - Una colección de expresiones concatenadas por operadores Y/O. <property name=”Condiciones”> <property name=”Condicion”> <property name=”Descripción”> //Descripción multilingüe <property name=”Si IBI e importe superior a 1000 euros” idioma=”código idioma”/> <property name=”If IBI and amount exceeding EUR 1000” idioma=”código idioma”/> </property>

	<property name="Expresiones"> <property name="Expresión"> <property name="("> {Ninguna, (, ((, (((, …} </property> <property name="Tipo"> ---- </property> <property name="Elemento"> ---- </property> <property name="Operador"> ---- </property> <property name="Valor"> ---- </property> <property name=")"> {Ninguna,),)),))), …} </property> <property name="Y/O"> {Ninguno, Y, O} </property> </property> </property>
Autorizaciones (0-n): Property	Definición de las autorizaciones sobre el procedimiento. Una autorización se identifica por un perfil de actuación y las acciones que éste puede realizar. <property name="Autorizaciones"> <property name="Autorización_1"> <property name="ROL1"/> //perfil de actuación <property name="Acciones"> <property name="ACCION_1"/> …… <property name="ACCION_N"/> </property> </property> </property>
Formato de numeración: String	Tipo de numeración que se aplicará a los expedientes del procedimiento. Se seleccionará de un conjunto predefinido de numeraciones posibles.
Grupo de numeración: String	Especificación del grupo de numeración que se aplicará a los expedientes del procedimiento.
Código interno: String	Especificación del código interno que se aplicará a los expedientes del procedimiento.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.

Figura 5.5. Atributos del Procedimiento.

5.6.2.2. Fase

Atributo	Descripción
Código: Object	Identificador unívoco de la fase en el procedimiento.
Obligatoriedad: String	Obligatoriedad de la fase dentro del procedimiento: SI/NO.
Descripción (1-n): String	Nombre descriptivo de la fase en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Orden	Valor numérico.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.
Vigente: String	Especificar si la tarea está vigente: SI/NO.

Figura 5.6. Atributos de la Fase.

5.6.2.3. Tarea Simple

Atributo	Descripción
Alias: Object	Identificador unívoco de la tarea en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta la tarea. Referencia al pool definido en el procedimiento.
Inputs (1-n): Artifact	Especificación de las entradas que generalmente serán documentos o componentes.
Outputs (1-n): Artifact	Especificación de las salidas que generalmente serán documentos o componentes.

Descripción (1-n): String	Nombre descriptivo de la tarea simple en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Orden	Valor numérico.
Category (0-n): String	Categoría/s asignadas a la tarea simple desde un catálogo predefinido de categorías. Category[0] = categoría 1 ⋮ Category[n] = categoría n
Prioridad: String o numérico	Prioridad asignada a la tarea simple desde un catálogo predefinido de prioridades.
Fase: Object	Fase a la que pertenece la tarea.
Estado: String	Especificación del estado en el que se encuentra la tarea: en definición, publicado o no vigente.
Modo de acceso: String	Indicar si la tarea es accesible externamente por el ciudadano mediante tramitación telemática para su consulta o también para su edición. {No accesible, Modificable, Consultable}
Modelo de tarea: SI/NO	Indicar si se trata de un modelo de tarea en cuyo servirá de referencia para crear nuevas tareas o modelos.
Nombre del modelo de tarea original (0-1): String	En el caso de que la tarea se haya creado a partir de un modelo de tarea, se especificará el nombre de dicho modelo.
Autorización de ejecución (1-n): String	Grupos organizativos o roles con permiso de ejecución sobre la tarea seleccionados del pool de unidades organizativas.
Autorización de consulta de datos (1-n): String	Grupos organizativos o roles con permiso de consulta de datos sobre la tarea seleccionados del pool de unidades organizativas.
Autorización de responsable de la tarea (0-1): String	Grupo organizativo o rol seleccionado del pool de unidades organizativos como responsable de la tarea.

Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.
Actuaciones (0-n): Property	Actuaciones definidas para la tarea cada una de ellas identificadas por una acción y una condición. Una actuación se identifica por: - Una descripción de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - El identificador de la acción seleccionada de un catálogo de acciones predefinidas. - El identificador de la condición seleccionada de entre todas las creadas para el procedimiento. <property name="Actuaciones"> <property name="Actuación"> <property name="Descripción_Actuación> //Descripción multilingüe <property name="Abrir tarea"/> …… <property name="To open task"/> </property> <property name="Acción"> Id acción </property> <property name="Condición"> Id condición </property> </property> </property>
Obligatoriedad: String	Indica si la tarea es de obligada ejecución o por el contrario es opcional: SI/NO.
Tarea inicio de procedimiento: String **Tarea inicial**: String **Tarea fin de procedimiento**: String **Tarea final:** String	Especificación de datos adicionales sobre la tarea con respecto al inicio y finalización del procedimiento: SI/NO.
Dependencia con la tarea/bloque anterior: String	Indica si la tarea depende de la tarea anterior: SI/NO.
Visible desde Web: String	Indica si la tarea es accesible vía Web: SI/NO.
Finalizable masivamente: String	Especifica si la tarea puede finalizarse masivamente: SI/NO.

Condiciones de inicio de tarea (0-n): String	Condiciones de entre todas las creadas para el procedimiento que deberán cumplirse para que la tarea pueda comenzar. CondicionesInicio[0] = Id_Condición ⋮ CondicionesInicio[n] = Id_Condición
Condiciones de fin de tarea (0-n): String	Condiciones de entre todas las creadas para el procedimiento que deberán cumplirse para que pueda finalizar la tarea. CondicionesFin[0] = Id_Condición ⋮ CondicionesFin[n] = Id_Condición

Figura 5.7. Atributos de la Tarea Simple.

5.6.2.4. Bloque

Atributo	Descripción
Alias: Object	Identificador unívoco del bloque en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta el evento. Referencia al pool definido en el procedimiento.
Inputs (1-n): Artifact	Especificación de las entradas que generalmente serán documentos o componentes.
Outputs (1-n): Artifact	Especificación de las salidas que generalmente serán documentos o componentes.
GraphicalElements (0-n): Object	Aquellos elementos que forman parte del subproceso.
No reglado False: Boolean	Su valor por defecto es False. Especifica si el bloque es reglado o no. Las actividades dentro de un bloque no reglado no son controladas o secuenciadas en un orden particular, su ejecución es determinada por el ejecutor de las actividades.
Descripción (1-n): String	Nombre del bloque en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre

Orden	Valor numérico.
Category (0-n): String	Categoría/s asignadas al bloque desde un catálogo predefinido de categorías. Category[0] = categoría 1 ⋮ Category[n] = categoría n
Prioridad: String o numérico	Prioridad asignada al bloque desde un catálogo predefinido de prioridades.
Fase: Object	Fase a la que pertenece el bloque.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.
Actuaciones (0-n): Property	Actuaciones definidas para la tarea cada una de ellas identificadas por una acción y una condición. Una actuación se identifica por: - Una descripción de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - El identificador de la acción seleccionada de un catálogo de acciones predefinidas. - EL identificador de la condición seleccionada de entre todas las creadas para el procedimiento. <property name=”Actuaciones”> <property name=”Actuación”> <property name=”Descripción_Actuación> //Descripción multilingüe <property name=”Abrir tarea”/> …… <property name=”To open task”/> </property> <property name=”Acción”> Id acción </property> <property name=”Condición”> Id condición </property> </property> </property>
Modelo de tarea: SI/NO	Indicar si se trata de un modelo de tarea en cuyo servirá de referencia para crear nuevas tareas o modelos.
Nombre del modelo de tarea original (0-1): String	En el caso de que la tarea se haya creado a partir de un modelo de tarea, se especificará el nombre de dicho modelo.

Obligatoriedad: String	Indica si el bloque es de obligada ejecución o por el contrario es opcional: SI/NO.
Bloque inicio de procedimiento: String **Bloque inicial**: String **Bloque fin de procedimiento**: String **Bloque final:** String	Especificación de datos adicionales sobre el bloque con respecto al inicio y finalización del procedimiento: SI/NO.
Dependencia con la tarea/bloque anterior: String	Indica si el bloque depende de la tarea/bloque anterior: SI/NO.
Visible desde Web: String	Indica si el bloque es accesible vía Web: SI/NO.
Finalizable masivamente: String	Especifica si el bloque puede finalizarse masivamente: SI/NO.
Condiciones de inicio de bloque(0-n): String	Condiciones de entre todas las creadas para el procedimiento que deberán cumplirse para que el bloque pueda comenzar. CondicionesInicio[0] = Id_Condición ⋮ CondicionesInicio[n] = Id_Condición
Condiciones de fin de bloque (0-n): String	Condiciones de entre todas las creadas para el procedimiento que deberán cumplirse para que pueda finalizar el bloque. CondicionesFin[0] = Id_Condición ⋮ CondicionesFin[n] = Id_Condición
Modo de acceso: String	Indicar si la tarea es accesible externamente por el ciudadano mediante tramitación telemática para su consulta o también para su edición. {No accesible, Modificable, Consultable}
Autorización de ejecución (1-n): String	Grupos organizativos o roles con permiso de ejecución sobre la tarea seleccionados del pool de unidades organizativas.

Autorización de consulta de datos (1-n): String	Grupos organizativos o roles con permiso de consulta de datos sobre la tarea seleccionados del pool de unidades organizativas.
Autorización de responsable de la tarea (0-1): String	Grupo organizativo o rol seleccionado del pool de unidades organizativos como responsable de la tarea.

Figura 5.8. Atributos del Bloque.

5.6.2.5. Documento

Atributo	Descripción
Id: Object	Identificador unívoco del documento en el procedimiento.
RequiredForStart True: Boolean	Su valor por defecto es True y significa que su entrada es requerida para que una tarea o bloque comience. Si es False, la tarea o bloque puede comenzar sin su entrada pero pueden aceptar más de una después de que haya comenzado.
ProducedAtCompletion True: Boolean	El valor por defecto de este atributo es True y significa que será producido como salida cuando la tarea o bloque haya sido completado. Si es False, la tarea o bloque puede producirlo como salida y varias veces antes de que se haya completado.
Name (1-n): String	Nombre del documento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Documentación requerida: String	Especificar si el documento pertenece al grupo de documentación requerida por el procedimiento: SI/NO.
Clasificación archivística: String	Tipo de documento si procede seleccionado de un catálogo de tipos de documentos predefinido.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.

Figura 5.9. Atributos del Documento.

5.6.2.6. Componente

Atributo	Descripción
Alias: Object	Identificador unívoco del documento en el procedimiento.
RequiredForStart True: Boolean	Su valor por defecto es True y significa que su entrada es requerida para que una tarea o bloque comience. Si es False, la tarea o bloque puede comenzar sin su entrada pero pueden aceptar más de una después de que haya comenzado.
ProducedAtCompletion True: Boolean	El valor por defecto de este atributo es True y significa que será producido como salida cuando la tarea o bloque haya sido completado. Si es False, la tarea o bloque puede producirlo como salida y varias veces antes de que se haya completado.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.
Descripción (1-n): String	Nombre del documento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Sección: String	Sección de la página en la que se ubicará este componente. Se seleccionará de entre todas las creadas para el procedimiento.
Condiciones de: **- Creación (0-n)**: String **- Visibilidad (0-n)**: String **- Acceso (0-n)**: String	Especificación de las condiciones de creación, visibilidad y de acceso del componente. Dichas condiciones se seleccionarán de entre todas las creadas para el procedimiento.

Campos (1-n): Property	Especificación de los campos incorporados en el componente. Cada uno de ellos se identifica por los siguientes atributos de los cuales sólo son modificables en el momento de la incorporación del componente al procedimiento la etiqueta, la obligatoriedad y la visibilidad: - Tipo de dato - Código - Longitud máxima - Valor por defecto - Máscara de entrada - Obligatoriedad - Visibilidad - Etiqueta. Atributo multilingüe. Los idiomas se seleccionarán de un catálogo predefinido. <property name=”Campos”> <property name=”NUMEXP”> //código del campo <property name”Tipo”> ---- </property> <property name=”Longitud”> ---- </property> <property name=”Etiqueta”> //Etiqueta multilingüe <property name=”Referencia al expediente idioma=”código idioma”/> ….. <property name=”Reference to the dossier” idioma=”código idioma”/> </property> <property name=”Obligatoriedad”> {SI/NO} </property> ……. <property name=”Valor por defecto”> ---- </property>

Figura 5.10. Atributos del Componente.

5.6.2.7. Condición

Atributos Exclusiva Data-Based (XOR)	Descripción
Id: Object	Identificador unívoco de la condición en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta la condición. Referencia al pool definido en el procedimiento.

XORType (Data | Event) Data: String	Tipo de pasarela XOR: basado en datos o en eventos.
Gates (0-n): Gate	Especificación de las distintas ramas en las que se bifurca el flujo en base a las distintas condiciones a valorar. Si la pasarela está actuando como decisión debe especificarse al menos una bifurcación, dos en el caso de que no haya una por defecto.
[Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Cada bifurcación debe tener asociado un flujo de secuencia cuyo atributo de condición debe ser una expresión válida. Si sólo hay una bifurcación, el flujo de secuencia debe tener su condición configurada a Ninguna. Una o más expresiones de asignación pueden ser definidas para cada bifurcación especificada. La asignación se ejecutará cuando la bifurcación correspondiente sea seleccionada.
DefaultGate (0-1): Gate	Especificar una bifurcación por defecto.
[DefaultGate: Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Especificación del flujo de secuencia de salida asociado a la bifurcación por defecto si la hubiese. Dicho flujo deberá tener un indicador y un atributo de condición por defecto. Una o más expresiones de asignación pueden definirse para la bifurcación por defecto. La asignación será ejecutada cuando ésta sea seleccionada.
Name (1-n): String	Nombre de la condición en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual de la condición expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.11. Atributos de la Condición Exclusiva Data-Based.

Atributos Exclusiva Event-Based (XOR)	Descripción
Id: Object	Identificador unívoco de la condición en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta la condición. Referencia al pool definido en el procedimiento.
XORType (Data | Event) Data: String	Tipo de pasarela XOR: basada en datos o basada en eventos.
Gates (2-n): Gate	Deben especificarse 2 o más bifurcaciones de salida.
[Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Cada bifurcación definida debe tener asociado un flujo de secuencia de salida cuyo atributo de condición será establecido a Ninguno (no hay una evaluación de una expresión de condición). Para cada bifurcación se pueden definir una o más expresiones de asignación. La asignación será ejecutada cuando la bifurcación correspondiente sea seleccionada.
Name (1-n): String	Nombre de la condición en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual de la condición expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.12. Atributos de la Condición Exclusiva Event-Based.

Atributos No exclusiva (OR)	Descripción
Id: Object	Identificador unívoco de la condición en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta la condición. Referencia al pool definido en el procedimiento.

Gates (0-n): Gate	Especificación de las distintas ramas en las que se bifurca el flujo en base a las distintas condiciones a valorar. Si la pasarela está actuando como decisión debe especificarse al menos una bifurcación, dos en el caso de que no haya una por defecto.
[Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Cada pasarela debe tener asociada un flujo se secuencia de salida cuyo atributo de condición se establece a una expresión válida. La expresión de condición debe ser única para todas las bifurcaciones dentro de la pasarela. Si sólo hay una bifurcación el flujo de secuencia de salida debe tener establecido su atributo de condición a Ninguno. Para cada bifurcación se pueden definir más de una expresión de asignación. La asignación será ejecutada cuando la bifurcación correspondiente sea seleccionada.
DefaultGate (0-1): Gate	Especificar una bifurcación por defecto.
[DefaultGate: Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Si hay una bifurcación por defecto ésta debe tener asociado un flujo de secuencia de flujo de salida con un indicador y un atributo de condición por defecto. Para la bifurcación por defecto se pueden definir más de una expresión de asignación que será ejecutada cuando ésta sea seleccionada.
Name (1-n): String	Nombre de la condición en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual de la condición expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.13. Atributos de la Condición No Exclusiva.

Atributos Gateway con caminos paralelos (AND)	Descripción
Id: Object	Identificador unívoco de la condición en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta la condición. Referencia al pool definido en el procedimiento.
GatewayType (XOR | OR | AND) XOR: String	El tipo de pasarela determinará su comportamiento, tanto para el flujo de secuencia entrante como para el de salida, y determinará el indicador interno. Su valor por defecto es XOR.
Gates (0-n): Gate	Especificación de las distintas ramas en las que se bifurca el flujo. Si la pasarela está actuando como una bifurcación debe especificarse al menos dos gates.
[Gate] OutgoingSequenceFlow: SequenceFlow **Assigments** (0-n): Assignment	Cada bifurcación definida debe tender asociado un flujo de secuencia de salida cuyo atributo de condición debe estar establecido a Ninguna. Para cada bifurcación se pueden definer varias expresiones de asignación. Esta asignación se ejecutará cuando la correspondiente gate sea seleccionada.
Name (1-n): String	Nombre de la condición en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual de la condición expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.14. Atributos de la Condición AND.

5.6.2.8. Evento

Atributos Evento Start	Descripción
Id: Object	Identificador unívoco del evento en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta el evento. Referencia al pool definido en el procedimiento.
Trigger: (None | Message | Timer | Ruler | Link | Multiple) None: String	Tipo de disparador que provoca dicho evento en el procedimiento. Por defecto toma el valor None que es el que aplicaremos en nuestro proyecto.
Name (1-n): String	Nombre del evento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual del evento expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.15. Atributos del Evento Start.

Atributos Evento End	Descripción
Id: Object	Identificador unívoco del evento en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta el evento. Referencia al pool definido en el procedimiento.
Result: (None | Message | Error | Cancel | Compensation | Link | Terminate | Multiple) None: String	Tipo de resultado que se produce al finalizar el evento. Por defecto toma el valor None.

Name (1-n): String	Nombre del evento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre
Descripción adicional (0-n): String	Opcionalmente, descripción textual del evento expresada en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción

Figura 5.16. Atributos del Evento End.

Atributos Evento Intermedio Timer	Descripción
Código: Object	Identificador unívoco del evento en el procedimiento.
Pool: Pool	Rol o entidad que ejecuta el evento. Referencia al pool definido en el procedimiento.
Trigger: (None | Message | Timer | Error | Cancel | Compensation | Link | Rule | Multiple) Message: String	Tipo de disparador esperado para ejecutar dicho evento. Por defecto toma el valor Message.
Target (0-1): Object	Si lo hay, el objeto asociado al evento debe ser una tarea o un bloque. Esto significa que el evento está adherido a la frontera de la actividad y es usado para expresar una excepción o compensación sobre la misma.
Denominación (1-n): String	Nombre del evento en los distintos idiomas seleccionados de un catálogo predefinido. Name[0] = código idioma + nombre ⋮ Name[n] = código idioma + nombre

Category (0-n): String	Categoría/s asignadas al evento desde un catálogo predefinido de categorías. Category[0] = categoría 1 ⋮ Category[n] = categoría n
Fecha de vencimiento	Especificación de la fecha de vencimiento del plazo definida bien como una duración o en base a una fecha del procedimiento. <property name=”FechVencimiento”> Id fecha </property> Ó <property name=”FechVencimiento”> <property name= Id fecha /> <property name=”años”> ---- </property> <property name=”meses”> ---- </property> <property name=”días”> ---- </property> * </property> * Ejemplos: 10 días naturales / días hábiles / semanas
Calendario: String	Calendario seleccionado.
Fecha de alarma: String	Especificación de aviso ante la proximidad de la fecha vencimiento del plazo. Ejemplos: 10 días naturales/días hábiles/semanas/meses/años
Actuaciones (0-n): Property	Actuaciones definidas para el plazo cada una de ellas identificadas por una acción y una condición. Una actuación se identifica por: - Una descripción de la misma en los distintos idiomas seleccionados de un catálogo predefinido. - El identificador de la acción seleccionada de un catálogo de acciones predefinidas. - EL identificador de la condición seleccionada de entre todas las creadas para el procedimiento. <property name=”Actuaciones”> <property name=”Actuación”> <property name=”Descripción_Actuación”> //Descripción multilingüe <property name=”Vencimiento” idioma=”código idioma”/> …… <property name=”Maturity” idioma=”código idioma”/>

	</property> <property name=”Acción”> Id acción </property> <property name=”Condición”> Id condición </property> </property> </property>
Excepciones (0-n): Property	Especificación de aquellas situaciones de excepción que modifican el estado del plazo y afectan a su cómputo. Una excepción se identifica por: - Código de la fecha seleccionada de entre todas las creadas para el procedimiento y que establece el inicio de la excepción. - Acción de control sobre el plazo: iniciar, finalizar, pausar, continuar o reiniciar <property name=”Excepciones”> <property name=”Excepción”> <property name=”Id fecha”/> <property name=”Comportamiento”> ---- </property> </property> </property>

Figura 5.17. Atributos del Evento Intermedio Timer.

5.6.2.9. Conexión entre objetos

Atributos Flujo de secuencia	Descripción
Id: Object	Identificador unívoco del conector en el procedimiento.
Source: Object	Identifica el objeto origen del conector.

Target: Object	Identifica el objeto destino del conector.
ConditionType (None | Expression | Default) None: String	Por defecto, el tipo de condición de un flujo de secuencia es Ninguno, es decir, en tiempo de ejecución para determinar si será utilizada el flujo de secuencia.
[ConditionType Expression only] **Condición**: String	Condición seleccionada de entre todas las creadas para el procedimiento. Será evaluada en tiempo de ejecución.

Figura 5.18. Atributos del Flujo de Secuencia.

Atributos Asociación	Descripción
Id: Object	Identificador unívoco del conector en el procedimiento.
Source: Object	Identifica el objeto origen del conector.
Target: Object	Identifica el objeto destino del conector.
Direction (None | To | From | Both) None: String	Define si la asociación muestra o no alguna dirección mediante una flecha. El valor por defecto es Ninguna (sin flecha). Un valor de To significa que la flecha deberá estar en el objeto fuente. Un valor de From significa que la flecha deberá estar en el objeto-artefacto destino. Un valor de Both significa que deberá haber una flecha en ambos extremos de la línea de asociación.

Figura 5.19. Atributos del Flujo de Asociación.

5.6.2.10. Nota

Atributo	Descripción
Id: Object	Identificador unívoco del conector en el procedimiento.
Text (1-n): Property	Texto para comunicar información adicional al lector del procedimiento.
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.

Figura 5.20. Atributos de la Nota.

5.6.2.11. Grupo organizativo o rol

Atributos Pool	Descripción
Id: Object	Identificador unívoco del grupo organizativo o rol en el procedimiento.
Name: String	Nombre del grupo organizativo o rol seleccionado desde los Servicios Comunes.
Descripción adicional (0-n): String	Opcionalmente, descripción textual del grupo organizativo o rol en los distintos idiomas seleccionados de un catálogo predefinido. DA[0] = código idioma + descripción ⋮ DA[n] = código idioma + descripción
Icono: String	Imagen seleccionada de un catálogo de iconos predefinido para identificar de forma clara y rápida esta entidad de tramitación.

Figura 5.21. Atributos del Grupo Organizativo.

5.6.2.12. Otros

Atributos Assignment	Descripción
To: Property	El objeto de la asignación debe ser una propiedad del proceso o de la actividad en sí misma.
From: Expression	La expresión debe estar formada por una combinación de valores, propiedades y atributos separados por operadores tales como la suma o la multiplicación.
AssignTime (0-1) (Start | End) Start: String	Una asignación puede tener asociado un tiempo de establecimiento de la misma. Si el objeto es una tarea o bloque, se debe definir este atributo. Un valor de Start significa que la asignación ocurrirá al comienzo de la actividad. Puede usarse para asignar propiedades de más alto nivel (global) de los procesos a las propiedades locales de la actividad como entrada a la misma. Un valor de End significa que la asignación ocurrirá al final de la actividad. Puede usarse para asignar propiedades de más alto nivel (global) de los procesos a las propiedades locales de la actividad como salida de la misma.

Figura 5.22. Atributos de Assignment.

Atributos Expression	Descripción
Expression: String	Debe ser una expresión matemática para ser testeada como True o False o ser evaluada para actualizar el valor de las propiedades.

Figura 5.23. Atributos de Expression.

Atributos Object	Descripción
Id: String	Identificador del objeto.

Figura 5.24. Atributos de Object.

Atributos Property	Descripción
Name: String	Nombre descriptivo de la propiedad.
Type: String	Tipo (por ejemplo, string). Puede ser de tipo conjunto que permite la especificación de propiedades hijas (esta opción nos permitirá agregar la característica de multi-idioma de los distintos atributos de las entidades).

Figura 5.25. Atributos de Property.

5.6.3. Restricciones a tener en cuenta

Eventos de comienzo

- No tiene flujos de secuencia entrantes salvo cuando el evento es usado en un subproceso expandido, en cuyo caso se adhiere a los límites del subproceso.

- Debe ser el origen de un flujo de secuencia saliente. Dicho flujo no tendrá asociada ninguna condición.

- Si hay un evento de fin, debe haber al menos un evento de comienzo.

Eventos de fin

- No tiene flujos de secuencia de salida salvo cuando el evento es usado en un subproceso expandido, en cuyo caso se adhiere a los límites del subproceso.

- Un evento de fin del procedimiento puede tener múltiples flujos de secuencia entrantes.

Subproceso

- Un subproceso puede tener múltiples flujos de secuencia entrantes.

Tarea simple

- Una tarea simple puede tener múltiples flujos de secuencia salientes, en cuyo caso se está representado que se tratan de caminos paralelos.

Pasarela o decisión exclusiva basada en condiciones

- Puede tener múltiples flujos de secuencias entrantes y salientes.

- Si actúa como decisión, si tiene múltiples flujos de secuencia entrantes, todos ellos serán necesarios para ejecutar la decisión para continuar con el flujo del proceso.

- Si actúa como punto de convergencia de más de un flujo de secuencia, sólo uno de ellos será seleccionado durante la ejecución del proceso. La selección se hará en base a la condición definida para cada flujo.

Pasarela o decisión exclusiva basada en eventos

- Puede tener múltiples flujos de secuencias entrantes y salientes.

5.7. MODELADO DE LA ARQUITECTURA UTILIZADA

La arquitectura de un sistema es una especificación de las partes y los conectores del sistema, así como las reglas para la interacción de dichas partes.

La OMG ha definido el estándar MDA, que tiene como objetivo proporcionar un entorno de desarrollo capaz de evolucionar con un esfuerzo mínimo. Para ello, este estándar se apoya en múltiples tecnologías, algunas creadas junto con MDA, y otras ya existentes pero que encajan perfectamente en la finalidad de este proyecto. Las principales tecnologías con las que nos encontramos son el tan conocido lenguaje de modelado UML, un framework para la definición de metamodelos, MOF, y un formato de intercambio de modelos basado en XML y denominado XMI.

En nuestro caso, hemos utilizado MOF porque esta arquitectura tiene un conjunto de características que la diferencian de otras arquitecturas:

- El modelo MOF está orientado a objetos y formado por elementos totalmente compatibles con el lenguaje de modelado UML.

- Las capas de la arquitectura MOF no son fijas y aunque normalmente utilizaremos cuatro niveles estos podrían ser más o menos en función de cómo se quiera utilizar MOF.

- Estas capas son simplemente un mecanismo para entender las relaciones entre los diferentes tipos de datos y metadatos.

- Un modelo no está limitado únicamente a un único meta-nivel.

Nivel	Descripción
M3 (Meta-metamodelo)	**MOF,** define un lenguaje para especificar metamodelos
M2 (Metamodelo)	**metamodelos**, instancias de los elementos MOF Define un lenguaje para especificar modelos
M1 (Modelo)	**modelos**, instancia de un metamodelo MOF
M0 (Instancia)	el **sistema,** objetos y datos, instancias de los elementos de un modelo

Figura 5.26. Arquitectura de metamodelado de cuatro niveles.

Para el desarrollo del proyecto se ha utilizado la plataforma Eclipse Europa, que ofrece un lenguaje de modelado mediante el Framework EMF, denominado Ecore.

MOF y Ecore se basan en elementos de modelado orientado a objetos:

- Clases y Atributos.
- Asociaciones en MOF y referencias entre objetos en Ecore.
- Agregación en MOF.
- Generalización.
- Paquetes.

6. DESARROLLO DEL SISTEMA

6.1. INTRODUCCIÓN

En este capítulo se va a explicar como se ha llevado a cabo el desarrollo de la aplicación Definidor Visual.

El diseño del software es un proceso y un modelado a la vez. El proceso de diseño es un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del sistema a construir. A lo largo del diseño se evalúa la calidad del desarrollo del proyecto con un conjunto de revisiones técnicas.

El diseño debe implementar todos los requisitos explícitos contenidos en el modelo de análisis y debe acumular todos los requisitos implícitos que desea el cliente. Debe ser una guía que puedan leer y entender los que construyan el código y los que prueban y mantienen el software.

Debe proporcionar una completa idea de lo que es el software, enfocando los dominios de datos, funcional y comportamiento desde el punto de vista de la implementación.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Proceso de desarrollo de modelado con Eclipse.
- Crear el meta-modelo.
- Generación de código.
- Diseño de la aplicación GMF a partir del meta-modelo.
- Generación de la herramienta gráfica.
- Creación de un modelo gráfico.
- Obtención del fichero XML a partir del modelo gráfico.

6.2. PROCESO DE DESARROLLO DE MODELADO CON ECLIPSE

Para el desarrollo del proyecto se ha utilizado la plataforma Eclipse, a continuación se muestra gráficamente, el proceso de desarrollo llevado a cabo en esta plataforma, para obtener los modelos necesarios para la implementación de la aplicación final.

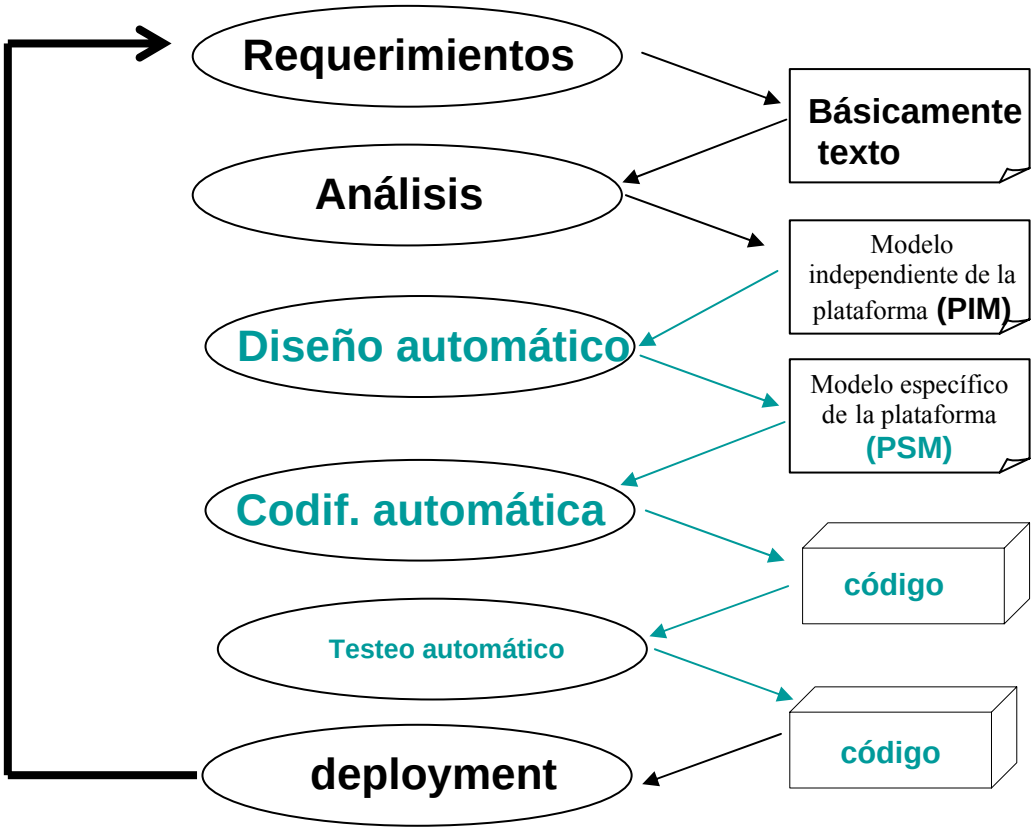


Figura 6.1. Ciclo de desarrollo.

¿Cómo se hace?

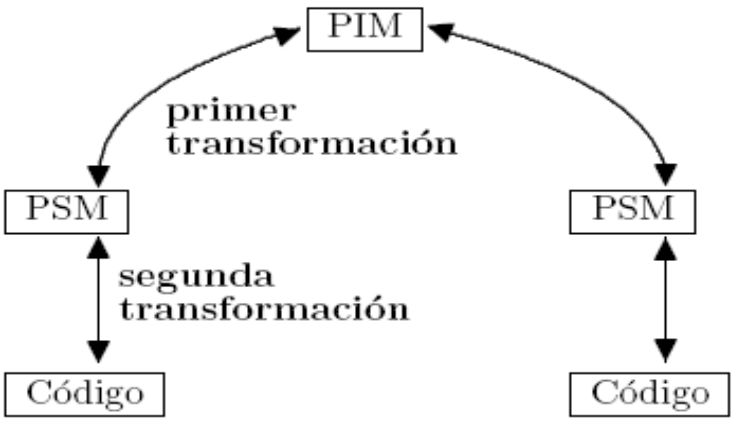


Figura 6.2. Ciclo de desarrollo. ¿Cómo se hace?

Necesitamos algunos elementos.

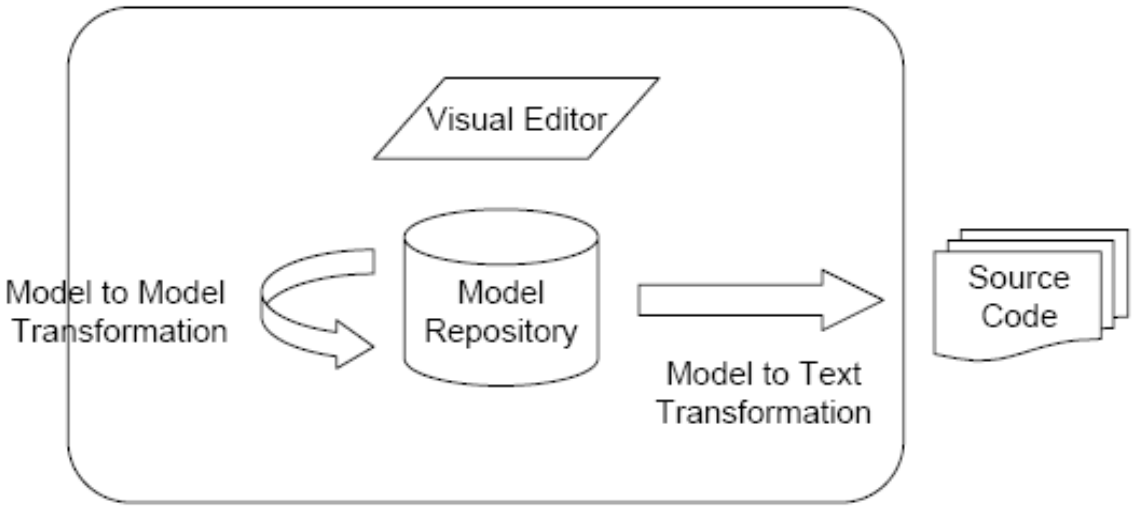


Figura 6.3. Ciclo de desarrollo. Elementos necesarios.

Definiendo el lenguaje de modelado y el repositorio de modelos, EMF (Ecore)

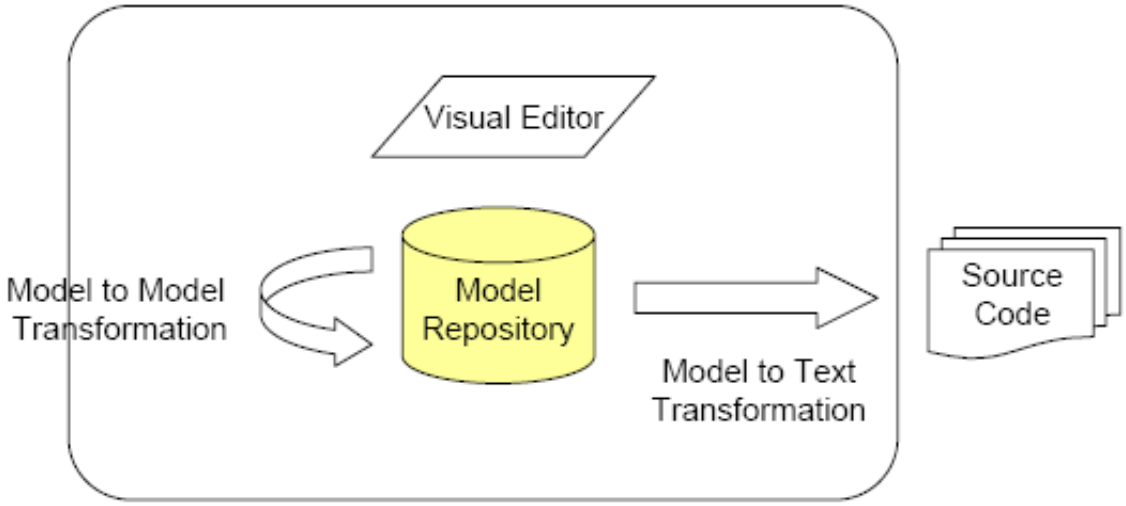


Figura 6.4. Ciclo de desarrollo. Lenguaje de modelado y repositorio de modelos (EMF).

Definiendo los editores gráficos, GMF.

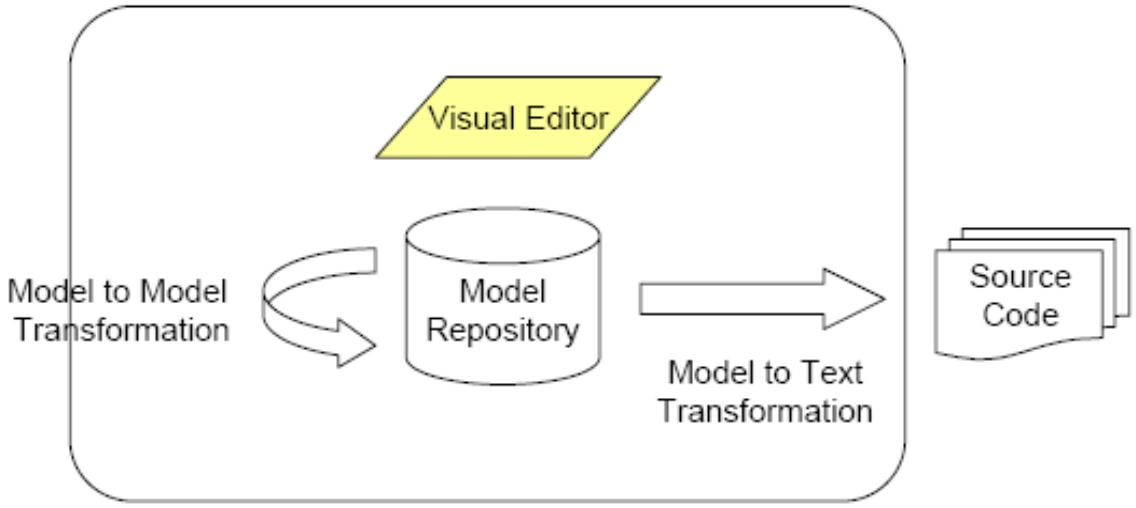


Figura 6.5. Ciclo de desarrollo. Definición de editores gráficos (GMF).

En los siguientes apartados, se explica de manera formal como se han ido obteniendo todos los modelos necesarios y las transformaciones que se han llevado a cabo para obtener el resultado final.

6.3. CREAR EL META-MODELO (.ECORE)

El punto de partida para el desarrollo de la herramienta del Definidor Visual, es crear un proyecto EMF vacío, y a partir de éste generar los diferentes modelos, en la carpeta "model".

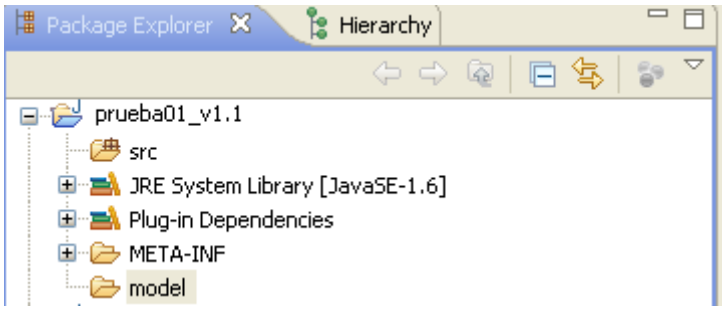


Figura 6.6. Proyecto EMF vacío.

La plataforma Eclipse ofrece una serie de opciones para generar diferentes modelos dentro de un proyecto EMF, en nuestro caso en particular vamos a empezar generando un Ecore Model.

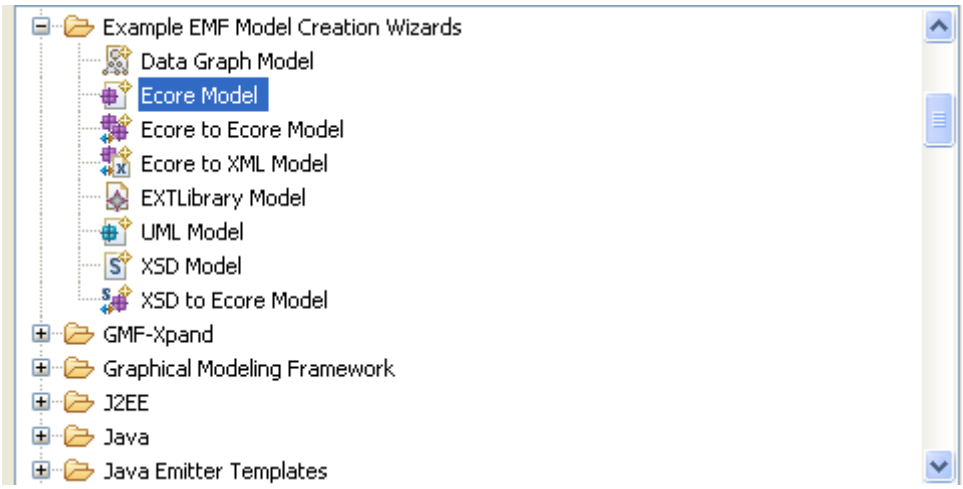


Figura 6.7. Proyecto EMF. Ventana selección Ecore Model.

Una vez que hemos obtenido el Ecore Model vacío, podemos empezar a definir el meta-modelo .Ecore, en base a los requisitos especificados en la fase de análisis.

En primer lugar, empezamos dando nombre al modelo y generando desde la raíz de dicho modelo los elementos necesarios. Los tipos de elementos que se pueden definir se exponen en las siguientes pantallas.

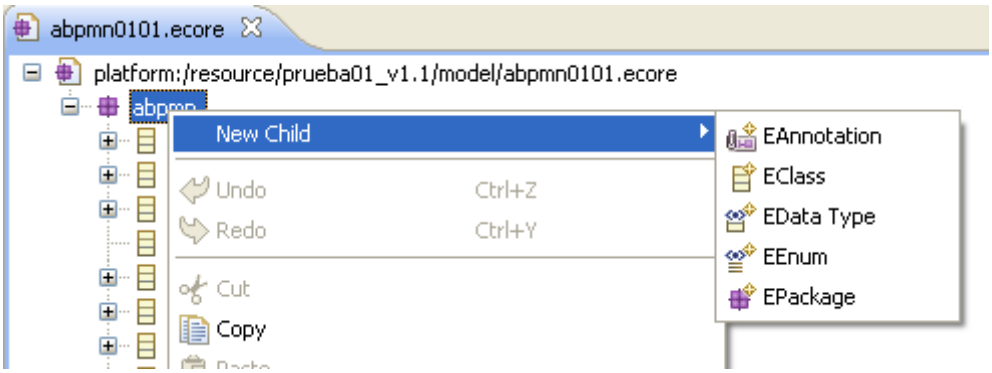


Figura 6.8. Ventana selección elementos para el raíz del Ecore Model.

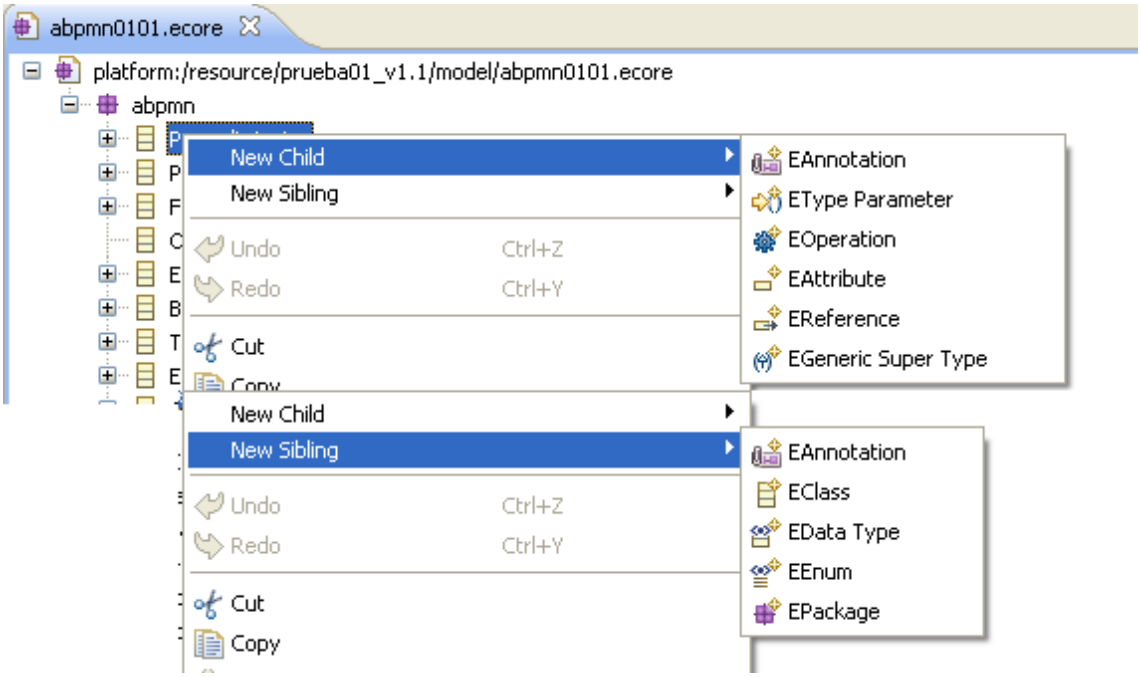


Figura 6.9. Ventana selección elementos del Ecore Model.

A continuación, se muestra el resultado de la definición del elemento Procedimiento y todos sus atributos y relaciones, el resto de elementos se generarían igual, cada uno de ellos con los atributos y relaciones necesarias.

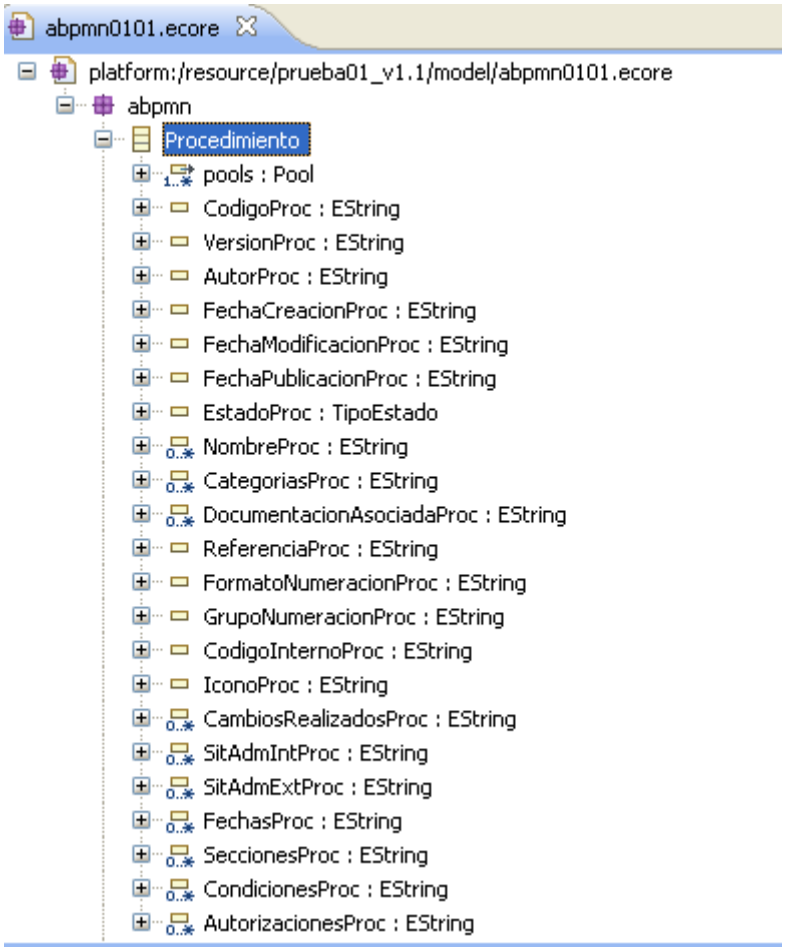


Figura 6.10. Atributos del elemento Procedimiento del Ecore Model.

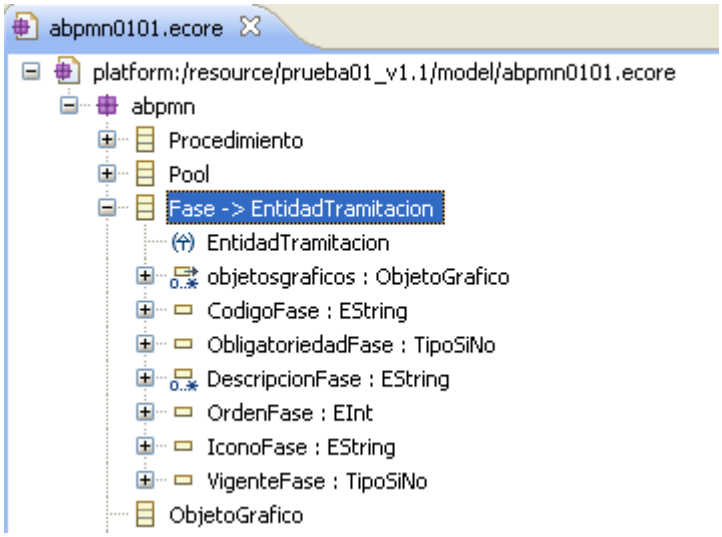


Figura 6.11. Atributos del elemento Fase del Ecore Model.

Una vez definido el tipo de elemento, cada uno de ellos tiene asociada una ventana de propiedades en la que se definirán, el nombre, tipo, etc para cada uno de ellos. De esta forma se completa la definición de los elementos que forman parte del modelo .Ecore.

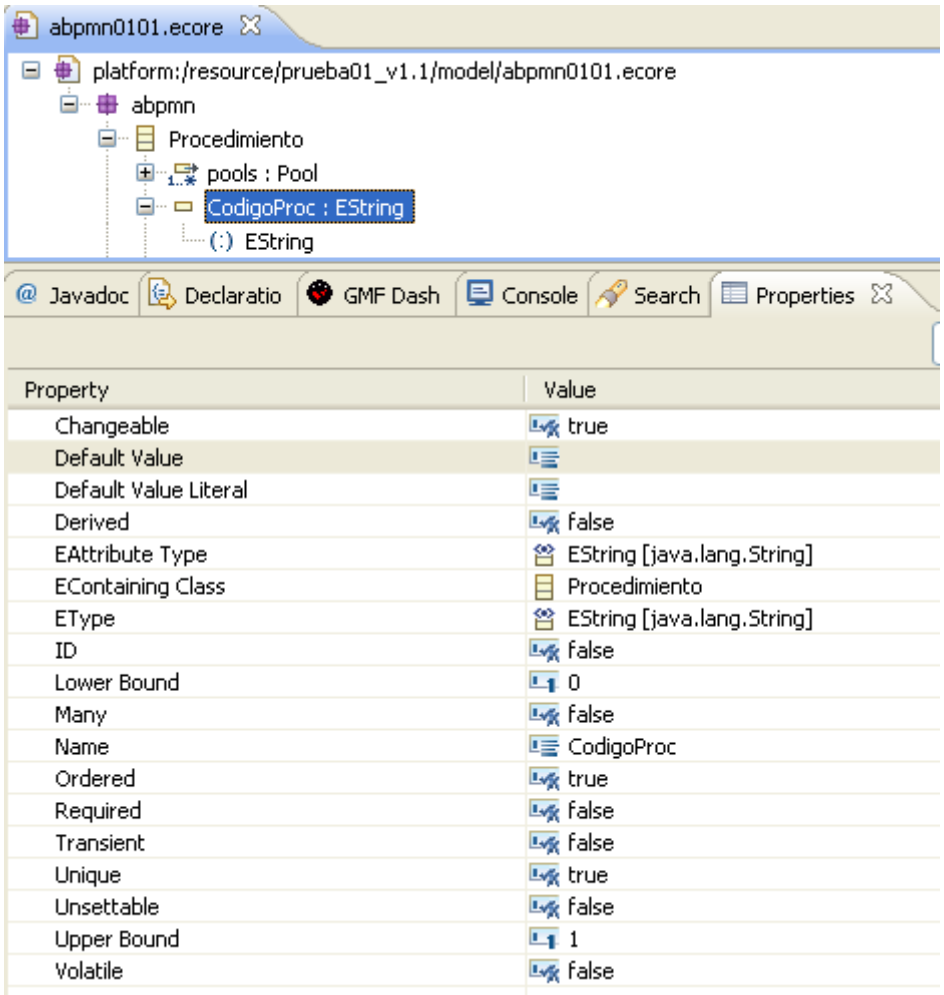


Figura 6.12. Ventana de propiedades del atributo CodigoProc del elemento Procedimiento.

En la siguiente imagen se muestra el modelo completo abpmn0101.ecore que se ha generado para el Definidor Visual.

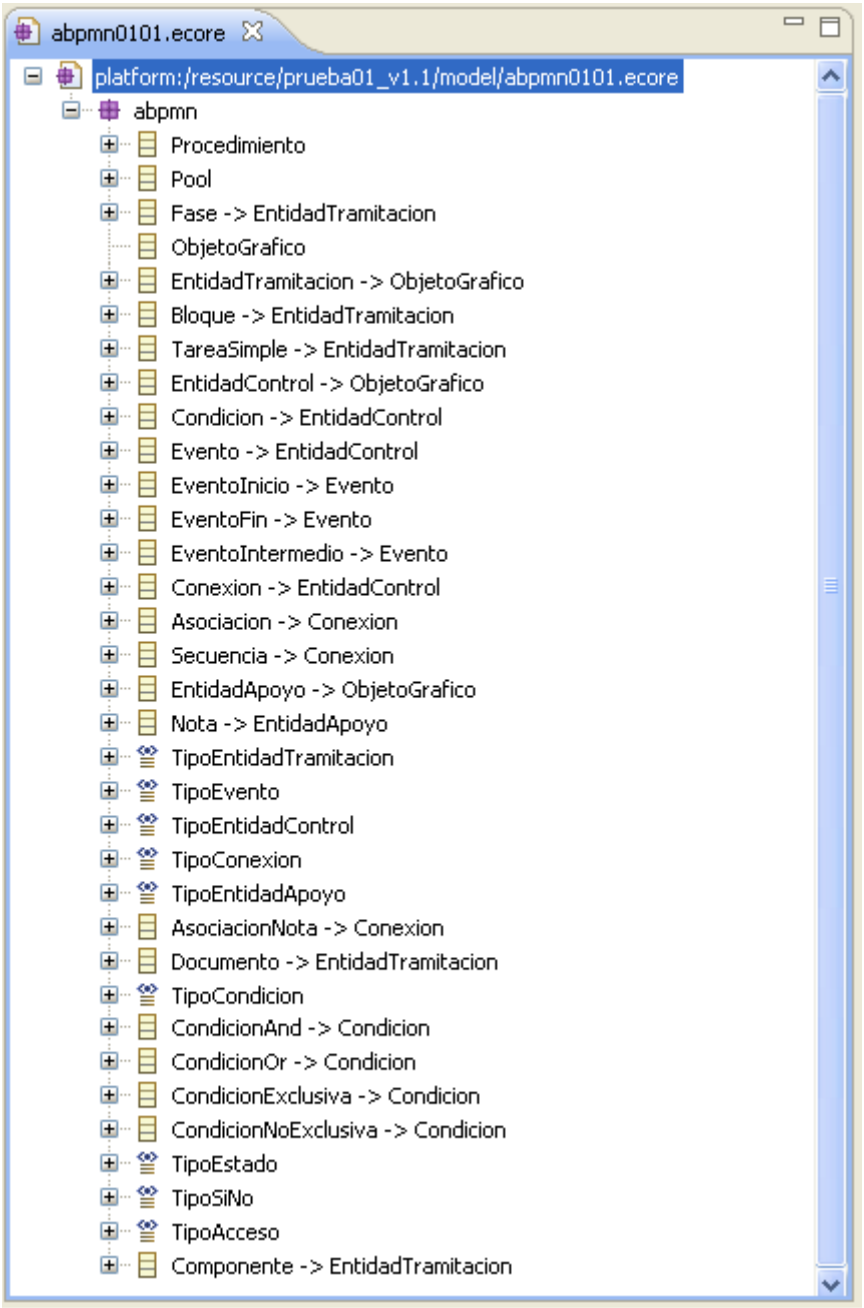


Figura 6.13. abpmn0101 .ecore.

En base al fichero .ecore, generamos y obtenemos el fichero .ecore_diagram que se aprecia en la siguiente figura. Dicho fichero expresa el fichero .ecore como si fuera un diagrama de clases de UML.

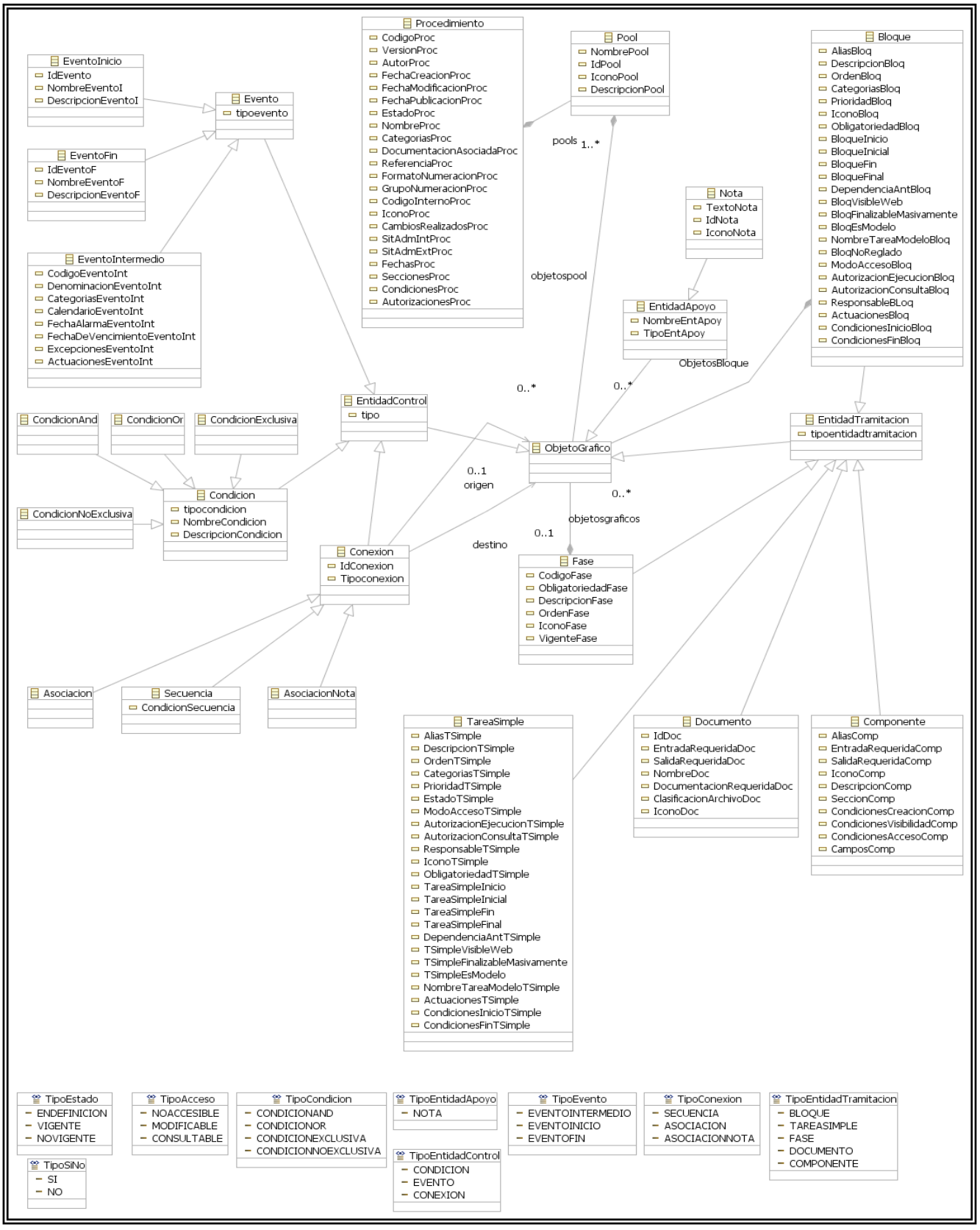


Figura 6.14. abpmn0101 .ecore_diagram.

También podemos obtener el fichero .xmi en base al fichero .ecore. Dicho fichero expresa el fichero .ecore en formato xmi. A continuación se muestra un pequeño fragmento de dicho fichero.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="abpmn"
    nsURI="abpmn" nsPrefix="abpmn">
  <eClassifiers xsi:type="ecore:EClass" name="Procedimiento">
    <eStructuralFeatures xsi:type="ecore:EReference" name="pools"
lowerBound="1" upperBound="-1"
        eType="#//Pool" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="CodigoProc"
eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"
        defaultValueLiteral=""/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="VersionProc" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="AutorProc"
eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="FechaCreacionProc" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="FechaModificacionProc"
        eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="FechaPublicacionProc" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="EstadoProc"
eType="#//TipoEstado"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="NombreProc"
upperBound="-1"
        eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="CategoriasProc" upperBound="-1"
        eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"
defaultValueLiteral=""/>
    <eStructuralFeatures xsi:type="ecore:EAttribute"
name="DocumentacionAsociadaProc"
        upperBound="-1" eType="ecore:EDataType
```

Figura 6.15. abpmn0101 .ecore en formato XMI.

6.4. GENERACIÓN DE CÓDIGO

Una vez que tenemos generado el fichero .ecore y .ecore_diagram, el siguiente paso es generar el fichero .genmodel.

También en este apartado se explicará como generar la carpeta **src** que contiene el código generado a partir de los ficheros .Ecore y .Genmodel, así como la generación de los ficheros no gráficos .Edit y .Editor.

6.4.1. Generar el Fichero .Genmodel a partir del .Ecore

A continuación, se explica como se ha obtenido el fichero .genmodel a partir del .ecore ya generado.

Nos posicionamos sobre el fichero .ecore y pulsamos el botón derecho del ratón, de esta manera obtenemos la ventana que se observa a continuación. Pulsaremos New → Other…

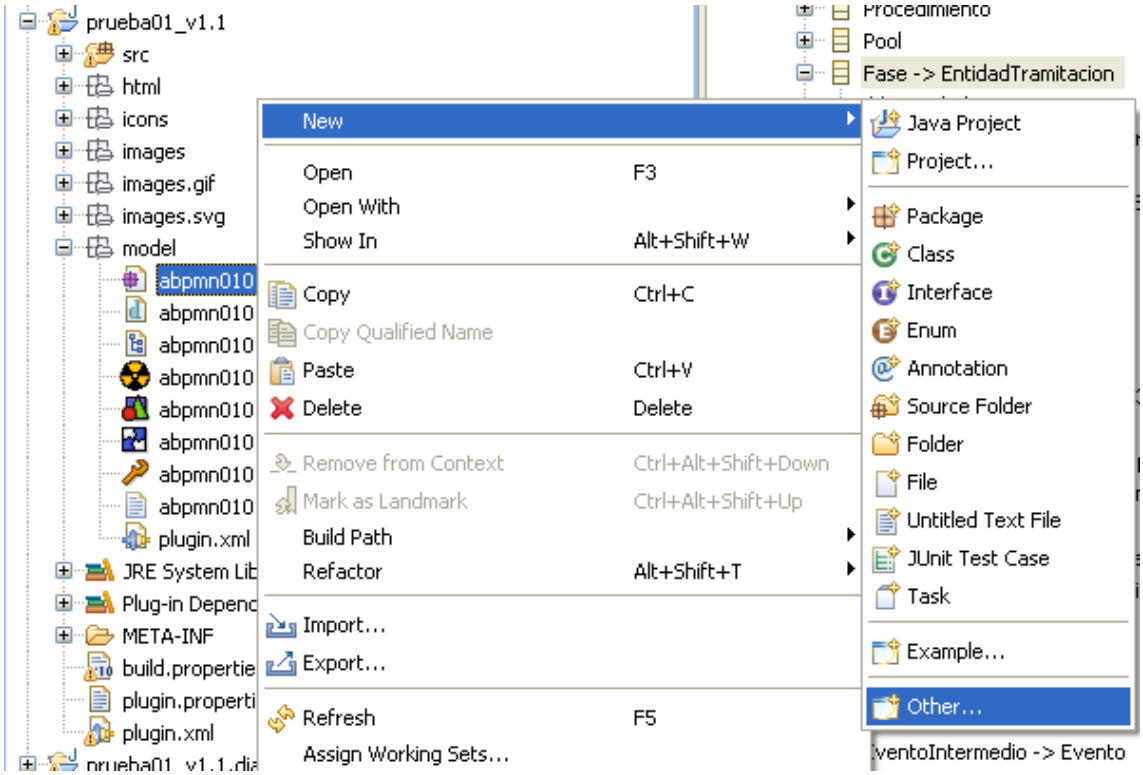


Figura 6.16. Generar .Genmodel a partir del .Ecore.

Y obtendremos la siguiente ventana donde seleccionaremos “EMF Model” dentro de la opción “Eclipse Modeling Framework”.

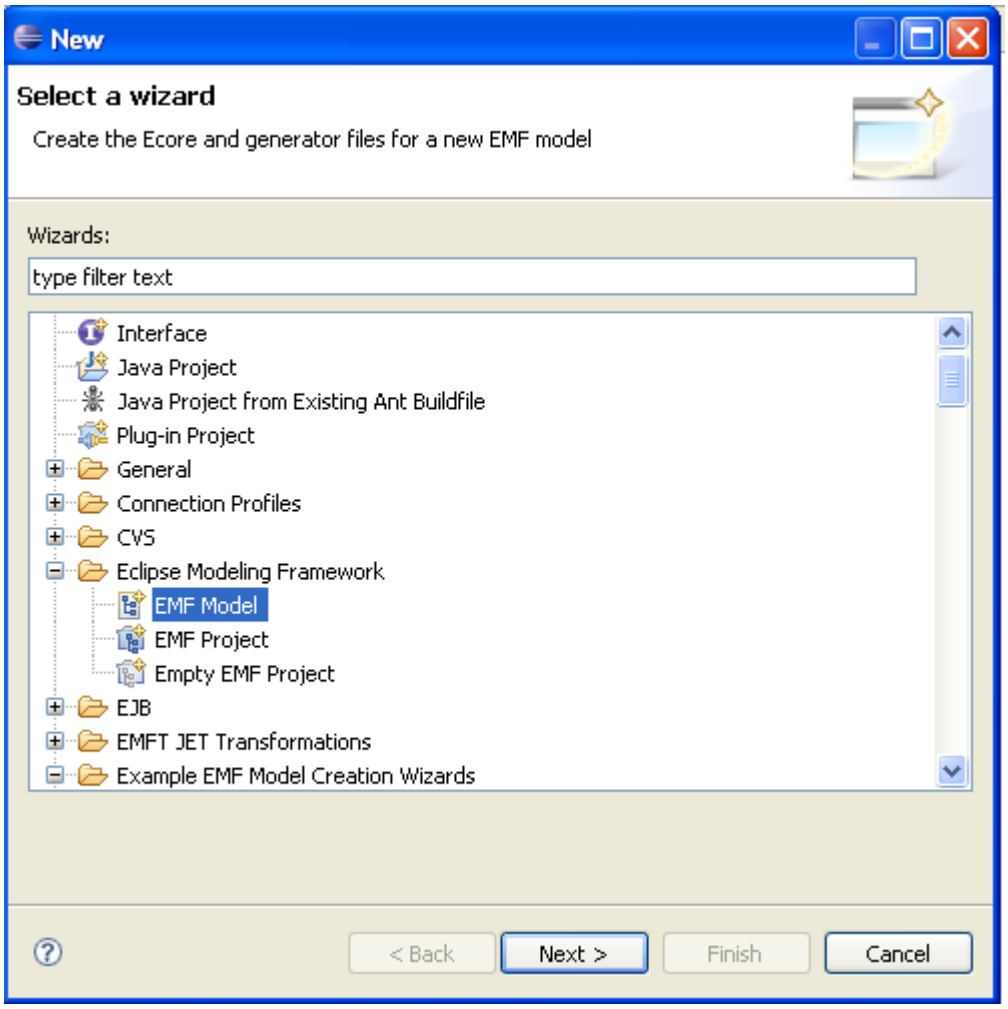


Figura 6.17. Generar .Genmodel. Ventana New.

De esta manera se desplegará la siguiente ventana donde pondremos el nombre del nuevo fichero .genmodel. En nuestro caso abpmn0101.genmodel.

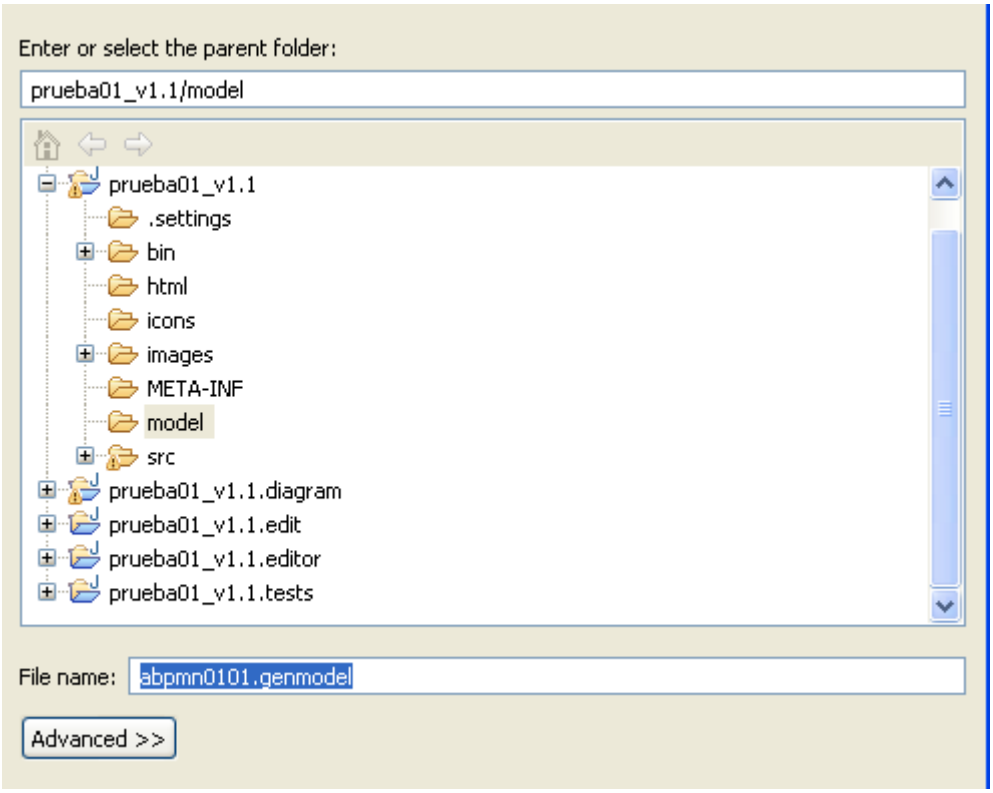


Figura 6.18. Generar .Genmodel. Ventana Nombre del Fichero.

Una vez que hemos completado el nombre del fichero, pasaremos a la siguiente ventana, donde seleccionamos el tipo de modelo del que queremos obtener el nuevo fichero .genmodel. En nuestro caso seleccionaremos "Ecore Model", puesto que lo vamos a obtener basándonos en el fichero .ecore generado anteriormente.

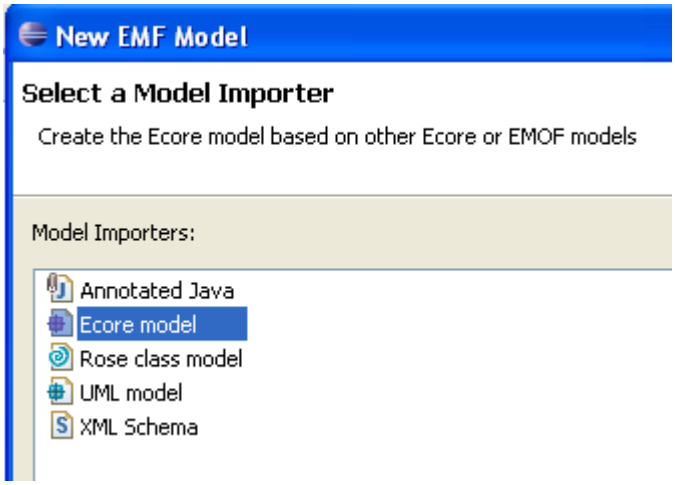


Figura 6.19. Generar .Genmodel. Ventana Seleccionar Modelo.

A continuación, pasamos a la siguiente pantalla donde seleccionamos y cargamos el fichero abpmn0101.ecore. De esta manera obtenemos el nuevo fichero abpmn0101.genmodel.

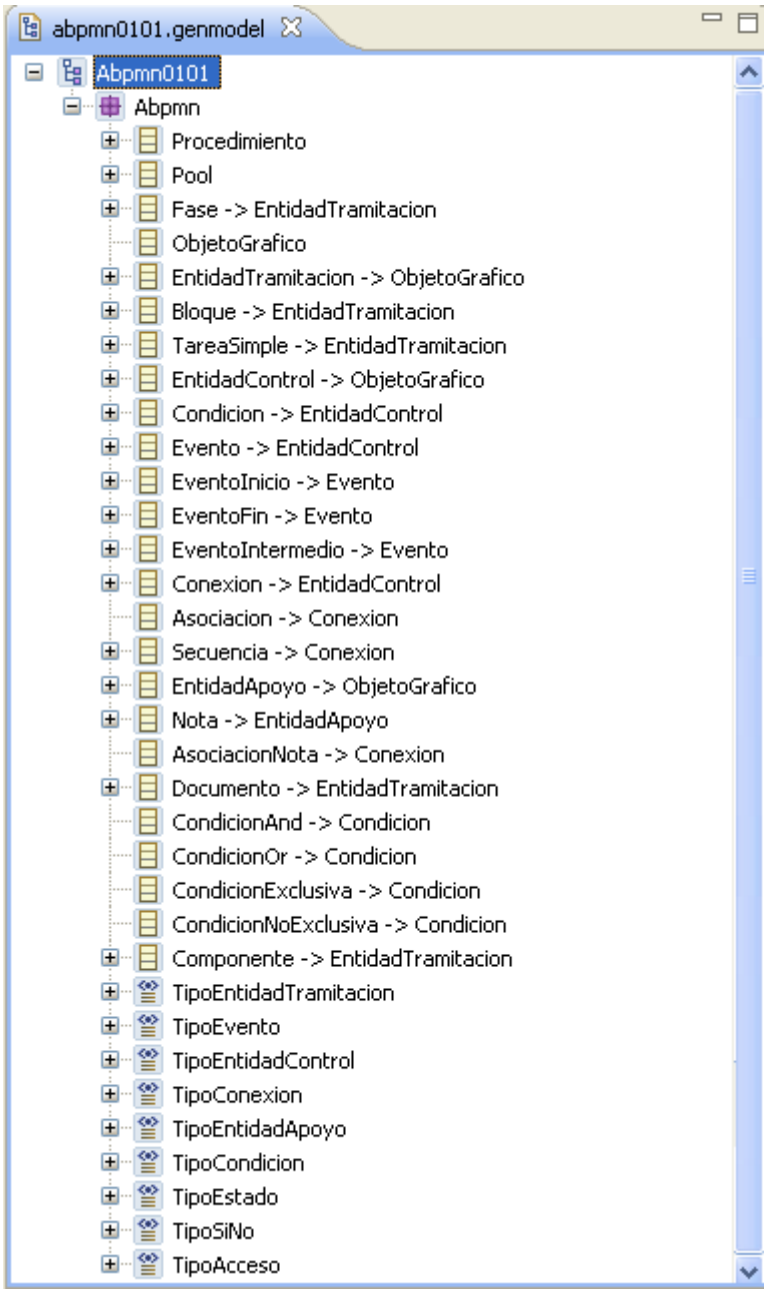


Figura 6.20. Abpmn0101 .genmodel.

6.4.2. Generar el Código asociado al Meta-Modelo (src)

Una vez que tenemos generado el fichero abpmn0101.genmodel, ya podemos generar el código asociado a dicho modelo. El código que se genera se sitúa por defecto en la carpeta "src" que se genera vacía cuando se crea el proyecto inicial. A continuación se explica como generar dicho código.

Abrimos el fichero abpmn0101.genmodel y se abrirá una ventana con dicho modelo, nos situamos sobre el raíz del abpmn0101.genmodel y pulsamos el botón derecho del ratón, se despliega una ventana con una serie de opciones, en nuestro caso seleccionamos "Generate Model code" para generar el código del proyecto.

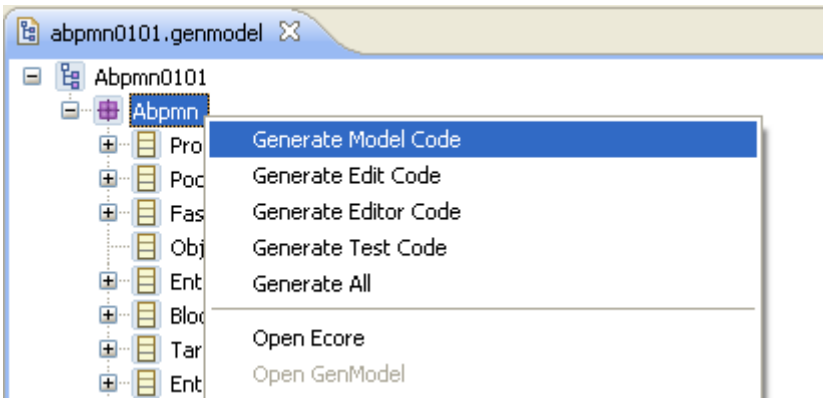


Figura 6.21. Abpmn0101 .genmodel. Ventana Generar código.

De esta manera obtenemos el código del proyecto. En la siguiente imagen se muestran los ficheros que se han generado mediante esta opción.

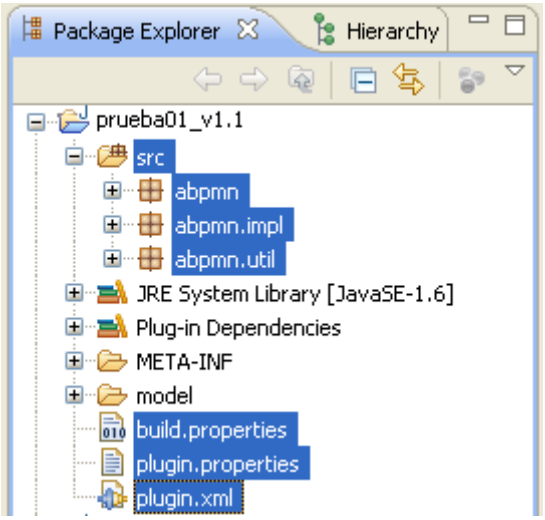


Figura 6.22. Abpmn0101 .genmodel. Código Generado.

6.4.2.1. Modificaciones del Código (src)

Para cumplir con las especificaciones de la fase de análisis, ha sido necesario realizar una serie de modificaciones al código generado por la herramienta.

En este caso se trata de añadir la validación de los atributos tipo fecha, es decir, fecha de creación, fecha de modificación y fecha de publicación del procedimiento.

Para ello ha sido necesario generar una nueva clase con el contenido referente a dicha validación denominada "TratamientoFechas.java" e incorporarla a los ficheros generados en la carpeta "src/abpmn.impl/".

Por otro lado, se ha tenido que añadir el código necesario al fichero "src/abpmn.impl/ProcedimientoImpl.java" para llevar a cabo dicha validación desde la herramienta.

A continuación se muestra el contenido del fichero "TratamientoFechas.java" generado.

```
package abpmn.impl;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.Date;
import java.util.Locale;
import java.util.Calendar;
import java.lang.Integer;

public class TratamientoFechas {

     /*CARGAR FECHA SISTEMA*/
     public static String FechaSistema()
     {
        Calendar calendario = Calendar.getInstance();
        String fecha,d1, m1;
        int y, m, d;
        y = calendario.get(Calendar.YEAR);
        m = calendario.get(Calendar.MONTH)+1;
        d = calendario.get(Calendar.DAY_OF_MONTH);
        if ((d < 10) && (m < 10)){
           d1 = "0" + d;
           m1 = "0" + m;
           fecha=d1+"/"+m1+"/"+y;
        }else  if (d < 10){
           d1 = "0" + d;
           fecha=d1+"/"+m+"/"+y;
        }else if (m < 10){
```

```
            m1 = "0" + m;
            fecha=d+"/"+m1+"/"+y;
         }else{
            fecha=d+"/"+m+"/"+y;
         }
         return fecha;
      }
      /*FIN CARGAR FECHA SISTEMA*/


      /*VALIDAR FECHA*/

      /*validar el formato de entrada*/
      /*year*/
      public static int convertYear(String pYear){
         try{
            int year = Integer.parseInt(pYear);
            if(year<0) return 0;
            return year;
         }catch(Exception e){
            return 0;
         }
      }

      /*month: [1..12]*/
      public static int convertMonth(String pMonth){
         try{
            int month = Integer.parseInt(pMonth);
            if(month<1 || month>12) return 0;
            return month;
         }catch(Exception e){
            return 0;
         }
      }

      /*day: [1..31] depende de  month and year*/
      public static int convertDay(String pYear, String pMonth, String
pDay){
         try{
            int year = convertYear(pYear);
            if(year == 0) return 0;
            int month = convertMonth(pMonth);
            if(month == 0) return 0;
            int day = Integer.parseInt(pDay);
            int daysOfMonth = getDaysOfMonths(pYear,pMonth);
            if(daysOfMonth>0){
               if(day>=1 && day<=daysOfMonth) return day;
               return 0;
            }
            return 0;
         }catch(Exception e){
            return 0;
         }
      }

      public static int getDaysOfMonths(String pYear, String pMonth){
         try{
            int year = convertYear(pYear);
            if(year == 0) return 0;
            int month = convertMonth(pMonth);
            if(month == 0) return 0;
```

```
          /*january, march, may, july, august,october, december*/
          if(month == 1 || month == 3 || month == 5 || month == 7 ||
month == 8 || month == 10 || month == 12) return 31;
          /*april, june,september,november*/
          if(month == 4 || month == 6 || month == 9 || month == 11 )
return 30;
          /*february*/
          if(month == 2 && isLeap(pYear)) return 29;
          if(month == 2 && !isLeap(pYear)) return 28;
          return 0;
        }catch(Exception e){
          return 0;
        }
     }

     public static boolean isLeap(String pYear){
        try{
          int year = convertYear(pYear);
          if(year == 0) return false;
          /*divisible per 4 but not divisible per 100*/
          if(year%4 == 0 && year%100 != 0) return true;
          return false;
        }catch(Exception e){
          return false;
        }
     }


     public static String ValidarFecha(String fecha){
        String fecha1,m1,d1;
        int y, m, d;
        if (fecha == "") fecha1 = fecha;
        else fecha1 = "Incorrecta";
        if (fecha.length() == 10)
        {
          String s1=fecha.substring(2,3);
          String s2=fecha.substring(5,6);
          if ((s1.equals("/")) && (s2.equals("/")))
          {
            String pYear=fecha.substring(6,10);
            y = convertYear(pYear);
            if (y != 0){
              String pMonth=fecha.substring(3,5);
              m = convertMonth(pMonth);
              if (m != 0){
                String pDay=fecha.substring(0,2);
                d = convertDay(pYear, pMonth, pDay);
                if (d != 0){
                  if ((d < 10) && (m < 10)){
                    d1 = "0" + d;
                    m1 = "0" + m;
                    fecha1=d1+"/"+m1+"/"+y;
                  }else if (d < 10){
                    d1 = "0" + d;
                    fecha1=d1+"/"+m+"/"+y;
                  }else if (m < 10){
                    m1 = "0" + m;
                    fecha1=d+"/"+m1+"/"+y;
                  }else{
                    fecha1=d+"/"+m+"/"+y;
                  }
```

```
                }
              }
            }
          }
        }
          return fecha1;
     }


      public static int CompararFecha(String fechaC,String fecha ){
        int valor = 0;
        DateFormat
df=DateFormat.getDateInstance(DateFormat.SHORT,Locale.getDefault());
        try {
          Date dC = df.parse(fechaC);
          Date d = df.parse(fecha);
          valor = dC.compareTo(d);
        } catch (ParseException e) {
          e.printStackTrace();
        }
        return valor;
     }


     /*           FIN VALIDAR FECHA         */
}
```

Figura 6.23. TratamientoFechas.java.

En la siguiente tabla se muestran las modificaciones realizadas al fichero "ProcedimientoImpl.java" para incorporar las validaciones necesarias en los campos fecha del procedimiento.

```
import java.util.Calendar;

protected static final String FECHA_MODIFICACION_PROC_EDEFAULT =
TratamientoFechas.FechaSistema();


public void setFechaCreacionProc(String newFechaCreacion) {

     String fechaOK =TratamientoFechas.ValidarFecha(newFechaCreacion);
        if (fechaOK == "Incorrecta") fechaCreacionProc = "Error, el
formato debe ser: (dd/mm/yyyy)";
        else
        {
          int Error
=TratamientoFechas.CompararFecha(fechaOK,fechaModificacionProc);
          if (Error == 1) fechaCreacionProc = "Error, la fecha de
creación no puede ser mayor que la de modificación";
          else fechaCreacionProc = fechaOK;
```

```
        }

     String oldFechaCreacion = fechaCreacionProc;
     if (eNotificationRequired())
       eNotify(new ENotificationImpl(this, Notification.SET,
AbpmnPackage.PROCEDIMIENTO__FECHA_CREACION_PROC, oldFechaCreacion,
fechaCreacionProc));
   }


public void setFechaPublicacionProc(String newFechaPublicacion) {
     String oldFechaPublicacion = fechaPublicacionProc;
     if (newFechaPublicacion == null)
     {
         fechaPublicacionProc = newFechaPublicacion;
     }else
     {
       String fechaOK
=TratamientoFechas.ValidarFecha(newFechaPublicacion);
       if (fechaOK == "Incorrecta") fechaPublicacionProc = "Error, el
formato debe ser: (dd/mm/yyyy)";
       else
       {
         int Error
=TratamientoFechas.CompararFecha(fechaModificacionProc,fechaOK);
         if (Error == 1) fechaPublicacionProc = "Error, la fecha de
publicación debe ser mayor o igual que la de creación y modificación";
         else fechaPublicacionProc = fechaOK;
       }
     }
     if (eNotificationRequired())
       eNotify(new ENotificationImpl(this, Notification.SET,
AbpmnPackage.PROCEDIMIENTO__FECHA_PUBLICACION_PROC,
oldFechaPublicacion, fechaPublicacionProc));
   }
```

Figura 6.24. Modificaciones en el fichero ProcedimientoImpl.java.

6.4.3. Generar el Código asociado a los Editores no Gráficos (.Edit y .Editor)

Las carpetas .edit y .editor se generan también a partir del fichero abpmn0101.genmodel.

Para ello, abrimos el fichero abpmn0101.genmodel y se muestra una ventana con dicho modelo, nos situamos sobre el raíz del abpmn0101.genmodel y pulsamos el botón derecho del ratón, se despliega una ventana con una serie de opciones, en nuestro caso seleccionamos "Generate Edit code" para generar la carpeta prueba01_v1.1.edit.

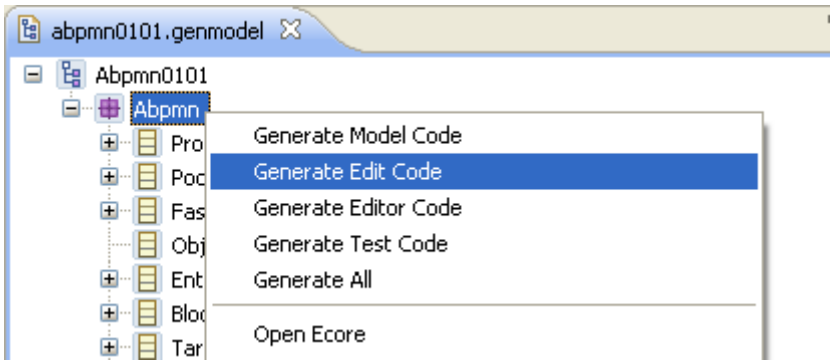


Figura 6.25. abpmn0101 .genmodel. Ventana Generar carpeta .edit.

De esta manera obtenemos la carpeta prueba01_v1.1.edit. En la siguiente imagen se muestran los ficheros que se han generado mediante esta opción dentro de la carpeta prueba01_v1.1.edit.

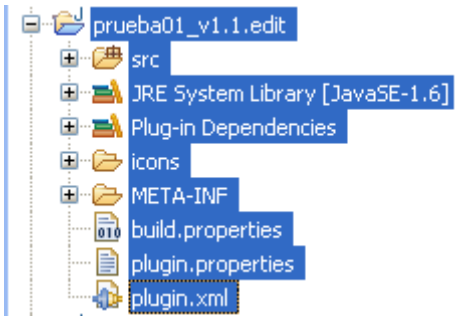


Figura 6.26. Carpeta .edit.

Para generar la carpeta .editor lo haríamos igual que en el caso anterior, pero ahora seleccionamos "Generate Editor code" para generar la carpeta prueba01_v1.1.editor.

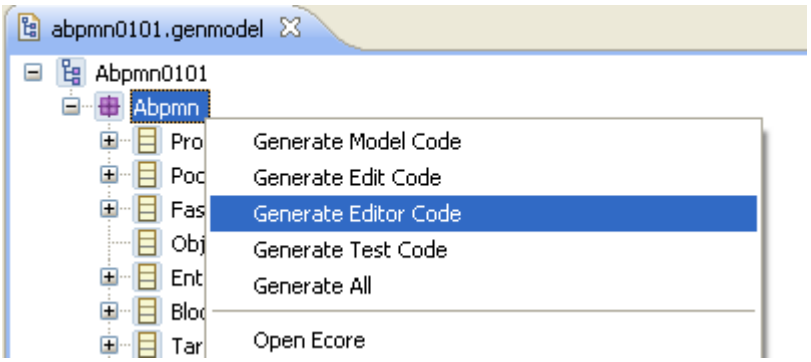


Figura 6.27. abpmn0101 .genmodel. Ventana Generar carpeta .editor.

De esta manera obtenemos la carpeta prueba01_v1.1.editor. En la siguiente imagen se muestran los ficheros que se han generado mediante esta opción en dicha carpeta.

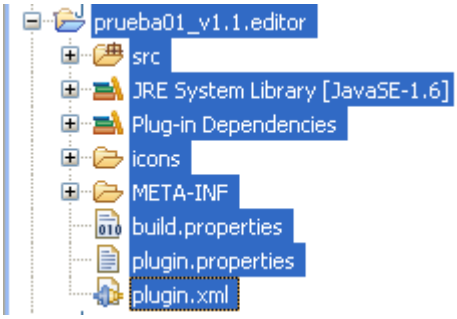


Figura 6.28. Carpeta .editor.

6.4.3.1. Modificaciones del Código del Editor no Gráfico .Edit

Para cumplir con las especificaciones de la fase de análisis, ha sido necesario realizar una serie de modificaciones al código generado por la herramienta en la carpeta .edit.

En este caso, se trata de añadir el código necesario para cargar la lista con los valores del atributo "Categoria" de las clases Bloque, Procedimiento y Tarea Simple.

Para ello ha sido necesario generar una base de datos, en la que se ha incorporado una tabla llamada Categoria, con los campos Id y Nombre. En dicha tabla se han insertado los valores para el atributo categoría. De esta manera, cada vez que cambien dichos valores, simplemente habrá que modificarlos en la base de datos y no hará falta modificar el código.

Para poder conectar la base de datos a la aplicación, será necesario realizar una conexión DNS a dicha base de datos desde el equipo donde se ejecuta la aplicación.

Por otro lado, se ha tenido que añadir el código necesario a los ficheros BloqueItemProvider.java, TareaSimpleItemProvider.java y ProcedimientoItemProvider.java que se encuentran en ".edit/src/abpmn.provider/", para poder cargar la lista de valores del atributo Categoría.

A continuación se muestra el contenido de los ficheros modificados.

BloqueItemProvider.java

```
protected void addCategoriasBloqPropertyDescriptor(Object object) {
     itemPropertyDescriptors.add
       (new ItemPropertyDescriptor

  (((ComposeableAdapterFactory)adapterFactory).getRootAdapterFactory(),
          getResourceLocator(),
          getString("_UI_Bloque_CategoriasBloq_feature"),
          getString("_UI_PropertyDescriptor_description",
"_UI_Bloque_CategoriasBloq_feature", "_UI_Bloque_type"),
          AbpmnPackage.Literals.BLOQUE__CATEGORIAS_BLOQ,
          true,
          false,
          false,
          ItemPropertyDescriptor.GENERIC_VALUE_IMAGE,
          null,
          null){
         @Override
         protected Collection<String> getComboBoxObjects(Object
object) {
            List<String> Valores = new ArrayList<String>();
            Connection con=null;
            try
            {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection("jdbc:odbc:Epac");
            Statement stmt=con.createStatement();
            ResultSet rs=stmt.executeQuery("Select Nombre from
Categoria");
            while(rs.next())
            {
               String s=rs.getString("Nombre");
               Valores.add(s);
            }
            }catch (Exception e){
               System.out.println("Excepcion e:"+e.getMessage());
               e.printStackTrace();
            }finally{
               if (con!=null){
                  try{
                     con.close();
                  }catch(Exception e){}
               }
            }
         return Valores;
         }
       }
         );
       }
```

ProcedimientoItemProvider.java

```
protected void addCategoriasProcPropertyDescriptor(Object object) {
     itemPropertyDescriptors.add
       (new ItemPropertyDescriptor

  (((ComposeableAdapterFactory)adapterFactory).getRootAdapterFactory(),
          getResourceLocator(),
```

```
            getString("_UI_Procedimiento_CategoriasProc_feature"),
            getString("_UI_PropertyDescriptor_description",
"_UI_Procedimiento_CategoriasProc_feature", "_UI_Procedimiento_type"),
            AbpmnPackage.Literals.PROCEDIMIENTO__CATEGORIAS_PROC,
            true,
            false,
            false,
            ItemPropertyDescriptor.GENERIC_VALUE_IMAGE,
            null,
            null){
          @Override
          protected Collection<String> getComboBoxObjects(Object
object) {
             List<String> Valores = new ArrayList<String>();
             Connection con=null;
             try
             {
             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             con=DriverManager.getConnection("jdbc:odbc:Epac");
             Statement stmt=con.createStatement();
             ResultSet rs=stmt.executeQuery("Select Nombre from
Categoria");
             while(rs.next())
             {
               String s=rs.getString("Nombre");
               Valores.add(s);
             }
             }catch (Exception e){
               System.out.println("Excepcion e:"+e.getMessage());
               e.printStackTrace();
             }finally{
               if (con!=null){
                  try{
                    con.close();
                  }catch(Exception e){}
               }
             }
           return Valores;
           }
        }
           );
        }
```

TareaSimpleItemProvider.java

```
protected void addCategoriasTSimplePropertyDescriptor(Object object) {
     itemPropertyDescriptors.add
       (new ItemPropertyDescriptor

  (((ComposeableAdapterFactory)adapterFactory).getRootAdapterFactory(),
            getResourceLocator(),
            getString("_UI_TareaSimple_CategoriasTSimple_feature"),
            getString("_UI_PropertyDescriptor_description",
"_UI_TareaSimple_CategoriasTSimple_feature", "_UI_TareaSimple_type"),
            AbpmnPackage.Literals.TAREA_SIMPLE__CATEGORIAS_TSIMPLE,
            true,
            false,
            false,
            ItemPropertyDescriptor.GENERIC_VALUE_IMAGE,
            null,
```

```
           null){
          @Override
          protected Collection<String> getComboBoxObjects(Object
object) {
             List<String> Valores = new ArrayList<String>();
             Connection con=null;
             try
             {
             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             con=DriverManager.getConnection("jdbc:odbc:Epac");
             Statement stmt=con.createStatement();
             ResultSet rs=stmt.executeQuery("Select Nombre from
Categoria");
             while(rs.next())
             {
               String s=rs.getString("Nombre");
               Valores.add(s);
             }
             }catch (Exception e){
               System.out.println("Excepcion e:"+e.getMessage());
               e.printStackTrace();
             }finally{
               if (con!=null){
                  try{
                    con.close();
                  }catch(Exception e){}
               }
             }
          return Valores;
          }
       }
          );
       }
```

Figura 6.29. Modificaciones para cargar lista valores del atributo Categoria.

6.4.3.2. Modificación de los iconos del Editor no Gráfico .Edit

Por defecto en la carpeta ".edit\icons\full\obj16" se generan los iconos en formato .gif que aparecerán en la paleta de la herramienta gráfica. A continuación se muestra un ejemplo de algunos de estos iconos:

Bloque.gif Asociacion.gif

Pero en nuestro caso, la paleta debe mostrar los iconos con la misma imagen asociada al elemento que queremos pintar en la pizarra, por lo tanto, se han tenido que generar nuevos iconos .gif para que cumplieran con esta especificación requerida en la fase de análisis.

Una vez generados los iconos nuevos en formato .gif, se añaden a la carpeta ".edit\icons\full\obj16" para que aparezcan en la paleta de la herramienta gráfica cuando esta sea generada.

A continuación, se muestran los iconos generados.

Nombre del icono	Imagen icono
Asociación.gif	
AsociaciónNota.gif	
Bloque.gif	
Componente.gif	
CondicionAnd.gif	
CondicionExclusiva.gif	
CondicionNoExclusiva.gif	
CondicionOr.gif	
Documento.gif	
EventoFin.gif	
EventoInicio.gif	
EventoIntermedio.gif	
Fase.gif	
Nota.gif	
Pool.gif	
Secuencia.gif	
TareaSimple.gif	

Figura 6.30. Nuevos iconos generados en .edit.

6.5. DISEÑO DE LA APLICACIÓN GMF A PARTIR DEL META-MODELO

El siguiente paso, es generar el fichero abpmn0101.gmfgraph, dicho fichero se genera a partir del fichero abpmn0101.ecore.

El modelo .gmfgraph nos permite definir las figuras, los nodos, los enlaces, etc que se visualizarán posteriormente en el diagrama.

6.5.1. Generar los elementos Gráficos (.Gmfgraph)

A continuación, se explica como se ha obtenido el fichero .gmfgraph a partir del .ecore ya generado.

Nos posicionamos sobre el fichero .ecore y pulsamos el botón derecho del ratón, de esta manera obtenemos la ventana que se observa a continuación. Pulsaremos New → Other…

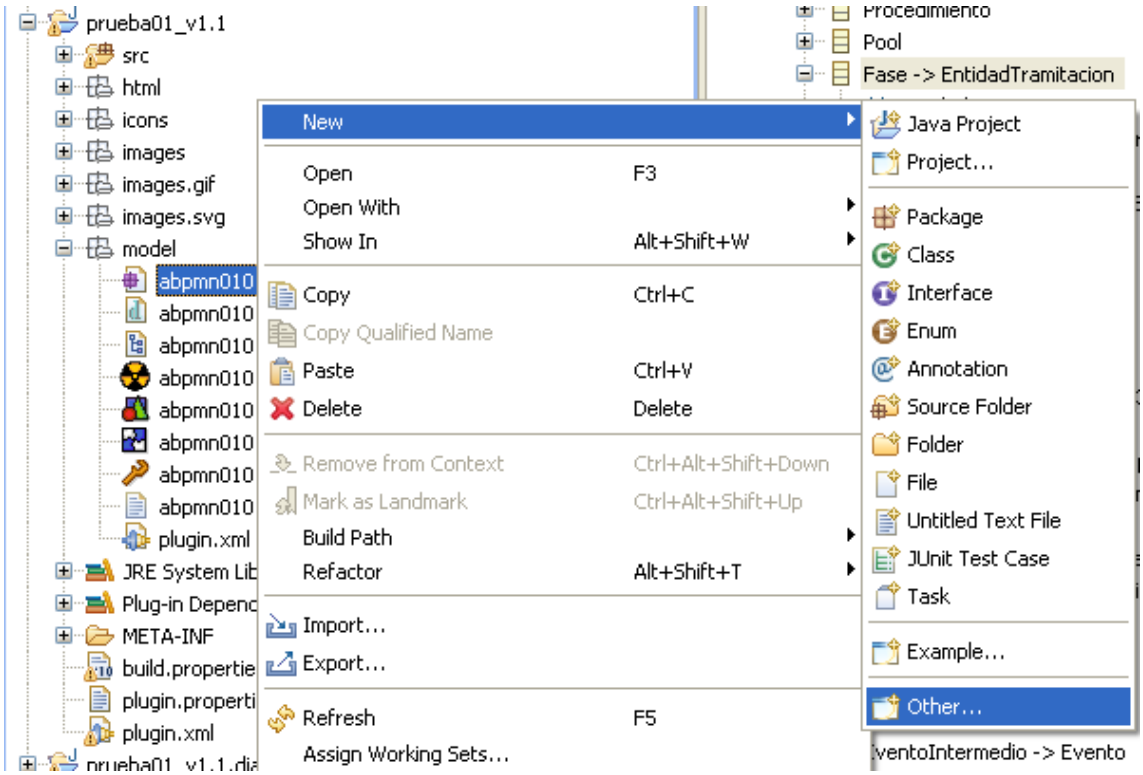


Figura 6.31. Generar .Gmfgraph a partir del .Ecore.

Y obtendremos la siguiente ventana donde seleccionaremos "GMFGraph Model" dentro de la opción Graphical Modeling Framework → Default.

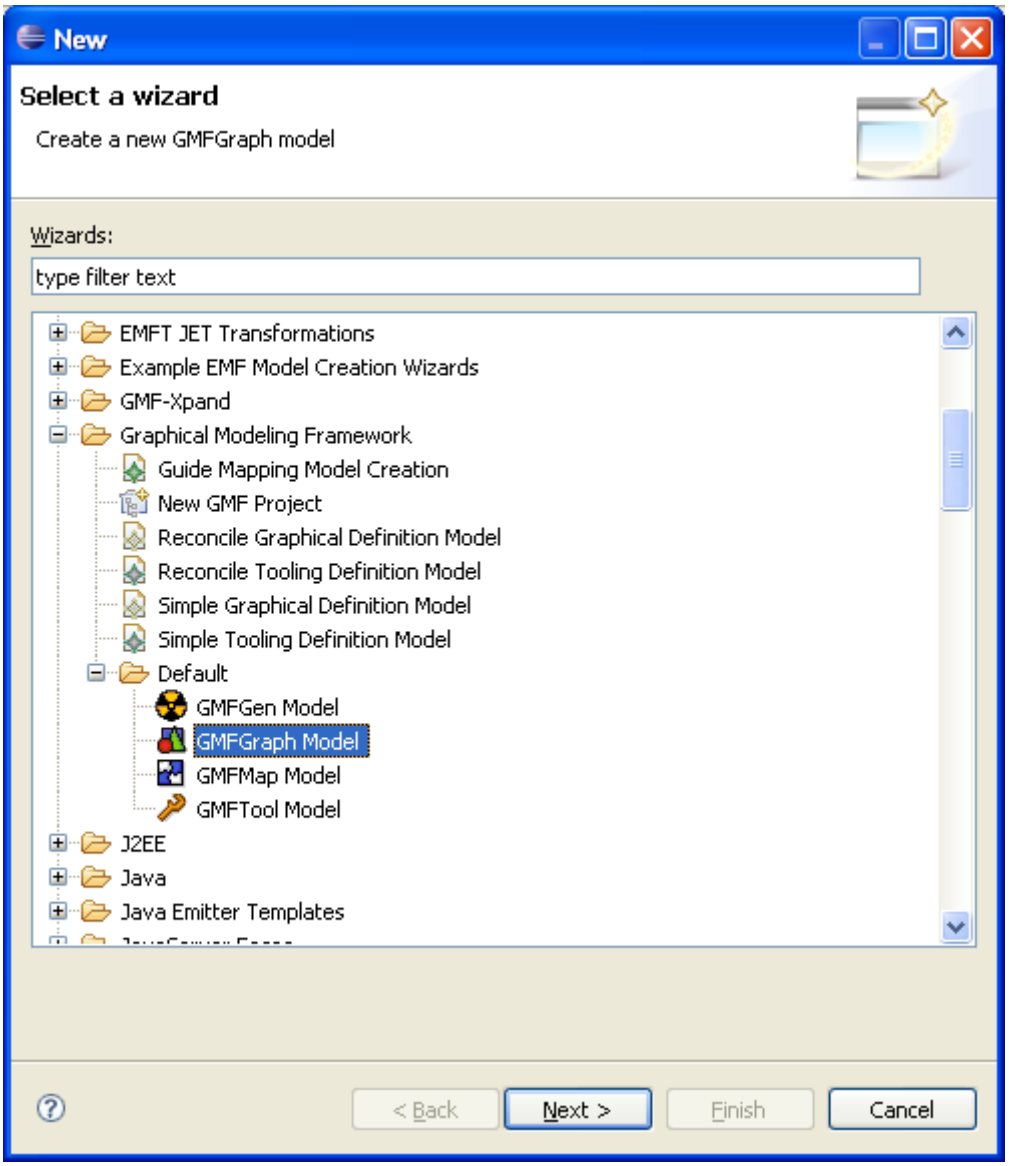


Figura 6.32. Generar .Gmfgraph. Ventana New.

De esta manera se desplegará la siguiente ventana donde pondremos el nombre del nuevo fichero .gmfgraph. En nuestro caso abpmn0101. gmfgraph.

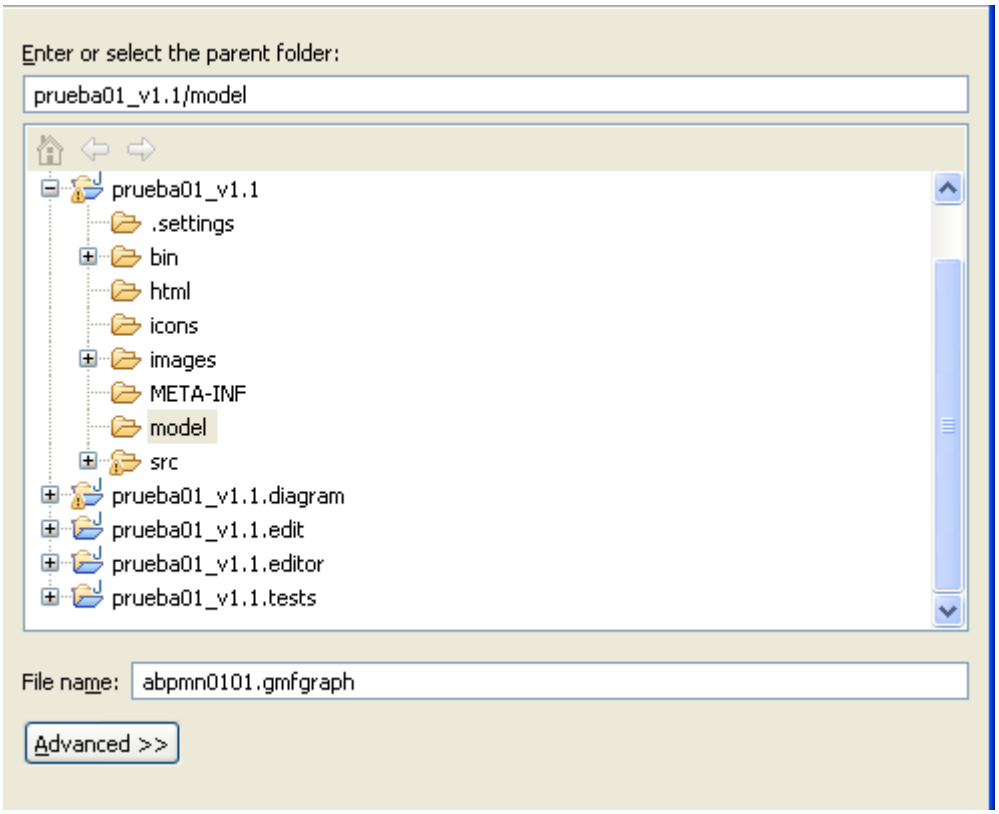


Figura 6.33. Generar .Gmfgraph. Ventana Nombre del Fichero.

Una vez que hemos completado el nombre del fichero, pasaremos a la siguiente ventana, donde seleccionamos la ruta del modelo del que queremos obtener el nuevo fichero .gmfgraph. En nuestro caso seleccionaremos la ruta donde se encuentra el fichero abpmn0101.ecore generado anteriormente.

Observamos que al cargar el .ecore, aparecen en la pantalla todos los elementos que forman parte del modelo.

Dichos elementos son las figuras, los nodos, los enlaces, etc que se visualizarán posteriormente en la herramienta gráfica generada.

Debemos seleccionar, cual será el elemento root del modelo .gmfgraph, en nuestro caso seleccionamos Procedimiento.

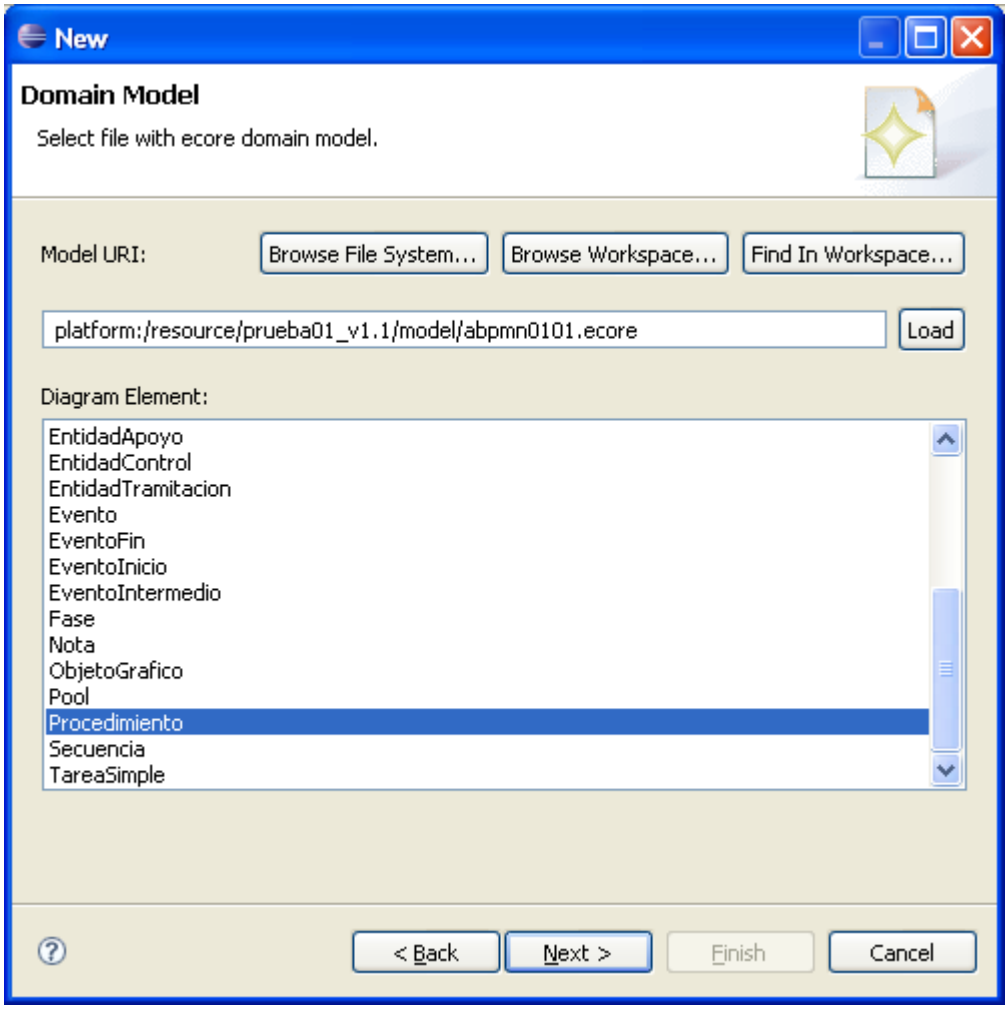


Figura 6.34. Generar .Gmfgraph. Ventana Seleccionar Modelo y Root.

A continuación, pasamos a la siguiente pantalla donde aparecen todos los elementos del modelo con sus atributos correspondientes, en esta pantalla se pueden seleccionar o deseleccionar los elementos como nodos, enlaces y atributos.

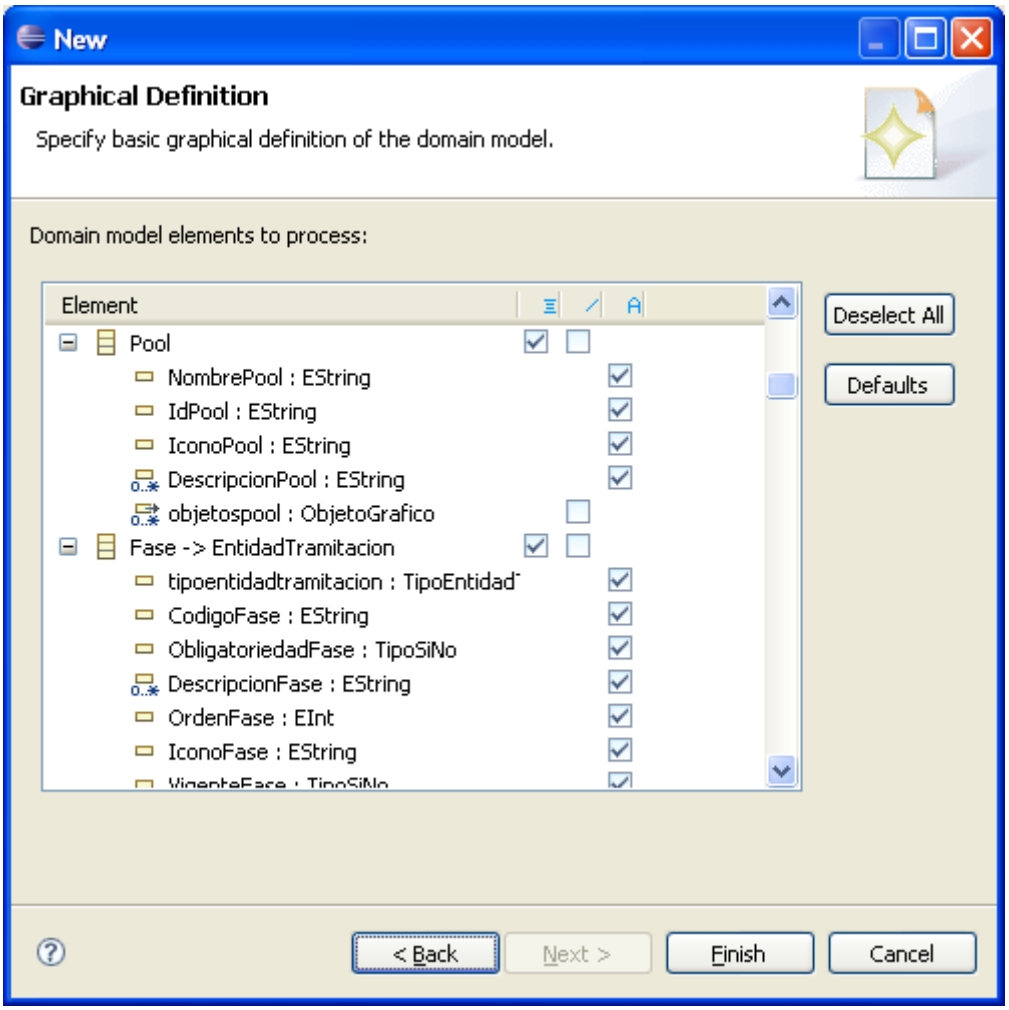


Figura 6.35. Generar .Gmfgraph. Ventana nodos, enlaces y atributos.

En este caso dejamos los valores que la herramienta ha generado por defecto, para más tarde modificar dichos valores y cumplir con las especificaciones solicitadas. De esta manera obtenemos el nuevo fichero abpmn0101.gmfgraph del que se muestra una parte a continuación.

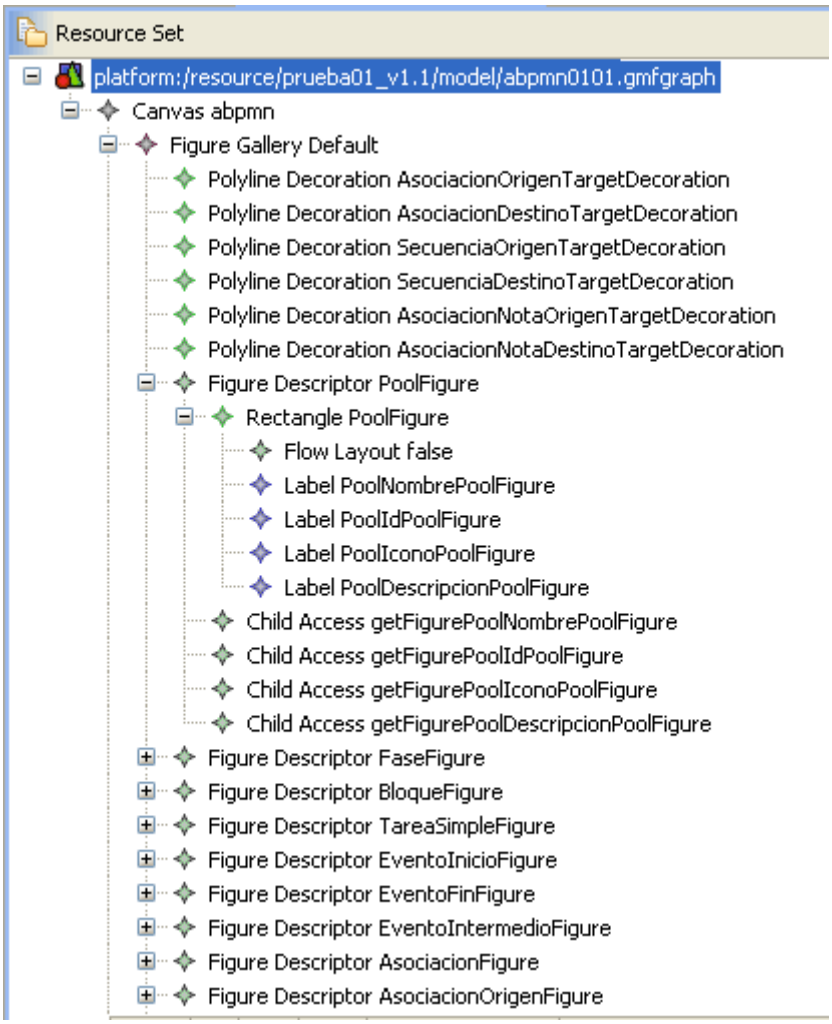


Figura 6.36. abpmn0101 .gmfgraph.

6.5.1.1. Modificaciones del Fichero .Gmfgraph

El fichero .gmfgraph que se obtiene por defecto, no cumple las especificaciones solicitadas de nuestra herramienta gráfica. Por lo que se ha tenido que transformar incorporando una serie de modificaciones y eliminando elementos que en nuestro caso no servían.

A continuación se explican las modificaciones que se han realizado al fichero para obtener uno nuevo acorde a las especificaciones solicitadas.

En primer lugar vamos a comenzar con los link o enlaces de la herramienta, es decir, las conexiones de asociación y flujo de secuencia.

Vamos a describir los pasos necesarios para generar la **conexión de flujo de secuencia** que va a utilizar nuestra herramienta gráfica y que se corresponde con la siguiente imagen ⟶ .

Para ello es necesario definir un elemento del tipo "Figure Descriptor" que será la figura correspondiente a la conexión flujo de secuencia, en este caso la denominamos "Figure Descriptor FlujoSecuenciaFigure" y dentro de ésta otro elemento del tipo "Polyline Connection", que denominamos "Polyline Connection FlujoSecuenciaFigure". Mediante el elemento Polyline Connection obtenemos una línea que mediante la ventana Properties que ofrece la herramienta, le asignaremos los atributos necesarios. Algunos de esos atributos son nombre, tipo de línea, la figura a la que hace referencia, decoración adicional en el origen o destino de la línea, etc.

Para generar la punta de la flecha, es necesario crear un nuevo elemento del tipo "Polygon Decoration" llamado "Polygon Decoration FlujoSecuenciaDecoration" y al que haremos referencia en la propiedad Target Decoration, es decir, en el final de la línea para que nos dibuje la imagen representada en dicho elemento.

En el elemento "Polygon Decoration FlujoSecuenciaDecoration", creamos un polígono con la figura correspondiente a la punta de la flecha, y le asignamos el color, en este caso negro, para que rellene dicho polígono.

El resto de elementos de tipo conexión se crearían de manera similar a la explicada, siguiendo las especificaciones necesarias para obtener la imagen de cada elemento.

En la siguiente imagen, se muestra como quedaría el fichero con los cambios realizados para generar los elementos **conexión flujo de secuencia y asociación.**

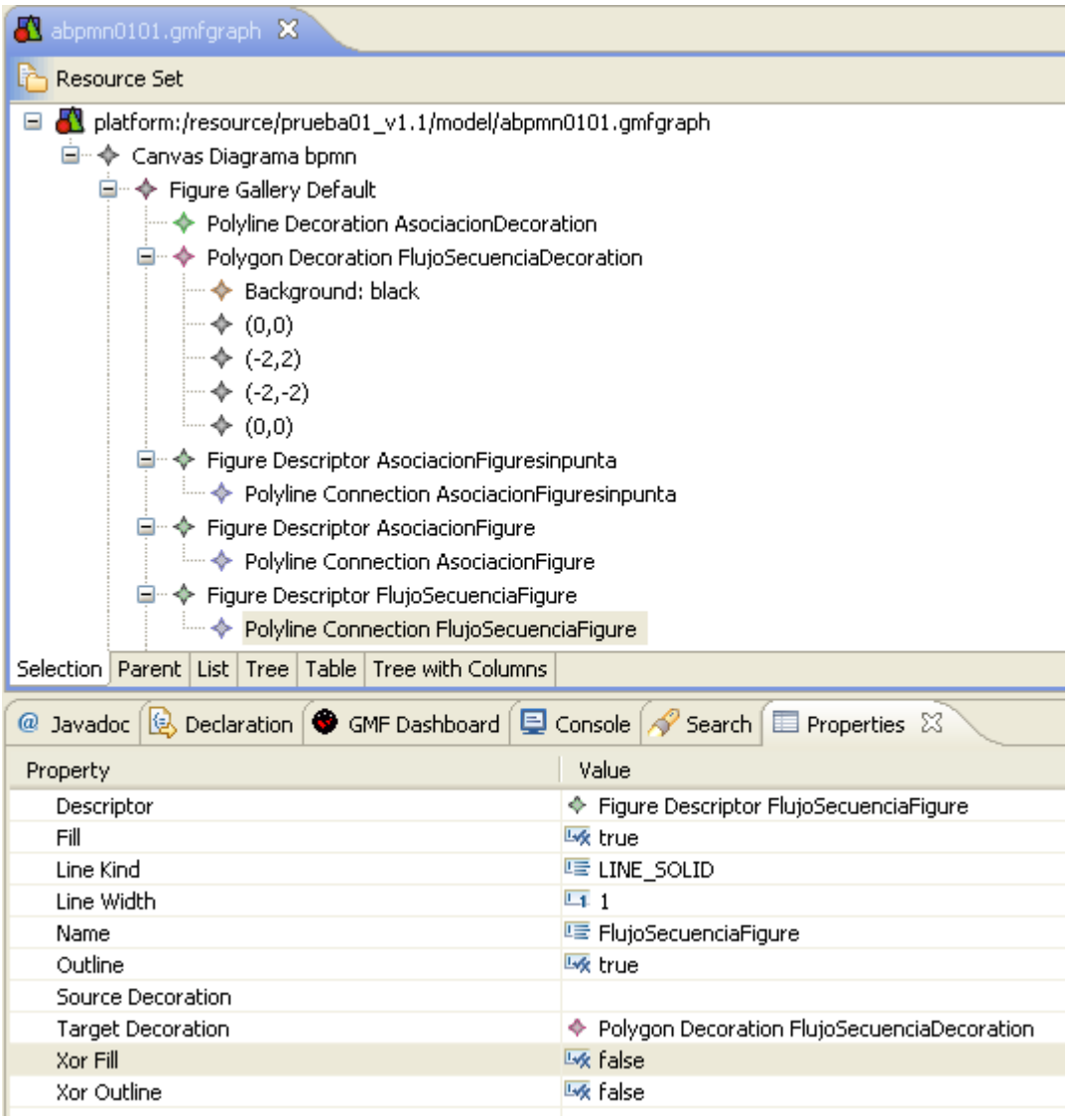


Figura 6.37. abpmn0101 .gmfgraph. Definir figuras de conexiones.

Para terminar con el proceso de creación de los elementos de conexión, faltaría crear un elemento de tipo "Connection", que haga referencia a las figuras generadas anteriormente.

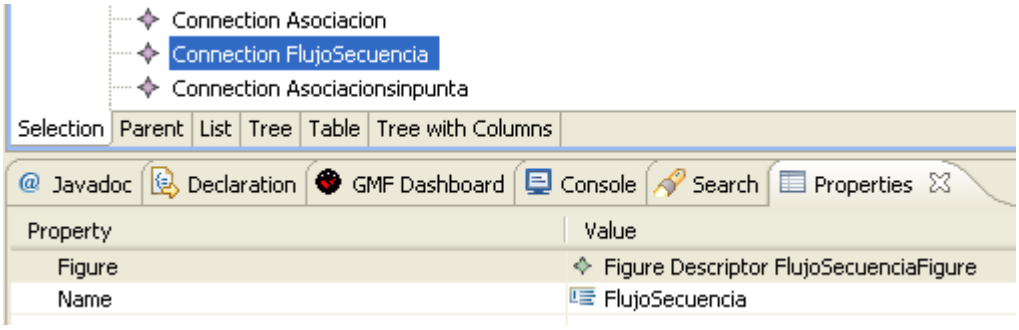


Figura 6.38. abpmn0101 .gmfgraph. Definir conexiones.

Ahora se va a explicar como generar el resto de elementos que se van a poder dibujar en la herramienta gráfica final: bloque, fase, pool, tarea simple, texto anotación, documento, evento inicio, evento fin, evento intermedio, condición exclusiva, condición no exclusiva, condición or, condición and y componente.

Las figuras Bloque, fase, tarea simple, pool, texto anotación y documento se crearían de la misma manera, por lo que se va a explicar únicamente la creación del elemento **Bloque** que sirve como ejemplo del resto, y que se corresponde con la siguiente imagen .

Para ello es necesario definir un elemento del tipo "Figure Descriptor", en este caso la denominamos "Figure Descriptor BloqueFigure" y dentro de ésta otro elemento del tipo "Rounded Rectangle", que denominamos "BloqueRoundedRectangleFigure". Mediante el elemento "Rounded Rectangle" obtenemos un rectángulo con las líneas redondeadas y mediante la ventana Properties que ofrece la herramienta, le asignaremos los atributos necesarios. Algunos de esos atributos son nombre, la figura a la que hace referencia, tamaño, etc.

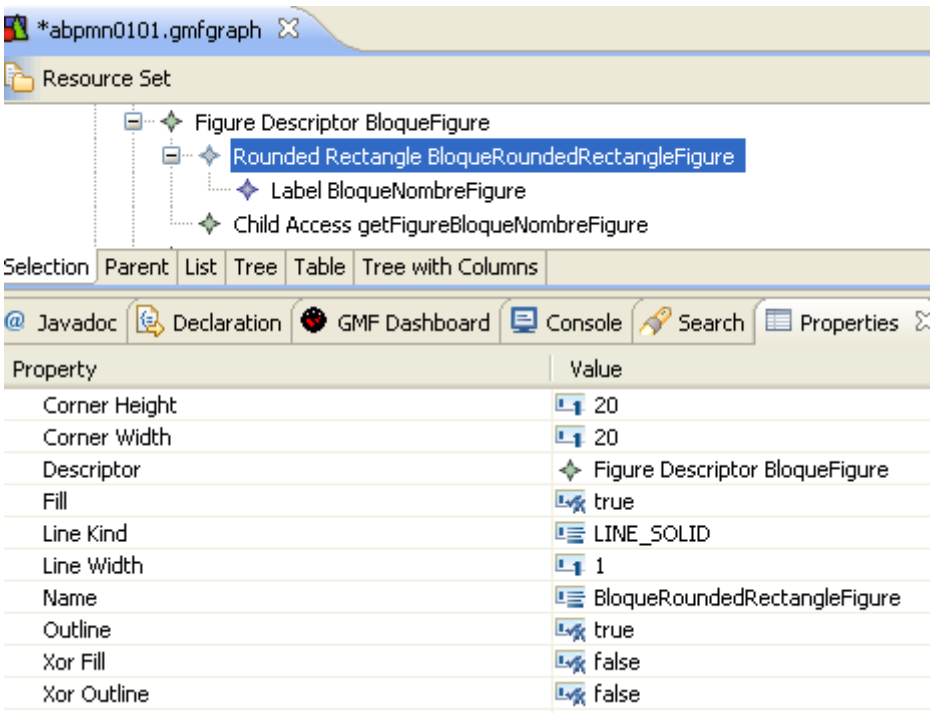


Figura 6.39. abpmn0101 .gmfgraph. Definir figura Bloque.

También generamos un elemento de tipo "Label", llamado "Label BloqueNombreFigure", para poder escribir el nombre del bloque en la figura. Para acceder a este elemento es necesario crear un elemento de tipo "Child Access" llamado "Child Access getFigureBloqueNombreFigure".

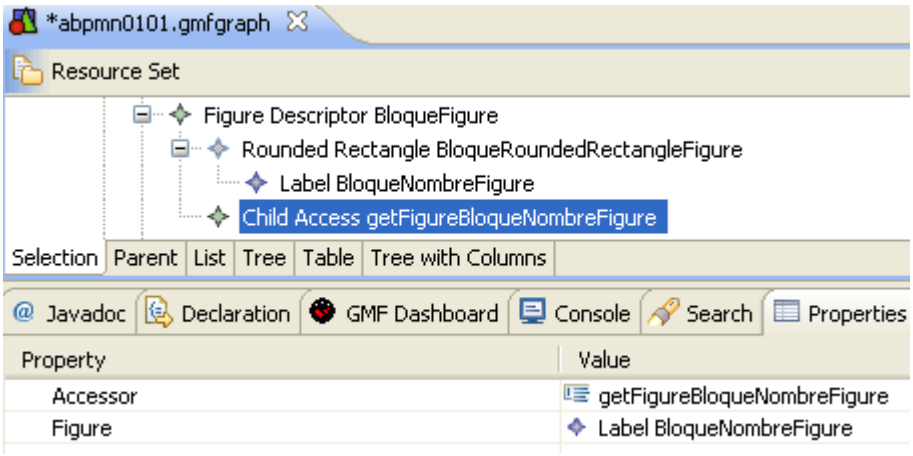


Figura 6.40. abpmn0101 .gmfgraph. Definir etiqueta nombre figura Bloque.

Y a su vez para acceder a "Child Access getFigureBloqueNombreFigure", debemos crear un elemento de tipo "Diagram Label" llamado "Diagram Label BloqueNombre".

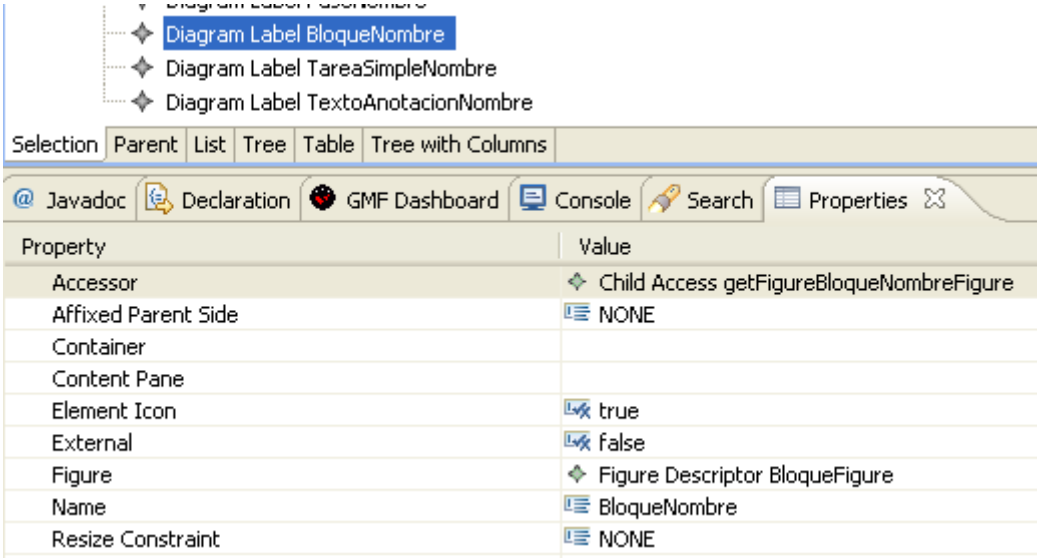


Figura 6.41. abpmn0101 .gmfgraph. Definir diagram label figura Bloque.

La figura bloque debe poder desplegarse y plegarse cuando tiene elementos dentro, para ello es necesario definir un elemento de tipo “Compartment” que haga referencia a la figura creada para dicho elemento.

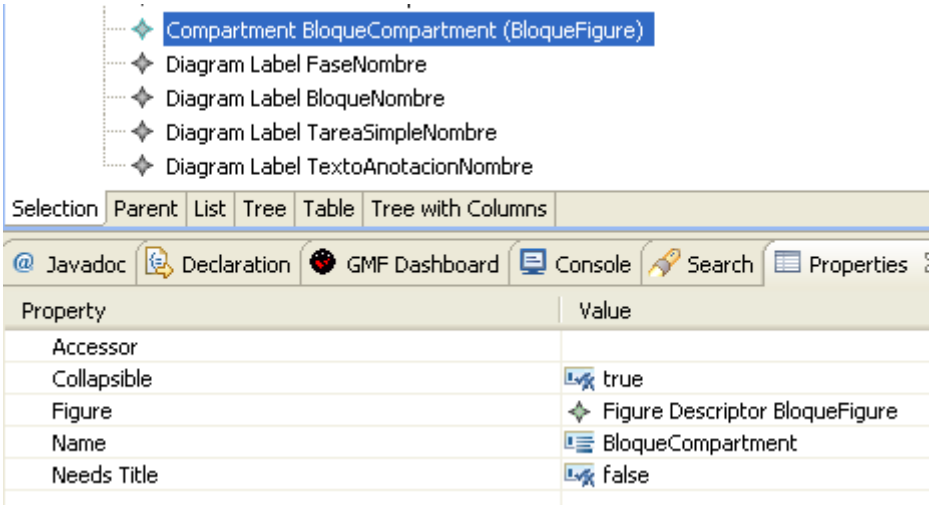


Figura 6.42. abpmn0101 .gmfgraph. Definir compartment figura Bloque.

En el caso de los elementos tarea simple, pool, fase, texto anotación y documento, no es necesario crear un elemento tipo “Compartment”, se crea un elemento tipo “Node” que hace referencia a las figuras creadas para dichos elementos.

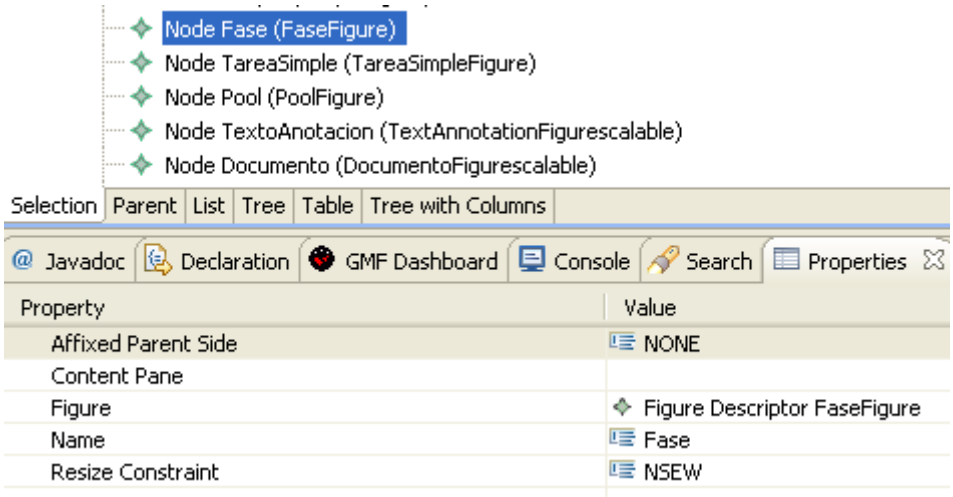


Figura 6.43. abpmn0101 .gmfgraph. Definir Node Fase.

En la siguiente imagen, se muestra como quedaría el fichero con los cambios realizados para generar los elementos **bloque, fase, pool, tarea simple, texto anotación y documento.**

Figura 6.44. abpmn0101 .gmfgraph. Figuras y nodos.

Por último se va a explicar cómo generar el resto de elementos que se van a poder dibujar en la herramienta gráfica final: evento inicio, evento fin, evento intermedio, condición exclusiva, condición no exclusiva, condición or, condición and y componente.

Puesto que dichas figuras se crearían de la misma manera, se va a explicar únicamente la creación del elemento **Evento Fin** que sirve como ejemplo del resto, y que se corresponde con la siguiente imagen .

Como en los elementos anteriores, es necesario definir un elemento del tipo “Figure Descriptor”, en este caso la denominamos “Figure Descriptor EventoFinFigure” y dentro de ésta otro elemento del tipo “Custom Figure”, que denominamos “EventoFinFigureSVG”.

Mediante el elemento "Custom Figure" se añade la ruta donde se encuentra la imagen correspondiente al elemento Evento Fin en formato SVG (Scalable Vector Graphics). Mediante la ventana Properties que ofrece la herramienta, le asignaremos los atributos necesarios. Esos atributos son el nombre, la figura a la que hace referencia, y el nombre de la clase que utilizaremos para visualizar dicho elemento.

Para poder visualizar correctamente el elemento en formato SVG, también es necesario realizar una serie de modificaciones en la carpeta .diagram que se generará posteriormente, dichas modificaciones, se explicarán en el apartado **6.6.2.1. Generar Imágenes .svg.**

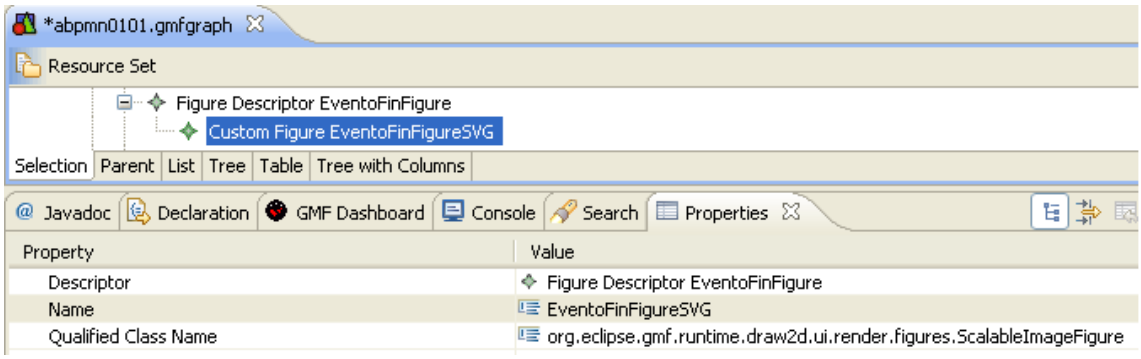


Figura 6.45. abpmn0101 .gmfgraph. Definir figura Evento Fin.

Para terminar, se crea un elemento tipo "Node" y poder hacer referencia a la figura creada para dicho elemento como en casos anteriores.

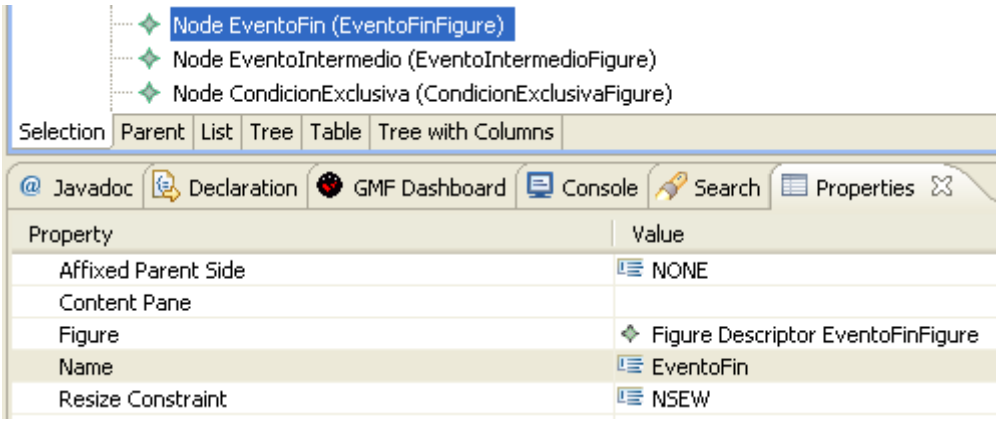


Figura 6.46. abpmn0101 .gmfgraph. Definir Node Evento Fin.

En la siguiente imagen, se muestra como quedaría el fichero con los cambios realizados para generar los elementos **evento inicio, evento fin, evento intermedio, condición exclusiva, condición no exclusiva, condición or, condición and y componente.**

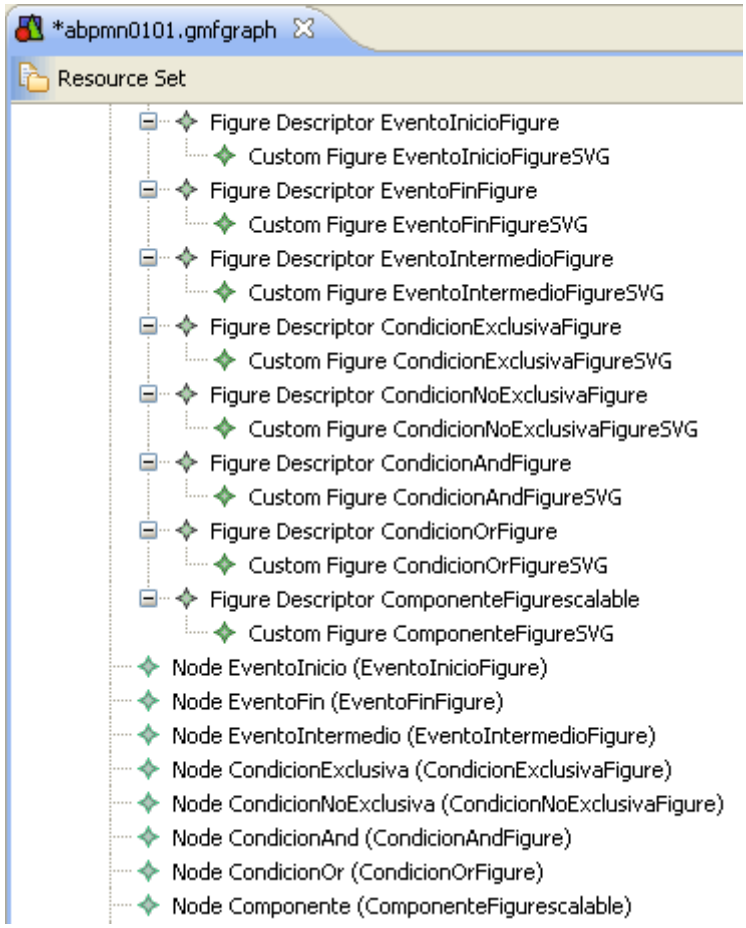


Figura 6.47. abpmn0101 .gmfgraph. Eventos, Condiciones y Componente .

6.5.2. Generar la Paleta de Herramientas (.Gmftool)

A continuación, se explica cómo se ha obtenido el fichero .gmftool a partir del .ecore ya generado.

Nos posicionamos sobre el fichero .ecore y pulsamos el botón derecho del ratón, de esta manera obtenemos la ventana que se observa a continuación. Pulsaremos New → Other…

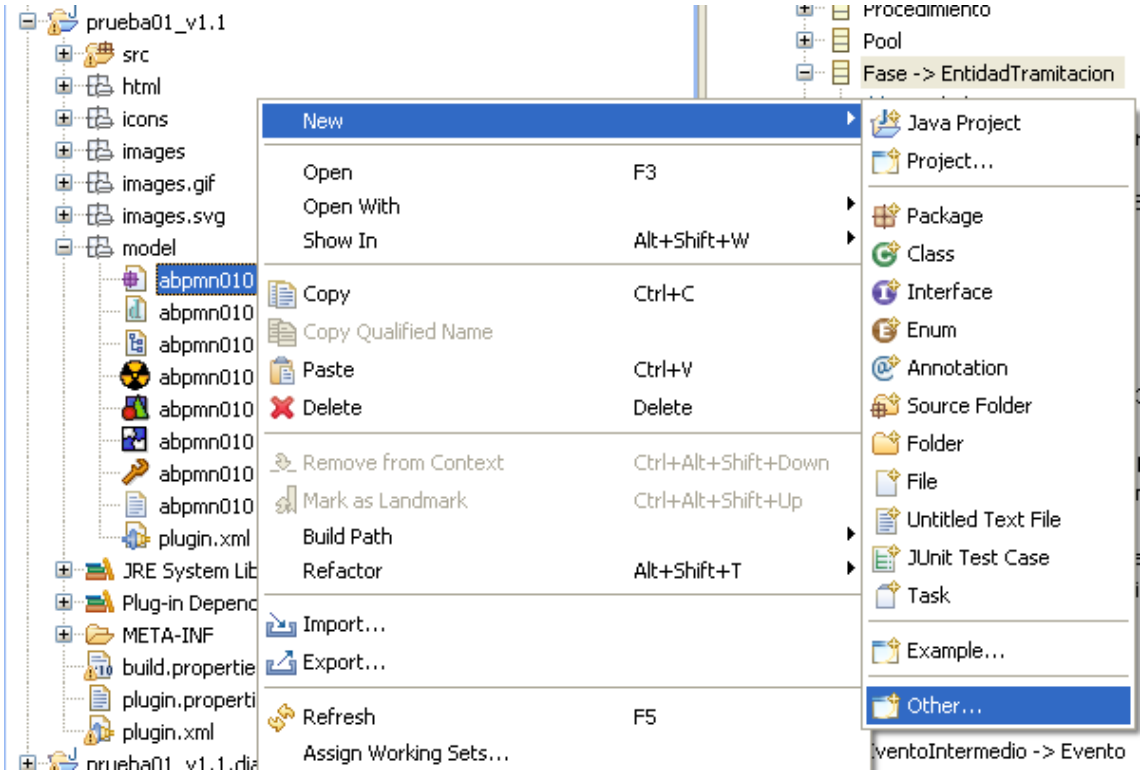


Figura 6.48. Generar .Gmftool a partir del .Ecore.

Y obtendremos la siguiente ventana donde seleccionaremos "GMFTool Model" dentro de la opción Graphical Modeling Framework → Default.

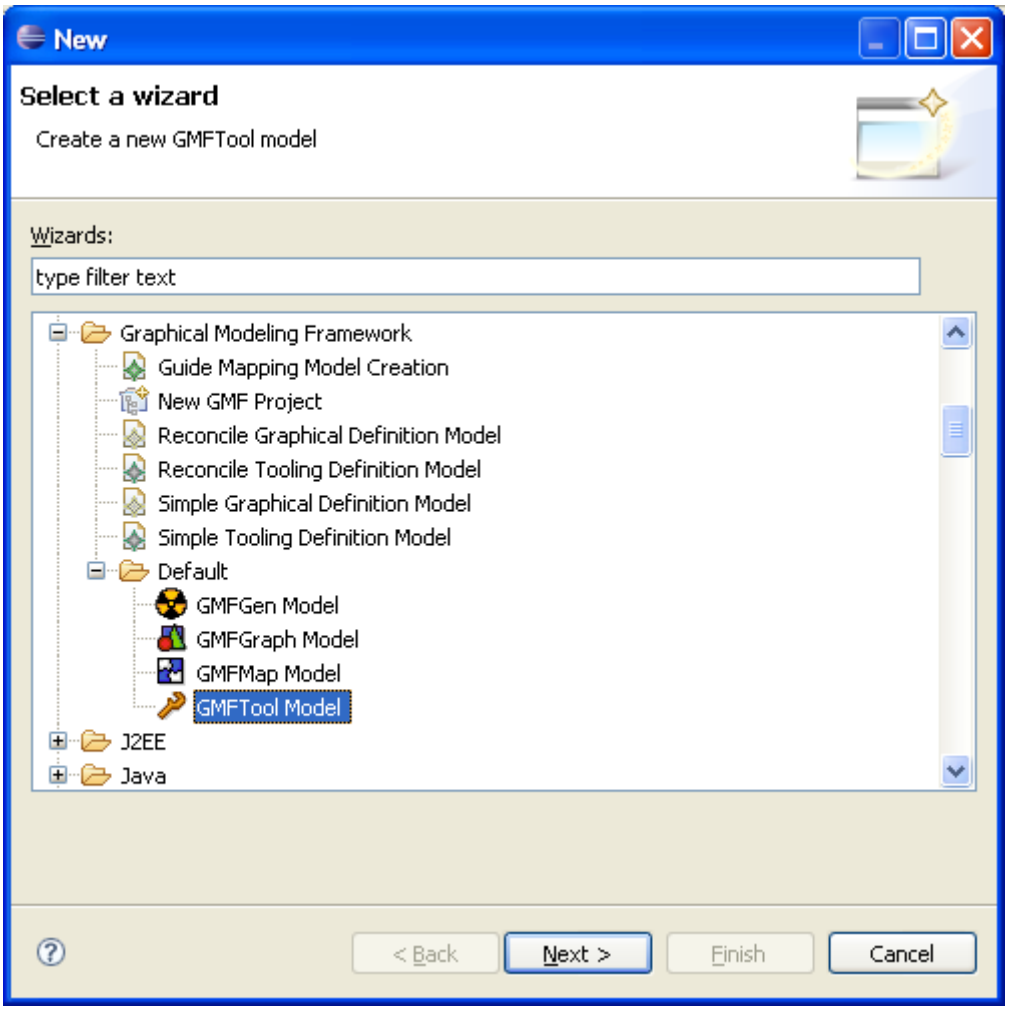


Figura 6.49. Generar .Gmftool. Ventana New.

De esta manera se desplegará la siguiente ventana donde pondremos el nombre del nuevo fichero .gmftool. En nuestro caso abpmn0101. gmftool.

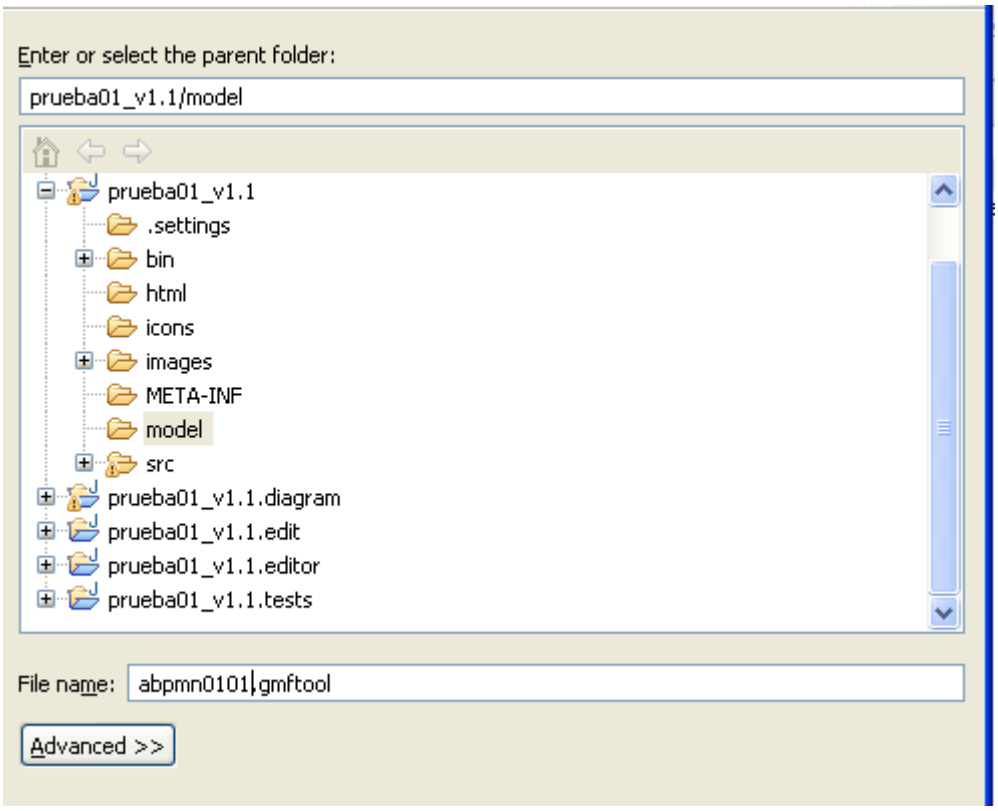


Figura 6.50. Generar .Gmftool. Ventana Nombre del Fichero.

Una vez que hemos completado el nombre del fichero, pasaremos a la siguiente ventana, donde seleccionamos la ruta del modelo del que queremos obtener el nuevo fichero .gmftool. En nuestro caso, seleccionaremos la ruta donde se encuentra el fichero abpmn0101.ecore generado anteriormente.

Observamos que al cargar el .ecore, aparecen en la pantalla todos los elementos que forman parte del modelo.

Dichos elementos se visualizarán posteriormente en la paleta de la herramienta gráfica generada.

Debemos seleccionar, cuál será el elemento root del modelo .gmftool, en nuestro caso, seleccionamos Procedimiento.

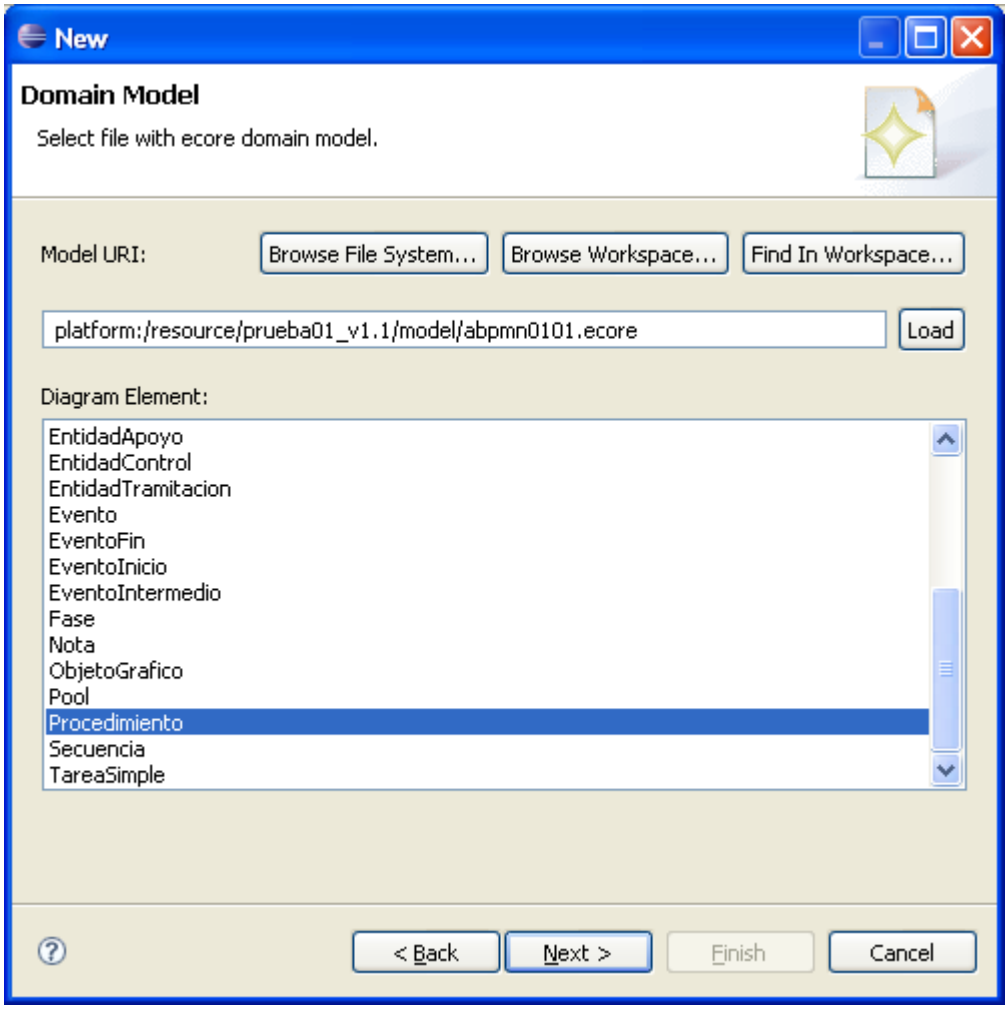


Figura 6.51. Generar .Gmftool. Ventana Seleccionar Modelo y Root.

A continuación, pasamos a la siguiente pantalla donde aparecen todos los elementos del modelo y que podrán aparecer o no en la paleta de la herramienta gráfica, para ello, en esta pantalla se pueden seleccionar o deseleccionar los elementos que se deseen.

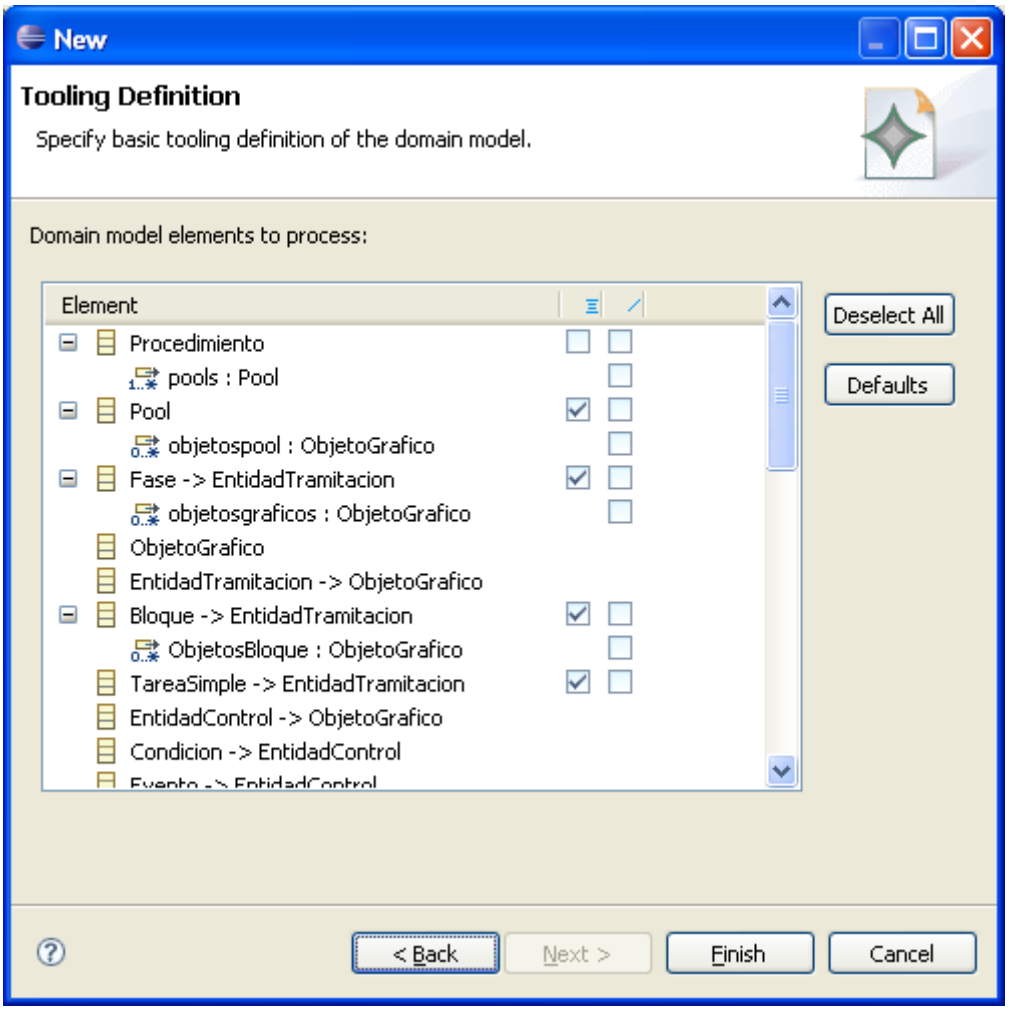


Figura 6.52. Generar .Gmftool. Ventana elementos.

En esta ocasión dejamos los valores que la herramienta ha generado por defecto, para más tarde modificar dichos valores para cumplir con las especificaciones solicitadas. De esta manera obtenemos el nuevo fichero abpmn0101.gmftool que se muestra a continuación.

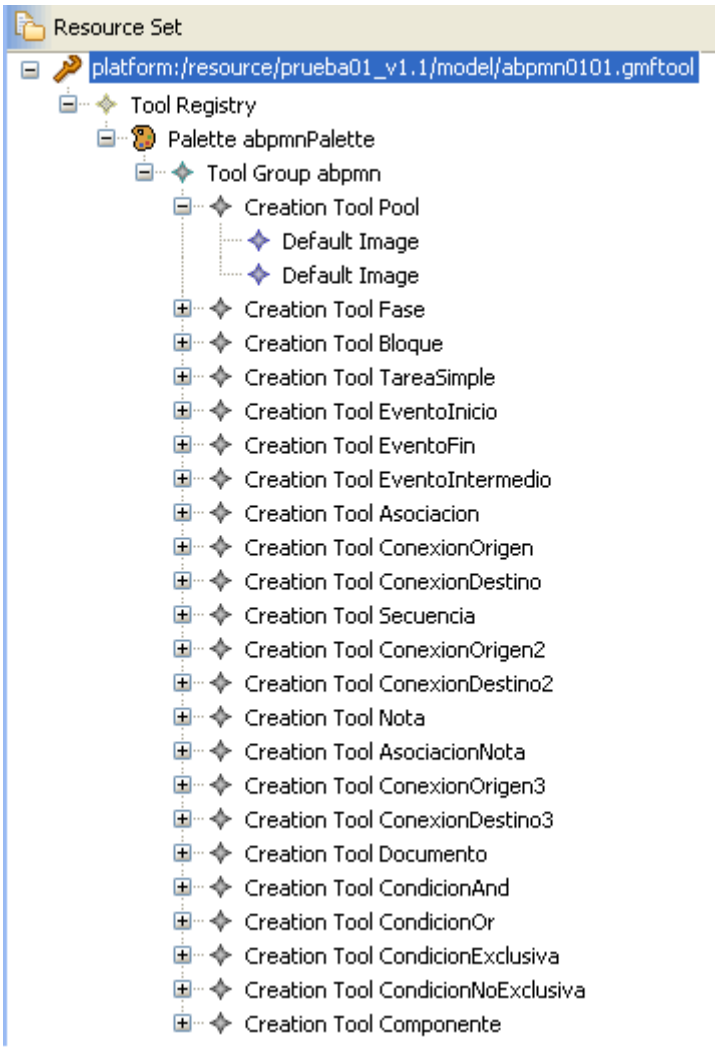


Figura 6.53. abpmn0101 .gmftool.

6.5.2.1. Modificaciones del Fichero .Gmftool

El fichero .gmftool que se obtiene por defecto, no cumple las especificaciones solicitadas de nuestra herramienta gráfica. Por lo que se ha tenido que transformar incorporando una serie de modificaciones y eliminando elementos que en nuestro caso no servían.

A continuación se explican las modificaciones que se han realizado al fichero para obtener uno nuevo acorde a las especificaciones solicitadas.

En primer lugar vamos a comenzar por dividir por grupos los elementos que van a formar parte de nuestra paleta, ya que como se puede observar en el .gmftool generado por defecto, se han agrupado todos los elementos en un solo grupo.

Vamos a dividir la paleta en 7 grupos (submenus), y dentro de cada uno de estos siete , crearemos los elementos que queremos incluir en cada uno de ellos.

En este momento, es necesario definir un elemento del tipo "Tool Group" para cada uno de los siete grupos que queremos incluir. Mediante la ventana Properties que ofrece la herramienta, asignaremos los atributos necesarios. Algunos de esos atributos son título, descripción, si el grupo se puede desplegar, etc.

En la siguiente imagen, se muestra cómo quedaría el fichero con los cambios realizados para generar los siete grupos (submenus) que formarán la paleta de la herramienta gráfica.

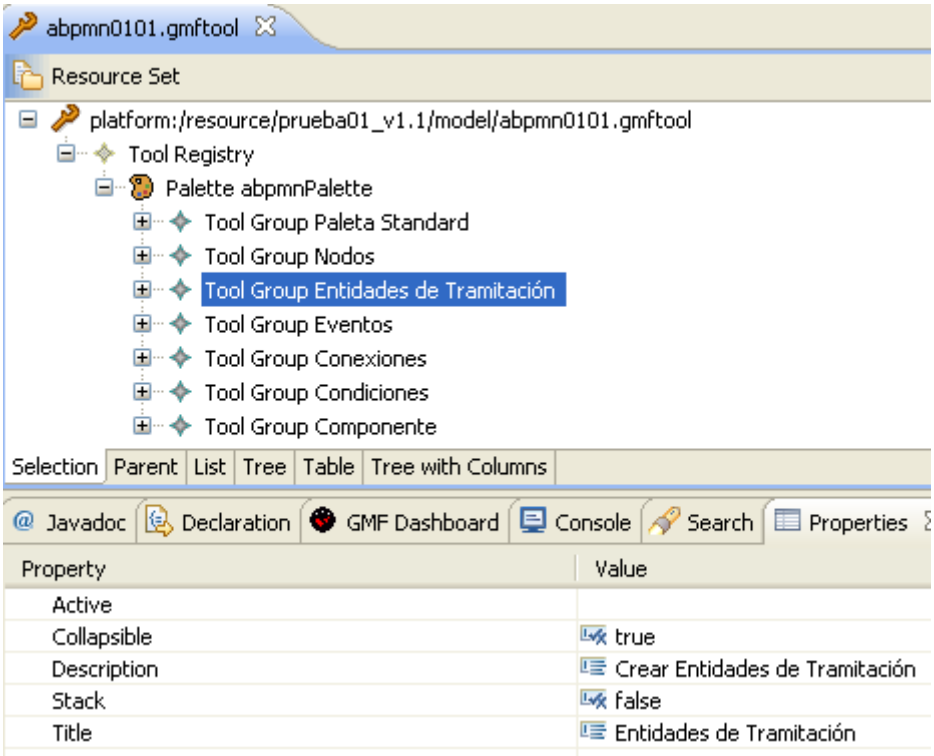


Figura 6.54. abpmn0101 .gmftool. Definir grupos (submenus) de la paleta.

Una vez que hemos creado los grupos, el siguiente paso consiste en crear cada uno de los elementos que queremos incluir en cada grupo. Para llevar a cabo dicho proceso, es necesario definir un elemento del tipo "Creation Tool" para cada uno de ellos. Mediante la ventana Properties que ofrece la herramienta, asignaremos los atributos necesarios, en este caso son título y descripción. En la siguiente imagen, se observa dicho proceso.

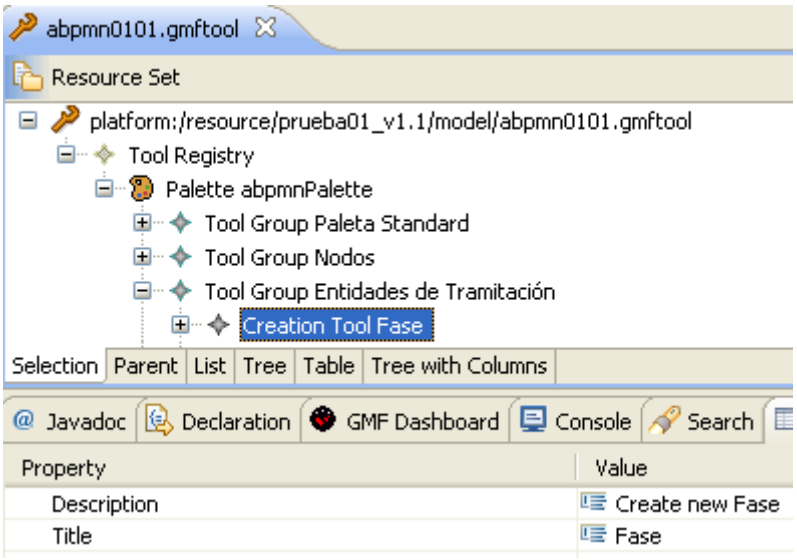


Figura 6.55. abpmn0101 .gmftool. Definir elementos de grupos de la paleta.

Una vez definidos todos los grupos y todos los elementos de cada grupo obtenemos el nuevo fichero abpmn0101.gmftool que se muestra a continuación.

Figura 6.56. abpmn0101 .gmftool definitivo.

6.5.3. Generar el Mapeo de los Elementos (.Gmfmap)

A continuación, se explica cómo se ha obtenido el fichero .gmfmap a partir del .gmfgraph ya generado.

Nos posicionamos sobre el fichero .gmfgraph y pulsamos el botón derecho del ratón, de esta manera obtenemos una ventana en la que pulsaremos New → Other… Y en la siguiente ventana seleccionaremos "GMFMap Model" dentro de la opción Graphical Modeling Framework → Default.

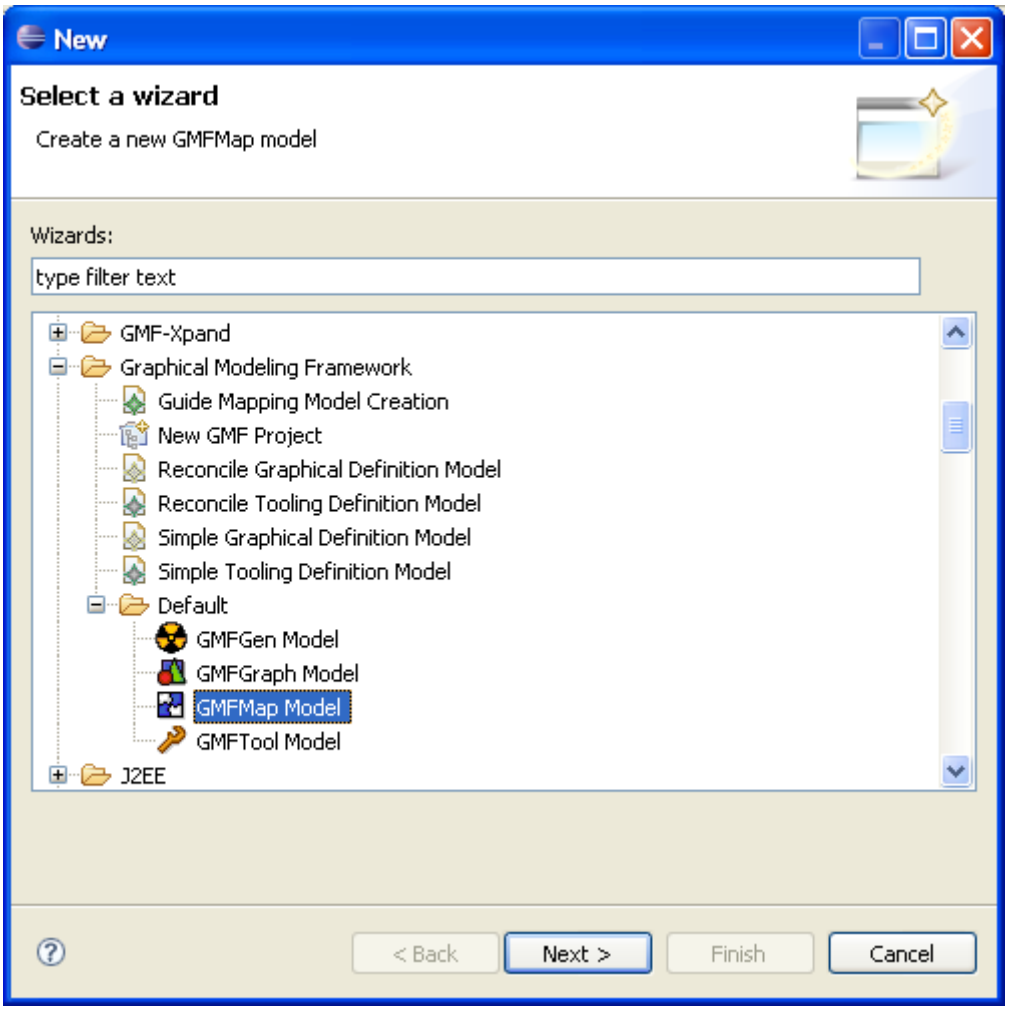


Figura 6.57. Generar .Gmfmap. Ventana New.

De esta manera se desplegará la siguiente ventana donde pondremos el nombre del nuevo fichero .gmfmap. En nuestro caso abpmn0101. gmfmap.

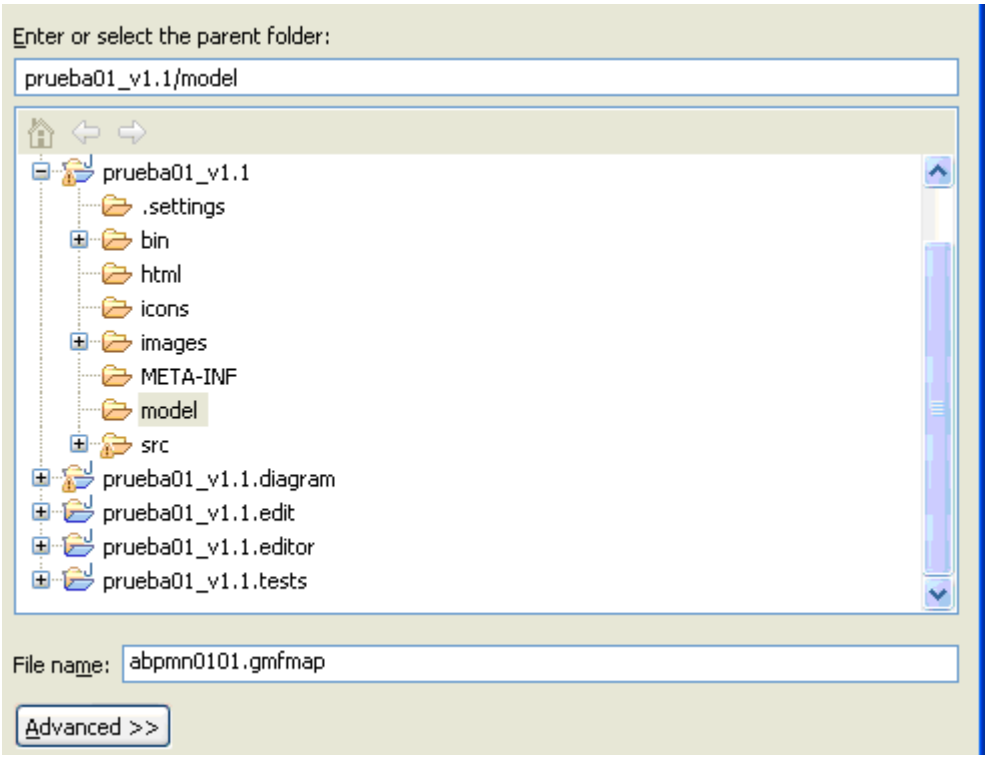


Figura 6.58. Generar .Gmfmap. Ventana Nombre del Fichero.

Una vez que hemos completado el nombre del fichero, pasaremos a una serie de ventanas, donde seleccionamos la ruta de los modelos de los que queremos obtener el nuevo fichero .gmfmap. En primer lugar seleccionaremos la ruta donde se encuentra el fichero abpmn0101.ecore generado anteriormente.

Observamos que al cargar el .ecore, aparecen en la pantalla todos los elementos que forman parte del modelo.

Dichos elementos se podrán dibujar posteriormente en la pizarra de la herramienta gráfica de la misma forma que se hayan definido en el fichero, especificando los elementos que pueden contener a otros, etc. Es decir, se realiza el mapeo de los elementos de la pizarra de la aplicación.

Debemos seleccionar, cual será el elemento root del modelo .gmfmap, en nuestro caso, seleccionamos Procedimiento.

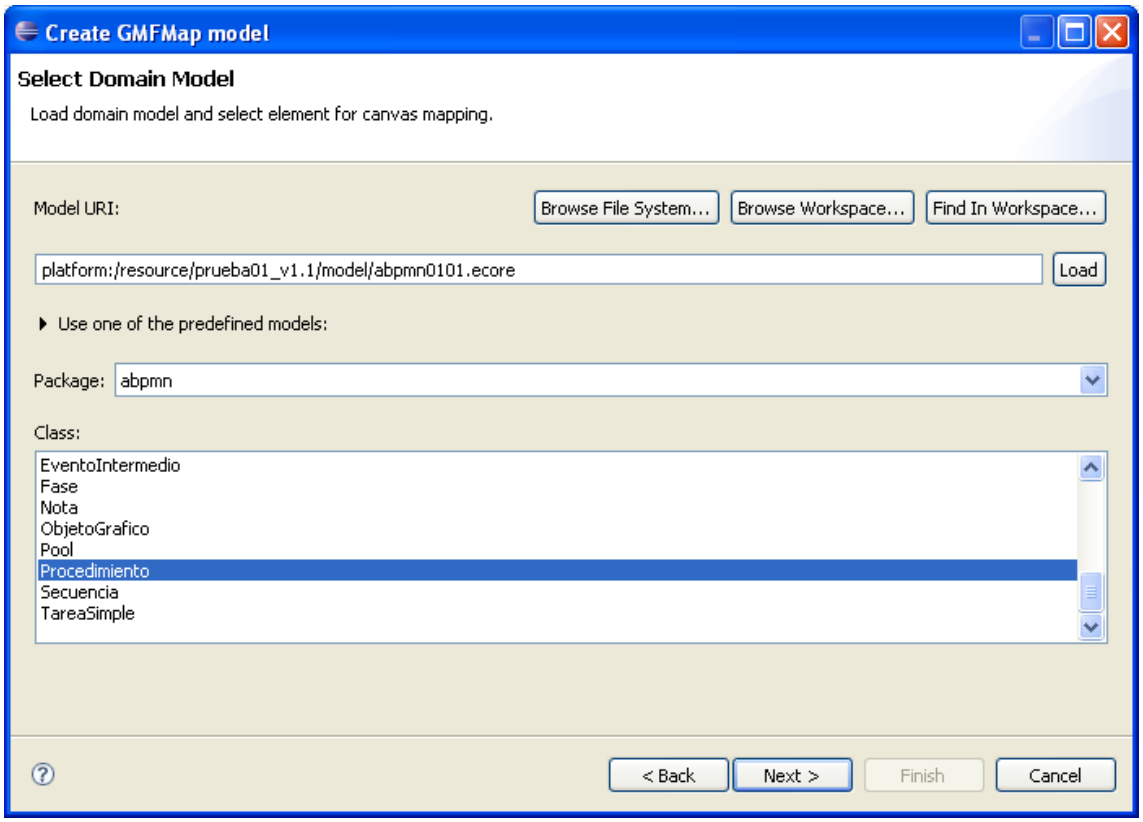


Figura 6.59. Generar .Gmfmap. Ventana Seleccionar .ecore y Root.

A continuación, pasamos a otra ventana en la que seleccionaremos la ruta donde se encuentra el fichero abpmn0101.gmftool generado anteriormente.

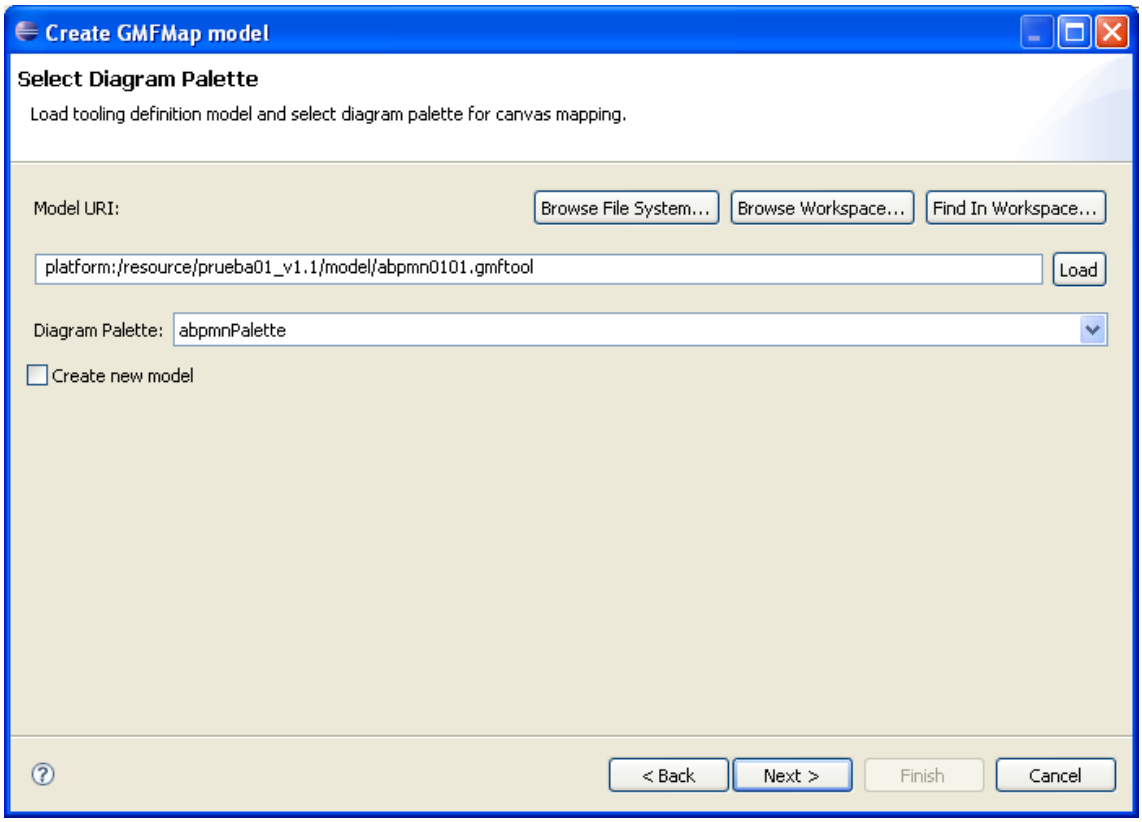


Figura 6.60. Generar .Gmfmap. Ventana Seleccionar .gmftool.

En la siguiente pantalla seleccionaremos la ruta del fichero abpmn0101.gmfgraph.

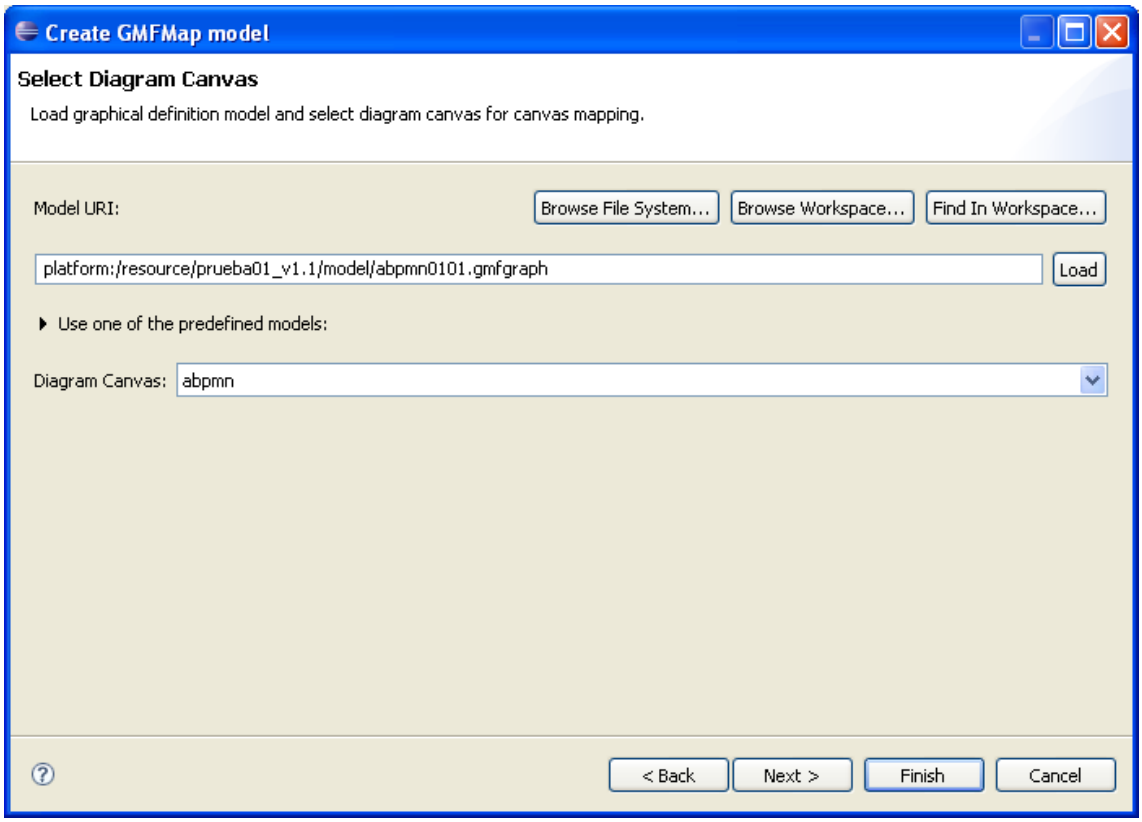


Figura 6.61. Generar .Gmfmap. Ventana Seleccionar .gmfgraph.

A continuación, pasamos a la siguiente pantalla donde aparecen todos los elementos del modelo y que se podrán dibujar o no en la paleta de la herramienta gráfica, para ello en esta pantalla se pueden seleccionar como nodos o enlaces los elementos que se deseen.

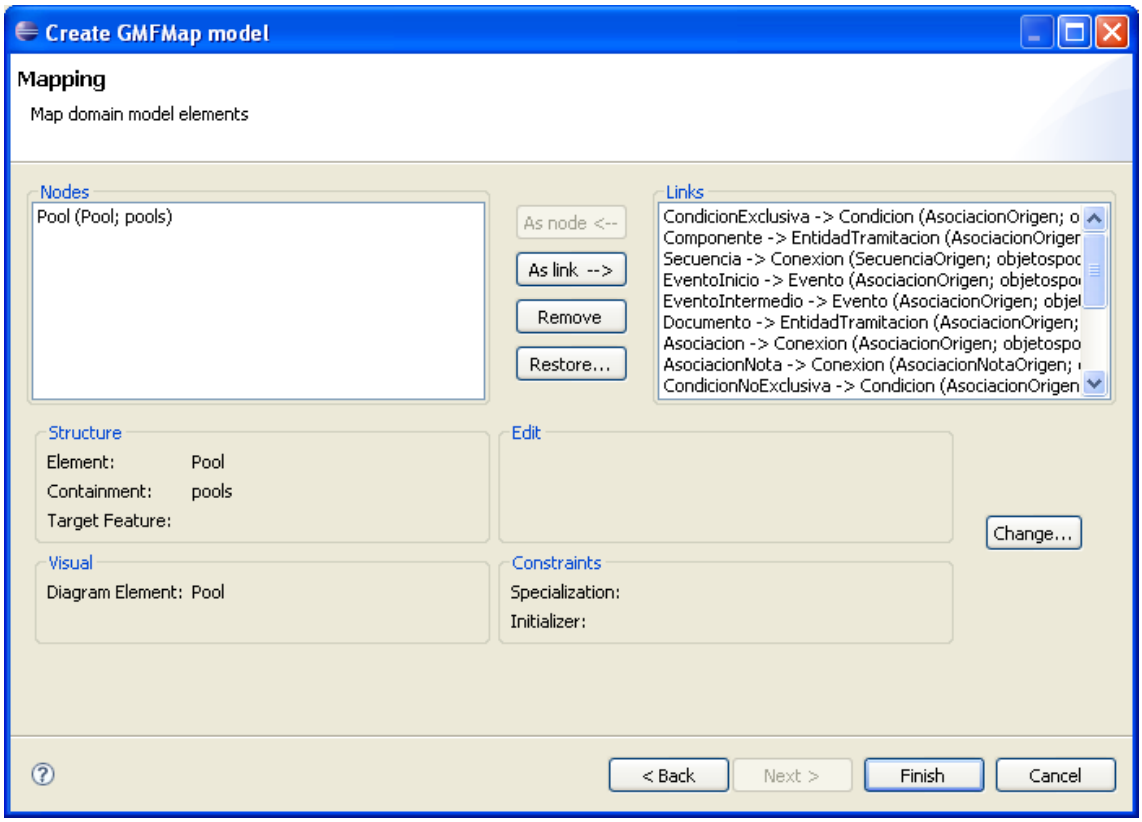


Figura 6.62. Generar .Gmfmap. Ventana elementos.

En este caso dejamos los valores que la herramienta ha generado por defecto, para más tarde modificar dichos valores y cumplir con las especificaciones solicitadas. De esta manera obtenemos el nuevo fichero abpmn0101.gmfmap que se muestra a continuación.

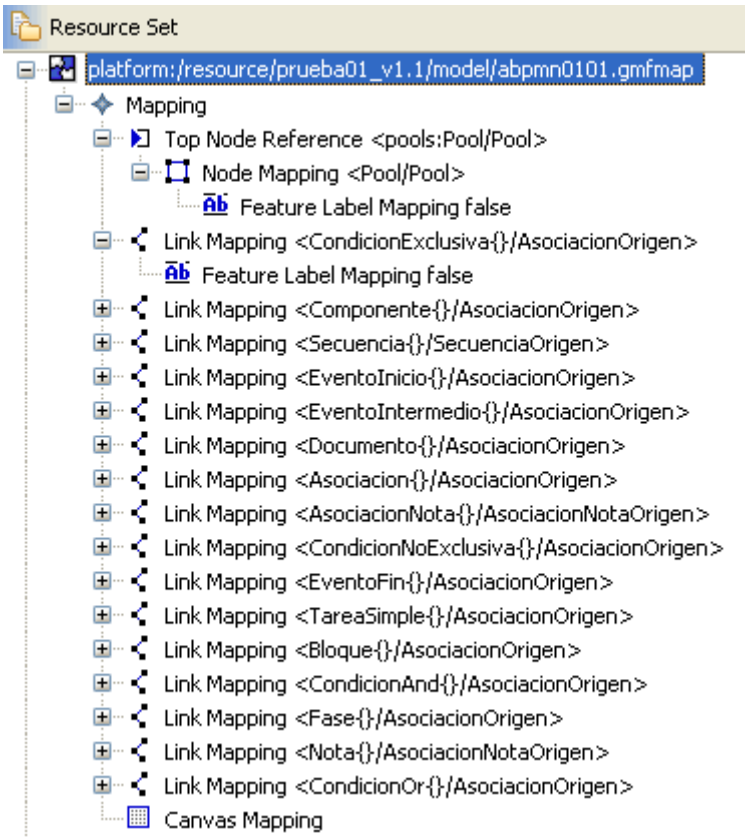


Figura 6.63. abpmn0101 .gmfmap.

6.5.3.1. Modificaciones del Fichero .Gmfmap

El fichero .gmfmap que se obtiene por defecto, no cumple las especificaciones solicitadas de nuestra herramienta gráfica. Por lo que se ha tenido que transformar incorporando una serie de modificaciones y eliminando elementos que en nuestro caso no servían.

En primer lugar vamos a comentar el objetivo de este fichero. Se trata de generar nodos y enlaces, los nodos serán todos los elementos que se quieran dibujar en la pizarra, y los enlaces las conexiones para unir dichos elementos. Por otro lado también se refleja en este fichero, el mapeo de dichos elementos. Por ejemplo, queremos que el primer elemento que se pueda dibujar en la pizarra sea POOL, y que dentro de éste, se dibujen el resto, etc.

A continuación se explican las modificaciones que se han realizado al fichero para obtener un nuevo fichero acorde a las especificaciones solicitadas.

Tenemos que generar todos los elementos que queremos dibujar en la pizarra, como **nodos**, y todos los enlaces entre elementos como **link**.

Las figuras Pool, fase, tarea simple, bloque, evento inicio, evento fin, evento intermedio, texto anotación, documento, condición and, condición or, condición exclusiva, condición no exclusiva y componente se crearían de la misma manera.

Partimos del elemento raíz POOL, que es el primer elemento que vamos a dibujar en la pizarra. En este caso ya lo tenemos generado por defecto en el fichero abpmn0101.gmfmap.

Ahora queremos generar el nodo Fase que será dibujado dentro de Pool. Primero necesitamos definir un elemento del tipo "Child Reference" y mediante la ventana Properties que ofrece la herramienta, le asignaremos los atributos necesarios. En este caso sólo le asignamos al atributo Containment Feature el valor objetospool, que habíamos definido en el modelo .ecore, dicho valor refleja que se van a poder dibujar de 0 a n elementos dentro del objeto pool. El atributo Child tomará el nombre del elemento que queremos dibujar, y dicho elemento será definido dentro de Child Reference como un nodo.

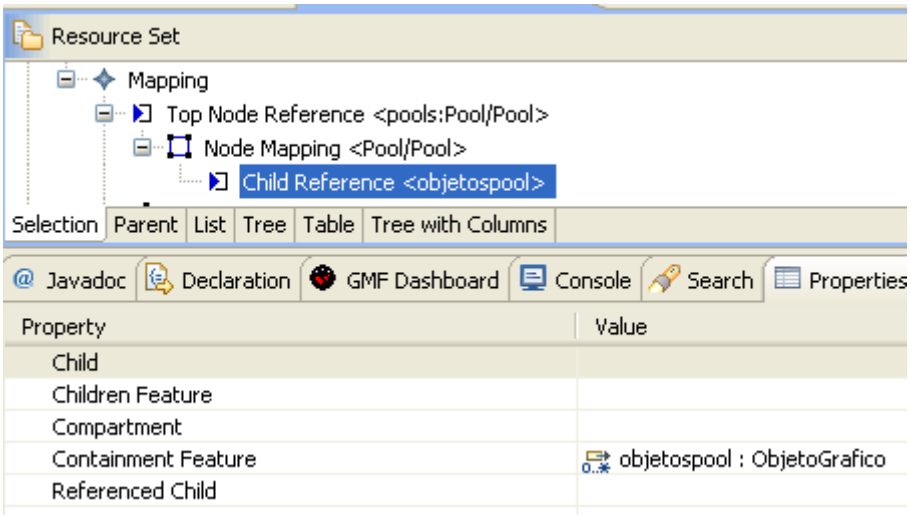


Figura 6.64. abpmn0101 .gmfmap. Definir Child Referente objeto pool.

Para ello es necesario definir un elemento del tipo "Node Mapping", mediante la ventana Properties que ofrece la herramienta y le asignaremos los atributos necesarios.

Dichos atributos son el elemento al que hace referencia en el modelo .ecore, la definición de dicho elemento en el modelo .gmfgraph y en el modelo .gmftool.

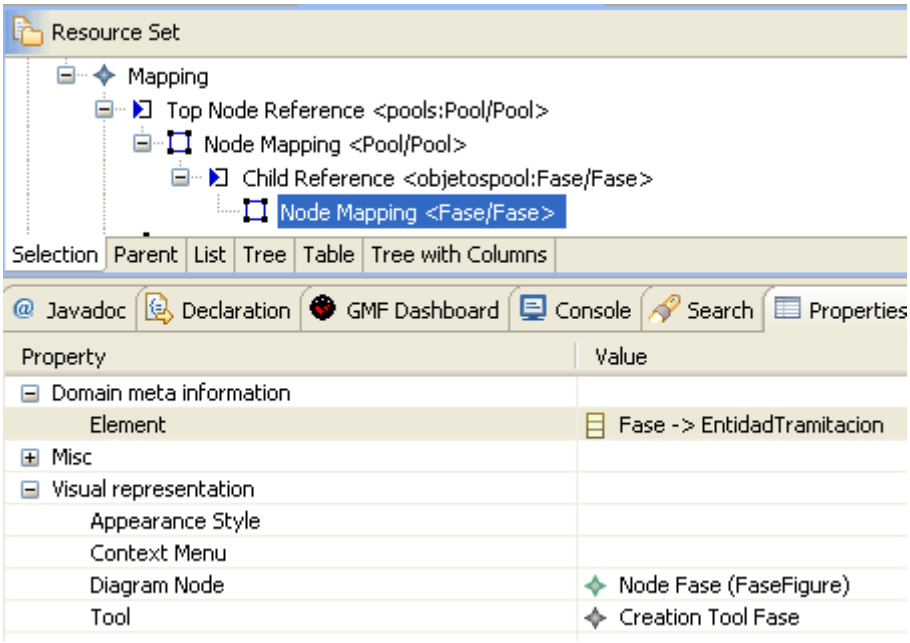


Figura 6.65. abpmn0101 .gmfmap. Definir Nodo Fase.

También queremos incluir una etiqueta al elemento Fase, para que se pueda escribir el nombre de la fase. Para ello necesitamos definir un elemento del tipo "Feature Label Mapping" dentro del nodo generado. Dicho elemento mediante sus propiedades, a parte de los atributos que pone por defecto le asignaremos, le asignaremos el valor "Diagram Label FaseNombre" en Diagram label, y "CodigoFase" en Features.

Para asignar el valor del atributo Features, se despliega una ventana con todos los atributos de la clase Fase, mediante la que se puede seleccionar el atributo que se quiere utilizar y poner el nombre de la fase.

A continuación, se muestra dicha ventana.

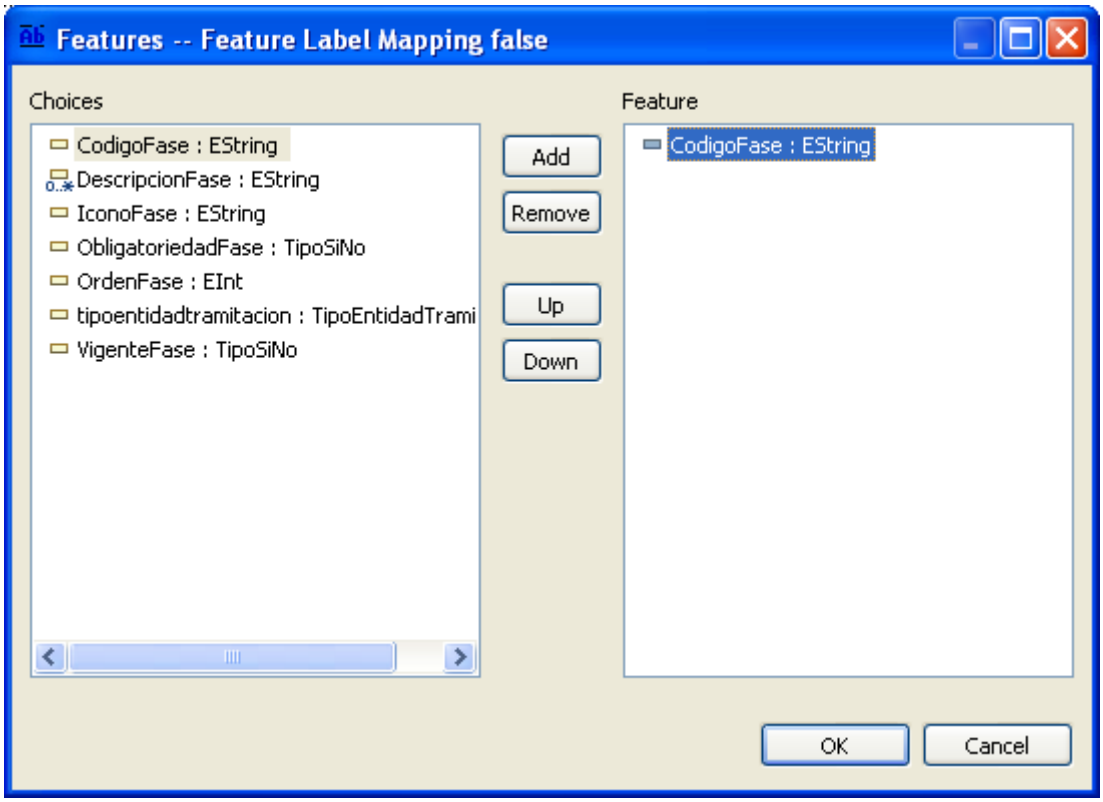


Figura 6.66. abpmn0101 .gmfmap. Seleccionar atributo de etiqueta Nodo Fase.

Una vez definidos los atributos necesarios, el elemento Feature Label Mapping quedaría de la siguiente manera.

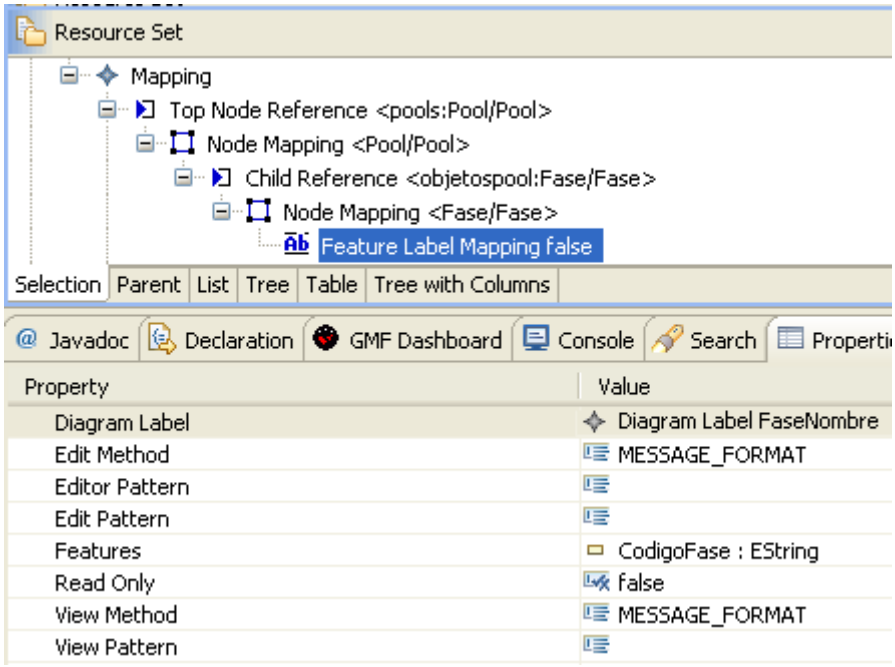


Figura 6.67. abpmn0101 .gmfmap. Definir etiqueta Nodo Fase.

El siguiente elemento a crear sería TextoAnotación, puesto que se puede dibujar también dentro de Pool, el proceso sería el mismo que el explicado para generar el elemento Fase. El fichero una vez incluido dicho elemento quedaría de la siguiente manera:

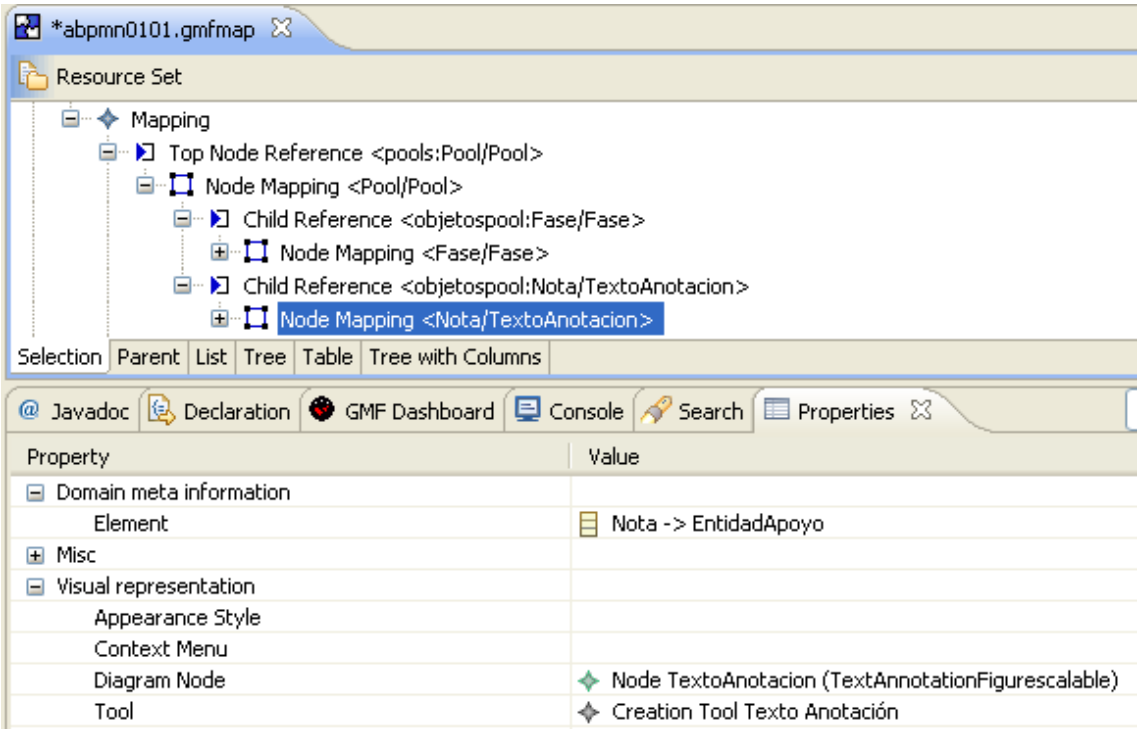


Figura 6.68. abpmn0101 .gmfmap. Pool, Fase y TextoAnotación.

Seguimos generando el fichero abpmn0101.gmfmap. Lo siguiente sería crear los elementos que se pueden dibujar dentro de Fase. El proceso a seguir sería el mismo que se ha explicado hasta ahora, se genera un elemento del tipo "Child Reference" y dentro de éste para cada elemento un elemento del tipo "Node Mapping", en este caso para los elementos tarea simple, bloque, evento inicio, evento fin, evento intermedio, texto anotación, documento, condición and, condición or, condición exclusiva, condición no exclusiva y componente.

Una vez incluidos estos elementos, el fichero quedaría como se muestra en la siguiente imagen.

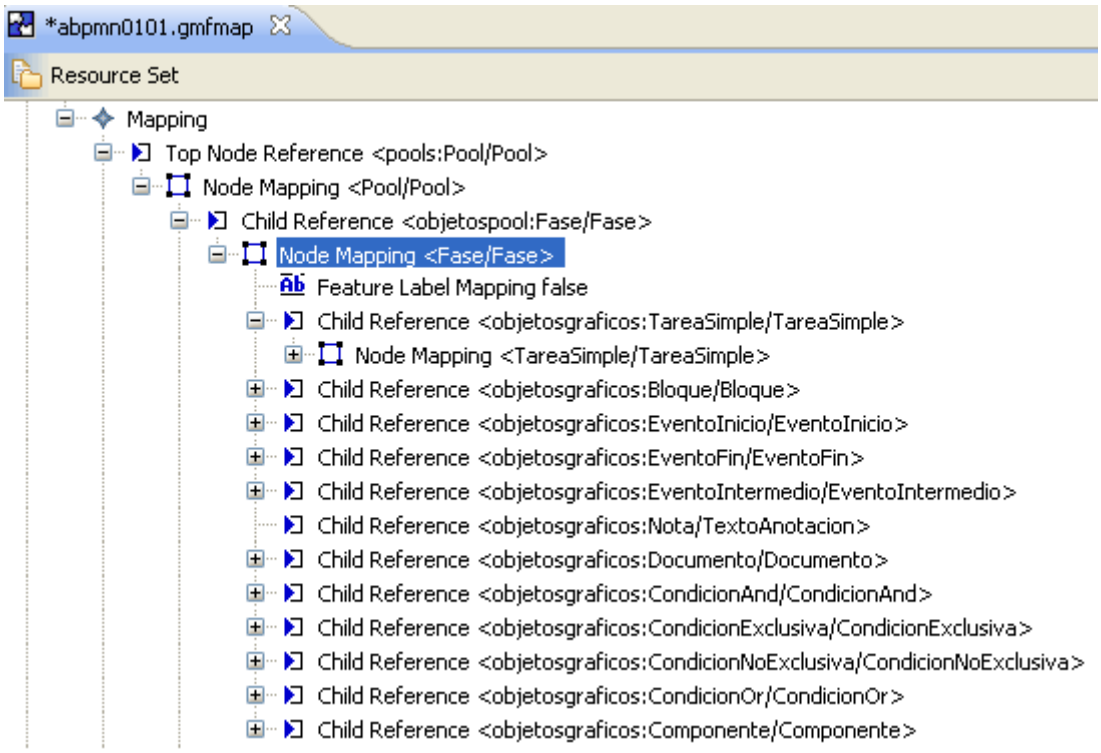


Figura 6.69. abpmn0101 .gmfmap. Nodos de Fase.

Puesto que la figura bloque debe poder desplegarse y plegarse cuando tiene elementos dentro, en el fichero .gmfgraph se definió un elemento tipo "Compartment". Ahora es necesario definir un elemento de tipo "Compartment Mapping" que haga referencia a dicho "Compartment".

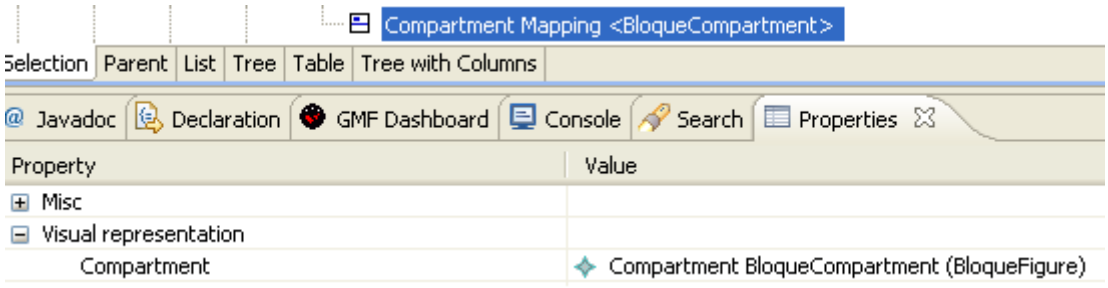


Figura 6.70. abpmn0101 .gmfmap. Compartment Bloque.

A parte, hay que crear los elementos que se pueden dibujar dentro de Bloque. El bloque puede contener los siguientes elementos: tarea simple, bloque, evento inicio, evento fin, evento intermedio, texto anotación, documento, condición and, condición or, condición exclusiva, condición no exclusiva y componente. El proceso a seguir sería el

mismo que se ha explicado hasta ahora, pero como ya tenemos generados los nodos para todos los elementos, sólo haría falta definir un elemento del tipo "Child Reference" que haga referencia a cada uno, y rellenar para cada elemento los atributos que se observan en la siguiente figura.

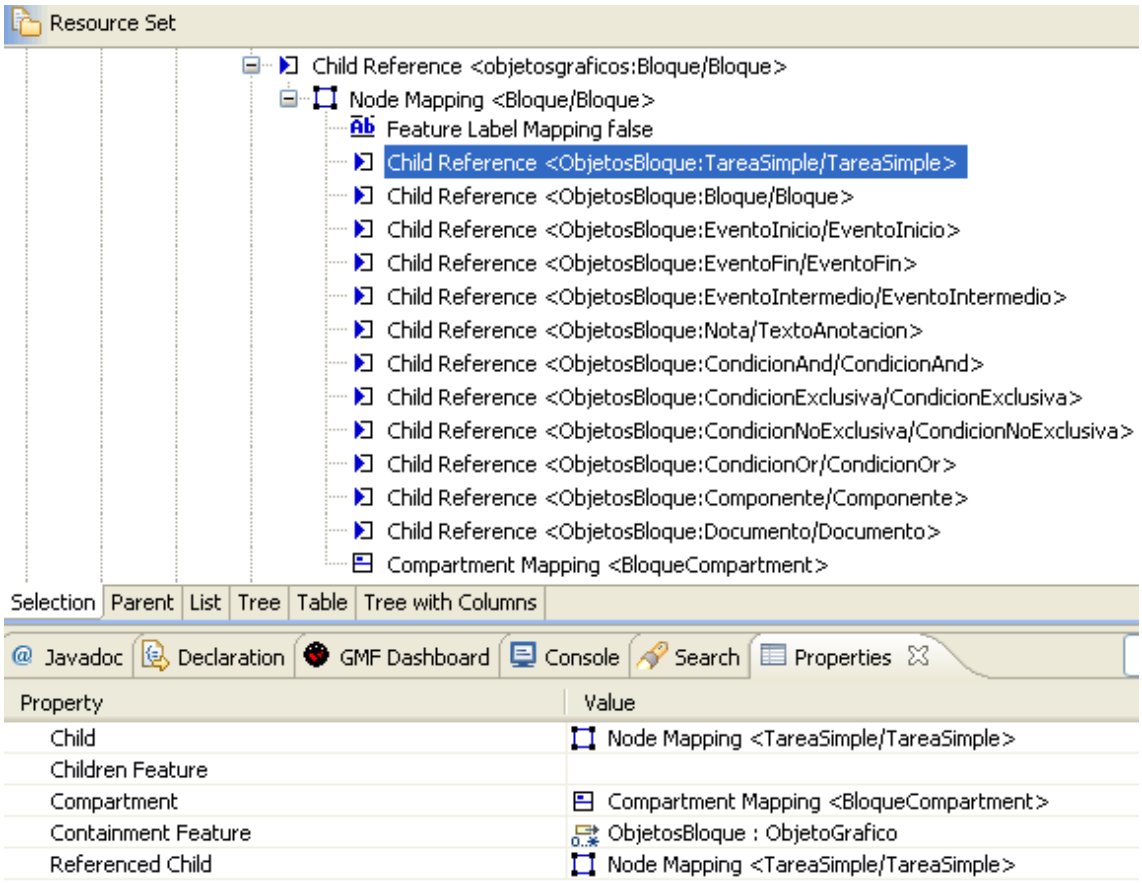


Figura 6.71. abpmn0101 .gmfmap. Child Reference Bloque.

Una vez incluidos estos elementos, el fichero quedaría como se muestra en la siguiente imagen.

Figura 6.72. abpmn0101 .gmfmap. Nodos.

Por último, quedaría definir los elementos de conexión que sirven para relacionar los elementos de la paleta. Los elementos de conexión serían: secuencia, asociación y asociación nota.

Puesto que los tres elementos se definen igual, se va a explicar el proceso para crear el elemento Secuencia que nos servirá de ejemplo para el resto.

Es necesario definir un elemento del tipo “Link Mapping” y rellenar los atributos como se observa en la siguiente figura.

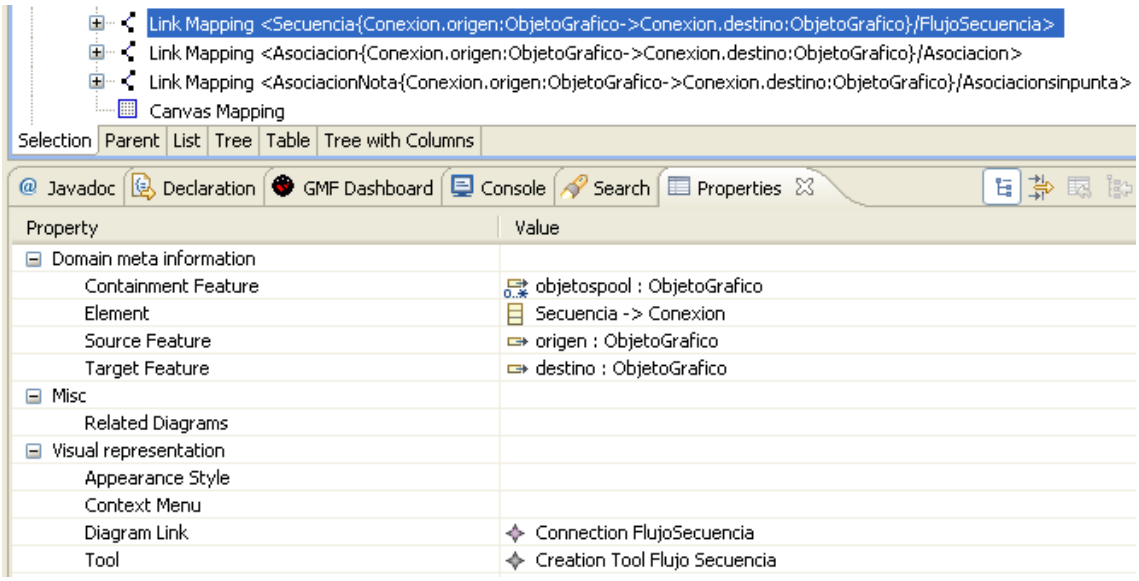


Figura 6.73. abpmn0101 .gmfmap. Link Mapping Secuencia.

Una vez generados los enlaces, ya tendríamos el fichero abpmn0101.gmfmap completo, dicho fichero se muestra en la siguiente imagen.

Figura 6.74. abpmn0101 .gmfmap definitivo.

6.5.3.2. Añadir Restricciones OCL

Mediante el lenguaje notacional OCL vamos a escribir las restricciones sobre nuestro modelo de objetos. Estas restricciones son particularmente útiles, en la medida en que nos permiten crear un amplio conjunto de reglas que rigen el aspecto de un objeto individual.

Cada expresión OCL se escribe en el contexto de una instancia de un tipo específico. En una expresión OCL, la palabra reservada self se usa para referirse a la instancia contextual.

Vamos a utilizar la operación "OclIsKingOf (t : OclType) : Bolean" . El resultado de esta operación es verdad si el tipo de self y t son el mismo. Además determina si t es el tipo directo de uno de los supertipos de un objeto.

A continuación, se exponen las restricciones que se han implementado para nuestro modelo en el fichero abpmn0101.gmfmap.

En primer lugar, vamos a asignar a cada elemento el tipo al que pertenece según la definición del modelo .ecore. Excepto para el nodo Pool, para el resto de los nodos generados en el fichero abpmn0101.gmfmap, se crea dentro de cada Node Mapping, un elemento del tipo "Feature Seq Initializer" para referirnos a la instancia del elemento del .ecore, por ejemplo, Fase→Entidad Tramitación.

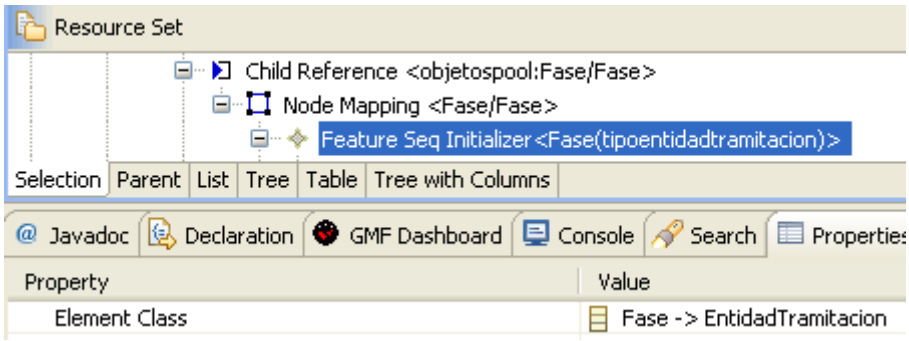


Figura 6.75. abpmn0101 .gmfmap. Restricción OCL tipo.

Y dentro de Feature Seq Initializer, se crea otro elemento del tipo "Feature Value Spec" para especificar el nombre del tipo al que pertenece.

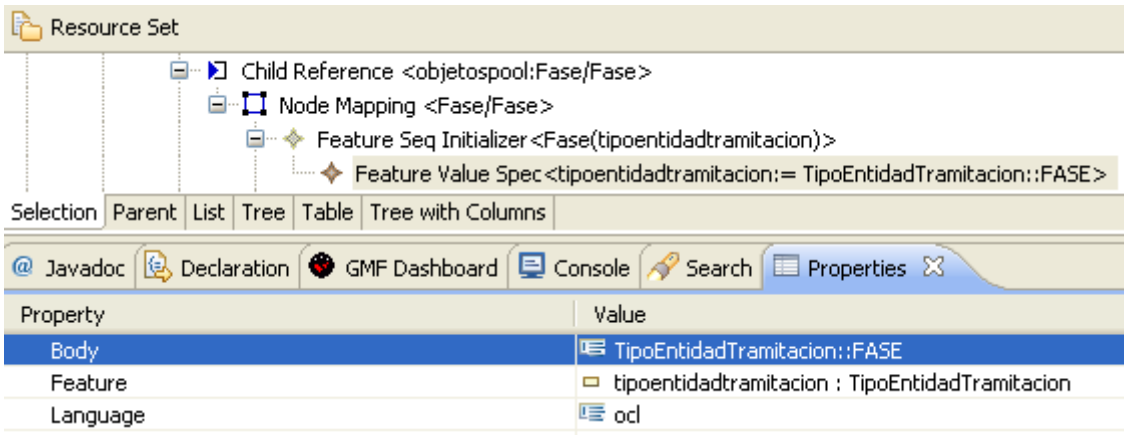


Figura 6.76. abpmn0101 .gmfmap. Restricción OCL nombre tipo.

En el atributo Body se especifica el código necesario para asignar el nombre del elemento al tipo que pertenece, en este caso, TipoEntidadTramitacion::FASE.

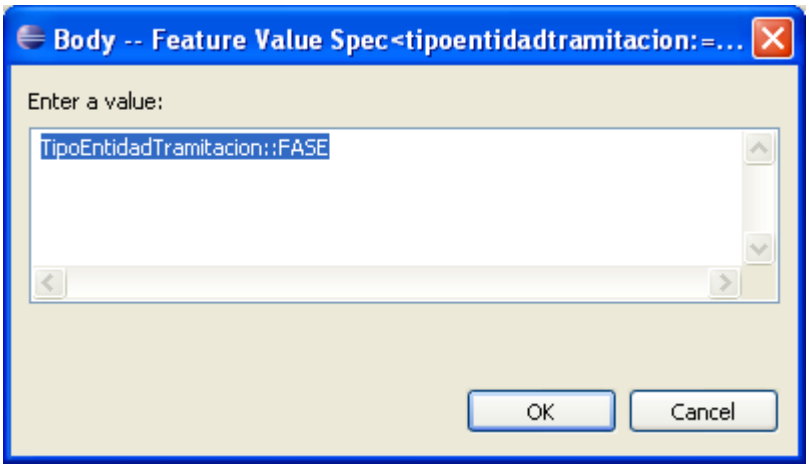


Figura 6.77. abpmn0101 .gmfmap. Restricción OCL código nombre tipo.

En el caso de los elementos de tipo link, tipo condición y tipo evento, sería necesario definir otro elemento del tipo "Feature Value Spec", puesto que están clasificados además de por tipo, por un subtipo dentro del tipo de entidad. A continuación se muestra un ejemplo de ello.

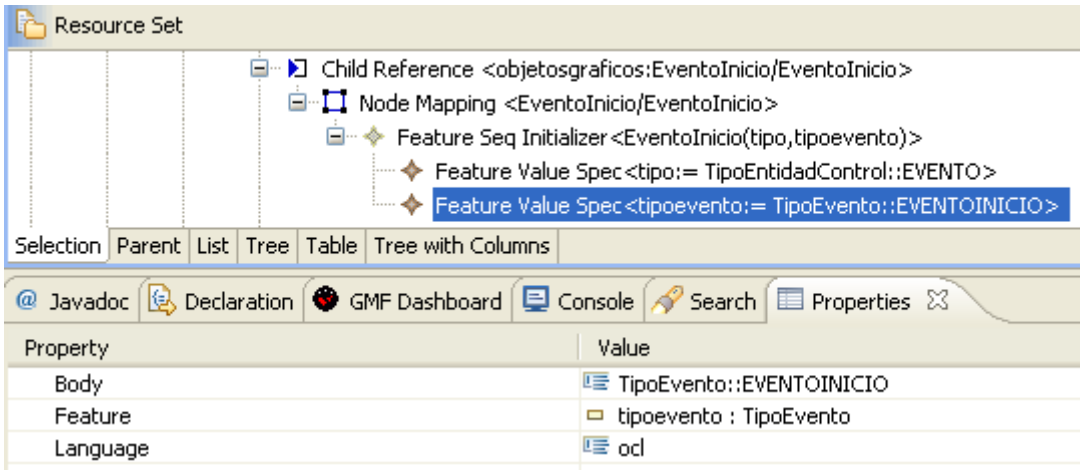


Figura 6.78. abpmn0101 .gmfmap. Restricción OCL nombre sub tipo.

Por último quedaría añadir las restricciones para los elementos de tipo link (conexión), es decir, qué elementos podemos conectar con cada tipo de enlace. Las restricciones serían las siguientes:

- **Link Secuencia:** No puede conectar elementos de tipo nota, ni documento: (not self.oclIsKindOf(Nota)) and (not self.oclIsKindOf(Documento)). Sólo puede conectar elementos de tipo bloque, tarea simple y entidades de control (eventos, condiciones, etc): self.oclIsKindOf(Bloque) or self.oclIsKindOf(TareaSimple) or self.oclIsKindOf(EntidadControl).

- **Link Asociación:** Sólo puede conectar elementos de tipo documento: self.oclIsKindOf(Documento).

- **Link AsociaciónNota:** Sólo puede conectar elementos de tipo nota: self.oclIsKindOf(Nota).

Para ello, tenemos que definir dentro del elemento “Link Mapping” correspondiente al enlace un elemento del tipo “Link Constraints”. Y dentro de éste un elemento del tipo “Constraint”, donde a través de sus propiedades podremos incluir las restricciones mediante el código necesario en el atributo Body.

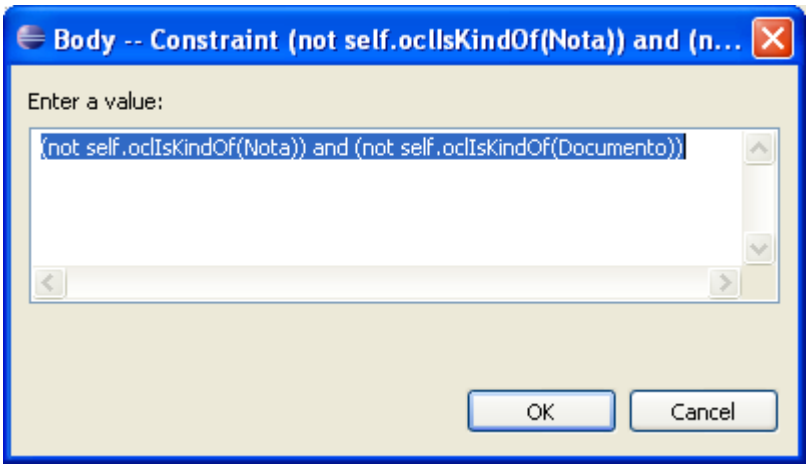


Figura 6.79. abpmn0101 .gmfmap. Restricción OCL código enlace secuencia.

A continuación, se muestra como quedaría el fichero abpmn0101.gmfmap con las restricciones de los enlaces incluidas.

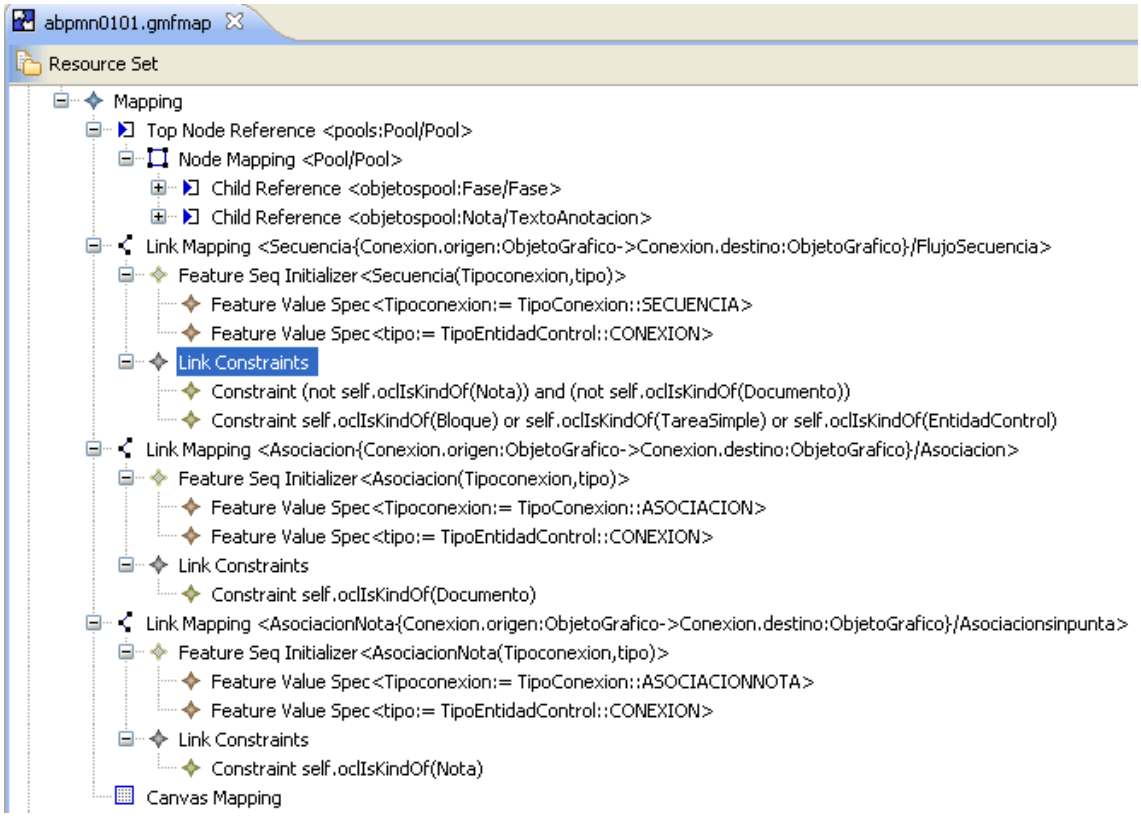


Figura 6.80. abpmn0101 .gmfmap. Restricciones OCL.

6.6. GENERACIÓN DE LA HERRAMIENTA GRÁFICA

Una vez que tenemos todos los ficheros necesarios obtenidos a partir del meta-modelo abpmn0101.ecore, el siguiente paso es la obtención del resto de ficheros y carpetas necesarias para poder generar la herramienta gráfica final.

6.6.1. Generar el Fichero .Gmfgen

A continuación, se explica cómo se ha obtenido el fichero .gmfgen a partir del .gmfmap ya generado.

Nos posicionamos sobre el fichero .gmfmap y pulsamos el botón derecho del ratón, de esta manera obtenemos una ventana en la que pulsaremos "Create generator model…". De esta forma se desplegará la siguiente ventana donde pondremos el nombre del nuevo fichero .gmfgen. En nuestro caso abpmn0101. gmfgen.

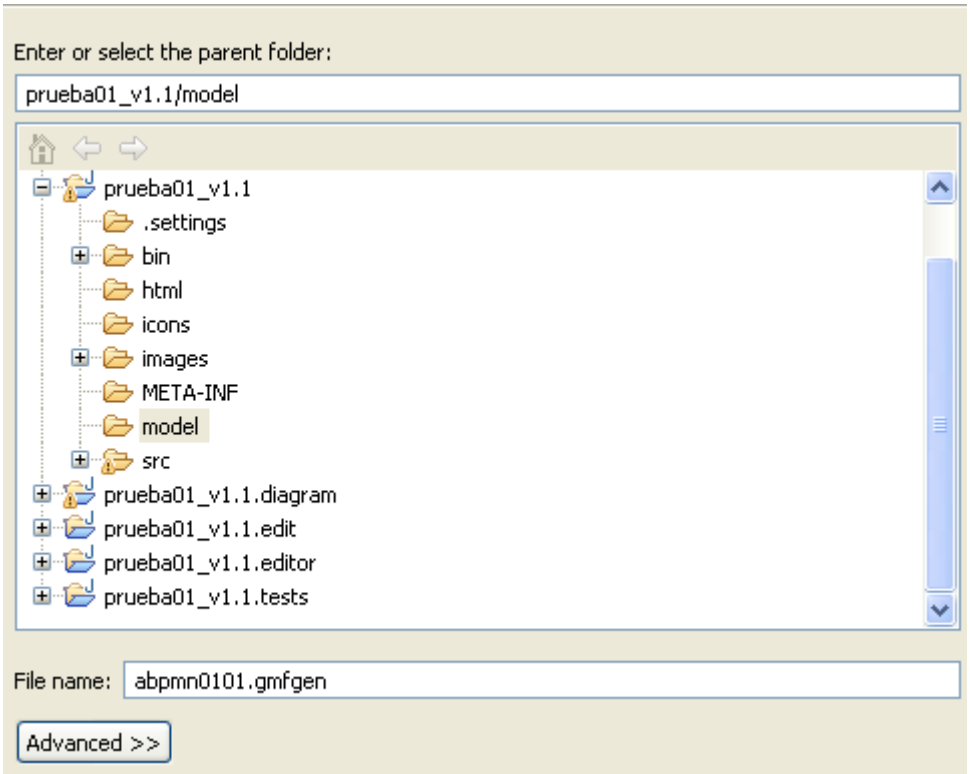


Figura 6.81. Generar .Gmfgen. Ventana Nombre del Fichero.

A continuación, pasamos a otra ventana donde seleccionaremos la ruta en la que se encuentra el fichero abpmn0101.gmfmap generado anteriormente.

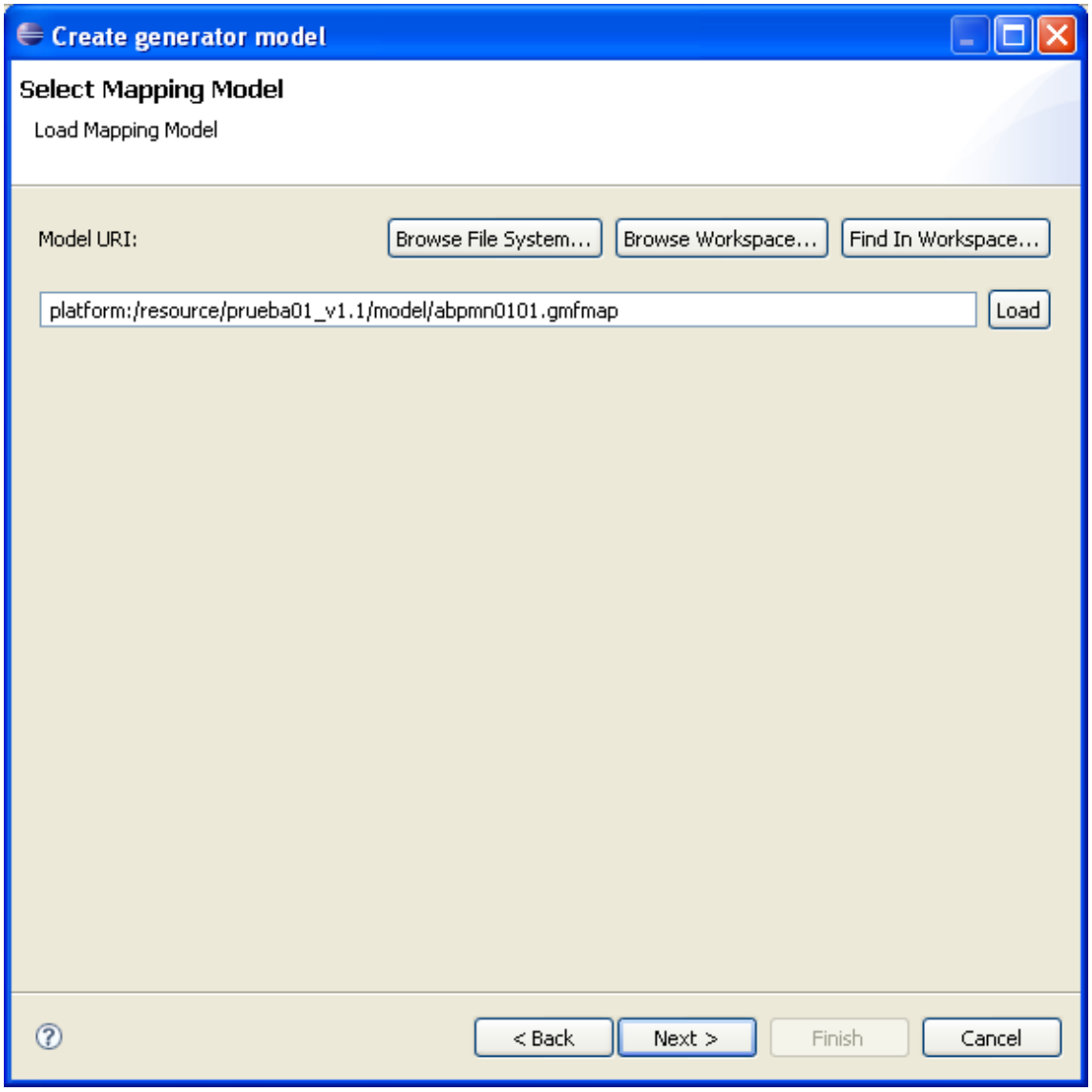


Figura 6.82. Generar .Gmfgen. Ventana Seleccionar .gmfmap.

A continuación, pasamos a otra ventana donde seleccionaremos la ruta en la que se encuentra el fichero abpmn0101.genmodel generado anteriormente.

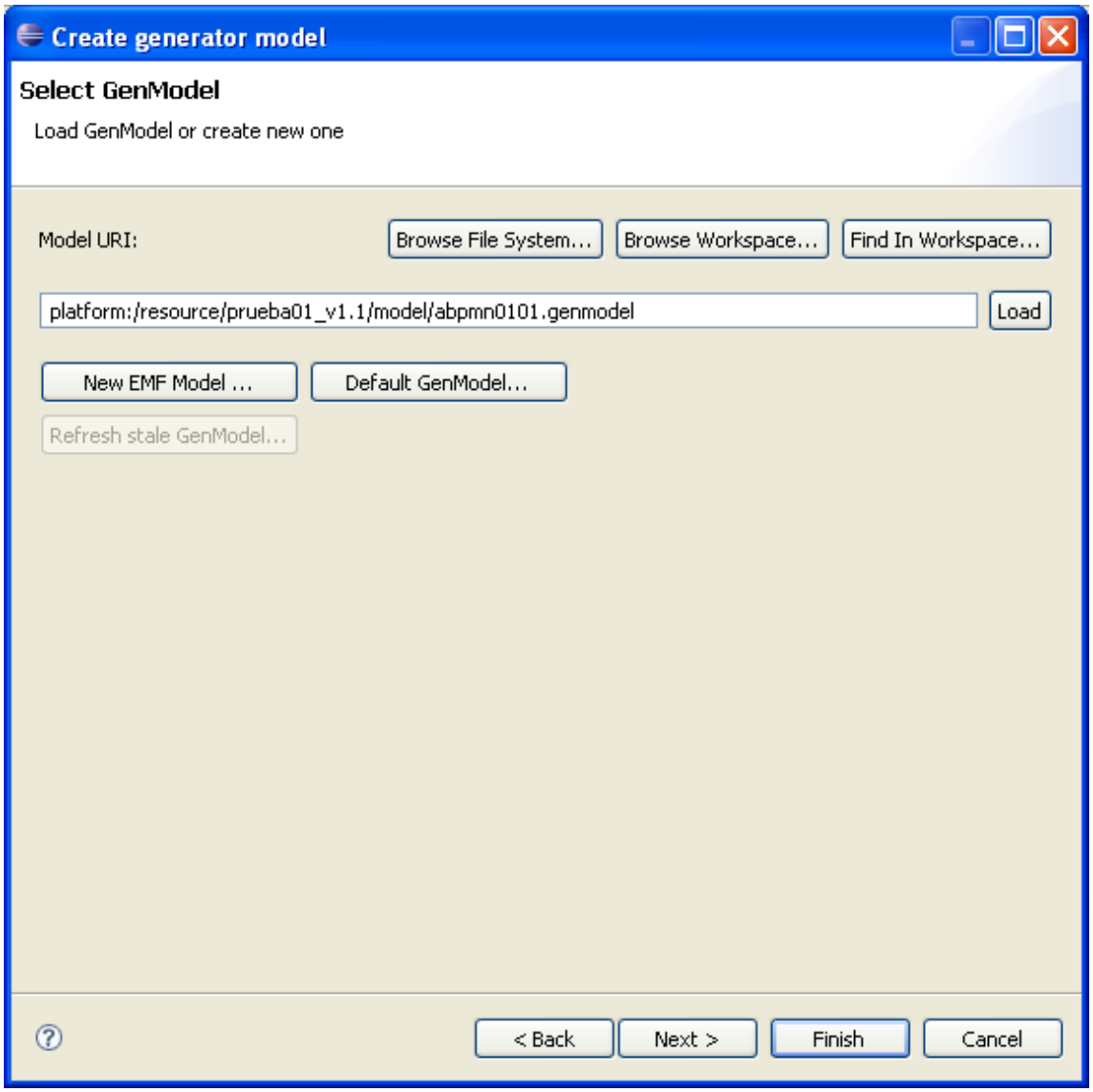


Figura 6.83. Generar .Gmfgen. Ventana Seleccionar .genmodel.

A continuación, pasamos a otra ventana donde seleccionaremos las especificaciones necesarias para la transformación del fichero .gmfgen.

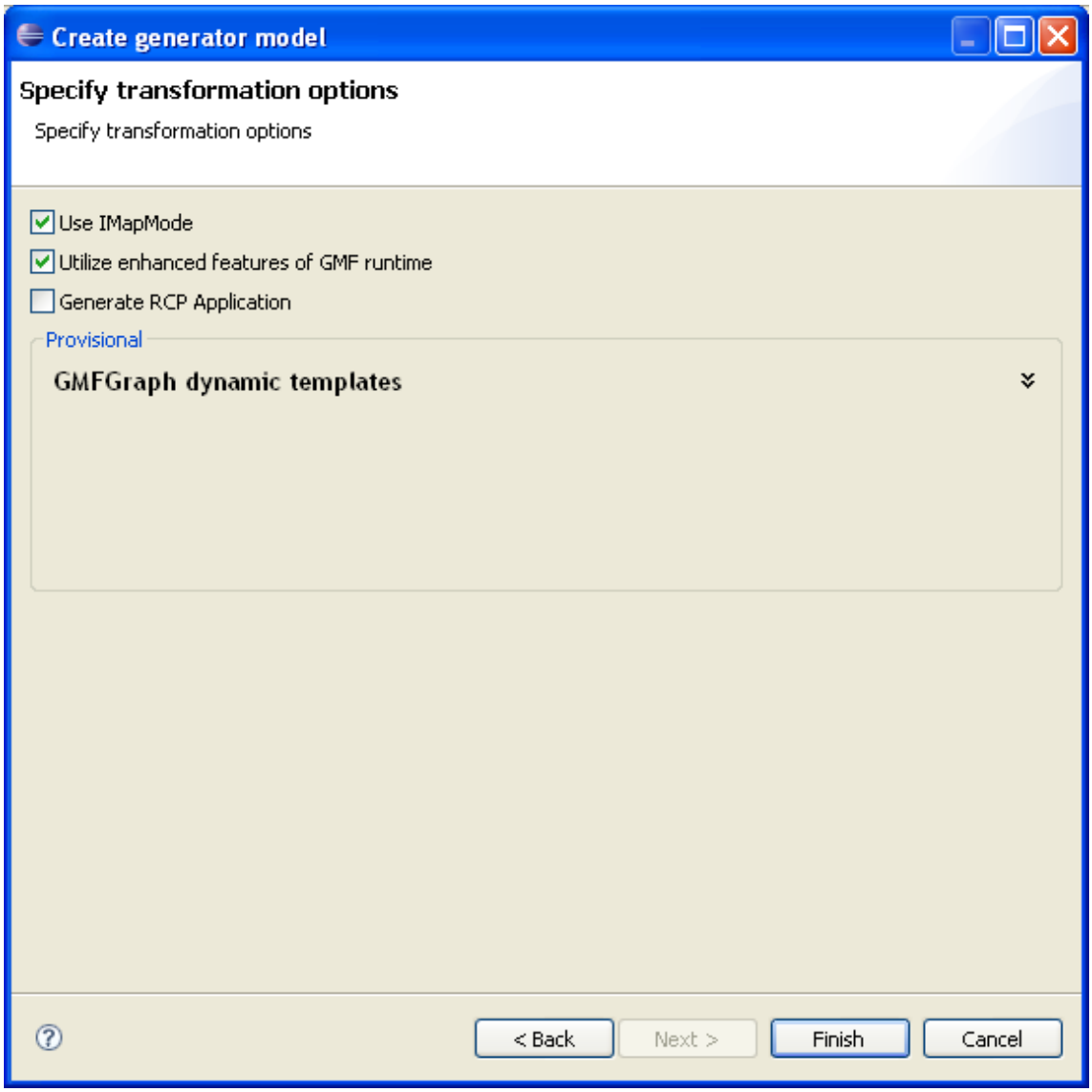


Figura 6.84. Generar .Gmfgen. Ventana Seleccionar Transformaciones.

Dejamos las opciones que aparecen por defecto y obtenemos el fichero abpmn0101.gmfgen que se muestra a continuación.

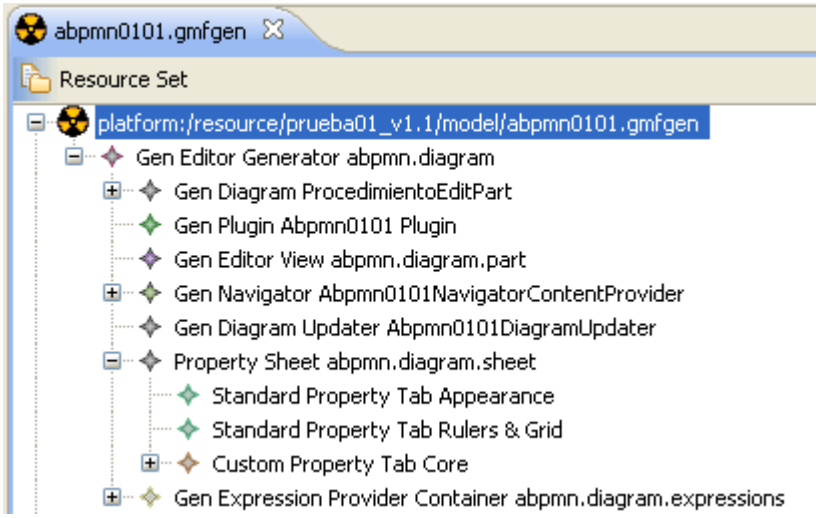


Figura 6.85. abpmn0101 .Gmfgen.

6.6.2. Generar la Carpeta .Diagram

Una vez que hemos generado el fichero .gmfgen, ya tenemos todos los ficheros y carpetas necesarias para poder generar la carpeta .Diagram.

El proceso a seguir sería el siguiente, nos posicionamos sobre el fichero .gmfgen y pulsamos el botón derecho del ratón, de esta manera obtenemos una ventana en la que pulsaremos "Generate diagram code" y directamente obtendremos la carpeta prueba01_v1.1.diagram.

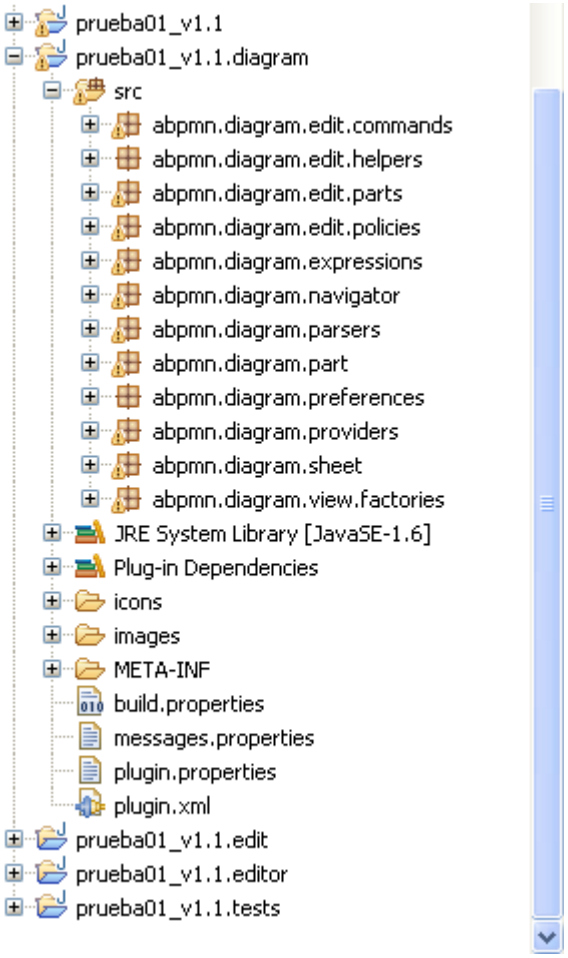


Figura 6.86. Carpeta prueba01_v1.1.diagram.

6.6.2.1. Generar Imágenes .Svg

Para visualizar correctamente en la paleta las siguientes figuras: evento inicio, evento fin, evento intermedio, condición exclusiva, condición no exclusiva, condición or, condición and y componente, es necesario, además de la definición que se hizo anteriormente en el fichero abpmn0101.gmfgraph, generar las imágenes .SVG a las que se hacía referencia en dicho fichero.

Una vez generadas las figuras en formato .SVG, es necesario añadirlas a la carpeta ".diagram\images\svg", para que se puedan visualizar.

También hay que remplazar la función **createNodeShape** para cada una de las figuras anteriores que se encuentra en la ruta ".diagram\src\abpmn.diagram.edit.parts\.

Por ejemplo para la figura Componente, en la ruta ".diagram\src\abpmn.diagram.edit.parts\ComponenteEditPart", el siguiente código.

```
   /**
    * @generated NOT
    */
   protected IFigure createNodeShape() {
     URL url =
FileLocator.find(Abpmn0101DiagramEditorPlugin.getInstance()
         .getBundle(), new Path("images/svg/Componente.svg"), null);
//$NON-NLS-1$
     return new
ScalableImageFigure(RenderedImageFactory.getInstance(url),
         true, true, true);
   }
```

Y por último, hay que añadir en la ruta ".diagram\plugin.xml" el plug-in "org.eclipse.gmf.runtime.draw2d.ui.render". Para ello hacemos doble clic en ".diagram\plugin.xml", se abre una ventana y en el apartado "Dependencies→Required Plug-ins" pulsamos el botón "Add", y buscamos el plug-in "org.eclipse.gmf.runtime.draw2d.ui.render", salvamos y de esta manera dicho plug-in queda añadido.

6.6.2.2. Modificaciones del Código .Diagram

Una vez que hemos generado la carpeta .Diagram, además de hacer las modificaciones necesarias para mostrar las imágenes de las figuras que aparecerán en la paleta, tenemos que realizar los últimos cambios para poder editar los atributos multi-idioma y los formularios correspondientes a algunos atributos de las clases.

Dichos cambios consisten en modificar el código del procedimiento "getPropertySource" que se encuentra en el fichero Abpmn0101PropertySection.java dentro de la ruta ".diagram\src\abpmn.diagram.sheet\.

A continuación se muestra como quedaría una vez realizadas las modificaciones necesarias.

```
public IPropertySource getPropertySource(Object object) {
     if (object instanceof IPropertySource) {
       return (IPropertySource) object;
     }
     AdapterFactory af = getAdapterFactory(object);
     if (af != null) {
```

```
        IItemPropertySource ips = (IItemPropertySource)
af.adapt(object,
                IItemPropertySource.class);
        if (ips != null) {
                  PropertySource ps= new PropertySource(object, ips){
                  @Override
                  protected
org.eclipse.ui.views.properties.IPropertyDescriptor
createPropertyDescriptor(IItemPropertyDescriptor
itemPropertyDescriptor){
                                return new PropertyDescriptor(object,
itemPropertyDescriptor){
                                @Override
                                public CellEditor
createPropertyEditor(Composite composite){
                                  CellEditor result =
super.createPropertyEditor(composite);
                                  Object genericFeature =
itemPropertyDescriptor.getFeature(object);
                                    String atributo =
((EStructuralFeature)genericFeature).getName();
                                  if
(((EStructuralFeature)genericFeature).getEType().getName().equals("ESt
ring")) //&& ((EStructuralFeature)genericFeature).isMany())
                                {
                                    if
(atributo.contains("CambiosRealizados") ||
atributo.contains("Descripcion") || atributo.contains("Denominacion")
|| atributo.contains("Etiqueta") ||
atributo.contains("Autorizaciones") || atributo.contains("Secciones")
|| atributo.contains("Fechas") || atributo.contains("Seccion") ||
atributo.contains("Vencimiento") || atributo.contains("SitAdm") ||
atributo.contains("Campos") || atributo.contains("Excepciones") ||
atributo.contains("Condicion") || atributo.contains("Actuaciones"))
                                    {
                                      result = new
TestExtendedDialogCellEditor(composite,
getEditLabelProvider(),itemPropertyDescriptor, object, atributo);
                                    }
                                    if (atributo.contains("Nombre"))
                                    {
                                      String container =
((EStructuralFeature)genericFeature).getContainerClass().getName();
                                      if (!container.equals("abpmn.Pool")
&& !container.equals("abpmn.Nota"))
                                      {
                                        result = new
TestExtendedDialogCellEditor(composite,
getEditLabelProvider(),itemPropertyDescriptor, object, atributo);
                                      }
                                    }
                                  }
                                  return result;
                                }
                                };
                  }
                  };
                  return ps;
        }
      }
      if (object instanceof IAdaptable) {
```

```
        return (IPropertySource) ((IAdaptable) object)
                .getAdapter(IPropertySource.class);
    }
    return null;
}
```

Además, es necesario añadir una serie de ficheros .java para la correcta edición de los formularios de algunos de los atributos. A continuación se muestra la lista con los ficheros que se han añadido a la ruta ".diagram\src\abpmn.diagram.sheet\.

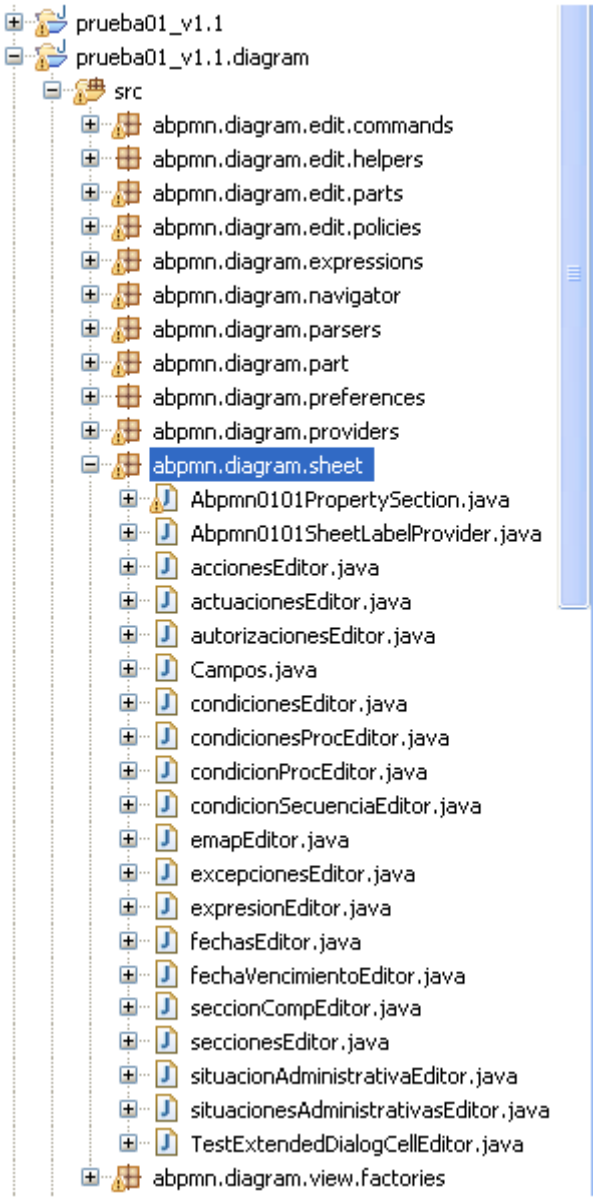


Figura 6.87. Carpeta prueba01_v1.1.diagram. Sheet.

6.7. CREACIÓN DE UN MODELO GRÁFICO

Para obtener el modelo gráfico generado, nos posicionamos sobre la carpeta .diagram, pulsamos el botón derecho y en la ventana que aparece se hace clic sobre la opción "Build Project" para construir el proyecto.

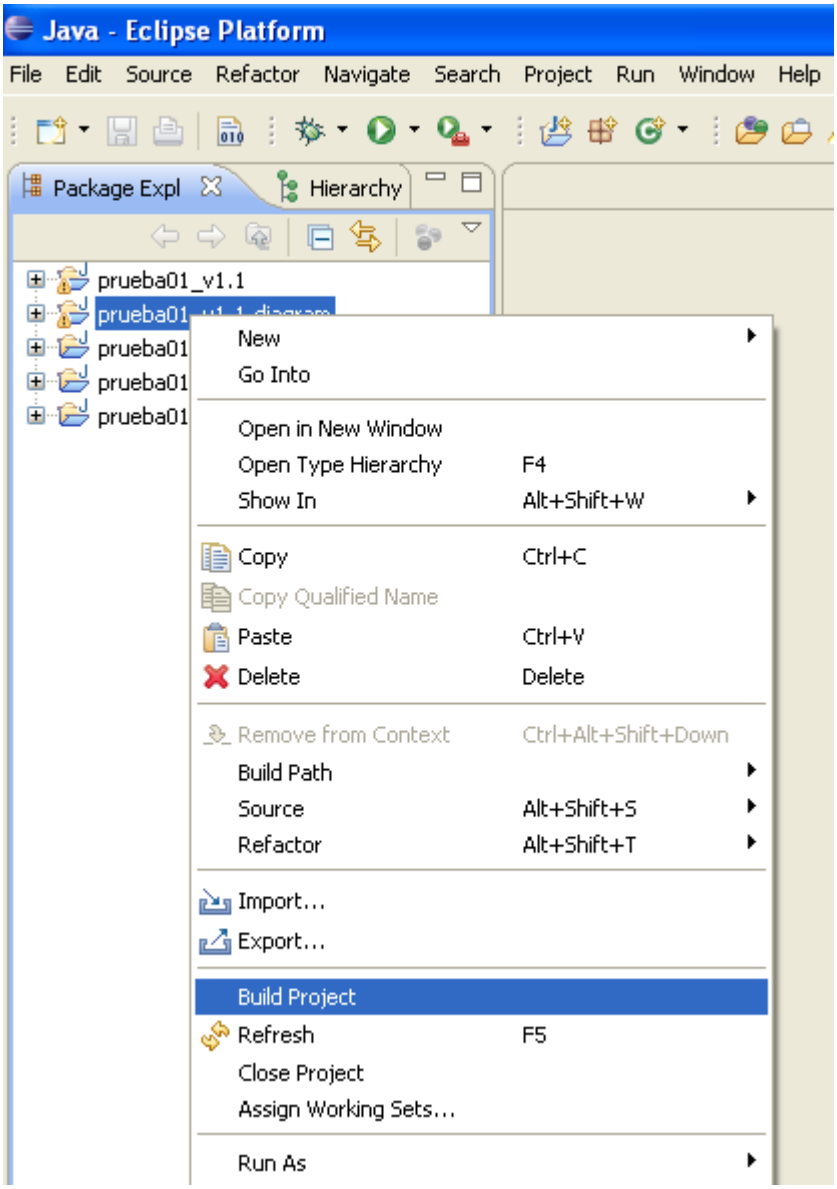


Figura 6.88. Carpeta prueba01_v1.1.diagram. Construir proyecto.

Nos aparece la siguiente ventana que nos indica que el proyecto se está generando.

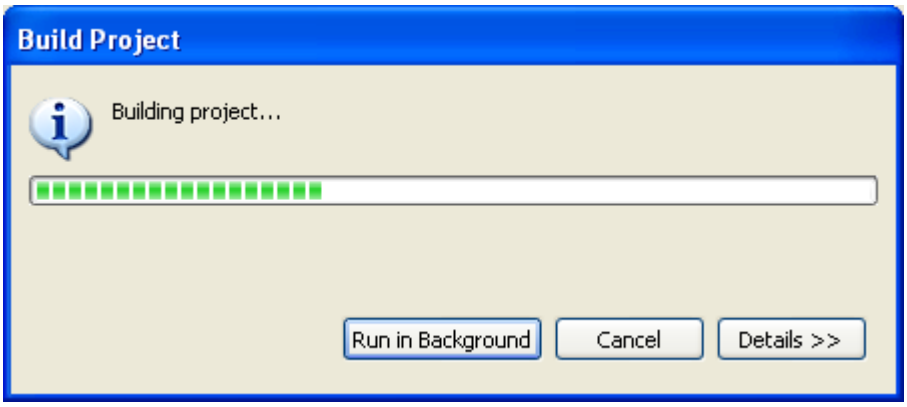


Figura 6.89. Carpeta prueba01_v1.1.diagram. Ventana generando proyecto.

Nos posicionamos sobre la carpeta .diagram, pulsamos el botón derecho y en la ventana que aparece seleccionamos la opción "Run As → Eclipse Application" para ejecutar el proyecto construido y poder obtener la herramienta gráfica.

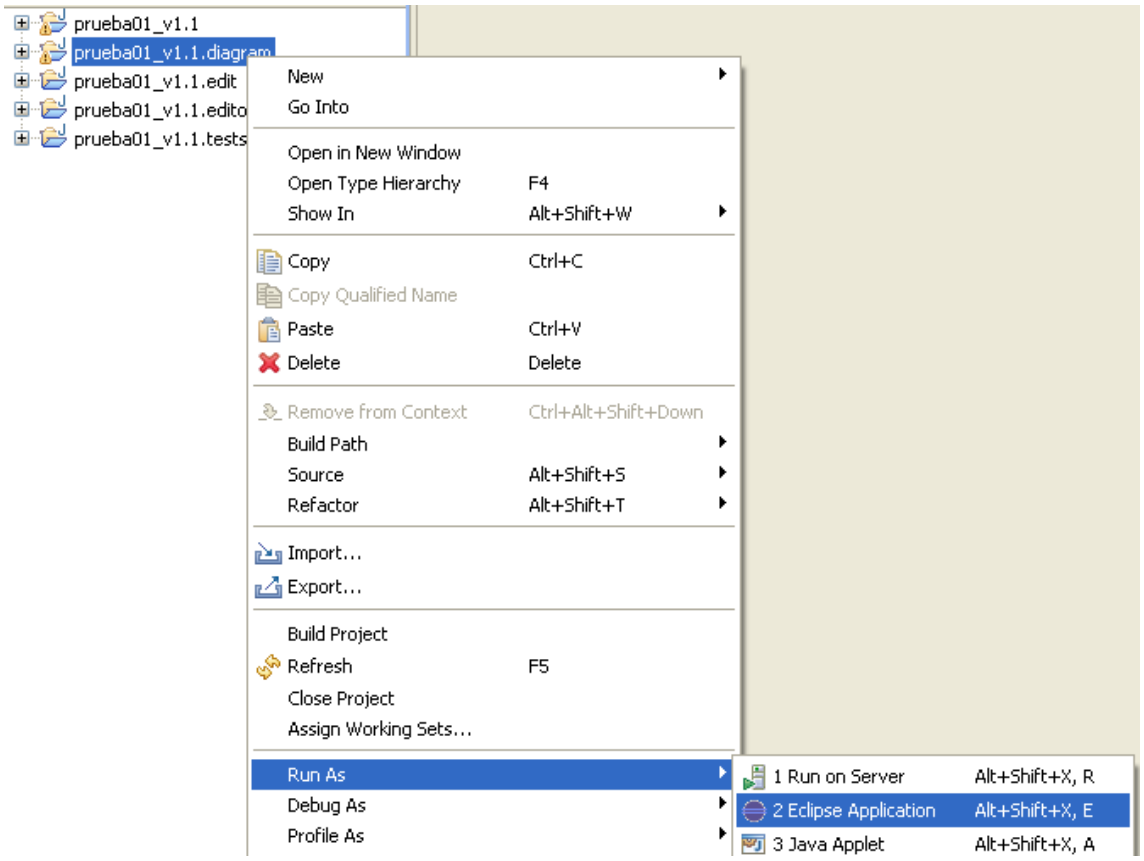


Figura 6.90. Carpeta prueba01_v1.1.diagram. Ejecutar proyecto.

Observamos que se abre una pantalla vacía.

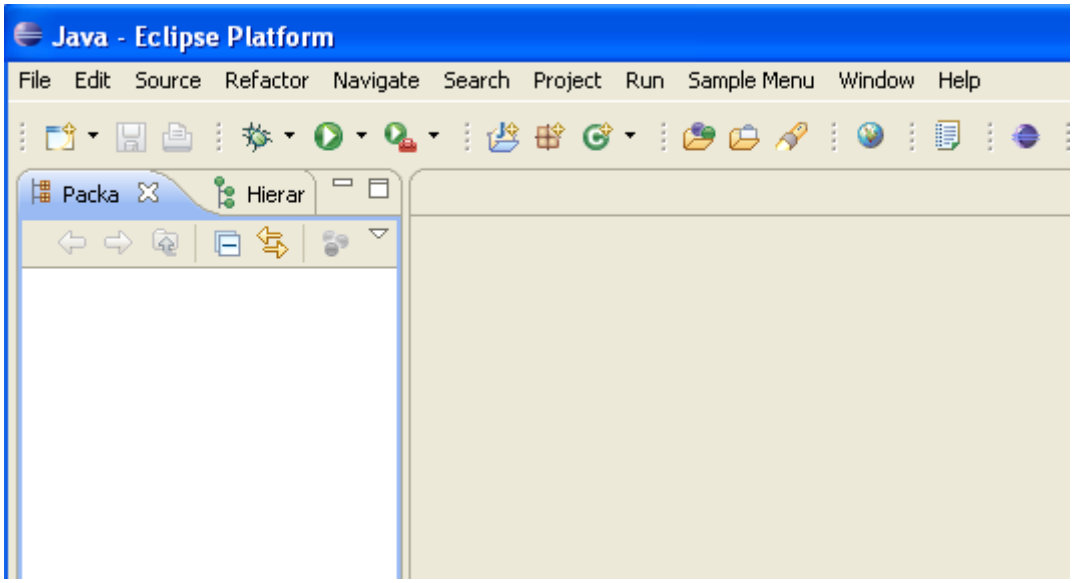


Figura 6.91. Definidor Visual. Generar proyecto.

Tenemos que crear un nuevo proyecto, para ello accedemos al menú principal, y seleccionamos "File → New → Project".

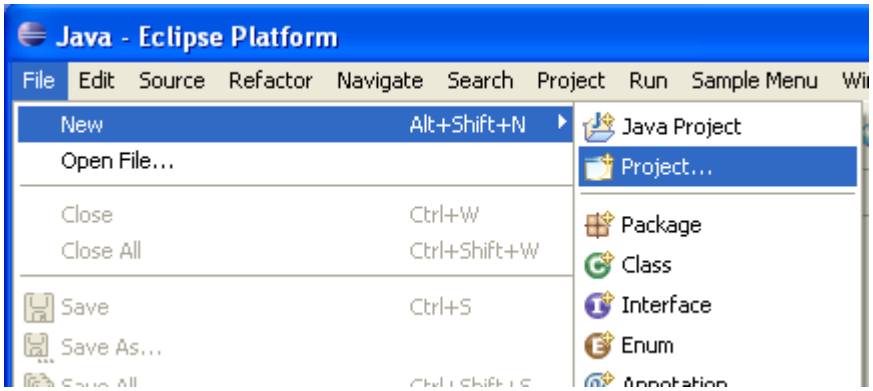


Figura 6.92. Definidor Visual. Ventana New.

Se abrirá la siguiente ventana, donde seleccionaremos la opción "General → Project".

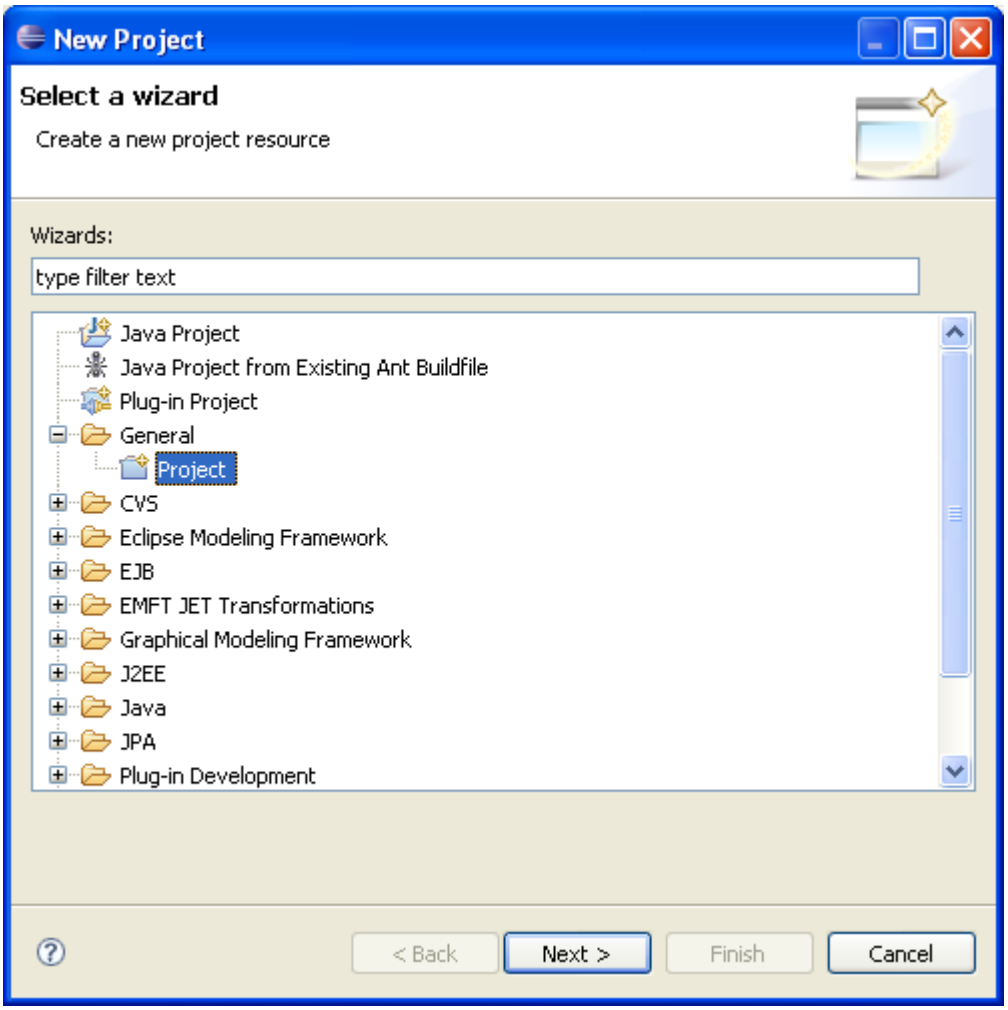


Figura 6.93. Definidor Visual. Ventana General.

En la siguiente ventana asignamos el nombre del proyecto.

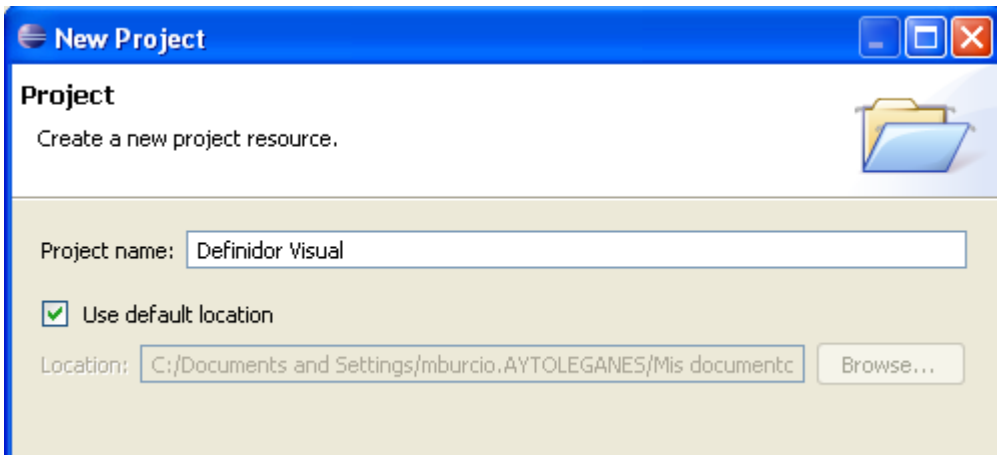


Figura 6.94. Definidor Visual. Ventana Nombre del proyecto.

Una vez que tenemos el proyecto creado, tenemos que crear una instancia de nuestra aplicación. Para ello, hacemos clic con el botón derecho del ratón sobre el proyecto creado anteriormente, seleccionamos la opción "New → Example" y se abrirá una nueva ventana donde elegiremos "Abpmn0101 Diagram".

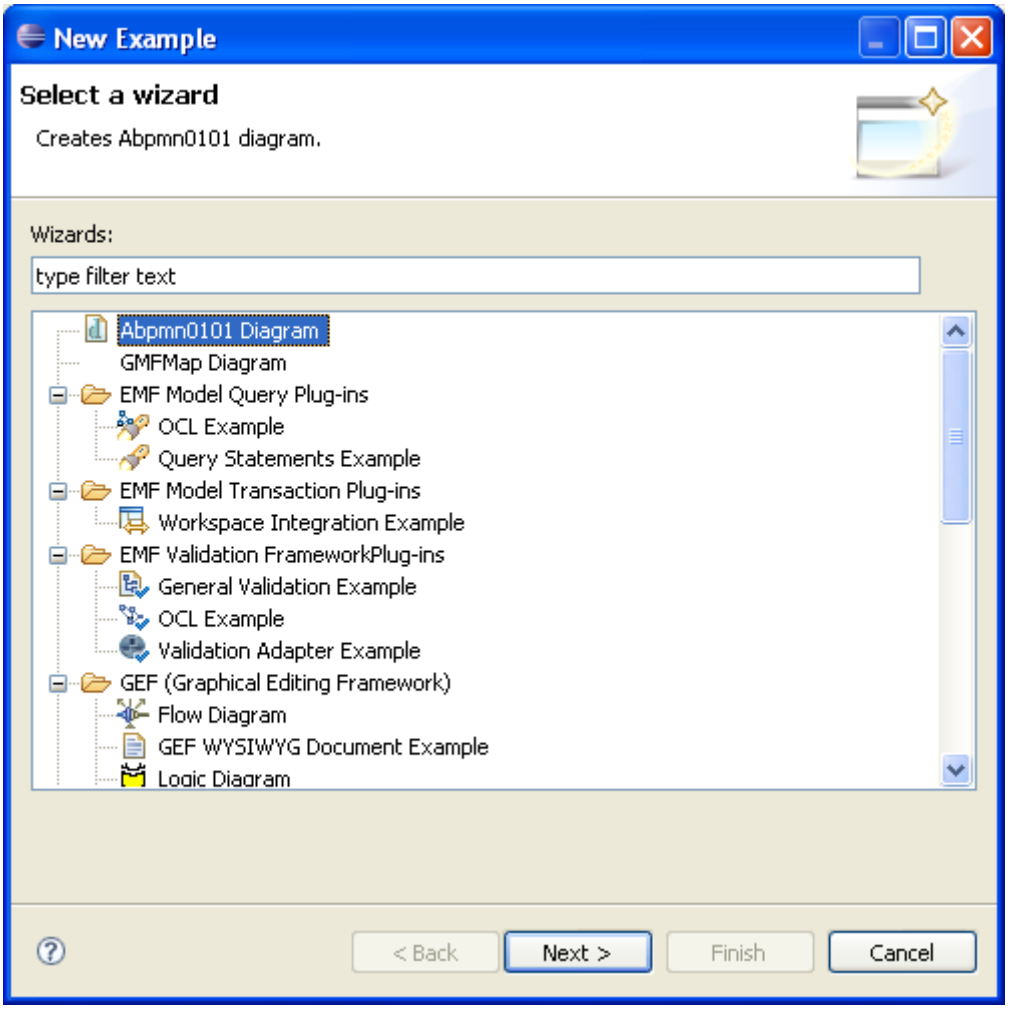


Figura 6.95. Definidor Visual. Ventana New Example.

En la siguiente ventana asignaremos el nombre.

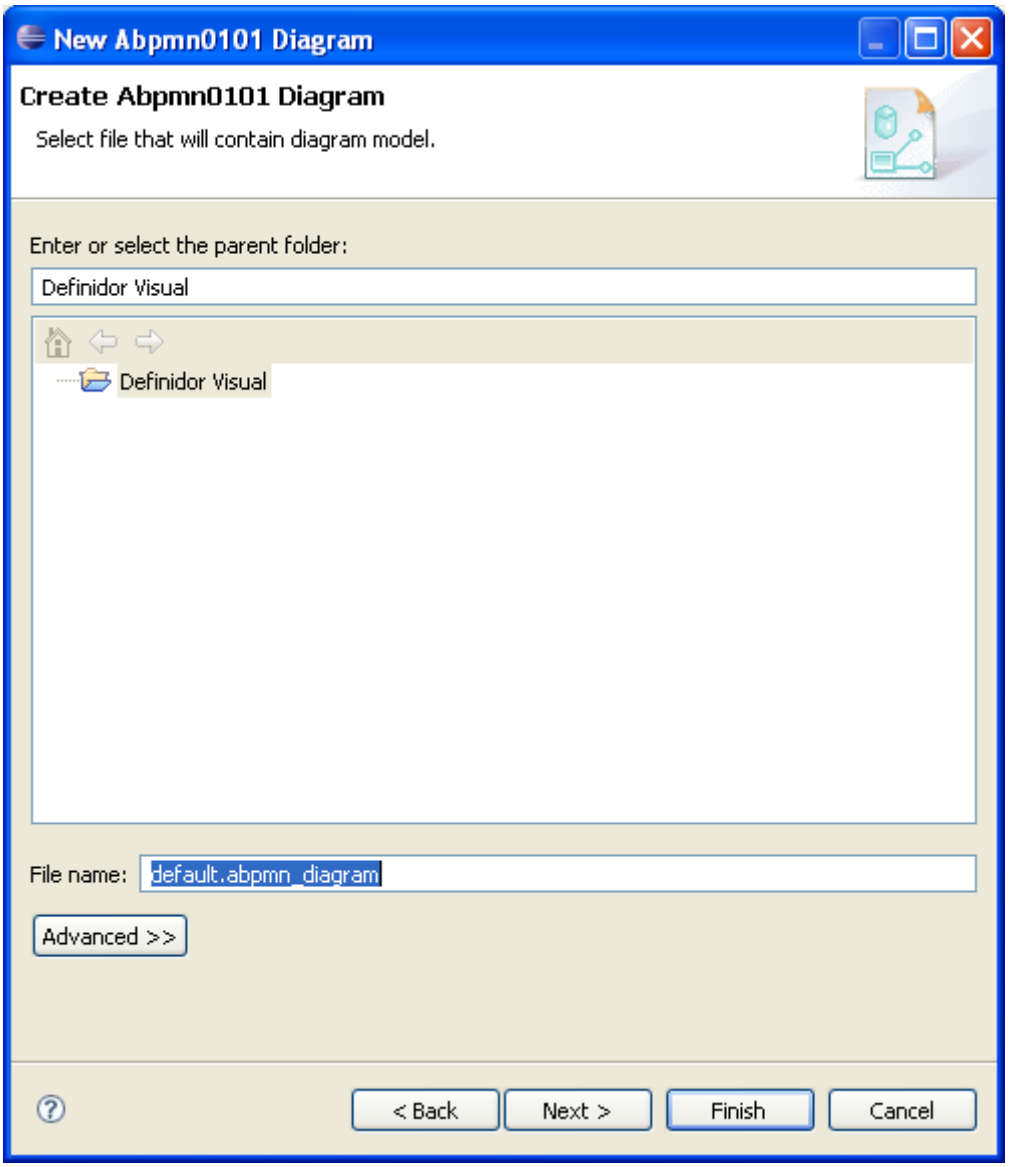


Figura 6.96. Definidor Visual. Ventana Nombre Instancia.

A continuación, nos aparecerá la pizarra, y a la derecha, la paleta con las distintas figuras de un procedimiento que se pueden dibujar sobre ella mediante la técnica "drag and drop". En la parte inferior de la pizarra mediante la opción "Properties", podemos definir y modificar los distintos atributos que definen cada figura.

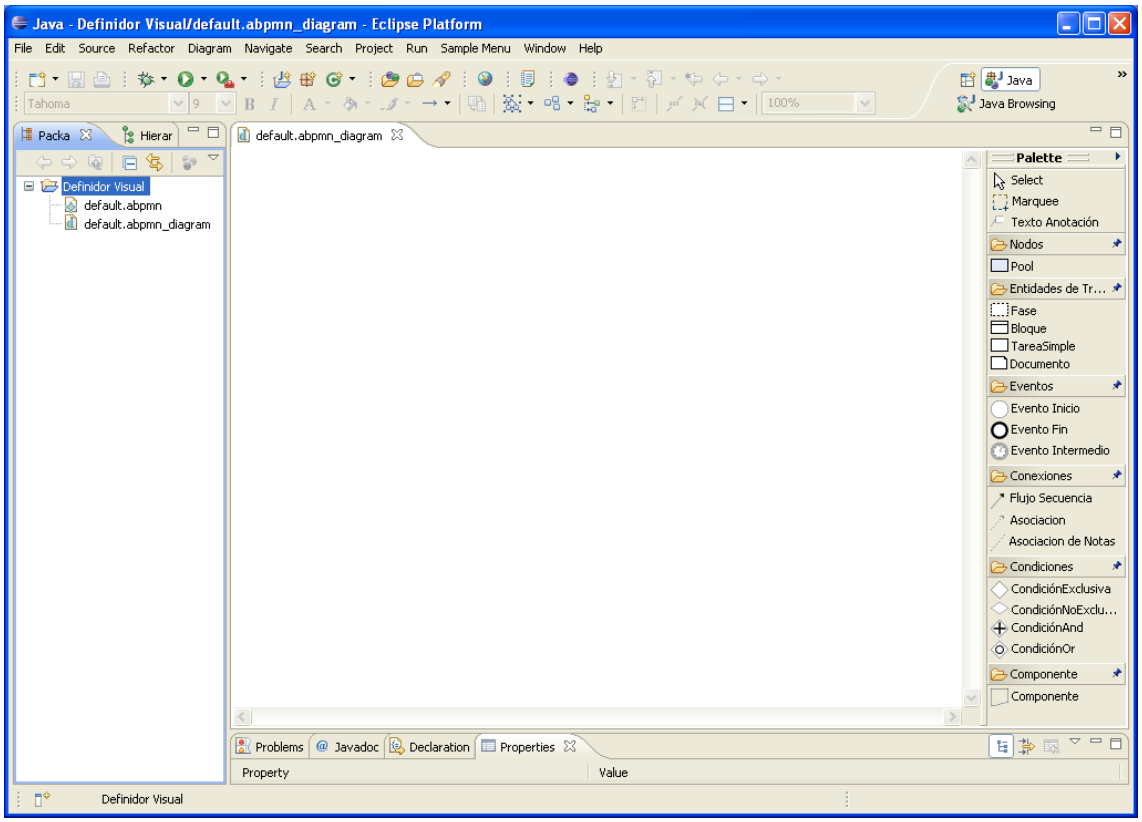


Figura 6.97. Definidor Visual.

6.8. OBTENCIÓN DEL FICHERO XML A PARTIR DEL MODELO GRÁFICO

Cuando tenemos ya generados una serie de elementos en la pizarra de la aplicación, podemos obtener un fichero en formato XML con la información de todos ellos.

Para ello, nos posicionamos en la parte izquierda, sobre el fichero "default.abpmn" y pulsamos el botón derecho del ratón, obtenemos un menú de opciones, seleccionamos "Open With → Text Editor".

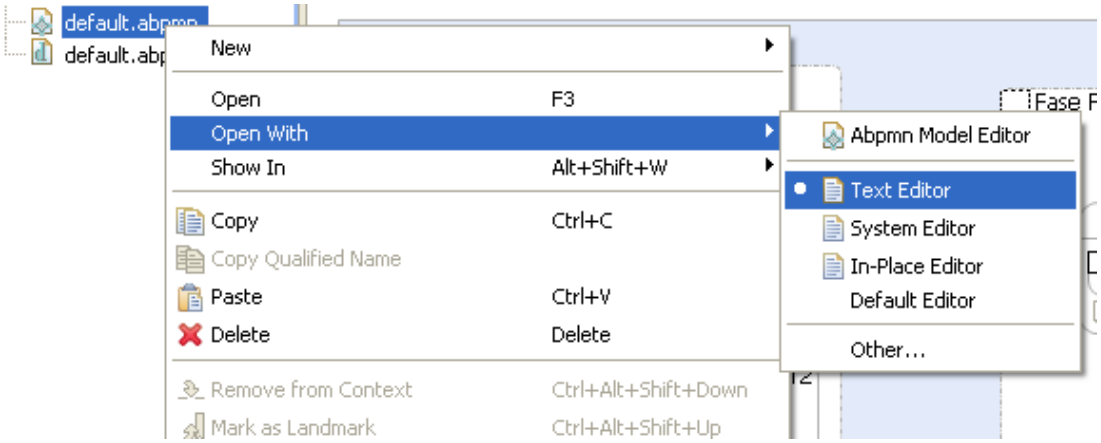


Figura 6.98. Abrir fichero XML.

Se abrirá en la parte derecha una ventana con toda la información en formato XML. A continuación se muestra un ejemplo de un fichero XML generado con el definidor visual.

default.abpmn

```
<?xml version="1.0" encoding="UTF-8"?>
<abpmn:Procedimiento xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www
  <pools IdPool="Pool Principal">
    <objetospool xsi:type="abpmn:Fase" tipoentidadtramitacion="FASE" CodigoFase="Fase Inicial">
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="Bloque 1 FI"/>
      <objetosgraficos xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      <objetosgraficos xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      <objetosgraficos xsi:type="abpmn:Documento" tipoentidadtramitacion="DOCUMENTO"/>
      <objetosgraficos xsi:type="abpmn:Asociacion" tipo="CONEXION" origen="//@pools.0/@objetosp
      <objetosgraficos xsi:type="abpmn:Nota" NombreEntApoy="Nota"/>
    </objetospool>
    <objetospool xsi:type="abpmn:Fase" tipoentidadtramitacion="FASE" CodigoFase="Fase Final">
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="b1">
        <ObjetosBloque xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
        <ObjetosBloque xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      </objetosgraficos>
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="b2"/>
    </objetospool>
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.0/@
    <objetospool xsi:type="abpmn:AsociacionNota" tipo="CONEXION" origen="//@pools.0/@objetospoo
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.3"
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.1/@
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.1/@
  </pools>
  <NombreProc>0-Proc 1</NombreProc>
  <NombreProc>1-Procd 1</NombreProc>
  <CategoriasProc>Categoria 1</CategoriasProc>
  <CategoriasProc>Categoria 2</CategoriasProc>
```

Figura 6.99. Fichero XML.

7. PRUEBAS DEL SISTEMA

7.1. INTRODUCCIÓN

En este capítulo se va a explicar cómo se han llevado a cabo las pruebas de la aplicación Definidor Visual.

La prueba del software es un elemento crítico para la garantía de la calidad del producto. El objetivo de esta etapa es garantizar la calidad del desarrollo. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: podemos probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar feedback mientras hay todavía tiempo y recursos para hacer algo.

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Pruebas realizadas.

7.2. PRUEBAS REALIZADAS

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software.

Las características generales son:

- La prueba comienza en el nivel de módulo y trabaja "hacia afuera".
- En diferentes puntos son adecuadas a la vez distintas técnicas de prueba.
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos) un grupo de pruebas independientes.
- La prueba y la depuración son actividades diferentes.

Una estrategia de prueba para el software debe constar de pruebas de bajo nivel, así como de pruebas de alto nivel.

Más concretamente, los objetivos de la estrategia de prueba son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de unidad, integración y las pruebas de sistema. Las pruebas de unidad y de integración son necesarias dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.

- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.

- Realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probados de nuevo y posiblemente devueltos a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados.

Para llevar a cabo las pruebas de la aplicación, en esta fase se han realizado los siguientes tipos de pruebas:

- Pruebas de unidad.

- Pruebas de integración.

- Pruebas de sistema.

7.2.1. Pruebas de unidad

Las pruebas de unidad se centran en el módulo. Usando la descripción del diseño como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. Es decir, se hacen pruebas de cada módulo por separado.

Las pruebas de unidad se han realizado para cada una de las figuras que se pueden modelar a través de la herramienta gráfica. El proceso que se ha seguido, es implementar la figura, comprobar que exista una entrada en la paleta de la aplicación para dibujar dicha figura, y que ésta se sitúa correctamente sobre la pizarra, con todas sus propiedades.

En este tipo de pruebas es donde se han encontrado mayor número de errores, debido a que todavía no se conocía con detalle la forma de implementar las figuras, y el

procedimiento a seguir para generarlas correctamente en cada uno de los ficheros de la aplicación.

7.2.2. Pruebas de integración

El objetivo de las pruebas de integración es coger los módulos probados en las pruebas de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

En este caso se ha llevado a cabo una integración incremental, es decir, la aplicación se construye y se prueba en pequeños segmentos.

Las pruebas de integración han consistido en integrar todas las figuras en la paleta de la aplicación y comprobar que no había errores al relacionar unas figuras con otras. Por ejemplo, conexiones entre ellas, unas figuras que pueden contener a otras, etc.

7.2.3. Pruebas del sistema

Estas pruebas verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Algunas de las pruebas llevadas a cabo son:

- Prueba de validación: Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido.

- Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.

8. INTEGRACIÓN E IMPLANTACIÓN DEL SISTEMA

8.1. INTRODUCCIÓN

En este capítulo se va a explicar como se ha llevado a cabo la integración e implantación de la aplicación.

Las actividades previas al inicio de la producción incluyen la preparación de la infraestructura necesaria para configurar el entorno, la instalación de los componentes, la activación de los procedimientos manuales y automáticos asociados y, cuando proceda, la migración o carga inicial de datos.

A continuación, se muestran los temas principales que se tratarán con más profundidad a lo largo del capítulo:

- Integración e implantación del sistema.

8.2. INTEGRACIÓN E IMPLANTACIÓN DEL SISTEMA

Una vez que se ha finalizado todo el desarrollo de la aplicación, el siguiente y último paso es el proceso de integración del sistema.

El proceso de integración e implantación del sistema tiene como objetivo principal la entrega y aceptación del sistema en su totalidad, y la realización de todas las actividades necesarias para el paso a producción del mismo.

Todo lo necesario para este fin está explicado en el manual de usuario de la aplicación que se encuentra en el siguiente apartado de esta memoria.

9. MANUAL DE USUARIO

9.1. ESPECIFICACIONES DE INSTALACIÓN

El primer paso que hay que realizar para poder ejecutar la aplicación, es instalar la máquina virtual de Java, sino está instalada en el equipo donde se va ejecutar la aplicación.

En segundo lugar, para poder ejecutar la aplicación, es necesario instalar la plataforma ECLIPSE EUROPA con todos los plugins necesarios. Para ello simplemente copiamos la carpeta Sun Eclipse que se encuentra en el raíz del CD del proyecto, en nuestro disco duro C:\ y ejecutamos el fichero "eclipse.exe". De esta manera ya podemos acceder a Eclipse Europa.

9.2. MANUAL DE USUARIO

9.2.1. Ejecutar la aplicación

Para ejecutar la aplicación desde Eclipse hay que seleccionar en el menú principal la opción "Run → Run". Tras hacerlo, se visualizará una nueva pantalla que aparecerá vacía. A continuación, desde el menú principal, creamos un nuevo proyecto (File → New → Project). En el wizard que se muestra tras seleccionar esta operación aparecen diversas opciones, en nuestro caso seleccionaremos la opción "General → Project" y le asignaremos un nombre a nuestro proyecto.

El siguiente paso, es crear una instancia de nuestra aplicación. Para ello, haciendo clic con el botón derecho del ratón sobre el proyecto creado anteriormente, seleccionamos la opción "New → Example". Tras hacerlo, se visualizará un wizard con diversas opciones, nosotros elegiremos "Abpmn0101 Diagram" y le asignaremos un nombre.

Tras crear la instancia, nos aparecerá la pizarra y a la derecha la paleta con las distintas entidades de un procedimiento que se pueden dibujar sobre ella mediante la técnica "drag and drop". En función de la entidad que estemos editando nos aparecerá en la parte inferior de la pizarra los distintos atributos que definen la entidad y que pueden ser modificados por el usuario.

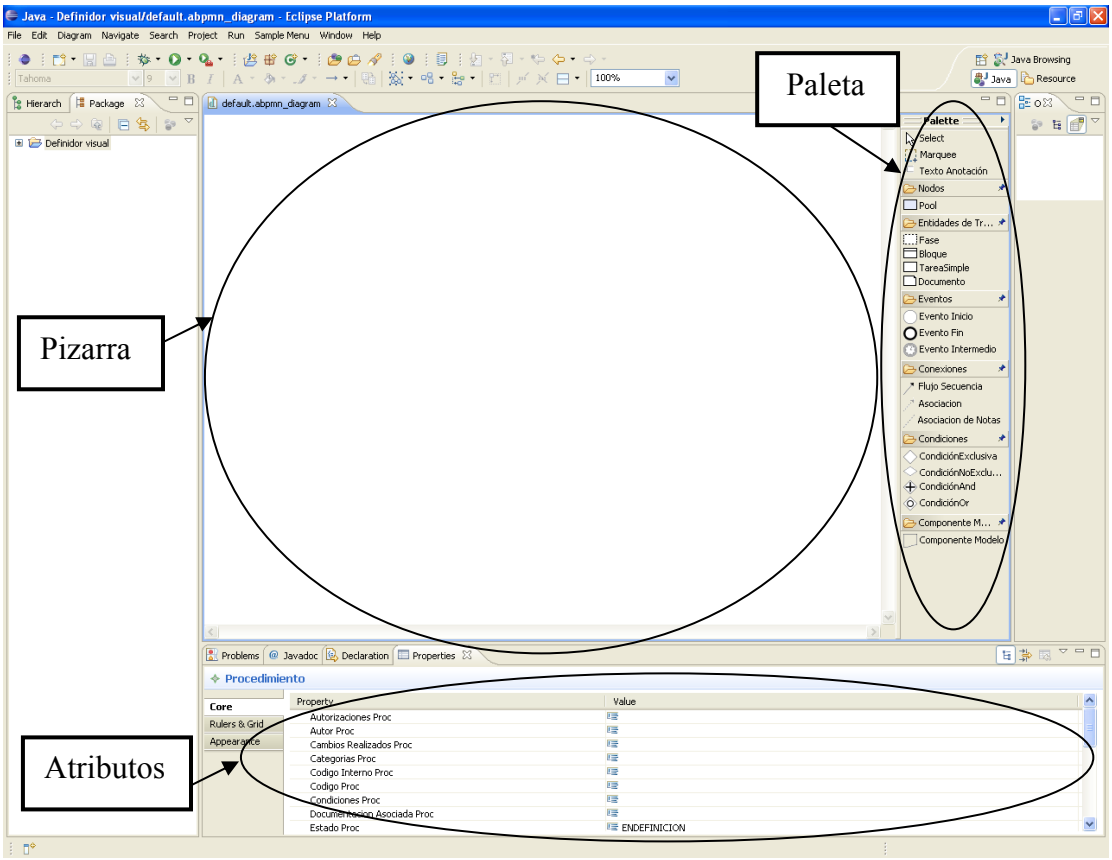


Figura 9.1. Entorno Definidor Visual.

A continuación se muestra un ejemplo de cómo quedaría la pizarra de la aplicación, con diferentes elementos generados en el DefinidorVisual.

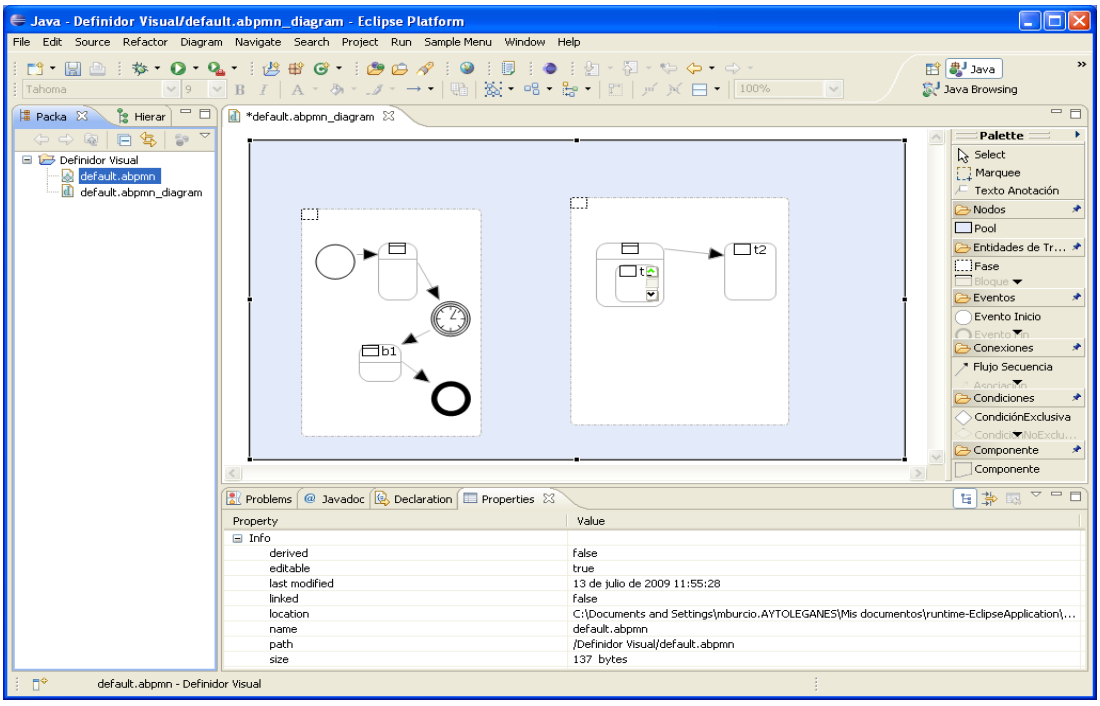


Figura 9.2. Ejemplo Definidor Visual.

9.2.2. Obtener fichero XML

Podemos obtener un fichero en formato XML, con toda la información de los elementos que aparezcan dibujados en la pizarra del Definidor Visual.

Suponemos que tenemos generados una serie de elementos en la pizarra de la aplicación y queremos obtener el fichero en formato XML con la información de todos los elementos. Para ello, nos posicionamos en la parte izquierda, sobre el fichero "default.abpmn" y pulsamos el botón derecho del ratón, obtenemos un menú de opciones, seleccionamos "Open With → Text Editor".

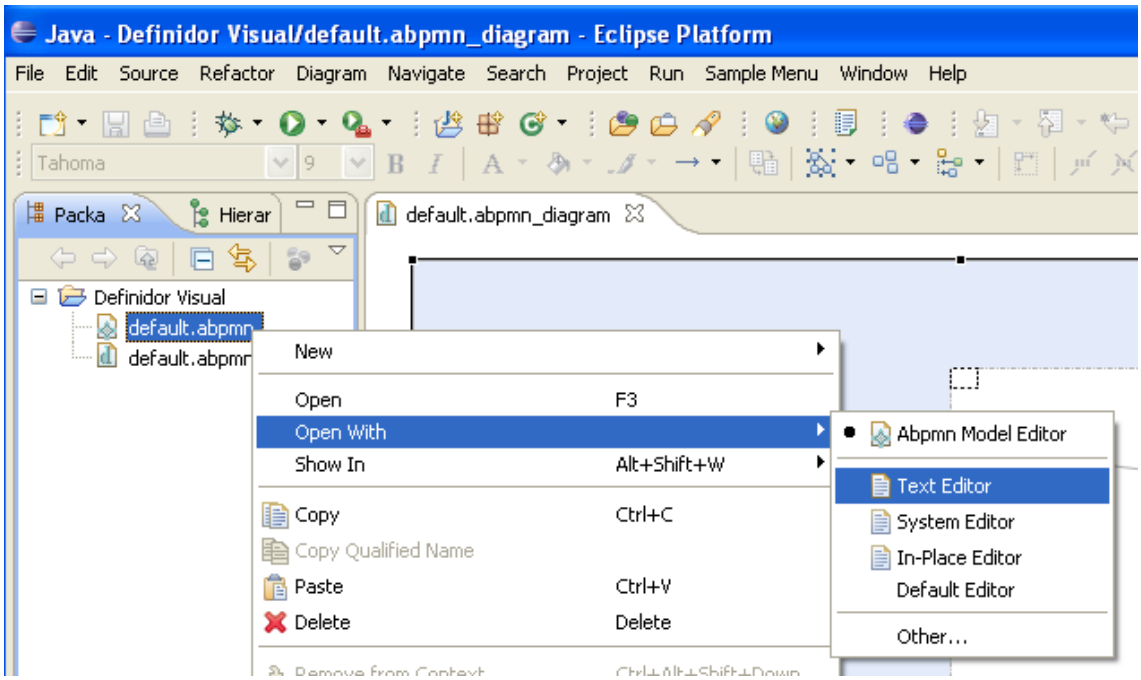
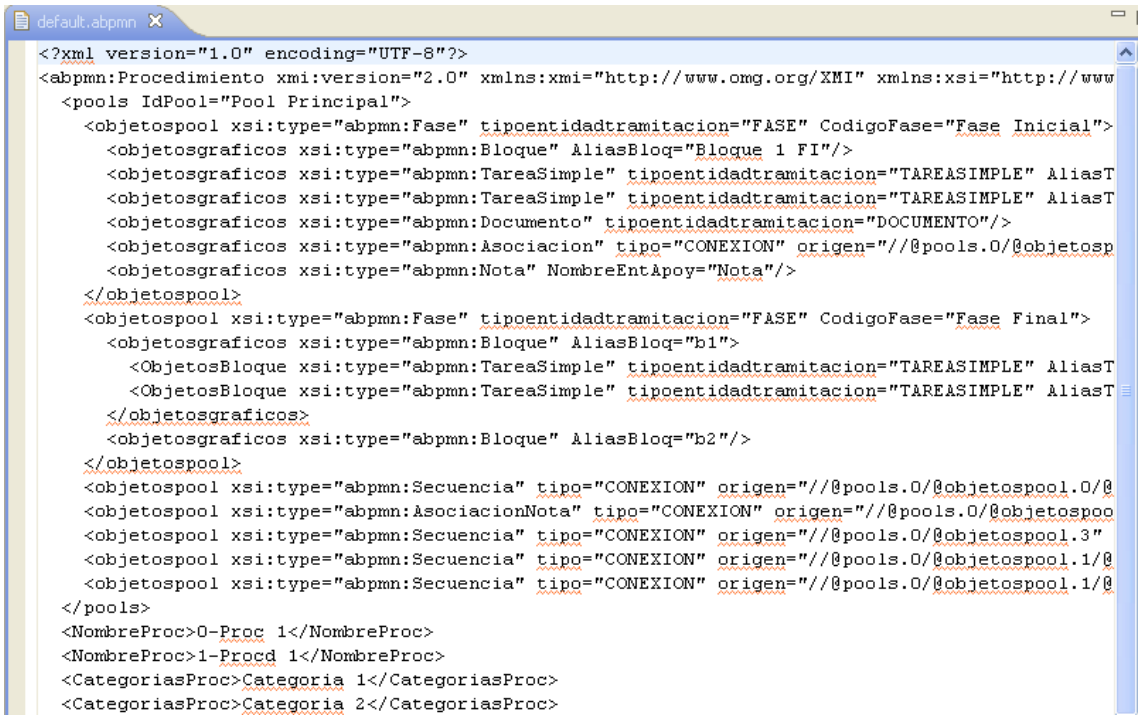


Figura 9.3. Generar fichero XML.

Se abrirá en la parte derecha una ventana con toda la información en formato XML. A continuación se muestra un ejemplo de un fichero XML generado con el definidor visual.



default.abpmn

```
<?xml version="1.0" encoding="UTF-8"?>
<abpmn:Procedimiento xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www
  <pools IdPool="Pool Principal">
    <objetospool xsi:type="abpmn:Fase" tipoentidadtramitacion="FASE" CodigoFase="Fase Inicial">
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="Bloque 1 FI"/>
      <objetosgraficos xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      <objetosgraficos xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      <objetosgraficos xsi:type="abpmn:Documento" tipoentidadtramitacion="DOCUMENTO"/>
      <objetosgraficos xsi:type="abpmn:Asociacion" tipo="CONEXION" origen="//@pools.0/@objetosp
      <objetosgraficos xsi:type="abpmn:Nota" NombreEntApoy="Nota"/>
    </objetospool>
    <objetospool xsi:type="abpmn:Fase" tipoentidadtramitacion="FASE" CodigoFase="Fase Final">
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="b1">
        <ObjetosBloque xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
        <ObjetosBloque xsi:type="abpmn:TareaSimple" tipoentidadtramitacion="TAREASIMPLE" AliasT
      </objetosgraficos>
      <objetosgraficos xsi:type="abpmn:Bloque" AliasBloq="b2"/>
    </objetospool>
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.0/@
    <objetospool xsi:type="abpmn:AsociacionNota" tipo="CONEXION" origen="//@pools.0/@objetospoo
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.3"
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.1/@
    <objetospool xsi:type="abpmn:Secuencia" tipo="CONEXION" origen="//@pools.0/@objetospool.1/@
  </pools>
  <NombreProc>0-Proc 1</NombreProc>
  <NombreProc>1-Procd 1</NombreProc>
  <CategoriasProc>Categoria 1</CategoriasProc>
  <CategoriasProc>Categoria 2</CategoriasProc>
```

Figura 9.4. Ejemplo fichero XML.

9.2.3. Formularios de atributos

Mediante la ventana de propiedades de las figuras de la pizarra, se pueden editar y modificar los atributos asociados a cada una de ellas. A continuación se muestran los distintos formularios que se han implementado.

9.2.3.1. Procedimiento

La pizarra representa el procedimiento en su conjunto. Las propiedades que el usuario puede editar para completar la definición del procedimiento son:

- Autorizaciones: En el procedimiento puede definirse "n" autorizaciones distintas especificando para cada una de ellas el perfil de actuación y las acciones qué pueden realizarse.

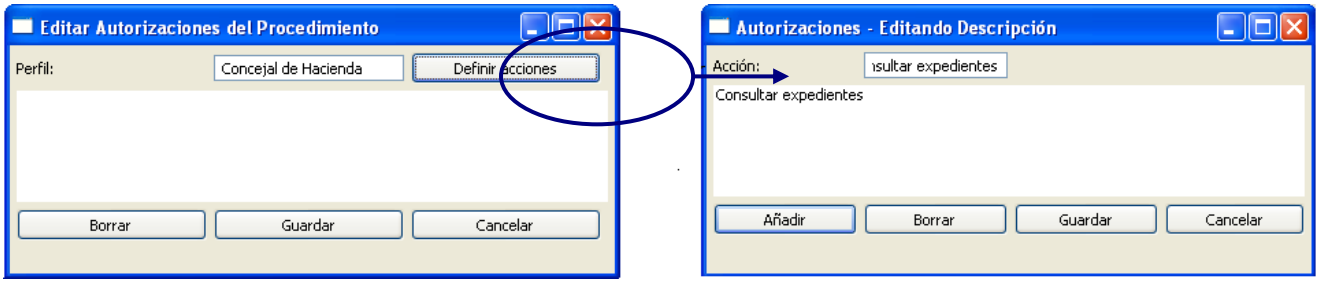


Figura 9.5. Formulario Procedimiento. Autorizaciones.

- Autor: El usuario en modo texto identificará quien es el autor del procedimiento.

- Cambios realizados: Su definición puede ser multi-idioma. Para ello el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

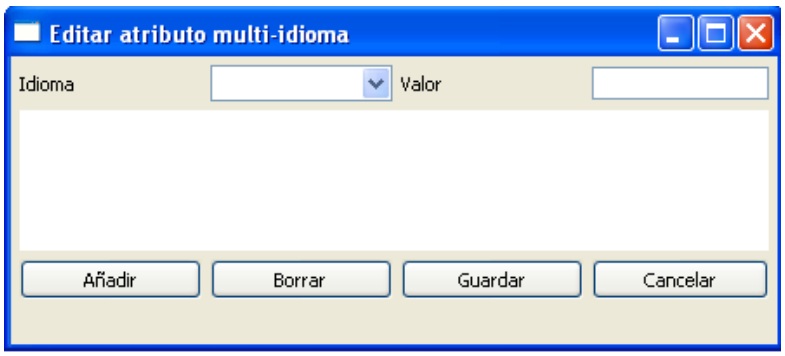


Figura 9.6. Formulario Procedimiento. Cambios realizados.

- Categorías: El procedimiento puede tener asignado "n" categorías distintas. El usuario deberá seleccionarlas de entre todas las disponibles.

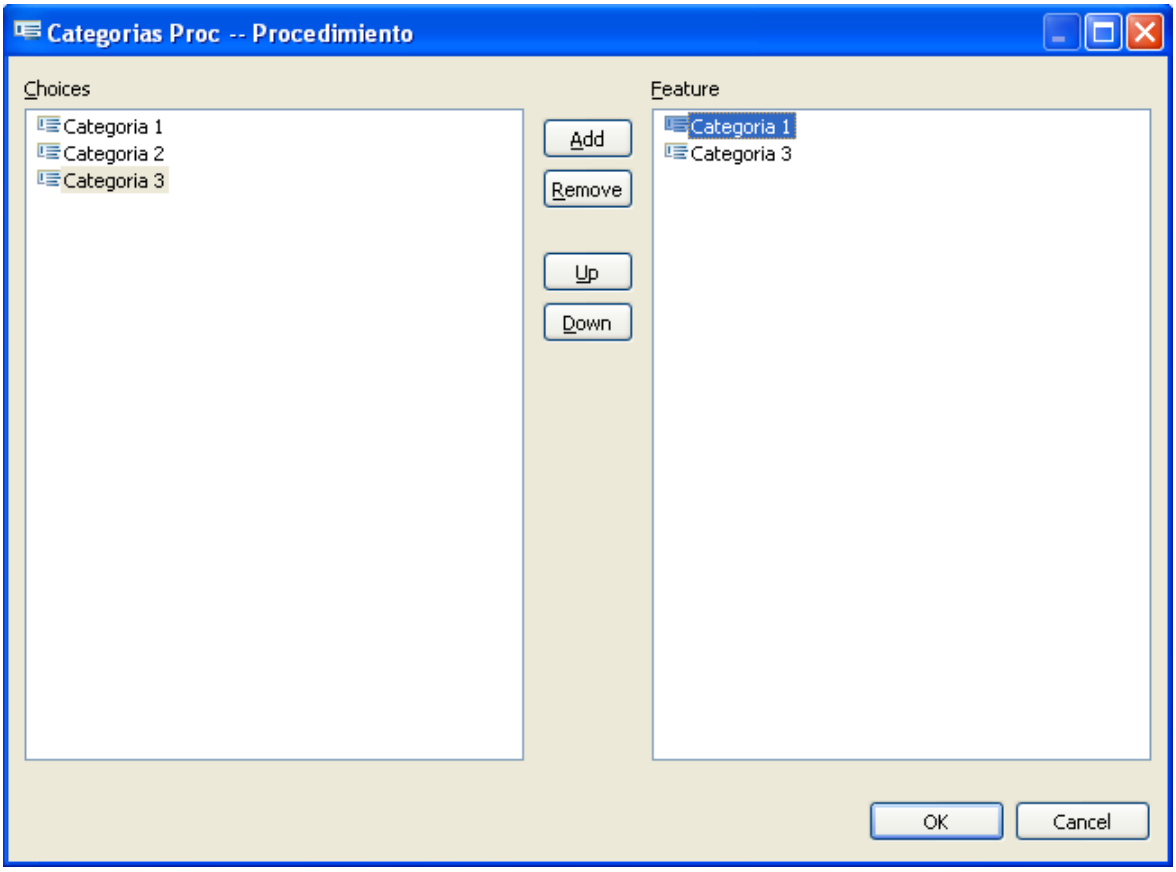


Figura 9.7. Formulario Procedimiento. Categorías.

- Código interno: En modo texto el usuario define el código que a nivel interno se aplicará a los expedientes del procedimiento.

- Código: El usuario define en modo texto el identificador unívoco del procedimiento.

- Condiciones: A través de formularios, el usuario especifica la colección de condiciones que van a modelar el comportamiento de las entidades que intervienen en el procedimiento.

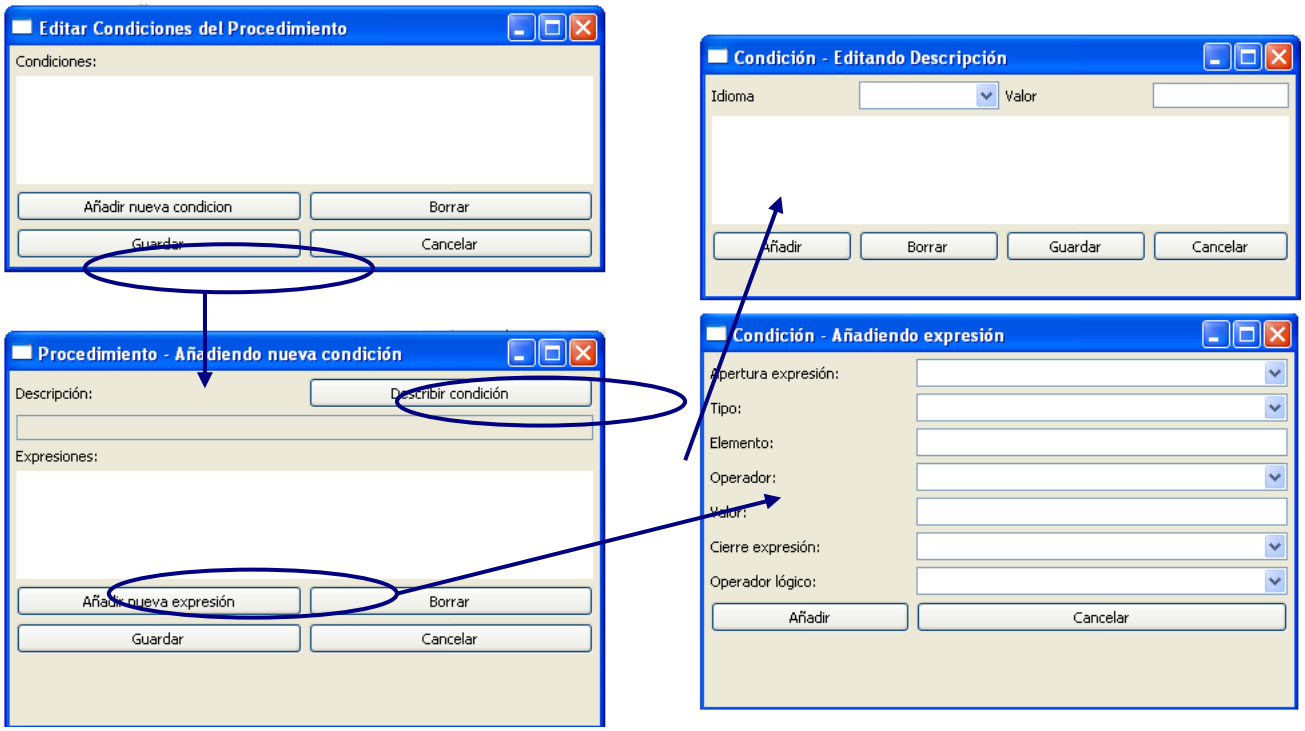


Figura 9.8. Formulario Procedimiento. Condiciones.

- Documentación asociada: El usuario puede asociar al procedimiento documentación de distinta índole. Para ello, deberá ir especificando una por una la distinta documentación que quiere asociar al procedimiento.

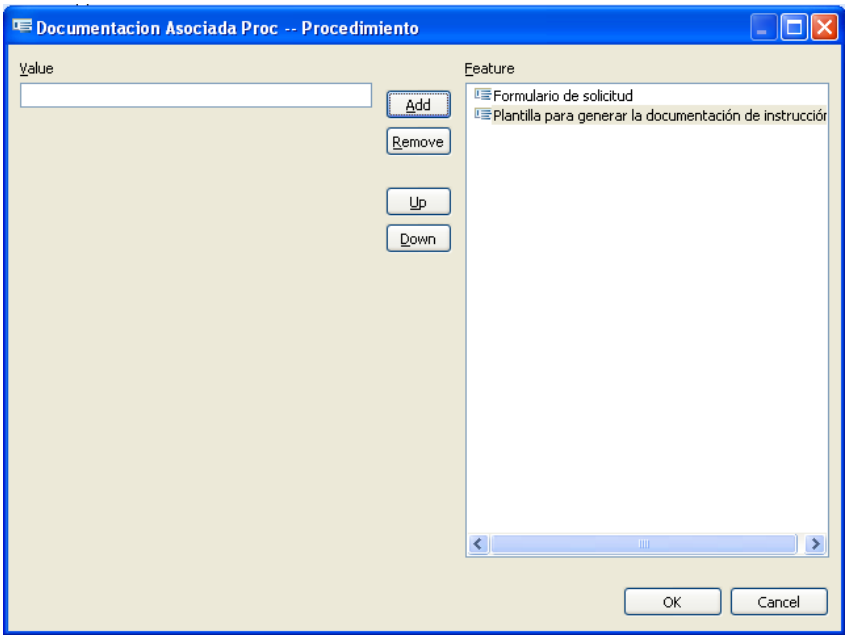


Figura 9.9. Formulario Procedimiento. Documentación asociada.

- Estado: A través de este atributo, el usuario define el estado en el que se encuentra el procedimiento: en definición, publicado o no vigente.

- Fecha de creación: En modo texto (dd/mm/aaaa), el usuario especifica la fecha de creación del procedimiento. Este atributo sólo es modificable por el usuario la primera vez que se crea el procedimiento, en posteriores modificaciones este campo sólo será de lectura.

- Fecha de modificación: Este atributo no es modificable por el usuario. Inicialmente, su valor es la fecha de creación del procedimiento. En posteriores modificaciones se le asignará automáticamente la fecha del sistema en el momento de la apertura.

- Fecha de publicación: Una vez que se concluye la definición del procedimiento, el usuario puede ponerlo a disposición de los usuarios tramitadores. A través de este atributo, el usuario indica en modo texto (dd/mm/aaaa) la fecha en la que se produce la publicación del procedimiento.

- Fechas: El usuario especifica aquellas fechas que tienen incidencia en el comportamiento del procedimiento.

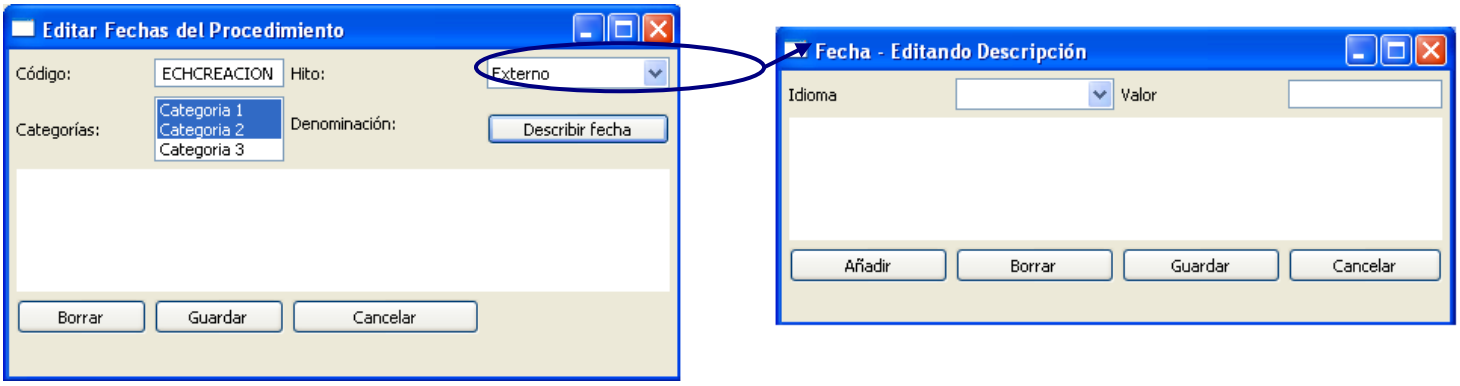


Figura 9.10. Formulario Procedimiento. Fechas.

- Formato de numeración: De entre todos los formatos de numeración disponibles, el usuario seleccionará aquel que se aplicará a los expedientes del procedimiento.

- Grupo de numeración: El usuario seleccionará de entre todos los disponibles aquel grupo de numeración que se aplicará a los expedientes del procedimiento.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente al procedimiento. Para ello, deberá seleccionarlo de entre todos los disponibles.

- Nombre del procedimiento: Su definición puede ser multi-idioma. Para ello el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

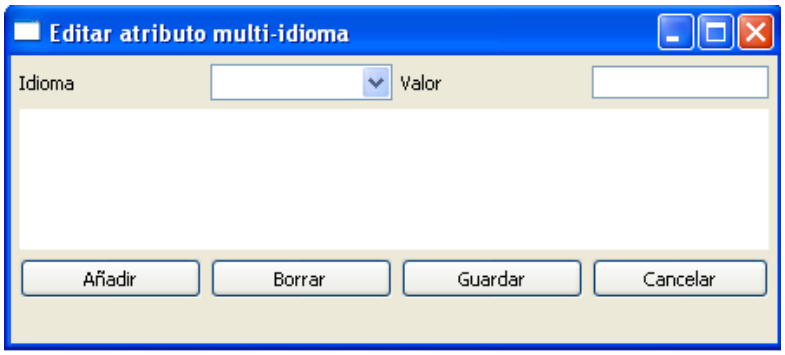


Figura 9.11. Formulario Procedimiento. Nombre.

- Referencia: El usuario seleccionará de entre todos los disponibles el formato del valor Referencia para los expedientes del procedimiento.

- Secciones: A través de este atributo, el usuario define las distintas secciones en las que se dividirá la interfaz de usuario con el fin de determinar la ubicación de las distintas entidades del procedimiento dentro de ella. Cada sección se identifica por el número de orden en el que aparece, el tipo de sección (cabecera, detalle o pie) y su modo de apertura (expandido o contraído).

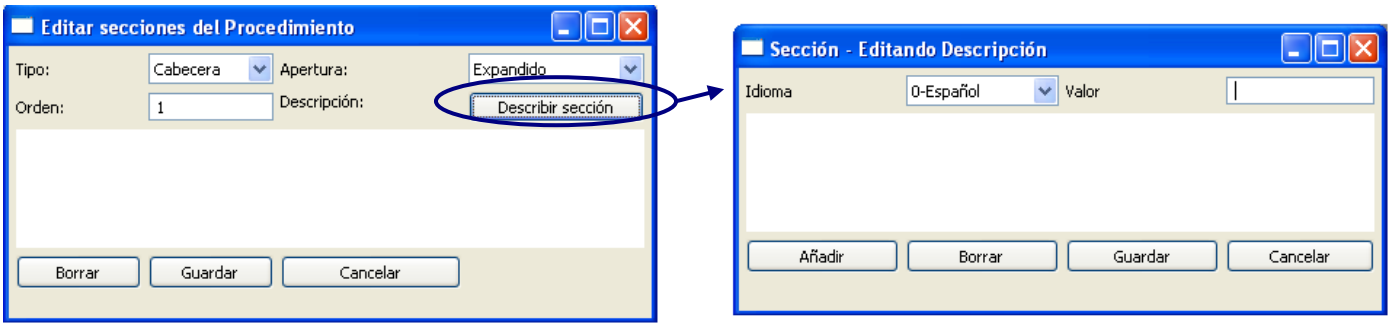


Figura 9.12. Formulario Procedimiento. Secciones.

- Situaciones administrativas externas: Con el fin de ayudar a conocer el estado de sus expedientes, el usuario define a través de este atributo cuáles son las distintas situaciones o estados en los que puede encontrarse un expediente del procedimiento.

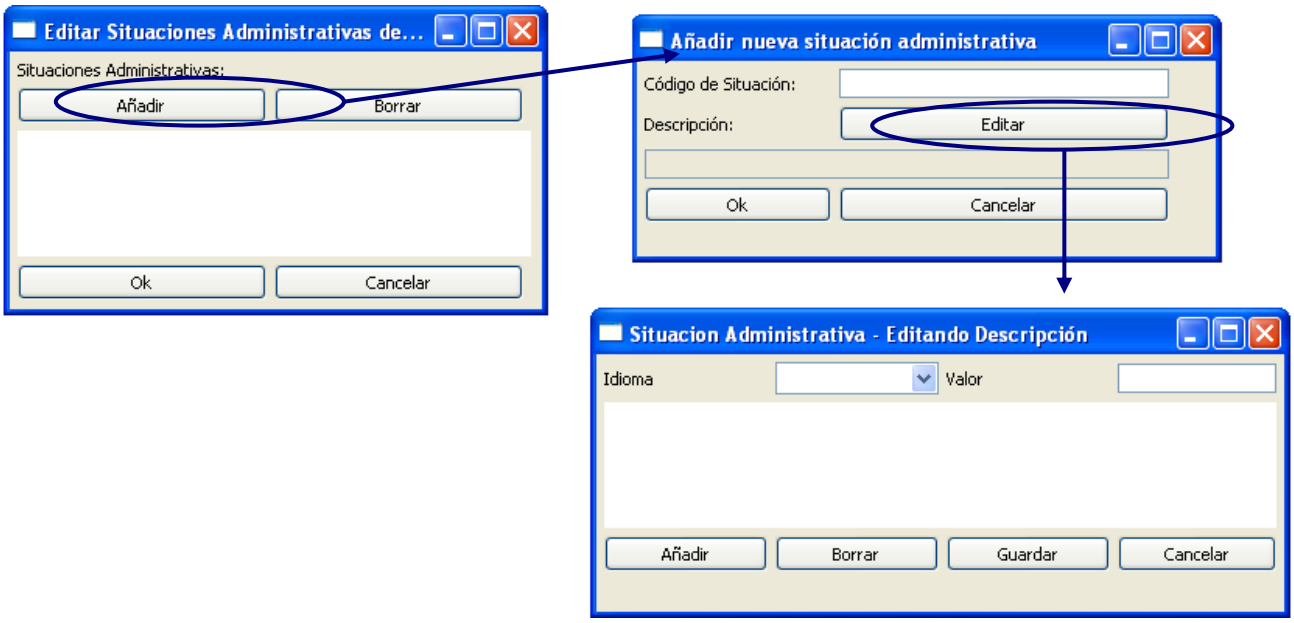


Figura 9.13. Formulario Procedimiento. Situaciones Administrativas externas.

- Situaciones administrativas internas: Como herramienta de gestión interna, el usuario define a través de este atributo cuáles son las distintas situaciones o estados en los que puede encontrarse un expediente del procedimiento.

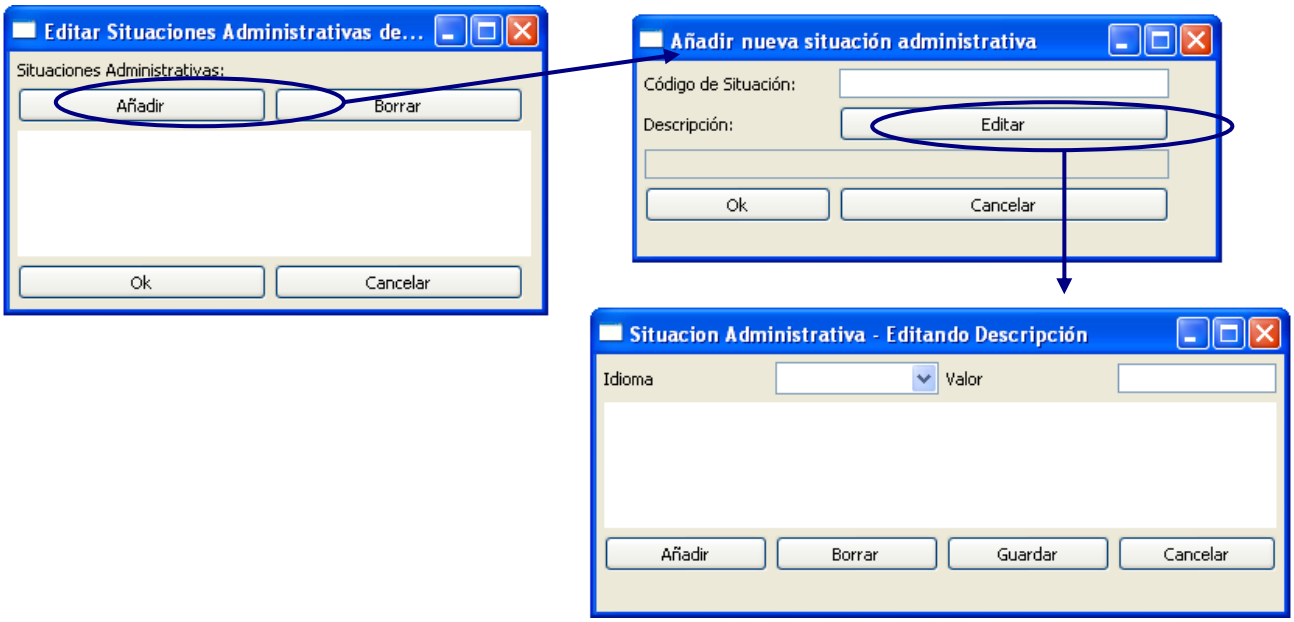


Figura 9.14. Formulario Procedimiento. Situaciones Administrativas internas.

- Versión: En modo texto, el usuario indica cuál es la nueva versión del procedimiento que se va a crear tras realizar las modificaciones oportunas.

9.2.3.2. Grupo organizativo o rol del procedimiento (Pool)

Para realizar la definición de un procedimiento, es obligatorio especificar un grupo organizativo o rol genérico en el que se englobará la definición del procedimiento. Las propiedades que el usuario puede editar para completar la definición del grupo organizativo son:

- Nombre: Desde los Servicios Comunes, el usuario seleccionará el grupo organizativo o rol al que pertenecerán los expedientes del procedimiento.

- Identificador: El usuario define a través de este atributo un identificador univoco para el grupo organizativo o rol seleccionado.

- Descripción: Opcionalmente, el usuario puede especificar en distintos idiomas información adicional acerca del grupo organizativo o rol seleccionado.

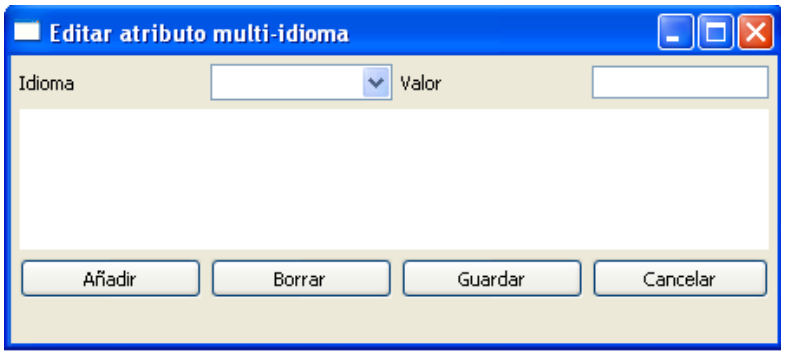


Figura 9.15. Formulario Pool. Descripción.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente al grupo organizativo o rol seleccionado con el fin de identificarlo de forma clara y rápida. Para ello, deberá seleccionarlo de entre todos los disponibles.

9.2.3.3. Fase

A excepción del grupo organizativo o rol seleccionado, el resto de las entidades se ubican dentro de fases. Por tanto, lo siguiente a especificar son las fases del procedimiento para continuar con su definición. Las propiedades que el usuario puede editar para completar la definición de una fase son:

- Código: En modo texto, el usuario define a través de este atributo un identificador univoco para la fase.

- Descripción: El usuario especifica un nombre descriptivo para la fase. Su definición es multi-idioma, esto es, el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

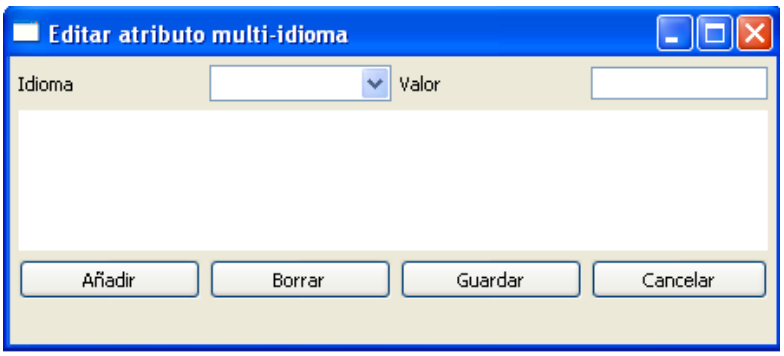


Figura 9.16. Formulario Fase. Descripción.

- Icono: Para identificar la fase el usuario puede asociar un icono que la represente. Para ello deberá seleccionarlo de entre todos los disponibles.

- Obligatoriedad: El usuario define a través de este atributo si la fase es opcional en la tramitación de los expedientes del procedimiento.

- Orden: Dado que las fases de un procedimiento son correlativas, el usuario especificará el orden de la fase dentro del procedimiento.

- Vigente: El usuario especifica el estado en el que se encuentra la fase: en definición, vigente o no vigente.

9.2.3.4. Bloque

Un bloque es un conjunto de tareas simples que se ejecutan para conseguir un objetivo común. Las propiedades que el usuario puede editar para completar la definición de un bloque son:

- Alias: En modo texto, el usuario define un identificador unívoco para el bloque.

- No reglado: El usuario define si el bloque es reglado (el sistema decide cuál es la siguiente tarea) o no reglado (el tramitador decide qué tarea ejecutar).

- Descripción: El usuario puede especificar en distintos idiomas información adicional acerca del bloque. Para ello, deberá ir seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

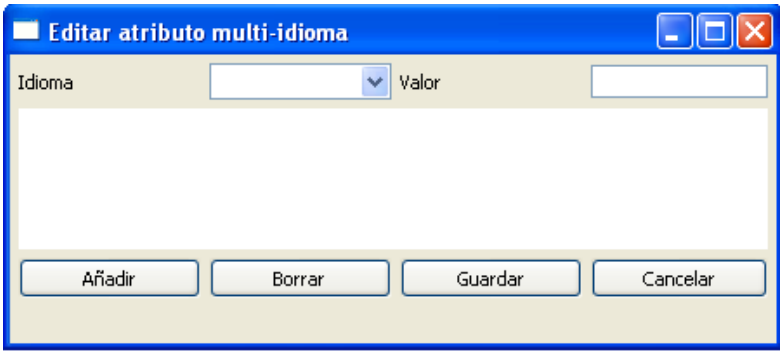


Figura 9.17. Formulario Bloque. Descripción.

- Orden: Con carácter informativo, el usuario define a través de este atributo cuál es el orden del bloque dentro de la fase en la que se ubica.

- Categorías: Un bloque puede tener asignadas "n" categorías distintas. Para ello, el usuario deberá seleccionarlas de entre todas las disponibles.

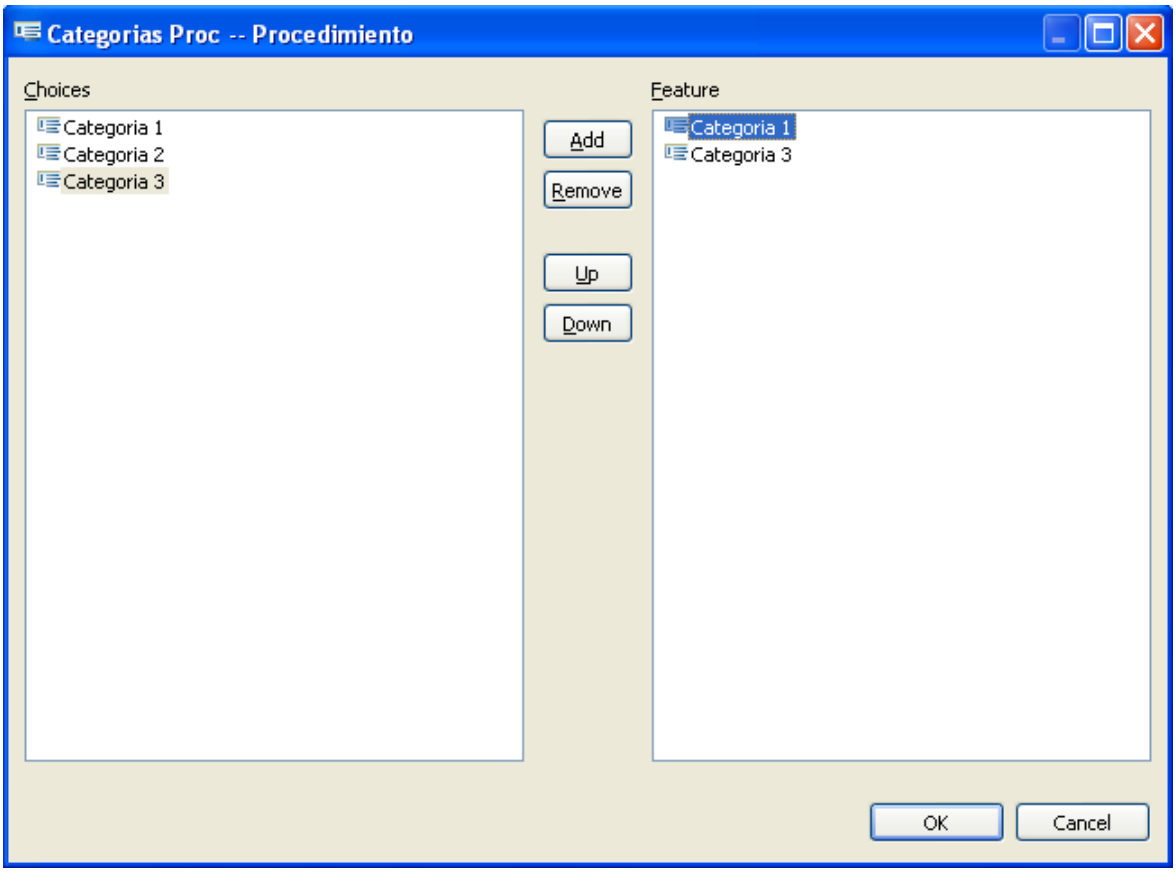


Figura 9.18. Formulario Bloque. Categorías.

- Prioridad: El usuario define cuál es la prioridad del bloque seleccionándola de entre todas las disponibles.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente al bloque y ayude a identificarlo rápidamente. Para ello, deberá seleccionarlo de entre todos los disponibles.

- Actuaciones: Mediante este atributo, el usuario tiene la posibilidad de definir cuáles son las actuaciones que se ejecutarán sobre el bloque como repuesta a eventos del usuario o del sistema. Una actuación está definida por una

descripción multi-idioma, una acción y una condición, éstas últimas seleccionadas de entre todas las disponibles.

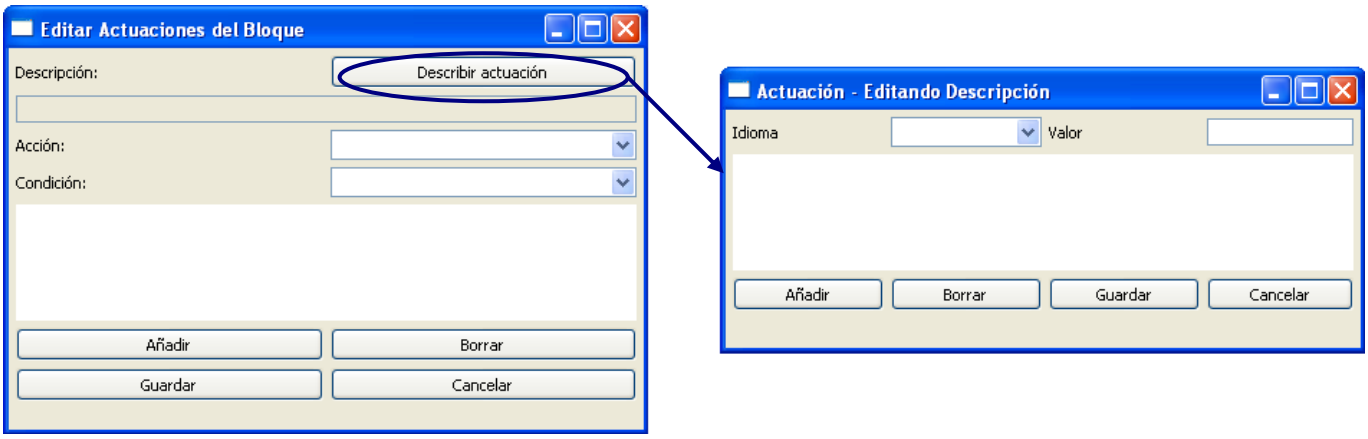


Figura 9.19. Formulario Bloque. Actuaciones.

- Es Modelo: El usuario especifica si el bloque podrá utilizarse posteriormente como plantilla para la creación de nuevas tareas y/o modelos.

- Nombre Tarea Modelo: Si el bloque se ha creado a partir de un modelo de tarea, el usuario deberá especificar el nombre de dicho modelo.

- Obligatoriedad: El usuario especifica si el bloque es de obligada ejecución o por el contrario es opcional.

- Bloque inicio, Bloque inicial, Bloque fin y Bloque final: A través de estos atributos, se especifica datos adicionales sobre el bloque concernientes al inicio y finalización del procedimiento.

- Dependencia con bloque anterior: El usuario especifica si el bloque depende de la tarea o bloque anterior.

- Visible desde Web: El usuario define si el bloque es accesible externamente vía Web.

- Finalizable Masivamente: A través de este atributo, el usuario especifica si el bloque puede finalizarse de forma masiva.

- Condiciones de inicio de bloque: El usuario especifica cuáles de las condiciones asociadas al procedimiento deberán cumplirse para que el bloque pueda comenzar.

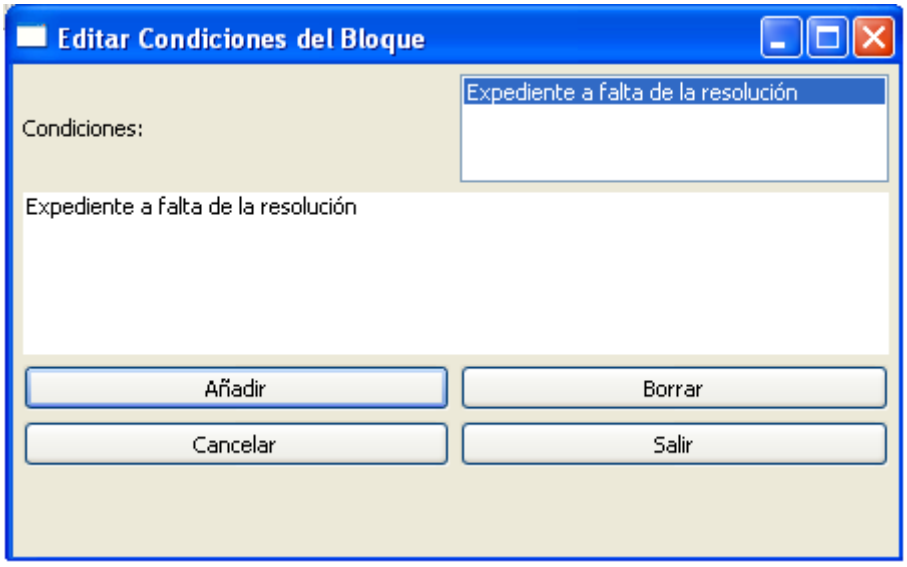


Figura 9.20. Formulario Bloque. Condiciones de inicio.

- Condiciones de fin de bloque: Al igual que en el atributo anterior, el usuario seleccionará de entre todas las condiciones disponibles cuáles de ellas deberán cumplirse para que pueda finalizar el bloque.

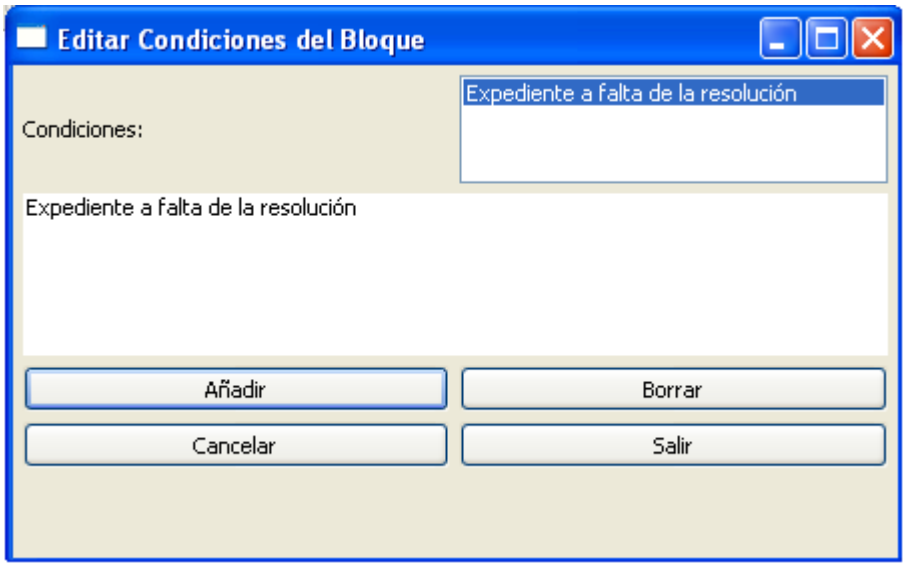


Figura 9.21. Formulario Bloque. Condiciones de fin.

- Modo de acceso: Mediante este atributo, el usuario puede especificar si el bloque será accesible externamente mediante tramitación telemática.

- Autorización de ejecución y Autorización de consulta: El usuario seleccionará aquellos grupos organizativos o roles que dispondrán de permiso de ejecución y/o de consulta sobre el bloque.

- Responsable: Opcionalmente, el usuario seleccionará de entre todas la unidades organizativas disponibles aquel grupo o rol responsable o encargante del bloque.

9.2.3.5. Tarea Simple

Una tarea simple es cada uno de los pasos que componen el proceso de gestión administrativa. Las propiedades que el usuario puede editar para completar la definición de una tarea simple son:

- Alias: En modo texto, el usuario define a través de este atributo un identificador univoco para la tarea.

- Descripción: Su definición es multi-idioma. El usuario deberá ir seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

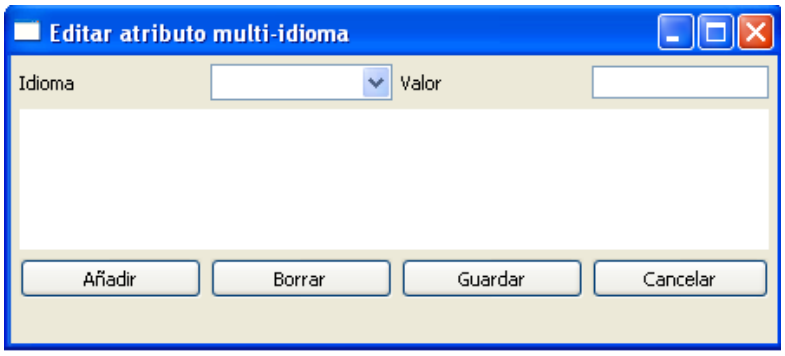


Figura 9.22. Formulario Tarea Simple. Descripción.

- Orden: Con carácter informativo, el usuario define cuál es el orden de la tarea dentro de la fase en la que se ubica.

- Categorías: Una tarea al igual que un bloque puede tener asignadas "n" categorías distintas. Para ello, el usuario deberá seleccionarlas de entre todas las disponibles.

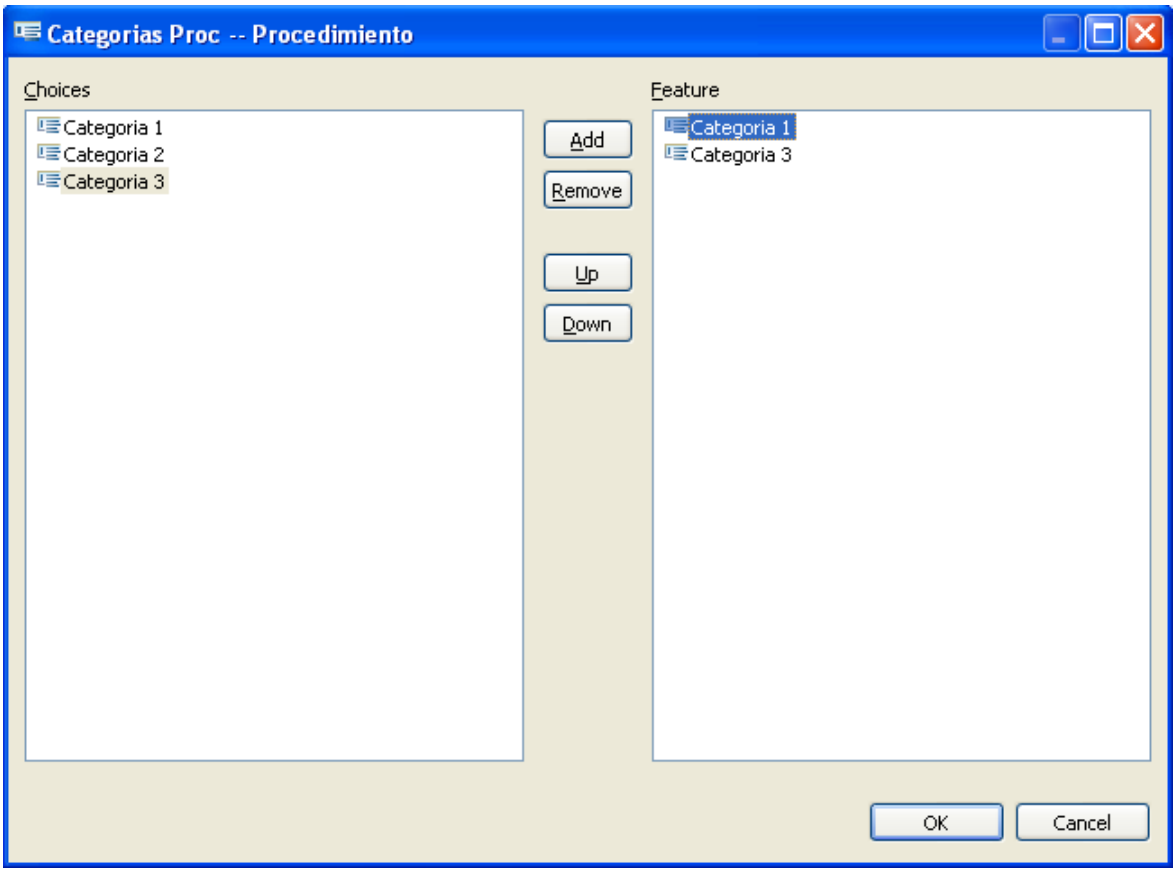


Figura 9.23. Formulario Tarea Simple. Categorías.

- Prioridad: A través de este atributo, el usuario indica cuál es la prioridad de la tarea seleccionándola de entre todas las prioridades disponibles.

- Estado: El usuario define el estado en el que se encuentra la tarea: en definición, vigente o no vigente.

- Modo de acceso: El usuario especifica si la tarea será accesible externamente mediante tramitación telemática.

- Es Modelo: El usuario especifica si la tarea podrá utilizarse posteriormente como plantilla para la creación de nuevas tareas y/o modelos.

- Nombre Tarea Modelo: Si la tarea se ha creado a partir de un modelo de tarea, el usuario deberá indicar el nombre de dicho modelo.

- Autorización de ejecución y Autorización de consulta: Desde los Servicios Comunes, el usuario seleccionará aquellos grupos organizativos o roles que dispondrán de permiso de ejecución y/o de consulta sobre la tarea.

- Responsable: Opcionalmente, el usuario seleccionará de entre todas la unidades organizativas disponibles aquel grupo o rol responsable de la tarea.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente a la tarea. Para ello, deberá seleccionarlo de entre todos los disponibles.

- Actuaciones: Mediante este atributo, el usuario tiene la posibilidad de definir cuáles son las actuaciones que se ejecutarán sobre la tarea como repuesta a eventos del usuario o del sistema. Una actuación está definida por una descripción multi-idioma, una acción y una condición, éstas últimas seleccionadas de entre todas las disponibles.

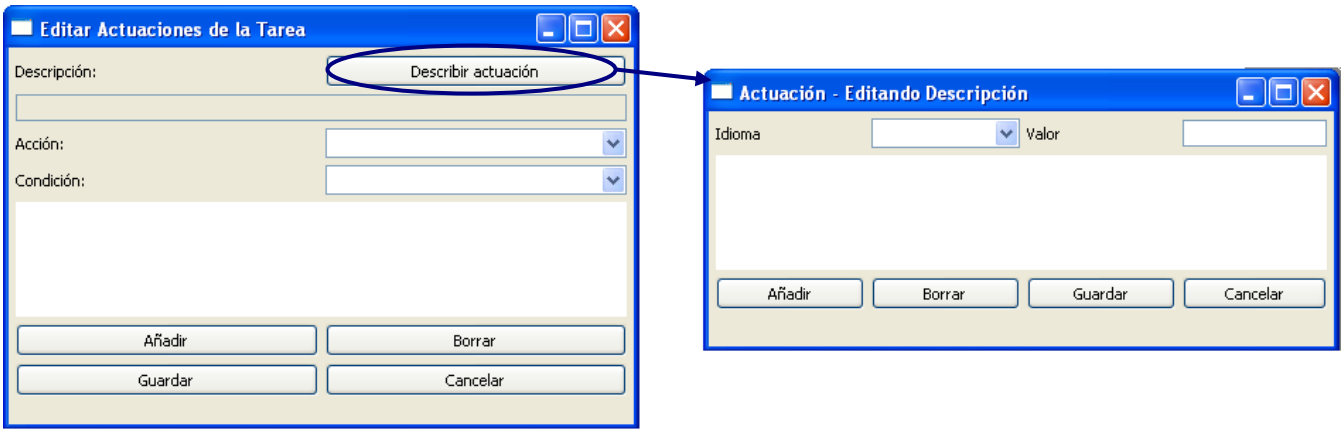


Figura 9.24. Formulario Tarea Simple. Actuaciones.

- Obligatoriedad: Mediante este atributo, el usuario especifica si la tarea es de obligada ejecución o por el contrario es opcional.

- Tarea inicio, Tarea inicial, Tarea fin y Tarea final: A través de estos atributos, se especifica datos adicionales sobre la tarea con respecto al inicio y finalización del procedimiento.

- Dependencia con tarea anterior: El usuario especifica si la tarea depende de la tarea o bloque anterior.

- Visible desde Web: El usuario define si la tarea es accesible externamente vía Web.

- Finalizable Masivamente: A través de este atributo, el usuario especifica si la tarea puede finalizarse de forma masiva.

- Condiciones de inicio de tarea: De entre todas las condiciones definidas para el procedimiento, el usuario seleccionará aquellas que deberán cumplirse para que la tarea pueda comenzar.

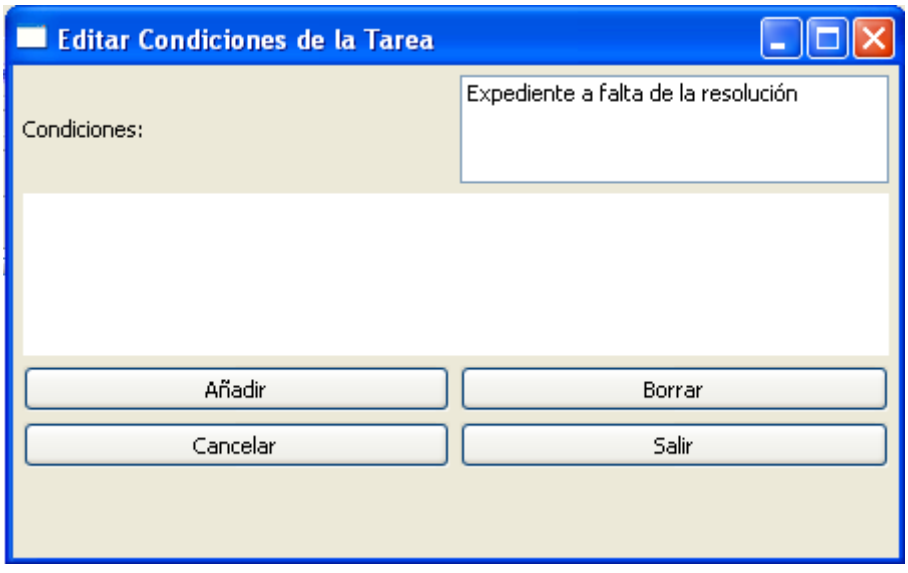


Figura 9.25. Formulario Tarea Simple. Condiciones de inicio.

- Condiciones de fin de tarea: El usuario seleccionará de entre todas las condiciones disponibles cuáles de ellas deberán cumplirse para que pueda finalizar la tarea.

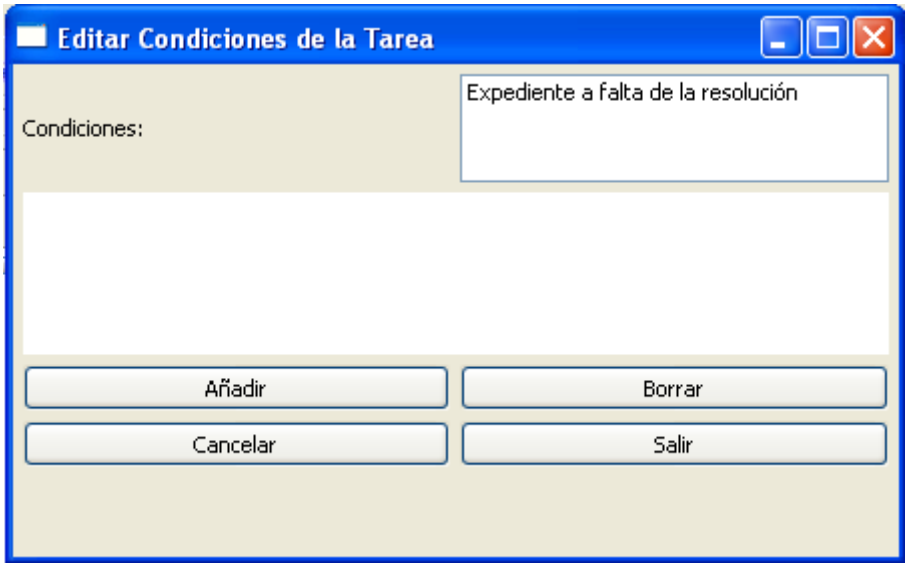


Figura 9.26. Formulario Tarea Simple. Condiciones de fin.

9.2.3.6. Documento

Un documento puede ser aportado como entrada a una tarea o bloque o bien puede ser generado como salida. Las propiedades que el usuario puede editar para completar la definición de un documento son:

- Id: En modo texto, el usuario define a través de este atributo un identificador univoco para el documento.

- Entrada requerida: El usuario especifica si el documento es requerido para que una tarea o bloque comience.

- Salida requerida: De forma similar al anterior atributo, se indica que el documento será generado una vez que la tarea o bloque haya sido completado.

- Nombre: Su definición puede ser multi-idioma. Para ello el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

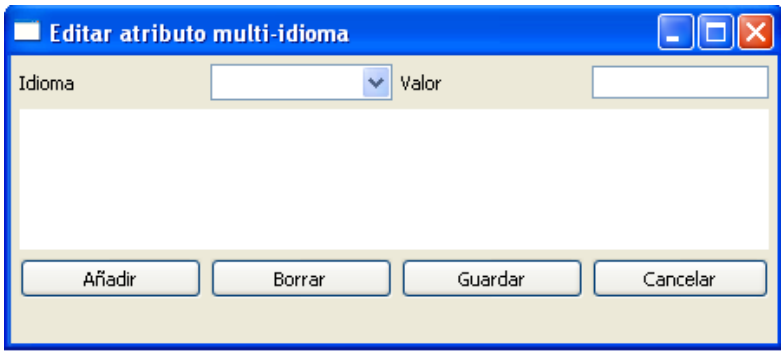


Figura 9.27. Formulario Documento. Nombre.

- Documentación requerida: Mediante este atributo, el usuario indica si el documento pertenece al grupo de documentación requerida por el procedimiento.

- Clasificación archivo: A través de este atributo, el usuario cataloga el documento en base a una clasificación archivística predefinida.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente al documento. Para ello, deberá seleccionarlo de entre todos los disponibles.

9.2.3.7. Eventos de comienzo y fin

Ambos elementos son usados para indicar el comienzo y fin de un expediente del procedimiento. Las propiedades que el usuario puede editar para completar la definición de cualquiera de estos eventos son:

- Id: En modo texto, el usuario define a través de este atributo un identificador univoco para el evento.

- Nombre: El usuario tiene la posibilidad de especificar el nombre del evento en distintos idiomas.

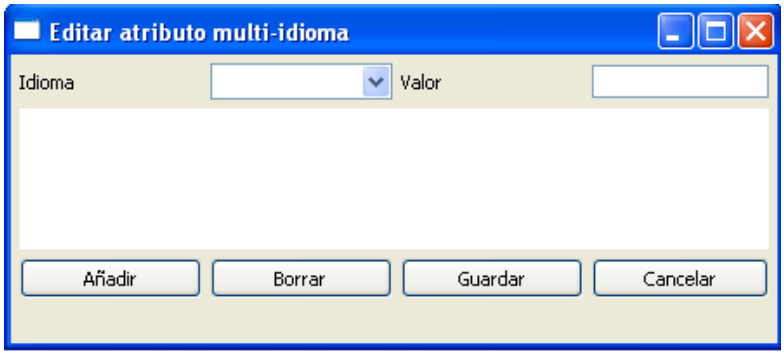


Figura 9.28. Formulario Evento. Nombre.

- Descripción: Opcionalmente, el usuario puede describir dicho evento en varios idiomas para una mayor clarificación. Para ello, el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

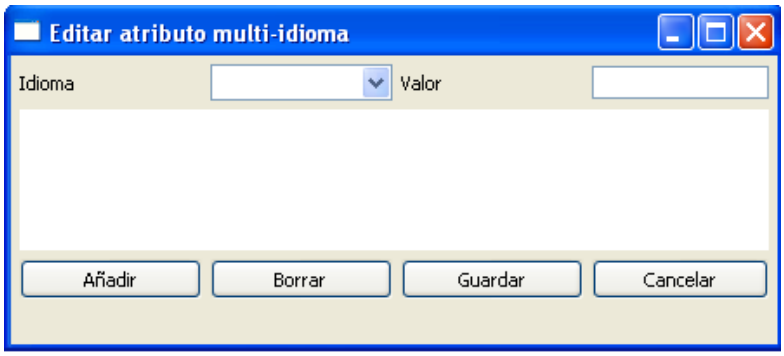


Figura 9.29. Formulario Evento. Descripción.

9.2.3.8. Componente

Los componentes son colecciones de datos, interfaz y lógica de negocio que el usuario podrá asociar a procedimientos, tareas y bloques. Las propiedades que el usuario puede editar para completar la definición de un componente son:

- Alias: En modo texto, el usuario define a través de este atributo un identificador univoco para el componente.

- Entrada requerida: El usuario especifica si el componente es requerido para que una tarea o bloque comience.

- Salida requerida: De forma similar al anterior atributo, se indica si el componente será generado una vez que la tarea o bloque haya sido completado.

- Descripción: Su definición puede ser multi-idioma. Para ello el usuario irá seleccionando idioma por idioma especificando para cada uno de ellos su traducción correspondiente.

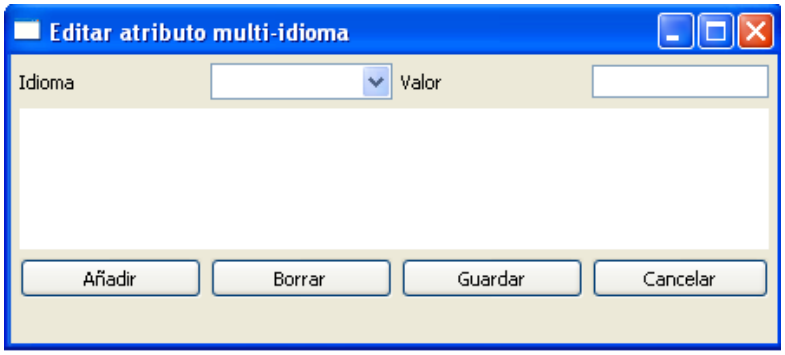


Figura 9.30. Formulario Componente. Descripción.

- Sección: Mediante este atributo, el usuario indica la sección de la página en la que se ubicará el componente. Para ello, deberá seleccionarla de entre todas las secciones definidas para el procedimiento.

- Icono: El usuario tiene la posibilidad de asociar un icono que represente al componente y permita identificarlo de manera clara y rápida. Para ello, deberá seleccionarlo de entre todos los disponibles.

- Condiciones de creación, visibilidad y acceso: De entre todas las condiciones definidas para el procedimiento, el usuario seleccionará aquellas relativas a la creación, visibilidad y acceso del componente.

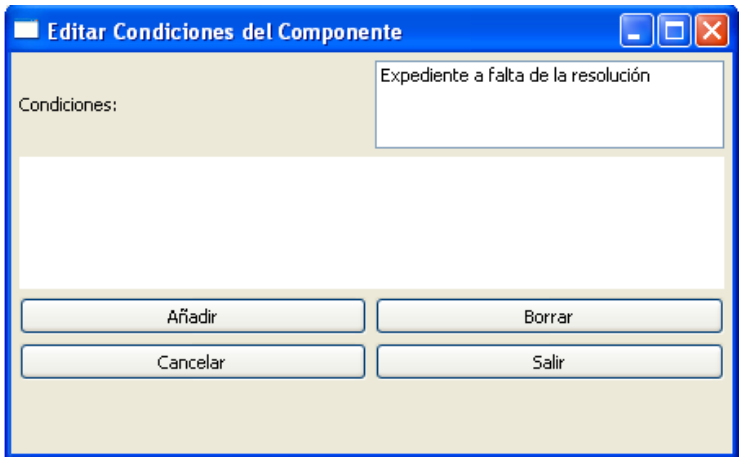


Figura 9.31. Formulario Componente. Condiciones.

- Campos: Un componente puede incorporar "n" campos. En el momento de la incorporación del componente al procedimiento el usuario sólo podrá modificar la etiqueta, la obligatoriedad y la visibilidad de cada uno de los campos.

Figura 9.32. Formulario Componente. Campos.

9.2.3.9. Flujos de secuencia y asociación

Ambos elementos se utilizan como mecanismo de conexión entre el resto de los elementos para definir cómo progresa el flujo dentro del procedimiento. Las propiedades que el usuario puede editar para completar la definición de estos elementos son:

Flujos de secuencia: Este tipo de flujo se utiliza para mostrar el orden en el que las tareas y los bloques se ejecutan. Tienen un solo origen y un solo destino.

- Id: En modo texto, el usuario define a través de este atributo un identificador univoco para el flujo.

- Condición: Un flujo de secuencia puede tener asociada una condición que el usuario deberá seleccionar de entre todas las definidas para el procedimiento.

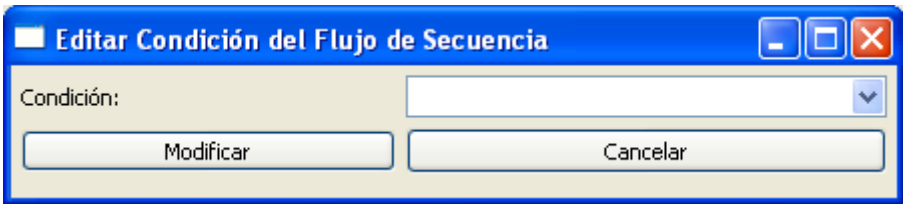


Figura 9.33. Formulario Flujo Secuencia. Condición.

Flujos de asociación de notas: El usuario tiene la posibilidad de incorporar notas y comentarios a los distintos elementos del procedimiento. Para asociar cada nota con su elemento se utilizan los flujos de asociación de notas.

- Id: En modo texto, el usuario define un identificador univoco para el flujo.

Flujos de asociación: Este tipo de flujo se utiliza para asociar documentos y componentes a tareas y bloques, ya sean de entrada o de salida.

- Id: En modo texto, el usuario define a través de este atributo un identificador univoco para el flujo.

10. CONCLUSIONES

10.1. INTRODUCCIÓN

Puesto que en apartados anteriores se ha explicado en qué consiste el proyecto, y se han definido sus funcionalidades básicas, en esta sección, se van a establecer las conclusiones básicas y los principales resultados obtenidos una vez que se ha finalizado el sistema, así como los problemas encontrados y las posibles mejoras y líneas futuras de investigación del proyecto.

A continuación, se muestran los puntos que se van a tratar en este capítulo:

- Resultados obtenidos.
- Problemas encontrados.
- Conclusiones.
- Mejoras y líneas futuras de investigación.

10.2. RESULTADOS OBTENIDOS

El resultado final del proyecto, ha sido muy satisfactorio, puesto que se han cumplido todos los objetivos previstos y expuestos en el primer capítulo de esta memoria.

Aunque el desarrollo de la aplicación ha sido costoso, se ha conseguido que el usuario tenga la sensación de que es una aplicación intuitiva, fácil de manejar y sobre todo útil, que era lo que se pretendía en un principio.

Se ha obtenido una aplicación modular, con el objetivo de que si en un futuro se quieren implementar e incorporar nuevos módulos no sea necesario modificar los módulos actuales de la aplicación. Es decir, que todo lo que se añada no afecte a lo que ya está implementado.

10.3. PROBLEMAS ENCONTRADOS

Durante el desarrollo del proyecto, se han ido encontrado por el camino una serie de problemas que afortunadamente se han ido subsanando a medida que se ha ido llevando a cabo la realización del mismo.

A continuación se exponen los problemas encontrados en la realización de este proyecto:

- Realizar modificaciones a los ficheros generados por defecto, para cumplir con las especificaciones iniciales.

- Generar nuevas imágenes en formato .svg para las figuras de la herramienta gráfica.

- Añadir código para la correcta visualización de los atributos de las figuras de la herramienta.

- Generar de nuevo todos los ficheros cada vez que se hace un cambio.

- Descubrir como hacer todos los cambios necesarios.

- Poca información sobre los principales conceptos técnicos del proyecto, EMF y GMF, y por lo tanto mucho tiempo dedicado en el estudio teórico de los mismos.

10.4. CONCLUSIONES

La experiencia personal adquirida durante la realización de este proyecto, ha sido especialmente enriquecedora.

Realmente resulta gratificante, ver que se ha llegado al final del proyecto, cumpliendo todos los objetivos trazados en un principio.

Durante el desarrollo del proyecto, se adquieren una serie de conocimientos, que hasta ahora resultaban desconocidos o muy novedosos, y que al final, se convierten en conceptos que no se olvidarán fácilmente.

En líneas generales, la realización de este proyecto ha supuesto una experiencia positiva y provechosa a nivel personal, aportándome una serie de conocimientos y experiencias inolvidables.

A continuación se exponen las conclusiones obtenidas tras la terminación del proyecto:

- El desarrollo de software dirigido por modelos, es un proceso tedioso debido a las transformaciones entre modelos.

- EMF y GMF son plug-ins de Eclipse muy útiles para la creación de modelos gráficos, pero se dispone de poca información sobre proyectos basados en éstos plug-ins, por lo que la transformación y sobre todo la adaptación de modelos a la herramienta generada ha sido costosa.

- Implementación difícil, hasta que se conoce el entorno de desarrollo.

10.5. MEJORAS Y LÍNEAS FUTURAS DE INVESTIGACIÓN

Por último cabe destacar un punto dedicado a futuras mejoras y evoluciones del proyecto, puesto que como es lógico en el mundo de la informática, un proyecto nunca se acaba totalmente.

Siempre hay alguna cosa que mejorar, ya sea porque se quieren añadir nuevos módulos o porque aparezcan nuevas tecnologías y versiones en el mercado, y como es lógico con el paso del tiempo, la tecnología o versión utilizada para el desarrollo del proyecto se quede obsoleta.

Por este motivo, si se quiere seguir dando funcionalidad a la aplicación, hay que actualizar la versión o la tecnología utilizada inicialmente, para adaptarla a tecnologías más recientes, todo esto claro, a largo plazo.

A corto plazo, se van a enumerar algunos de los puntos más importantes que se pueden mejorar:

- Generar un nuevo elemento en la herramienta de definición visual constituido por una gaveta de tareas modelo disponibles que esté contenido en la paleta de la aplicación. De modo que se pueda crear una nueva tarea modelo para el procedimiento haciendo un “drag and drop” desde esta gaveta a la pizarra.

- Añadir una nueva opción en la paleta de la aplicación para poder definir nuevas tareas modelo desde un icono de la gaveta de tareas modelo.

- Posibilidad de generar elementos en la pizarra a distintos niveles. Es decir, acceder o crear elementos en un nivel superior, pinchando en elementos existentes de un nivel anterior.

- Posibilidad de recuperar los datos de la pizarra grabados en ficheros XML con la norma BPMN para tratarlos con bases de datos, entrada para otras aplicaciones, etc.

Éstas son sólo algunas de las posibles mejoras y ampliaciones que se pueden realizar. Pero como se ha comentado antes, las posibilidades son infinitas, y la mejora de una aplicación informática puede no acabar nunca.

BIBLIOGRAFÍA

BIBLIOGRAFÍA WEB

[MDA]

http://www.omg.org/docs/omg/03-06-01.pdf

[MOF08]

http://www.omg.org/technology/documents/formal/mof.htm

[XML]

http://www.w3.org/XML/

[BPMN]

www.bpmn.org/

[ECLIPSE]

http://www.eclipse.org/

[EMF]

http://www.eclipse.org/modeling/emf/

[GMF]

http://www.eclipse.org/modeling/gmf/

[WIKIPEDIA WEB]

http://es.wikipedia.org/wiki/Gestión_de_proyectos

[IBM WEB]

http://www.software.ibm.com/ad/ocl

[LIFIA WEB]

http://www.lifia.info.unlp.edu.ar/papers/2003/Becker2003.pdf

[SISTEDES WEB]

http://www.sistedes.es/sistedes/pdf/2007/dsdm-07-bollati-revisionHerramientas.pdf

BIBLIOGRAFÍA

[DE AMESCUA 95]

De Amescua Seco, Antonio / García Sánchez, Luis / Martínez Fernández, Paloma / Díaz Pérez, Paloma;
Ingeniería del Software de Gestión. Análisis y diseño de aplicaciones
Madrid, Paraninfo, 1995

[KLEPPE 03]

Kleppe Anneke / Warmer Jos / Bast Wim;
MDA Explained: The Model Driven Architecture(TM): Practice and Promise
Addison-Wesley Professional, 2003.

[MOORE 04]

Moore William / Dean David / Gerber Anna / Wagenknecht Gunnar / Vanderheyden Philippe;
Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework"
IBM Redbooks, 2004.

[D'ANJOU 04]

D'Anjou Jim / Fairbrother Scott / Kehn Dan / Kellerman John / McCarthy Pat
Java Developer's Guide to Eclipse, The, 2nd Edition
Addison-Wesley Professional, 2004.

[STEINBERG 08]

Steinberg Dave / Budinsky Frank / Paternostro Marcelo/ Merks Ed
EMF: Eclipse Modeling Framework, 2nd Edition
Addison-Wesley Professional, 2008.