# Entorno de modelado para el desarrollo de sistemas de información

Proyecto Fin de Carrera

Julio de 2012

**Pedro Alexander Santana Santana**

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

**Proyecto Fin de Carrera:**

Nombre del PFC:

**Entorno de modelado para el desarrollo de Sistemas de Información**

Apellidos y nombre del alumno

**Santana Santana, Pedro Alexander**

Fecha

**Julio de 2012**

Tutores

**Ana María Placido Castro**

**José Juan Santana Hernández**

José Juan Hdez. Cabrera

# AGRADECIMIENTOS

*A mi familia por la paciencia,*

*los consejos y los ánimos que siempre me han dado.*

*A mis amigos, Jose, Fabio y Rayco, sin ellos*

*no lo hubiera conseguido.*

*A mis compañeros de trabajo, en especial*

*a Yeray por sus consejos sobre el desarrollo del software y*

*a Sergio por soportar todos los fallos de la aplicación.*

*Gracias a todos los que en algún momento habéis compartido*

*esta parte del camino conmigo.*

# CONTENIDO


*1. INTRODUCCIÓN ..... 13*

**1.1 Clasificación de los sistemas de información ..... 16**

**1.2 Evolución de los sistemas de información ..... 18**

**1.3 Los Sistemas Estratégicos de Información ..... 21**

**1.4 Desarrollo de los sistemas de información ..... 23**

**1.5 Objetivos ..... 30**

*2. ESTADO ACTUAL DEL ARTE ..... 31*

**2.1 Ingeniería dirigida por modelos ..... 31**

2.1.1 Model-Driven Architecture (MDA) ..... 32

**2.2 Herramientas para el desarrollo de Sistemas de Información ..... 33**

2.2.1 Herramientas CASE ..... 33

2.2.2 Lenguajes de cuarta generación (4GL) ..... 35

*3. METODOLOGÍA ..... 36*

*4. ANÁLISIS ..... 37*

**4.1 Análisis de requisito de usuarios ..... 37**

4.1.1 Modelo de dominio ..... 37

4.1.2 Lista de características ..... 38

4.1.2.1 Edición ..... 39

4.1.2.2 Vistas ..... 40

4.1.2.3 Depuración ..... 41

4.1.2.4 Publicación ..... 41

4.1.2.5 Control de errores ..... 42

4.1.2.6 Ayuda ..... 42

4.1.2.7 Configuración de la aplicación ..... 43

**4.2 Análisis de requisitos de software ..... 43**

4.2.1 Definición de casos de uso ..... 43

4.2.1.1 Desarrollar el Sistema de información. ..... 44

4.2.1.2 Mantener los S.I. ..... 48

4.2.1.3 Publicar S.I. ..... 51

*5. DISEÑO ..... 54*

**5.1 Diseño arquitectónico ..... 55**

**5.2 Puntos de extensión ..... 55**

5.2.1 Vistas ..... 56

5.2.2 Editores ..... 57

5.2.3 Perspectivas ... 58
5.2.4 Comandos y acciones ... 58
5.2.4.1 Comandos ... 59
5.2.4.2 Acciones ... 60
5.2.5 Asistentes ... 60
5.2.6 Generadores ... 60

**5.3 Diseño de la interfaz de usuario ... 62**
5.3.1 Vista general ... 62
5.3.2 Vista de explorador de proyectos ... 63
5.3.3 Vista de errores ... 64
5.3.4 Vista de tareas ... 65
5.3.5 Asistente de creación de proyecto ... 66
5.3.6 Asistente de creación de recursos ... 67

***6. DESARROLLO ... 68***

**6.1 Creación de plug-ins ... 69**

**6.2 Monet ... 71**
6.2.1 Estructura del proyecto ... 72

**6.3 Perspectiva ... 73**

**6.4 Editor ... 75**
6.4.1 Extensión del editor ... 76

**6.5 Asistentes y diálogos ... 77**
6.5.1 Asistente de creación de proyectos ... 78
6.5.2 Asistentes de creación de recursos ... 79

**6.6 Vistas ... 81**
6.6.1 Extensión de la vista ... 81
6.6.1.1 Declarar la categoría de una vista ... 81
6.6.1.2 Declarar una vista ... 82
6.6.1.3 Crear el comportamiento de la vista ... 82
6.6.2 Vista de dependencias ... 83
6.6.3 Explorador de proyectos ... 84
6.6.4 Vista de templates ... 84

**6.7 Generadores ... 85**
6.7.1 Monet Builder ... 86
6.7.2 Packaging Builder ... 87

**6.8 Marcadores ... 87**

**6.9 Naturalezas ... 89**
6.9.1 Declaración de la naturaleza ... 90
6.9.2 Asociar generadores y naturalezas ... 90

**6.10 Página de propiedades del proyecto ... 91**

**6.11 Publicación ... 91**

6.12 Sitio de actualización ..... 92
6.13 Proceso de instalación del entorno de modelado ..... 93
*7. CONCLUSIONES ..... 95*
*8. TRABAJO FUTURO ..... 97*
*9. ANEXO I ..... 98*
9.1 Eclipse ..... 98
9.1.1 Objetivos de la Plataforma Eclipse ..... 99
9.1.2 Arquitectura de la Plataforma Eclipse ..... 99
9.1.2.1 Plataforma de Ejecución (Platform Runtime) ..... 100
9.1.2.2 Workspace ..... 101
9.1.2.3 Workbench ..... 102
9.1.3 Arquitectura basada en plug-ins ..... 103
9.1.3.1 ¿Qué es un plug-in? ..... 103
9.1.3.2 Arquitectura y plug-in manifest ..... 105
9.1.3.3 Activación de los plug-ins ..... 106
*10. ANEXO II ..... 107*
10.1 Lenguaje de modelado ..... 107
10.1.1 Unidad de negocio y modelo de negocio ..... 107
10.1.2 Arquitectura de la información ..... 111
10.1.2.1 Nodo Formulario ..... 111
10.1.2.2 Nodo Colección ..... 111
10.1.2.3 Nodo contenedor ..... 112
10.1.2.4 Nodo Catálogo ..... 113
10.1.2.5 Nodo Documento ..... 113
10.1.2.6 Nodo Escritorio ..... 113
10.1.3 Mapas de trabajo ..... 114
10.1.3.1 Ejemplo de mapa de trabajo y diagrama de representación ..... 116
*11. BIBLIOGRAFÍA ..... 118*

# ÍNDICE DE ILUSTRACIONES


*Ilustración 1 - Sistema de Información de la Organización empresarial: funciones* 14
*Ilustración 2 - Núcleo competitivo de Porter* 22
*Ilustración 3 - Casos de uso principales* 43
*Ilustración 4 - Caso de uso: Desarrollar Sistema de Información* 44
*Ilustración 5 - Caso de uso: Mantener Sistema de Información* 48
*Ilustración 6 - Caso de uso: Publicar Sistema de Información* 51
*Ilustración 7 - Diseño Arquitectónico de la aplicación* 55
*Ilustración 8 - Clases ViewPart* 56
*Ilustración 9 - Clases EditorPart* 57
*Ilustración 10 - Punto de extensión de perspectivas* 58
*Ilustración 11 - Visión general de los comandos* 59
*Ilustración 12 - Punto de extensión para los builders* 61
*Ilustración 13 - Distribución general del entorno de modelado* 62
*Ilustración 14 - Diseño del explorador de proyectos* 63
*Ilustración 15 - Diseño de la vista de errores* 64
*Ilustración 16 - Diseño de la vista de tareas* 65
*Ilustración 17 - Diseño de asistente de creación de proyectos* 66
*Ilustración 18 - Diseño del asistente de creación de recursos* 67
*Ilustración 19 - Detalle de la vista* 70
*Ilustración 20 -Vista producida por la ejecución del plug-in "Hello World"* 70
*Ilustración 21 - Arquitectura de Monet* 71
*Ilustración 22 - Estructura de un proyecto de Monet* 72
*Ilustración 23 - Detalle de la perspectiva* 74
*Ilustración 24 - Detalle del editor* 76
*Ilustración 25 - Detalle de la extensión del editor y sus propiedades* 77
*Ilustración 26 - Asistente de creación de proyectos* 78
*Ilustración 27 - Asistente de creación de formularios* 80
*Ilustración 28 - Detalle de un fichero de tipo form* 80
*Ilustración 29 - Detalle de la declaración de la categoría Monet* 82
*Ilustración 30 - Vista de dependencias* 83
*Ilustración 31 - Explorador de proyectos de Monet* 84
*Ilustración 32 - Detalle de la vista de plantillas* 85
*Ilustración 33 - Pipeline del builder de Monet* 86
*Ilustración 34 - Marcadores en el editor* 88
*Ilustración 35 - Detalle de la vista de problemas* 88
*Ilustración 36 - Detalle de error en la vista de proyectos* 88
*Ilustración 37 - Detalle del fichero de proyecto* 89
*Ilustración 38 - Detalles de la naturaleza de Monet* 90
*Ilustración 39 - Detalle de la naturaleza y los builders asociados* 90
*Ilustración 40 - Página de propiedades del proyecto* 91
*Ilustración 41 - Error durante la publicación del modelo* 91
*Ilustración 42 - Editor del sitio de actualización* 92
*Ilustración 43 - Producto construido* 93
*Ilustración 44 - Diálogo de instalación de nuevo software en Eclipse* 93

*Ilustración 45 - Monet Modelling Environment URL* 94
*Ilustración 46 - Instalación del entorno de modelado* 94
*Ilustración 47 - Arquitectura de la plataforma Eclipse* 100
*Ilustración 48 - Árbol de recursos del Workspace* 101
*Ilustración 49 - Estructura de la ventana del Workbench* 103
*Ilustración 50 - Estructura básica de un plug-in.* 104
*Ilustración 51 - Esquema de la extensión de un plug-in mediante otro plug-in* 105
*Ilustración 52 - Unidades de negocio dependientes* 108
*Ilustración 53 - Relación entre el modelo de negocio y las definiciones* 108
*Ilustración 54 - Diferentes niveles de lenguajes en Monet* 109
*Ilustración 55 - Relación entre una colección y los nodos contenidos* 112
*Ilustración 56 - Relación entre un contenedor y el nodo que contiene* 112
*Ilustración 57 - Ejemplo de representación de un mapa de trabajo* 117

# ÍNDICE DE TABLAS

*Tabla 1 - Tipología de Sistemas de información* ..... 16
*Tabla 2 - Etapas de la evolución de los sistemas de información* ..... 19
*Tabla 3 - Comparación de enfoques de desarrollo de sistemas* ..... 28
*Tabla 4- Ventajas y desventajas de las herramientas CASE* ..... 34
*Tabla 5 - Lista de características: Edición* ..... 39
*Tabla 6 - Lista de características: Vistas* ..... 40
*Tabla 7 - Lista de características: Depuración* ..... 41
*Tabla 8 - Lista de características: Publicación* ..... 41
*Tabla 9 - Lista de características: Control de errores* ..... 42
*Tabla 10 - Lista de características: Ayuda* ..... 42
*Tabla 11 - Lista de características: Configuración* ..... 43
*Tabla 12 - Caso de uso: Crear Proyecto* ..... 45
*Tabla 13 - Caso de uso: Importar proyectos* ..... 45
*Tabla 14 - Caso de uso: Crear ficheros* ..... 46
*Tabla 15 - Caso de uso: Editar ficheros* ..... 46
*Tabla 16 - Caso de uso: Compilar Proyecto* ..... 47
*Tabla 17 - Caso de uso: Configurar proyecto* ..... 47
*Tabla 18 - Caso de uso: Gestionar versiones* ..... 49
*Tabla 19 - Caso de uso: Visualizar incidencias* ..... 49
*Tabla 20 - Caso de uso: Añadir incidencias* ..... 50
*Tabla 21 - Caso de uso: Resolver incidencias* ..... 50
*Tabla 22 - Caso de uso: Configurar destino* ..... 51
*Tabla 23 - Caso de uso: Empaquetar proyecto* ..... 52
*Tabla 24 - Caso de uso: Depurar proyecto* ..... 52
*Tabla 25 - Caso de uso: Desplegar proyecto* ..... 53

# 1. INTRODUCCIÓN

Durante los últimos años los sistemas de información constituyen uno de los principales ámbitos de estudio en el área de organización de empresas. El entorno donde las compañías desarrollan sus actividades se vuelve cada vez más complejo. La creciente globalización, el proceso de internacionalización de la empresa, el incremento de la competencia en los mercados de bienes y servicios, la rapidez en el desarrollo de las tecnologías de información, el aumento de la incertidumbre en el entorno y la reducción de los ciclos de vida de los productos originan que la información se convierta en un elemento clave para la gestión, así como para la supervivencia y crecimiento de la organización empresarial. Si los recursos básicos analizados hasta ahora eran tierra, trabajo y capital, ahora la información aparece como otro elemento fundamental a valorar en las empresas.

A la hora de definir un sistema de información existe un amplio abanico de definiciones[1]. Tal vez la más precisa sea la propuesta por Andreu, Ricart y Valor [1]:

> *Conjunto formal de procesos que, operando sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuyen selectivamente la información necesaria para la operación de dicha empresa y para las actividades de dirección y control correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo a su estrategia.*

Todo sistema de información utiliza como materia prima los datos, los cuales almacena, procesa y transforma para obtener como resultado final información, la cual será suministrada a los diferentes usuarios del sistema, existiendo además un proceso de retroalimentación o "feedback" en la cual se ha de valorar si la información obtenida se adecua a lo esperado.

[1] Otra definición de sistema de información serían las propuestas por K y J Laudon (1996), para los cuales un "sistema de información es aquel conjunto de componentes interrelacionados que capturan, almacenan, procesan y distribuyen la información para apoyar la toma de decisiones, el control, análisis y visión de una organización".

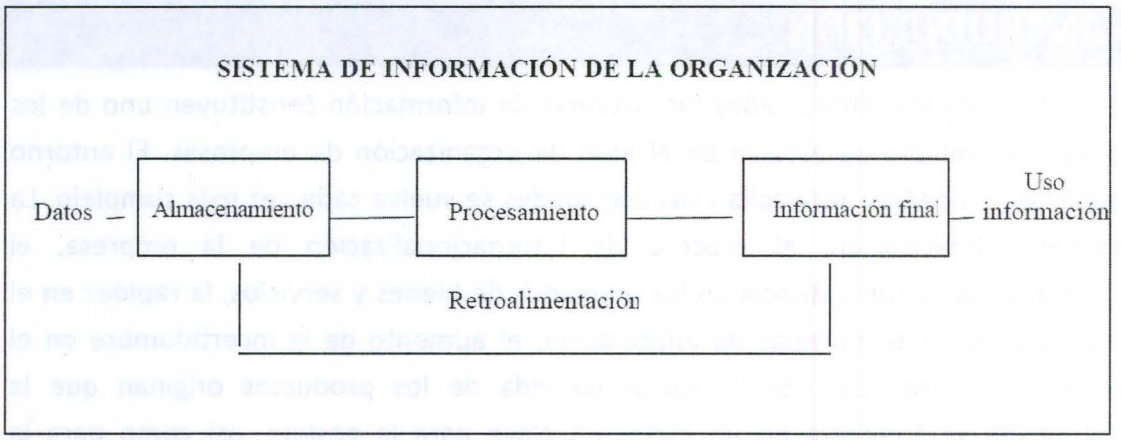


Ilustración 1 - Sistema de Información de la Organización empresarial: funciones

Junto con los datos, los otros dos componentes básicos que constituyen un sistema de información son los usuarios (personal directivo, empleados y en general cualquier agente de la organización que utilice la información en su puesto de trabajo) y los equipos (informáticos, software, hardware y tecnologías de almacenamiento de la información y de las telecomunicaciones).

En muchas ocasiones existe bastante confusión, pues al referirse a sistemas de información se piensa en un primer momento tanto en los ordenadores como en los programas informáticos. Una empresa puede adquirir nuevos ordenadores, instalar nuevos productos de telecomunicaciones, elaborar una página web, realizar comercio electrónico, pero ello no implica que exista en su organización un sistema de información. Un sistema de información abarca más que el aspecto meramente computacional, pues no sólo hemos de tener en cuenta estas herramientas, sino también el modo de organizar dichas herramientas y de obtener la información necesaria para el correcto funcionamiento de la empresa.

El personal responsable de elaborar el sistema de información de una empresa ha de poseer conocimientos tanto de las tecnologías de información disponible y que pueden utilizarse en la empresa, como del modo de organizarlas. Para ello en primer lugar tendrán que conocer la estrategia de la organización y el tipo de organización para posteriormente establecer las necesidades de información y adquirir las herramientas necesarias para el desarrollo del sistema de información.

Todo sistema de información va a poseer unos objetivos principales, los cuales se resumen a continuación:

- **Apoyar los objetivos y estrategias de la empresa:** el sistema de información ha de suministrar a la organización toda la información necesaria para su correcto funcionamiento. La información manejada

abarcará desde la actividad rutinaria de la empresa hasta aquella necesaria para el proceso de planificación a largo plazo de la empresa.

- **Proporcionar información para el control de la totalidad de actividades de la empresa,** pudiendo comprobar el cumplimiento de las metas establecidas por la organización. Los sistemas de información abarcan a todos los departamentos de la empresa y a la gestión global de la organización.
- **Adaptar las necesidades de información a la evolución de la empresa:** conforme la empresa va creciendo y desarrollándose, surgen nuevas necesidades de información que han de ser satisfechas por el sistema de información, evolucionando este último adecuándose a las nuevas circunstancias del entorno.
- **Interactuar con los diferentes agentes de la organización,** permitiendo que estos empleen el sistema de información para satisfacer sus necesidades de un modo rápido y eficaz. La interactividad y flexibilidad de los sistemas de información constituyen un punto clave en el éxito o fracaso.

Para la consecución de dichos objetivos, un buen sistema de información ha de ser capaz de recibir y procesar los datos del modo más eficaz y sin errores, suministrar los datos en el momento preciso, evaluar la calidad de los datos de entrada, eliminar la información poco útil evitando redundancias, almacenar los datos de modo que estén disponibles cuando el usuario lo crea conveniente, proporcionar seguridad evitando la perdida de información o la intrusión de personal no autorizado o agentes externos a la compañía y generar información de salida útil para los usuarios de sistemas de información, ayudando en el proceso de toma de decisiones.

## 1.1 CLASIFICACIÓN DE LOS SISTEMAS DE INFORMACIÓN

A la hora de clasificar los Sistemas de Información, existe una gran variedad de criterios. En la Tabla 1 podemos ver algunas de las principales tipologías de sistemas de información que nos podemos encontrar:

| Tipo de Sistema de Información | Tipos |
|---|---|
| **Grado de formalidad** | • Formales<br>• Informales |
| **Automatización** | • Manuales<br>• Informáticos |
| **Relación con la toma de decisiones** | • Estratégicos (alta dirección)<br>• Gerencial (nivel intermedio)<br>• Operativos (control operativo) |
| **Funcionalidad** | • Gestión comercial<br>• Gestión contable<br>• Gestión financiera<br>• Gestión de Recursos Humanos<br>• Gestión de la Producción |
| **Grado de especialización** | • Específicos<br>• Generales |

Tabla 1 - Tipología de Sistemas de información

Sin embargo la clasificación más útil es la propuesta por K y J Laudon [4]. En ella los sistemas de información se agrupan según su utilidad en los diferentes niveles de la organización empresarial. La organización consta de 4 niveles básicos: un nivel operativo referido a las operaciones diarias de la organización, un nivel del conocimiento que afecta a los empleados encargados del manejo de la información (generalmente el departamento de informática), un nivel administrativo (abarcaría a los gerentes intermedios de la organización) y un nivel estratégico (la alta dirección de la empresa).

Según estos niveles, K y J Laudon establecen la siguiente clasificación de sistemas de información:

a) **Sistemas de Procesamiento de Operaciones (SPO):** sistemas informáticos encargados de la administración de aquellas operaciones

diarias de rutina necesarias en la gestión empresarial (aplicaciones de nóminas, seguimiento de pedidos, auditoría, registro y datos de empleados). Estos sistemas generan información que será utilizada por el resto de sistemas de información de la compañía siendo empleados por el personal de los niveles inferiores de la organización (Nivel Operativo).

b) **Sistemas de Trabajo del Conocimiento (STC):** aquellos sistemas de información encargados de apoyar a los agentes que manejan información en la creación e integración de nuevos conocimientos para la empresa (estaciones de trabajo para la administración); forman parte del nivel de conocimiento.

c) **Sistemas de automatización en la oficina (SAO):** sistemas informáticos empleados para incrementar la productividad de los empleados que manejan la información en los niveles inferiores de la organización (procesador de textos, agendas electrónicas, hojas de cálculo, correo electrónico, etc.); se encuentran encuadrados en el nivel de conocimiento al igual que los STC.

d) **Sistemas de información para la administración (SIA):** sistemas de información a nivel administrativo empleados en el proceso de planificación, control y toma de decisiones proporcionando informes sobre las actividades ordinarias (control de inventarios, presupuesto anual, análisis de las decisiones de inversión y financiación). Son empleados por la gerencia y directivos de los niveles intermedios de la organización.

e) **Sistemas para el soporte de decisiones (SSD):** sistemas informáticos interactivos que ayudan en los distintos usuarios en el proceso de toma de decisiones, a la hora de utilizar diferentes datos y modelos para la resolución de problemas no estructurados (análisis de costes, análisis de precios y beneficios, análisis de ventas por zona geográfica). Son empleados por la gerencia intermedia de la organización.

f) **Sistemas de Soporte Gerencial (SSG):** sistemas de información a nivel estratégico de la organización diseñados para tomar decisiones estratégicas mediante el empleo de gráficos y comunicaciones avanzadas. Son utilizados por la alta dirección de la organización con el fin de elaborar la estrategia general de la empresa (planificación de ventas para 4 años, plan de operaciones, planificación de la mano de obra).

Todos estos sistemas de información a su vez podrían analizarse según las diferentes áreas de la empresa: ventas y mercadotecnia, manufactura y producción, finanzas, contabilidad y recursos humanos. Para cada una de estas áreas existe un conjunto específico de aplicaciones informáticas y equipos, los cuales han de estar coordinados entre sí. Si ello no se realizara, una empresa tendrá problemas de intercambio de datos entre las diferentes áreas, aparecerá la existencia de redundancia de datos y la existencia de ineficiencias e incrementos de costes de comunicación. Por ello resulta clave la correcta planificación y desarrollo de los sistemas de información.

## 1.2 EVOLUCIÓN DE LOS SISTEMAS DE INFORMACIÓN

Los Sistemas de Información han ido evolucionando durante los últimos años hasta constituir los denominados sistemas de información estratégicos. Primeramente los Sistemas de Información empresariales eran considerados como un instrumento simplificador de las distintas actividades de la empresa, una herramienta con la cual se facilitaban los tramites y reducía la burocracia. Su finalidad era básicamente llevar la contabilidad y el procesamiento de los documentos a nivel operativo.

Posteriormente el desarrollo de la informática y las telecomunicaciones permitieron incrementar la eficacia en la realización de las tareas, ahorrar tiempo en el desarrollo de las actividades y almacenar la mayor cantidad de información en el menor espacio posible, lo cual aumentó en las organizaciones el interés en los sistemas de información. Con el transcurrir del tiempo las empresas fueron observando cómo las tecnologías y sistemas de información permitían a la empresa obtener mejores resultados que sus competidores, constituyéndose por sí mismas como una fuente de ventaja competitiva y una poderosa arma que permitía diferenciarse de sus competidores y obtener mejores resultados que estos. De este modo los sistemas de información se constituyeron como una de las cuestiones estratégicas de la empresa, que ha de considerarse siempre en todo proceso de planificación empresarial.

Dada la clasificación de K y J Laudon, los primeros sistemas de información en desarrollarse fueron los Sistemas de Procesamiento de operaciones. Con el transcurrir del tiempo, fueron apareciendo en primer lugar los sistemas de información para la administración y finalmente los sistemas de apoyo a las decisiones así como los sistemas estratégicos. Se produjo un desarrollo vertical de los sistemas de información, partiendo de los niveles inferiores de la organización hasta abarcar al equipo directivo de la empresa.

A la hora de analizar el progreso de los sistemas de información, uno de los trabajos fundamentales fue el propuesto por Gibson y Nolan. Ellos describieron la evolución de los sistemas de información basándose en la evolución de las tecnologías de información (Ver Tabla 2). En la medida en que se desarrollaron los equipos informáticos, el software, el hardware, las bases de datos y las telecomunicaciones, los sistemas de información fueron adquiriendo una mayor relevancia en las organizaciones, empezándose a considerar como un elemento más del proceso de planificación.

| Etapas de la Evolución de los sistemas de información | Características |
|---|---|
| **Iniciación** | • Introducción de la informática en la empresa<br>• Aplicaciones informáticas orientadas a la mecanización y automatización de los procesos ordinarios.<br>• Escaso gasto en informática y escasa formación del personal |
| **Contagio** | • La aplicación de las tecnologías de información originan resultados espectaculares.<br>• Difusión de las tecnologías de información en todas las áreas de la empresa.<br>• Aumenta la cualificación del personal<br>• Existe gran descoordinación y poca planificación en el desarrollo de lo sistemas de información |
| **Control** | • La alta dirección de la organización se preocupa de los sistemas de información como consecuencia del alto coste en ellos.<br>• Centralización de los proyectos de inversión en tecnologías de información. |
| **Integración** | • Se controla el incremento del gasto<br>• Se produce la integración de los sistemas de información existentes er las distintas áreas de la empresa.<br>• Mejora y perfeccionan los sistemas de información. |
| **Administración de la información** | • El sistema de información adquiere una dimensión estratégica en la empresa.<br>• Descentralización de ciertas aplicaciones informáticas |
| **Madurez** | • Desarrollo de los Sistemas de Información en los niveles superiores de la organización apareciendo los Sistemas Estratégicos de información.<br>• Adquiere gran importancia la creatividad y la innovación. |

Tabla 2 - Etapas de la evolución de los sistemas de información

Esta clasificación de la evolución de los sistemas de información pueden agruparse en 4 grandes etapas, tal como establecen Andreu, Ricart y Valor [1]:

a) **Introducción de la informática en la organización:** los sistemas de información se aplicaban para simplificar y automatizar los procesos administrativos. Se usan las computadoras y los sistemas informáticos para mejorar el proceso de contabilidad, elaborar nóminas y facturación buscando sobre todo el ahorro de costes y tiempo en la realización de dichas operaciones. Existe una carencia de formación por parte de los empleados de la organización en dichos sistemas y no hay profesionales que puedan resolver dichos problemas dentro de la compañía.

b) **Etapa de contagio de las aplicaciones informáticas:** tras observar como la aplicación de los sistemas informáticos en algunas áreas de la empresa originan importantes mejoras, estos se van difundiendo por los diferentes departamentos de la empresa. Dicho "contagio" se desarrolla sin ninguna planificación por parte de la organización, con lo cual se produce un alto incremento de los costes. Aumenta la formación del personal en las tecnologías de la información y en las aplicaciones informáticas, existiendo ya en la organización personal capaz de solucionar los problemas planteados en el manejo del sistema de información.

c) **Coordinación de los Sistemas de Información y los objetivos de la empresa:** los sistemas de información son utilizados en la totalidad de la organización y ya son tenidos en cuenta por parte de la dirección como un elemento fundamental de la empresa. Se empiezan a elaborar procedimientos de planificación de los sistemas de información y aparece la necesidad de usar los sistemas de de información como un medio de cumplimiento de los objetivos de la empresa.

d) **Aparición de los Sistemas Estratégicos de información:** los sistemas de información son valorados como una fuente de ventaja competitiva sostenible, de tal modo que al elaborar la estrategia general de la compañía se establece la planificación y desarrollo de los sistemas de información como otros de los aspectos clave dentro del proceso directivo.

## 1.3 Los Sistemas Estratégicos de Información

En la última etapa de evolución, los sistemas de información constituyen los denominados Sistemas Estratégicos de Información. Monforte [5] define sistema estratégico de información como:

> *El dominio y control, por parte de una empresa, de una característica, habilidad, recurso o conocimiento que incrementa su eficiencia y le permite distanciarse de los competidores*

K y J Laudon a su vez definen sistemas estratégicos de información como:

> *Aquel sistema de información que forma parte del "ser" de la empresa, bien porque supone una ventaja competitiva por sí mismo, bien porque está unido de una forma esencial al negocio y aporta un atributo especial a los productos, operaciones o toma de decisiones*

De ambas definiciones podemos destacar el concepto de "ventaja competitiva", relacionado directamente con la estrategia de la empresa. La ventaja competitiva de una empresa se entiende como aquella característica de una empresa que la diferencia del resto de competidores colocándola en una posición relativa superior para competir.

Bueno y Morcillo [6] la definen como:

> *Sistemas computacionales a cualquier nivel en la empresa que cambian las metas, operaciones, servicios, productos o relaciones del medio ambiente para ayudar a la institución a obtener una ventaja competitiva*

Dicha posición de superioridad sobre los competidores ha de ser sostenible en el tiempo, pues solo así se lograrán los resultados para la organización.

Así un sistema de información permitiría a una organización obtener mejores resultados que el resto de agentes de la economía. La empresa se beneficiaría de una reducción de costes en la fabricación del producto, reducción del coste de comunicación entre las diferentes áreas de la empresa, mejor coordinación entre los

diferentes niveles jerárquicos de la empresa, una mejor conectividad con proveedores y clientes, rápida adaptación a las necesidades del consumidor, disminución del tiempo de entrega del producto, etc. De este modo se reforzaría la posible estrategia seguida por la empresa, por ejemplo las planteadas por Porter [7]: liderazgo en costes, diferenciación del producto y concentración.

Aquellas organizaciones que no valoren los sistemas de información como un elemento estratégico, o aunque los tengan presentes no lo desarrollen de una forma coherente con su estrategia, se enfrentarán a una gran diversidad de problemas: los competidores, proveedores y clientes pueden incrementar su poder a la hora de negociar con la empresa, aparece el establecimiento de objetivos empresariales inalcanzables con los sistemas de información actualmente disponibles en la empresa, surge duplicidad de esfuerzo, inexactitud de los sistemas, gestión inadecuada de la información, mala elección de las tecnologías de la información, etc. De este modo los sistemas estratégicos de información permiten a la empresa sobrevivir en entornos altamente competitivos y lograr un crecimiento de la organización. Una organización puede plantearse el modelo de fuerzas competitivas de Porter, donde la empresa relaciona las amenazas y oportunidades que puede encontrarse con los agentes externos y actuar en consecuencia.

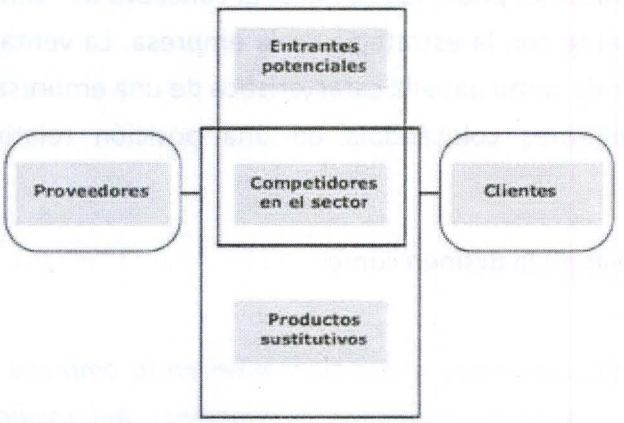


Ilustración 2 - Núcleo competitivo de Porter

Partiendo de dicho modelo, los sistemas de información nos servirían para poder competir y obtener unos mejores resultados:

a) **Amenaza de nuevos participantes:** los sistemas de información permitirían mejorar el control de los canales de distribución y aprovisionamiento para limitar el acceso de los competidores. Igualmente pueden utilizarse los sistemas de información para adecuar

mejor nuestros productos a las necesidades del cliente, explotar economías de escala con el fin de reducir precios y competir ante una posible guerra de precios con las empresas adversarias, siendo más agresivos en la estrategia de liderazgo en costes.

b) **Amenaza de productos o servicios sustitutivos:** se puede conseguir una mejor adaptación a las necesidades del cliente, establecer una correcta adecuación de calidad-precio de nuestro, identificar más fácilmente las necesidades insatisfechas de los consumidores y redefinir los segmentos del mercado.

c) **Rivalidad con los competidores actuales:** los sistemas de información constituyen un arma que permite reducir costes o mejorar la imagen de marca de la compañía.

d) **Poder negociador de clientes y proveedores:** los sistemas de información pueden ser utilizados como un medio para equilibrar el poder con los proveedores y los compradores de nuestro producto.

La empresa puede conseguir la ventaja competitiva bien diferenciando el producto (tratando de crear lealtad a la marca, intentando que el producto de la compañía cree más valor en el consumidor), utilizando estrategia de liderazgo en costes (los sistemas de información permiten disminuir los costes de comunicación e intercambio de la información dentro de las organizaciones, siendo evaluado dicho coste tanto en tiempo como en dinero) o bien utilizando una estrategia de concentración (utilizando una estrategia similar a las anteriores pero concentrándonos en un segmento concreto del mercado).

## 1.4 DESARROLLO DE LOS SISTEMAS DE INFORMACIÓN

La consecución de una ventaja competitiva utilizando los sistemas de información dependerá en gran medida del correcto desarrollo y puesta en funcionamiento del sistema de información. El desarrollo de un sistema de información no resulta sencillo. Aquellas organizaciones que simplemente adquieren tecnologías de información sin tener en cuenta las necesidades existentes en la compañía fracasarán, poniendo en peligro la supervivencia de la empresa. Por ello resulta fundamental los pasos a seguir en el desarrollo de los sistemas de información.

El proceso de desarrollo de los sistemas de información constaría de siete etapas fundamentales.

1. **Definición del proyecto:** en esta etapa se determinaría si la empresa presenta problemas y como esto pueden solucionarse mediante la implantación de un sistema de información. En ella se identificarán cuales son los objetivos del uso de los sistemas de información y como estos se ubican dentro de la estrategia global de la compañía. En esta fase resulta fundamental que la alta dirección considere los sistemas de información como un arma estratégica y crea realmente en ello.
2. **Análisis de sistemas:** Tras haber identificado los diferentes problemas de la organización, estos serán analizados más detenidamente, identificando las causas que lo originan y planteando diversas soluciones. En esta fase se producirá un estudio de factibilidad, para ver si las soluciones son posibles dados los recursos que posee la organización. Hablaremos de tres tipos de factibilidad:
    a. *Factibilidad técnica:* para que la solución pueda ser implantada hemos de analizar si la empresa posee los medios informáticos adecuados, o bien si hemos de adquirirlos del exterior.
    b. *Factibilidad económica:* se realizara un estudio y valoración económica de la solución comprobando que los beneficios de la mejora superan los costes de la implantación o modificación de los sistemas de información.
    c. *Factibilidad operativa:* hemos de valorar si la solución propuesta es deseable dada la organización interna de la empresa.
3. **Diseño de Sistemas:** una vez elegida aquella solución que resuelva los problemas, se detallará cómo el sistema de información satisface los requisitos planteados por la organización. A la hora de diseñar los sistemas, hemos de indicar que componentes de los sistemas de información utilizaremos (nivel hardware, software y tecnología de las telecomunicaciones) y como se relacionarán dichos componentes entre sí. De esta forma se producirá las especificaciones del sistema de información.
4. **Programación:** se traducirán las especificaciones del sistema desarrolladas en la etapa anterior, llevándose a cabo la programación y el desarrollo del software.
5. **Fase de pruebas:** para evaluar el correcto funcionamiento del sistema de información será necesario llevar a cabo un proceso exhaustivo y profundo para determinar si el sistema de información funciona en diversas condiciones y si los resultados se corresponden con lo que se esperaba. A la hora de establecer las pruebas, las empresas pueden realizarlas de tres tipos:

a. *pruebas de programas:* los diversos programas elaborados se probaran o separado, con el fin de garantizar que cada uno de ellos está libre de errores.
b. *pruebas del sistema:* se probará el sistema de información como un todo. La finalidad será comprobar el correcto funcionamiento del sistema en conjunto, puesto que en ocasiones puede que los programas funcionen correctamente de forma individual, pero a la hora de funcionar en conjunto el sistema de información no ofrezca los resultados esperados por la empresa.
c. *pruebas de aceptación:* pruebas realizadas por los usuarios finales del sistema de información. Cuando estos dan el visto bueno se proporciona la certificación final del correcto funcionamiento del sistema de información.

6. **Conversión:** una vez comprobado que el sistema de información funciona correctamente, se llevará a cabo la implantación de este, o bien la sustitución del antiguo sistema de información por el nuevo. A la hora de llevar a cabo esta conversión, las organizaciones van a poder optar por diversas estrategias:
   a. *la denominada estrategia en paralelo.* En ella durante un periodo de tiempo ambos programas van a convivir y utilizarse, funcionando tanto el nuevo sistema de información como el antiguo. Esta estrategia es la más fiable y segura, aunque sin embargo es la más costosa y podemos obtener información redundante.
   b. *se puede optar por un cambio directo,* remplazando el viejo sistema de información por el nuevo en una fecha determinada. Esta estrategia sería menos costosa, sin embargo ante cualquier problema que surja se puede originar la paralización de la actividad de la empresa. Igualmente requiere que el personal de la compañía haya recibido formación sobre el nuevo sistema de información, si no nos encontramos con empleados incapaces de saber manejar el nuevo sistema de información.
   c. *Llevar a cabo una experiencia piloto,* donde el nuevo sistema de información se utiliza en un área limitada de la compañía y tras comprobar su correcto funcionamiento se instala en la totalidad de la empresa. Sería una mezcla de las dos estrategias anteriores.
7. **Producción y mantenimiento:** una vez instalado el nuevo sistema de información se dice que el sistema está en producción. A partir de aquí

existe un proceso constante de evaluación del sistema de información por parte de los usuarios y personal especializado. Tras ello se identificaran nuevos errores y se planteará la corrección de estos.

La totalidad de las fases analizadas constituirán el denominado ciclo de vida de los sistemas de información. Sin embargo para muchas compañías desarrollar el sistema de información siguiendo la totalidad de las etapas anteriores puede resultarle muy costoso tanto en tiempo como en dinero. Otros inconvenientes vendrían dados por los continuos cambios de los requisitos de la información que puede originar que un sistema de información quede obsoleto incluso en la etapa de desarrollo. Por ello las empresas a la hora de desarrollar un sistema de información puede optar por otro conjunto de estrategias que le pueden permitir obtener resultados tan positivos como los conseguidos utilizando el ciclo de vida de los sistemas de información.

Otras posibles estrategias a adoptar por las empresas serían las siguientes:

- **Elaboración de prototipos:** la empresa desarrolla un sistema de información no funcional, el cual será una versión preliminar del sistema de información total. La elaboración de un prototipo supone la reducción de las etapas seguidas en el ciclo de vida de sistemas, buscando la rapidez en el desarrollo y la reducción de costes tanto en tiempo como en dinero. Los prototipos son evaluados por los empleados en su puesto de trabajo y se van continuamente adaptando a las necesidades de estos. Una vez que se comprueba su correcto funcionamiento el prototipo se irá extendiendo por el resto de áreas de la empresa.

  Indicar que el principal problema del uso de prototipos es su superficialidad, pues en muchas ocasiones la empresa se centra en la reducción de costes olvidándose de las necesidades de información de la empresa, la falta de elaboración de documentación sobre el funcionamiento del prototipo y la ausencia de pruebas para comprobar el correcto funcionamiento del prototipo. Suelen utilizarse en organizaciones pequeñas o con bajas necesidades de información; en grandes organizaciones no son muy recomendables.
- **Paquetes de software de aplicaciones:** adquisición por parte de la empresa de paquetes de software de aplicaciones ya existentes en el mercado y que utilizará para manejar la información. Resulta una solución muy sencilla para las organizaciones, pues la empresa simplemente adquiere el programa y lo instala en la organización. Los paquetes de software suelen aplicarse a una gran variedad de áreas de

la empresa y son muy útiles cuando la empresa no dispone del suficiente capital para poder desarrollar por ella misma el sistema de información. Sin embargo el principal inconveniente sería la ausencia de flexibilidad de estos para adaptarse a las necesidades específicas de la empresa. Ha de valorarse pues cuestiones tales como los recursos que posee la empresa, las funciones del software, el esfuerzo de instalación y mantenimiento del paquete de software, el coste y la facilidad de manejo de este.

- **Desarrollo por los usuarios finales:** el desarrollo de las aplicaciones informáticas durante los últimos años, permite que sean los propios usuarios finales quienes elaboren y desarrollen sus propios sistemas de información, existiendo una escasa participación por parte de los especialistas técnicos. Esta solución permite un mayor control del sistema por parte de los usuarios así como el ahorro en coste. Sin embargo, los principales inconvenientes serían la excesiva proliferación de sistemas de información sin control, el no-cumplimiento de unos mínimos de calidad y la falta de valoración de la organización desde un punto de vista global.
- **Subcontratación de los sistemas de información:** la empresa decide contratar empresas externas para que desarrollen los sistemas de información de la organización. Las empresas se beneficiarían del aprovechamiento de economías de escala por parte del proveedor, se asegurarían la calidad en el servicio, no existiría incertidumbre en los costes y la adaptación a las necesidades de las empresas sería más adecuada. Por otro lado subcontratar supone una cierta pérdida de control por parte de las empresas, siendo clave el poder negociador con el proveedor de los servicios informáticos. Igualmente, información considerada como estratégica para la organización es conocida por organizaciones ajenas a la empresa, surgiendo además una dependencia del proveedor.

En la siguiente tabla comparamos los diferentes enfoques de desarrollo de sistemas:

| Enfoque | Características | Ventajas | Desventajas |
|---|---|---|---|
| **Ciclo de vida de sistemas** | • Secuencial<br>• Realización de un proceso formal.<br>• Especificaciones y aprobaciones por escrito<br>• Los usuarios tienen un papel limitado | • Necesario para Sistemas y proyectos muy complejos y grandes | • Lento y costoso<br>• No estimula los cambios en la organización<br>• Se ha de elaborar mucha documentación. |
| **Elaboración de prototipos** | • Requerimientos especificados dinámicamente con Sistema experimental<br>• Proceso rápido, informal e iterativo<br>• Los usuarios interactúan rápido con el proceso | • Rápido y barato<br>• Útil cuando existe incertidumbre en los requisitos de información o los usuarios finales son importantes | • Inadecuado para sistemas grandes y complejos<br>• Puede ser superficial al obviar el análisis, documentación y pruebas |
| **Paquete de software para la aplicación** | • El software comercial evita la necesidad de programas desarrollados internamente | • Reducen el diseño, programación, instalación y mantenimiento<br>• Ahorro en tiempo y coste<br>• Disminuye la necesidad de poseer recursos internos | • Puede no satisfacer los requerimientos de la institución<br>• Puede no desempeñar bien algunas funciones |
| **Desarrollo de usuarios finales** | • Sistemas creado por y para usuarios finales<br>• Rápido e informal<br>• Poca influencia de especialistas de la información | • Usuarios controlan la construcción de los sistemas<br>• Ahorra el coste y tiempo de desarrollo | • Proliferación excesiva de sistemas sin interconexión entre ellos.<br>• En muchas ocasiones no cumplen las normas de calidad |
| **Fuentes externas** | • Sistemas construidos y operados por proveedores externos | • Reduce y controla mejor los costes<br>• Se obtienen sistemas cuando existe carencia de recursos en la empresa | • Pérdida de control sobre el área de sistemas de información<br>• Dependencia de la dirección técnica y la prosperidad de los proveedores externos |

Tabla 3 - Comparación de enfoques de desarrollo de sistemas

En los últimos años la ingeniería del software está siendo influenciada por la transformación de modelos, ya que con ello se apunta a resolver los problemas de tiempo, costos y calidad asociados a la creación de software. La forma de resolver este problema se enmarca bajo la denominación general de ingeniería dirigida por modelos (MDE, Model-Driven Engineering).

MDE es la disciplina dentro de la Ingeniería del Software que tiene por objetivo dar soporte a las actividades del ciclo de vida del software, utilizando modelos como principal artefacto de desarrollo.

Definir los procesos software utilizando modelos procesables por el ordenador, nos ofrece la posibilidad de aprovechar las oportunidades que ofrece la ingeniería dirigida por modelos, con el fin de dar soporte a la ejecución y evaluación de los procesos implantados.

Dado lo anterior, un modelo de procesos de negocio, construido por un analista, puede ser usado en un proceso de construcción de software.

Uno de los principales problemas a los que se tiene que enfrentar un ingeniero de sistemas es el no disponer de un entorno controlado que le proporcione todas las herramientas para desarrollar su función. Actualmente, para desarrollar sistemas de información son necesarias un conjunto de herramientas independientes, que no guardan ninguna relación entre sí y que muchas veces ralentizan el trabajo de los ingenieros.

Este trabajo parte de esta base e intenta responder a este problema planteando y desarrollando una arquitectura que permita la construcción de un entorno de modelado para desarrollar los sistemas de información.

## 1.5 OBJETIVOS

El objetivo de este trabajo es el de crear un entorno de modelado que facilite, el diseño, implementación y evaluación de los servicios de una unidad de negocio. El entorno de modelado será usado principalmente por los ingenieros de sistemas de información para describir los sistemas de las unidades de negocio.

A partir de los requisitos y el diseño que se desarrolle, el usuario usará este entorno para generar el modelo de negocio.

Los objetivos que pretendemos alcanzar al realizar este proyecto son los siguientes:

- Estudio del estado actual de los desarrollos en el campo de los sistemas de información. Este punto consistirá en la observación de los más relevantes entornos de desarrollo del mercado con el fin de extraer características comunes.
- Estudiar y comprender la plataforma donde estará integrado el entorno de modelado.
- Desarrollo de un prototipo con las operaciones básicas que se necesitan para desarrollar los sistemas de información.
- Introducción del prototipo en un entorno de desarrollo real y realizar un estudio y comparativa de las ventajas y desventajas encontradas durante la utilización de este tipo de software.

# 2. ESTADO ACTUAL DEL ARTE

Para una mejor comprensión del desarrollo de este proyecto y para poner en contexto al lector del presente documento, profundizaré en la definición de ciertos conceptos que están íntimamente relacionados con el proyecto.

## 2.1 INGENIERÍA DIRIGIDA POR MODELOS

La calidad en la gestión que las organizaciones de hoy necesitan para desarrollar o apoyar la competitividad y el desarrollo de los negocios, es el reto más importante que hoy enfrentan los responsables de implantar las tecnologías de la información en las empresas. Los modelos de gestión empresarial tradicionales son dinámicos y es por esto que los enfoques arquitectónicos que se seleccionen deben apoyar los procesos de negocio y no se obstáculo.

La Ingeniería Dirigida por Modelos ofrece un prometedor enfoque para superar la incapacidad de los lenguajes de tercera generación para gestionar de forma flexible el software que da soporte a los sistemas de información en las organizaciones. El objetivo que se persigue es reducir los desvíos que actualmente se observan entre las necesidades de gestión de la información en las organizaciones y los Sistemas de Información que realizan dicha gestión.

Tanto la creación como la gestión de un sistema de información son procesos complejos en los que habitualmente se presentan problemas de diversas naturaleza: desviaciones con respecto a la planificación y el presupuesto; falta de comprensión de los requerimientos del usuario; elección de tecnologías no adecuadas. Estos problemas surgen de la dificultad que supone gestionar correctamente proyectos de sistemas de información en los que hay que saber llegar a soluciones tanto relacionadas con la organización como con la tecnología.

La ingeniería dirigida por modelos (MDE) es una metodología de desarrollo de software que se centra en la creación y explotación de modelos de dominio en vez de en conceptos de programación [6]. Un modelo de dominio es una representación de conceptos u objetos en una situación real del dominio del problema. La aproximación de la ingeniería dirigida por modelos pretende incrementar la productividad mediante la reutilización de modelos estandarizados, simplificar el proceso de diseño mediante el uso de modelos que responden a patrones de diseño y promocionando la comunicación entre individuos y equipos de trabajo en el sistema mediante la estandarización de la terminología y el uso de las mejores prácticas usadas en el dominio de aplicación.

El modelado de un sistema es considerado efectivo dentro de MDE si todos los modelos tienen sentido desde el punto de vista del usuario que es familiar con el dominio de aplicación y si ellos pueden servir como base de implementación del sistema. Los modelos se desarrollan a través de una comunicación extensiva con todos los implicados, desde jefes de producto, diseñadores y desarrolladores hasta los usuarios que manejan el dominio de aplicación.

### 2.1.1 Model-Driven Architecture (MDA)

Desarrollado dentro del Object Management Group (OMG), es un marco de trabajo para el desarrollo de software que define una arquitectura orientada a modelos y unas guías de transformación entre los mismos para recoger las especificaciones de un sistema, donde los productos que se generan son modelos formales.

MDA define una aproximación a la especificación de sistemas de tecnologías de la información que separa la especificación de la funcionalidad del sistema de la especificación de la implementación de dicha funcionalidad en una plataforma tecnológica específica. Para tal fin, MDA define una arquitectura para modelos que proporcionan un conjunto de pautas para estructurar las especificaciones expresándolas como modelos [7]. MDA separa la lógica del negocio de la lógica de la aplicación de la plataforma tecnológica y representa esta lógica con modelos semánticos muy precisos. MDA, al ser un marco arquitectónico, perfecciona la estrategia arquitectónica y mejora la eficiencia del grupo de desarrollo.

El enfoque MDA y los estándares que lo soportan permiten que la misma funcionalidad del sistema especificada como modelo pueda explotarse en múltiples plataformas, y permite a diferentes aplicaciones integrarse mediante la comunicación de sus modelos, habilitando la integración y la interoperabilidad [8] [9] [10]. Por tanto, proporciona una solución para los cambios de negocio y de tecnología, permitiendo construir aplicaciones independientes de la plataforma e implementarlas en plataformas como CORBA, JEE o Servicios Web.

El proceso de desarrollo en MDA está basado en un esquema de compilaciones sucesivas por el que, partiendo de un nivel de abstracción muy alto, se va transformando los modelos iniciales en modelos más detallados. Este esquema de compilaciones sucesivas permite que se pueda introducir lógica del negocio en los productos intermedios. Si el modelo de negocio inicial no describe completamente el negocio, este esquema de compilaciones permite al programador introducir código donde lo necesite. En cierto modo, se asemeja bastante a lo que es un generador de

código, que a partir de un lenguaje de 4ª generación (4GL) generan código en lenguajes de 3ª generación (3GL).

## 2.2 HERRAMIENTAS PARA EL DESARROLLO DE SISTEMAS DE INFORMACIÓN

Las herramientas de ayuda al desarrollo de sistemas de información surgieron para intentar dar solución a los problemas inherentes a los proyectos de generación de aplicaciones informáticas: plazos y presupuestos incumplidos, insatisfacción del usuario, escasa productividad y baja calidad de los desarrollos.

Algunas de estas herramientas se dirigen principalmente a mejorar la calidad, como es el caso de las herramientas CASE (Computer Aided Software Engineering). Otras van dirigidas a mejorar la productividad durante la fase de construcción, como es el caso de los lenguajes de cuarta generación (4GL-Fourth Generation Language).

### 2.2.1 Herramientas CASE

Las herramientas CASE son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

El empleo de herramientas CASE permiten integrar el proceso de ciclo de vida:

- Análisis de datos y procesos integrados mediante un repositorio.
- Generación de interfaces entre el análisis y el diseño.
- Generación del código a partir del diseño.
- Control de mantenimiento.

Las herramientas CASE, en función de las fases del ciclo de vida abarcadas, se pueden agrupar en:

- **Herramientas integradas, I-CASE:** abarcan todas las fases del ciclo de vida del desarrollo de sistemas.
- **Herramientas de alto nivel, U-CASE:** orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo, análisis y diseño.
- **Herramientas de bajo nivel, L-CASE:** dirigidas a las últimas fases del desarrollo: construcción e implementación.
- **Juegos de herramientas o toolkits:** son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida.

Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

| Tipo de Case | Ventajas | Desventajas |
|---|---|---|
| **UpperCase** | • Se utiliza en plataforma PC, es aplicable a diferentes entornos.<br>• Menor costo | • Permite mejorar la calidad de los sistema, pero no mejora la productividad.<br>• No permite la integración del ciclo de vida. |
| **Lower Case** | • Permite lograr importantes mejoras de productividad a corto plazo.<br>• Permite un eficiente soporte al mantenimiento de sistemas. | • No garantiza la consistencia de los resultados a nivel corporativo.<br>• No garantiza la eficiencia del Análisis y Diseño.<br>• No permite la integración del ciclo de vida clásico. |
| **I-Case** | • Integra el ciclo de vida.<br>• Permite lograr importantes mejoras de productividad a medio plazo.<br>• Mantiene la consistencia de los sistemas a nivel corporativo | • No es tan eficiente para soluciones simples, sino para soluciones complejas.<br>• Depende del hardware y del software.<br>• Es costoso. |

Tabla 4- Ventajas y desventajas de las herramientas CASE

### 2.2.2 Lenguajes de cuarta generación (4GL)

La evolución de los lenguajes de programación ha sido constante dentro del siglo XX pasando por varias etapas bien definidas llamadas generaciones.

Los lenguajes de cuarta generación o bien 4GL son herramientas encargadas de optimizar el desarrollo de software automatizando la creación de este.

El término "Lenguajes de Cuarta Generación" (4GL) fue primeramente usado para describir a los lenguajes de especificación, que son lenguajes de alto nivel no procedurales (el usuario define qué se debe hacer, no cómo debe hacerse [11]). La denotación 4GL fue utilizada por primera vez por James Martín en 1982. Los lenguajes RPG de IBM e Informatics de IBM y MARK-IV respectivamente pueden ser considerados los primeros lenguajes de este tipo.

Los lenguajes de cuarta generación contemplan un grado de abstracción que viene a crear una verdadera caja negra pues el programador no dice como se realizaran las aplicaciones que se desee crear, nada más dará una serie de parámetros que aunque dependen del programa que se quiera crear siempre van a estar muy alejados de la máquina o hardware [12].

Los primeros 4GL han basado toda su potencia de generación automática de código en el prototipaje de GUI, dejando un déficit en la generación de código orientado al manejo de datos y los procesos funcionales [12].

Se debe considerar que los 4GL han cambiado mucho desde los primeros lenguajes incluidos dentro de este término. La evolución se ha visto marcada por varios aspectos, uno de ellos son las restricciones del sistema operativo y del mismo hardware, que han marcado en gran medida la pauta a seguir.

Uno de los cambios sustanciales que han tenido los lenguajes de cuarta generación ha sido su relación con la base de datos cambiando el uso de técnicas y recursos en comparación con los 4GL más viejos. Además del mayor manejo de interfaces por el usuario [13].

También se han visto influenciados por las tendencias de cambio que han sufrido la computación con el paso de los años, como la tendencia a programar con orientación a objetos, la incorporación de la arquitectura cliente / servidor, la ingeniería del software y la tendencia a trabajar en equipo [12].

# 3. METODOLOGÍA

Como mencionamos anteriormente, este software se desarrolla con el propósito de proporcionar herramientas que permitan mejorar la productividad en el desarrollo de los sistemas de información.

Este tipo de aplicaciones se ve sometido a una gran cantidad de cambios causados por diferentes motivos: evolución del lenguaje, cambio en los requisitos, realimentación de los usuarios, etc. Hay que tener en cuenta que el entorno de modelado constituiría la base de la pirámide en el desarrollo del sistema y debe soportar estos cambios además de evitar convertirse en un cuello de botella para la evolución del sistema de información. Otro de los requisitos de este tipo de software es la necesidad del usuario de usarlo inmediatamente lo que obliga a disponer de una versión estable cuanto antes.

Por estos motivos la mejor manera de desarrollar este tipo de proyectos es utilizar estrategias de desarrollo evolutivo o incremental que nos permitan obtener versiones cada vez más completas del software. Este tipo de estrategias nos aportan ventajas como:

- Obtener una realimentación por parte de los clientes.
- Refinar los requisitos a medida que el cliente usa la aplicación.
- Disponer de versiones cada vez más completas.

Algunos de los métodos más conocidos que utilizan esta aproximación son:

- Proceso Unificado de desarrollo (Rational Unified Process, RUP).
- Programación Extrema (Extreme Programming, XP).
- Método de desarrollo de sistemas dinámicos (MDSD).
- Desarrollo de software adaptativo (Agile Software Development, ASD).

# 4. ANÁLISIS

A continuación veremos el análisis previo realizado para la obtención final de un entorno que contenga y satisfaga todos los requisitos necesarios para la elaboración y posterior mantenimiento de los sistemas.

## 4.1 ANÁLISIS DE REQUISITO DE USUARIOS

En esta sección veremos el dominio del problema, la lista de características, la cual incluye la mayoría de las funcionalidades que se creen imprescindibles y las cuales se encuentran priorizadas debido a la magnitud del problema planteado.

### 4.1.1 Modelo de dominio

A continuación se detallan los posibles usuarios que usaran el sistema. Cuando se comience con el desarrollo formal del proyecto, se identificarán los usuarios definitivos, que no tienen por qué coincidir con los propuestos en esta sección.

Usuarios:

- ***Desarrollador del modelo***: Es el usuario encargado de desarrollar el sistema de información, crear los proyectos, añadir recursos, etc.

- ***Compilador***: Es el usuario encargado de comprobar que el sistema de información no contenga errores.

- ***Encargado de las pruebas***: Se encarga de probar el sistema de información antes de desplegarlo en su destino.

- ***Encargado del despliegue***: Una vez construido el sistema de información, este usuario se encarga de desplegarlo en la localización de destino.

- ***Encargado del mantenimiento:*** Este usuario se encarga de mantener un sistema de información que ya se encuentra desplegado.

### 4.1.2 Lista de características

La realización de una lista de características como paso previo al análisis de requisitos de software es una práctica que nos indica cual es la magnitud del problema a realizar y la prioridad que tiene cada una de las funcionalidades de la herramienta.

El estado en el que se pueden encontrar cada una de las características son:

- **Planificada:** funcionalidad que debe estar en las primeras iteraciones del desarrollo.
- **Aceptada:** Son funcionalidades con menor importancia a la hora de desarrollar los sistemas y pueden aplazarse para futuras iteraciones.
- **Postergada:** Son las últimas funcionalidades que se añadirán al entorno y pueden no aplicarse si así se decidiera.

Para organizar la lista tenemos el siguiente conjunto de categorías:

A. Edición
B. Vistas
C. Depuración
D. Publicación
E. Control de errores
F. Ayuda
G. Configuración

#### 4.1.2.1 *Edición*

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-A.1 | Editor de texto | La aplicación debe permitir al usuario editar los ficheros que componen el sistema de información. | Planificada |
| LC-A.2 | Editor gráfico | La aplicación permite al usuario la edición de los ficheros que componen el sistema mediante herramientas gráficas | Postergada |
| LC-A.3 | Asistente de contenido | Herramienta que visualiza sugerencias al usuario mientras escribe el código del sistema | Aceptada |
| LC-A.4 | Asistente de creación de proyecto | Asistente para facilitar al usuario la forma de crear los proyectos | Planificada |
| LC-A.5 | Asistente para crear recursos del sistema | Facilita al usuario la labor de crear cada uno de los recursos de los que se compone el sistema | Planificada |
| LC-A.6 | Refactorizar código | Permite al usuario renombrar variables o nombres importantes dentro del sistema | Postergada |
| LC-A.7 | Refactorizar ficheros | El usuario podrá renombrar ficheros del sistema | Aceptada |
| LC-A.8 | Asistente para importar proyectos | El usuario podrá importar en el entorno proyectos creados con anterioridad | Aceptada |

Tabla 5 - Lista de características: Edición

### *4.1.2.2 Vistas*

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-B.1 | Vista de proyectos | Vista que refleja todos los proyectos creados por el usuario | Planificada |
| LC-B.2 | Vista de problemas | Vista para visualizar los errores que existen en los sistemas desarrollados por el usuario | Aceptada |
| LC-B.3 | Vista de Tareas | Esta vista visualiza tareas que se tienen que realizar en el sistema. | Postergada |
| LC-B.4 | Vista de búsqueda | Esta vista permite realizar búsquedas dentro de uno o varios proyectos | Postergada |
| LC-B.5 | Vista de bugs | Esta vista permite visualizar errores que haya descubierto el encargado de probar el sistema. Se puede conectar con sistemas de seguimiento de errores como Mantis. | Postergada |
| LC-B.6 | Vista de publicación | Esta vista permite administrar los sitios de destino donde se publicarán los sistemas | Postergada |
| LC-B.7 | Vista de versiones | Esta viste permite asociar los proyectos a repositorios de control de versiones como subversion o git | Aceptada |

Tabla 6 - Lista de características: Vistas

#### 4.1.2.3 Depuración

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-C.1 | Compilar proyecto | Compila un proyecto completo | Planificada |
| LC-C.2 | Compilación incremental | Permite compilar los cambios producidos en el proyecto después de la anterior compilación. | Planificada |
| LC-C.3 | Depuración | Permite depurar el sistema de información | Postergada |
| LC-C.4 | Depuración remota | Permite depurar el sistema de información que se encuentran en otra máquina o servidor. | Postergada |

Tabla 7 - Lista de características: Depuración

#### 4.1.2.4 Publicación

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-D.1 | Publicación de un proyecto | El usuario puede publicar el sistema de información | Planificada |
| LC-D.2 | Compilar y publicar | El usuario puede compilar un proyecto y publicarlo | Planificada |
| LC-D.3 | Configurar destinos de despliegue | El usuario podrá configurar los destinos donde se desplegarán los sistemas | Postergada |

Tabla 8 - Lista de características: Publicación

### 4.1.2.5 *Control de errores*

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-E.1 | Detección de errores | El entorno es capaz de identificar todos los errores y mostrárselos al usuario | Planificada |
| LC-E.2 | Marcas de fichero | El entorno marca los errores en los ficheros que lo contengan permitiendo al usuario identificar los ficheros con problemas. | Planificada |
| LC-E.3 | Sincronizar vista de problemas con los ficheros | El usuario puede acceder a un fichero que contenga errores a partir del mensaje que se muestra en la vista de problemas | Aceptada |
| LC-E.4 | Marcas de editor | El usuario podrá ver en el editor de texto las líneas del fichero que contienen errores y un mensaje descriptivo del mismo. | Planificada |

Tabla 9 - Lista de características: Control de errores

### 4.1.2.6 *Ayuda*

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-F.1 | Ayuda del entorno | Manual de usuario del sistema | Postergada |
| LC-F.2 | Interfaz Multilenguaje | El usuario podrá seleccionar el idioma en el que visualizar la interfaz del entorno | Postergada |

Tabla 10 - Lista de características: Ayuda

#### 4.1.2.7 *Configuración de la aplicación*

| Código | Nombre | Descripción | Estado |
|---|---|---|---|
| LC-G.1 | Configurar proyecto | El usuario podrá configurar aspectos del proyecto. | Planificada |
| LC-G.2 | Configurar entorno | El usuario podrá cerrar o abrir vistas, configurar el idioma, etc. | Aceptada |

Tabla 11 - Lista de características: Configuración

## 4.2 ANÁLISIS DE REQUISITOS DE SOFTWARE

En esta sección se describirá completamente el comportamiento del sistema que vamos a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software.

### 4.2.1 Definición de casos de uso

Nuestro sistema se divide en tres casos de uso principales que se identifican con las tres acciones básicas que realiza un ingeniero de sistemas. Estos casos de uso son:

- Desarrollar el sistema de información
- Mantener el sistema de información
- Publicar el sistema de información

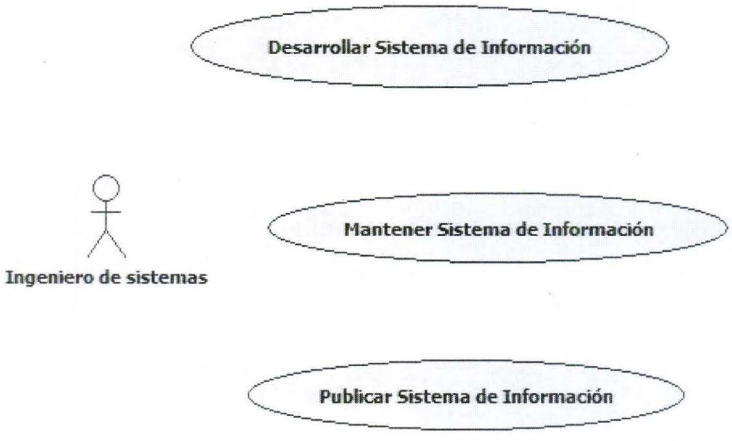


Ilustración 3 - Casos de uso principales

#### 4.2.1.1 ***Desarrollar el Sistema de información.***

La tarea principal que llevará a cabo los usuarios que utilicen este software será la de desarrollar los sistemas de información. El usuario usará el entorno de modelado para realizar operaciones como crear proyectos y/o ficheros, editar ficheros, compilar los proyectos para comprobar si existen errores, etc. A continuación describimos más en detalle estas operaciones.

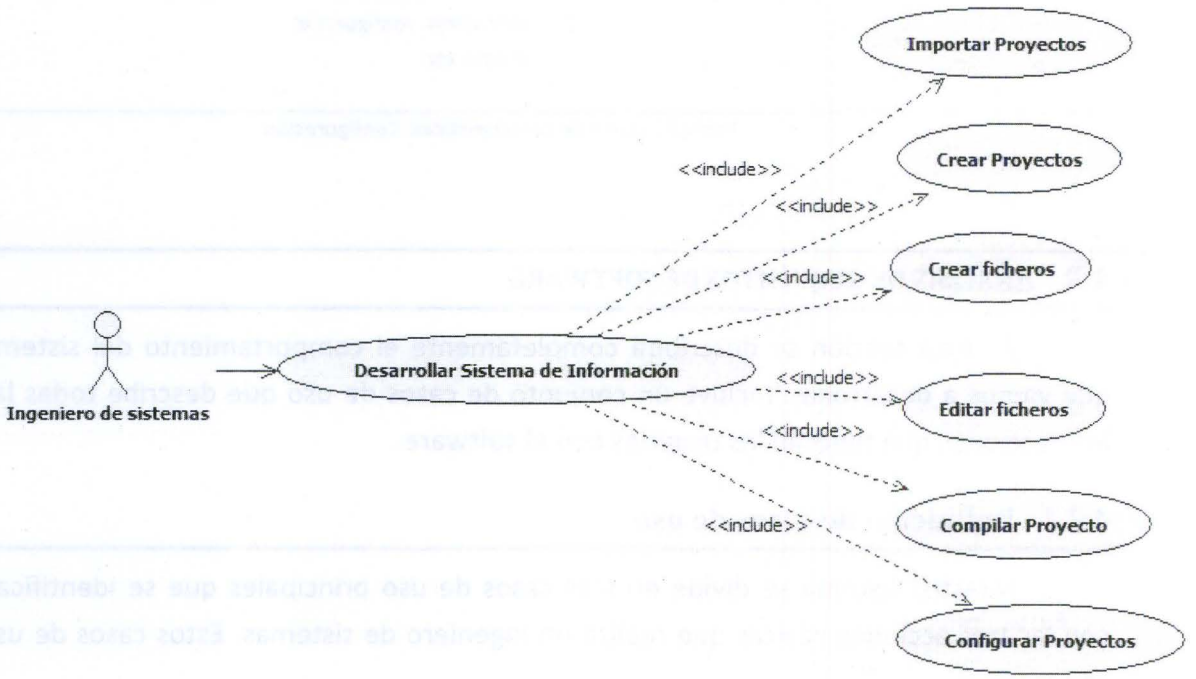


Ilustración 4 - Caso de uso: Desarrollar Sistema de Información

### Crear Proyecto

| Nombre | Crear Proyecto |
|---|---|
| **Descripción** | Permite al usuario la creación de un proyecto donde desarrollará el sistema de información. |
| **Parámetros de entrada** | • Nombre del proyecto.<br>• Directorio del proyecto.<br>• Versión |
| **Flujo** | • Se abre el asistente para que el usuario introduzca la información<br>• Se crea el proyecto y su estructura interna (carpetas, ficheros, etc.). |
| **Precondiciones** | Ninguna. |
| **Postcondiciones** | Se crea el proyecto con toda su estructura interna. |

Tabla 12 - Caso de uso: Crear Proyecto

### Importar proyectos

| Nombre | Importar Proyecto |
|---|---|
| **Descripción** | Permite al usuario importar proyectos que han sido creados previamente. |
| **Parámetros de entrada** | Directorio donde se encuentra el proyecto a importar. |
| **Flujo** | • Se abre el asistente para que el usuario introduzca la información<br>• Se visualiza en el entorno el proyecto existente. |
| **Precondiciones** | El proyecto debe existir y ser válido. |
| **Postcondiciones** | Se importa el proyecto. |

Tabla 13 - Caso de uso: Importar proyectos

### Crear Ficheros

| Nombre | Crear ficheros |
|---|---|
| **Descripción** | El usuario crea ficheros dentro de un proyecto seleccionado. |
| **Parámetros de entrada** | Como parámetros de entrada el usuario debe introducir el nombre del fichero y la información que necesite o se especifique en el lenguaje en el que se desarrolle el S.I. |
| **Flujo** | • Se abre el asistente para que el usuario introduzca la información<br>• Se añade al proyecto el nuevo fichero. |
| **Precondiciones** | El proyecto debe existir y ser válido.<br>El nombre del fichero no puede estar repetido |
| **Postcondiciones** | Se crea un nuevo fichero en el proyecto seleccionado. |

Tabla 14 - Caso de uso: Crear ficheros

### Editar ficheros

| Nombre | Editar ficheros |
|---|---|
| **Descripción** | El usuario puede editar el contenido de un fichero |
| **Parámetros de entrada** | Ninguno. |
| **Flujo** | • El usuario selecciona el fichero que quiere editar. |
| **Precondiciones** | Ninguna. |
| **Postcondiciones** | Se visualiza en el editor el contenido del fichero seleccionado por el usuario |

Tabla 15 - Caso de uso: Editar ficheros

### Compilar Proyecto

| Nombre | Compilar Proyecto |
|---|---|
| **Descripción** | El usuario compila el proyecto seleccionado |
| **Parámetros de entrada** | Proyecto a compilar |
| **Flujo** | • El usuario selecciona el proyecto que desea compilar<br>• Se visualiza un diálogo de progreso con el estado de la compilación. |
| **Precondiciones** | El proyecto debe estar abierto. |
| **Postcondiciones** | Se visualiza en la vista de problemas los errores encontrados durante la compilación. |

Tabla 16 - Caso de uso: Compilar Proyecto

### Configurar proyecto

| Nombre | Configurar Proyecto |
|---|---|
| **Descripción** | El usuario configura los parámetros del proyecto. |
| **Parámetros de entrada** | Ninguno |
| **Flujo** | • Se abre el asistente para que el usuario modifique la información del proyecto. |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | El proyecto actualiza su configuración |

Tabla 17 - Caso de uso: Configurar proyecto

### 4.2.1.2 *Mantener los S.I.*

Otra de las tareas básicas que realizarán los ingenieros de sistemas en el entorno de modelado será el de proporcionar mantenimiento a sistemas de información que se encuentren publicados o en fase de pruebas.

Durante el desarrollo de un sistema de información se pueden producir diferentes situaciones que dan lugar a incidencias que el ingeniero necesita resolver. Estas incidencias pueden deberse a errores que el cliente ha encontrado en el sistema o a cambios que el cliente propone una vez usado el sistema. Otro tipo de incidencias son las que se detectan en la fase de pruebas del sistema y que pueden tratarse de errores o bugs que deben corregirse antes de publicar el sistema en su destino final.

Muchas veces este mantenimiento se ayuda de aplicaciones externas como puede ser *subversión* o *git* para mantener las diferentes versiones que se han desarrollado para el sistema de información o Mantis para gestionar los errores o bugs detectados en el sistema. Como mencionamos al principio, uno de los principales objetivos del entorno de modelado, es centralizar todas las herramientas que use el ingeniero de sistemas para desarrollar las aplicaciones, por lo que debemos dar soporte a estas situaciones que se dan con mucha frecuencia en equipos de desarrollo actuales.

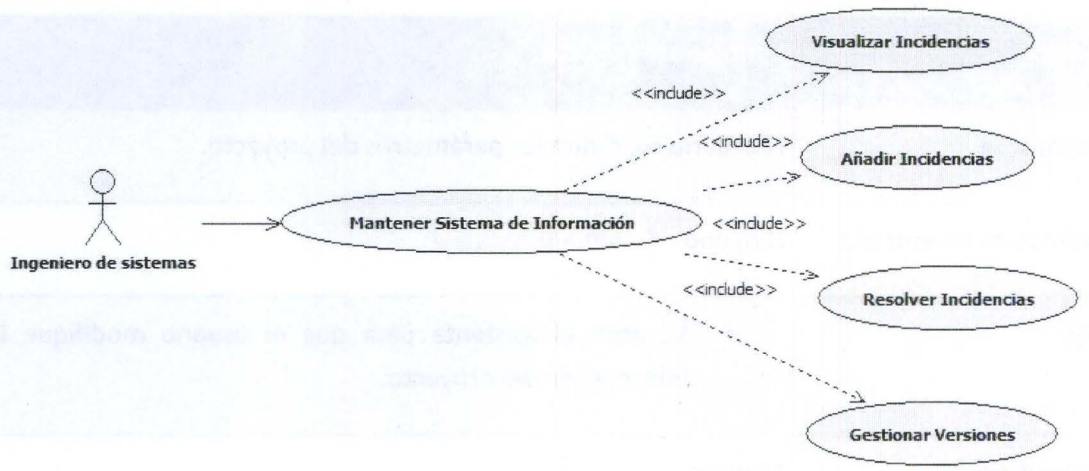


Ilustración 5 - Caso de uso: Mantener Sistema de Información

### Gestionar Versiones

| Nombre | Gestionar versiones |
|---|---|
| **Descripción** | El usuario puede seleccionar la versión del sistema de información con la que quiere trabajar. También puede añadir una nueva versión del sistema de información o eliminar una versión existente. |
| **Parámetros de entrada** | Ninguno |
| **Flujo** | • Se abre el asistente para que el usuario modifique la información del proyecto. |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | El proyecto actualiza su configuración |

Tabla 18 - Caso de uso: Gestionar versiones

### Visualizar Incidencias

| Nombre | Visualizar incidencias |
|---|---|
| **Descripción** | El usuario del entorno puede visualizar los errores detectados en el sistema de información que tiene asignado. |
| **Parámetros de entrada** | Ninguno |
| **Flujo** | • El usuario selecciona el proyecto que desea mantener.<br>• Una vez seleccionado se le muestra en la vista correspondiente el listado de incidencias asociadas a dicho proyecto. |
| **Precondiciones** | El proyecto debe encontrarse abierto. |
| **Postcondiciones** | Se visualiza las incidencias abiertas por los usuarios. |

Tabla 19 - Caso de uso: Visualizar incidencias

### Añadir incidencias

| Nombre | Añadir incidencias |
|---|---|
| **Descripción** | El usuario puede detectar errores o plantear cambios en el sistema de información y necesita que esa situación quede reflejada para el resto de usuarios del entorno. |
| **Parámetros de entrada** | Código de la incidencia, descripción, tipo, prioridad. |
| **Flujo** | • Se abre el asistente para que el usuario introduzca la información de la incidencia<br>• El usuario da de alta la nueva incidencia. |
| **Precondiciones** | El proyecto debe encontrarse abierto. |
| **Postcondiciones** | Se actualiza el listado de incidencias para ese proyecto |

Tabla 20 - Caso de uso: Añadir incidencias

### Resolver incidencias

| Nombre | Resolver incidencias |
|---|---|
| **Descripción** | Un usuario del entorno puede visualizar las incidencias de un determinado proyecto y encargarse de su resolución. |
| **Parámetros de entrada** | Código de la incidencia. |
| **Flujo** | • El usuario selecciona la incidencia que quiere tratar.<br>• Una vez seleccionada la incidencia se puede visualizar el fichero asociado a la misma.<br>• Al finalizar, el usuario puede cambiar el estado de la incidencia. |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | Se resuelve una incidencia. |

Tabla 21 - Caso de uso: Resolver incidencias

### 4.2.1.3 *Publicar S.I.*

La tarea final después de desarrollar el sistema de información es probar los resultados y desplegar el producto.

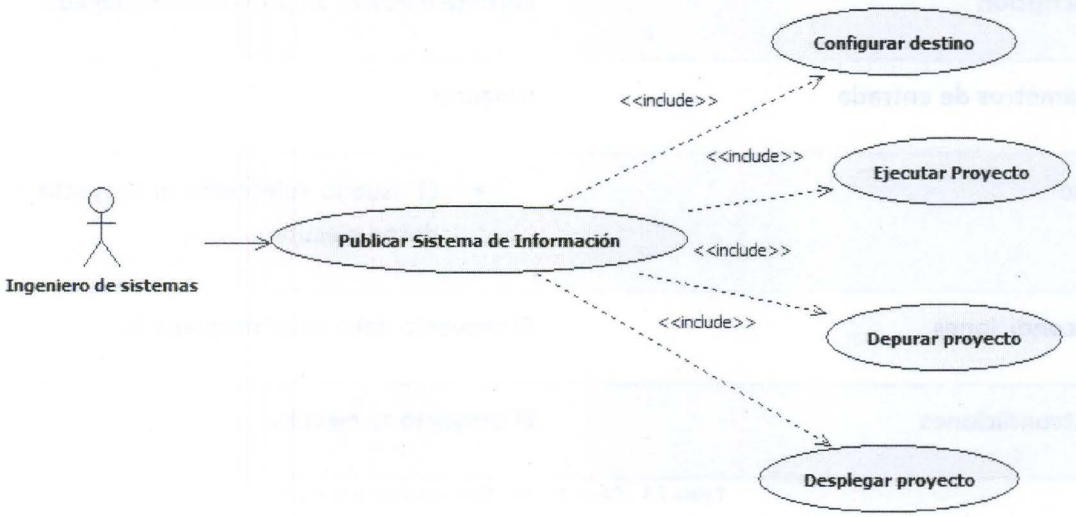


Ilustración 6 - Caso de uso: Publicar Sistema de Información

**Configurar destino**

| Nombre | Configurar destino |
|---|---|
| **Descripción** | El usuario puede configurar el destino donde se desplegará el sistema de información. |
| **Parámetros de entrada** | Nombre del destino, versión del lenguaje. |
| **Flujo** | • Se muestra el asistente para que el usuario configure un destino.<br>• El usuario puede seleccionar un proyecto predeterminado para este destino. |
| **Precondiciones** | Ninguna |
| **Postcondiciones** | Nuevo destino configurado |

Tabla 22 - Caso de uso: Configurar destino

### Ejecutar proyecto

| Nombre | Ejecutar proyecto |
|---|---|
| **Descripción** | Permite ejecutar un proyecto desplegado. |
| **Parámetros de entrada** | Ninguno |
| **Flujo** | • El usuario selecciona el proyecto que desee ejecutar |
| **Precondiciones** | El proyecto debe estar desplegado. |
| **Postcondiciones** | El proyecto se ejecuta. |

Tabla 23 - Caso de uso: Empaquetar proyecto

### Depurar proyecto

| Nombre | Depurar proyecto |
|---|---|
| **Descripción** | Permite al usuario depurar el sistema de información facilitándole la tarea de localizar errores. |
| **Parámetros de entrada** | Proyecto que desee depurar |
| **Flujo** | • El usuario selecciona el proyecto que desea ejecutar.<br>• El usuario ejecuta el proyecto en modo depuración. |
| **Precondiciones** | El proyecto debe estar desplegado en un destino. |
| **Postcondiciones** | El proyecto se ejecuta en modo depuración. |

Tabla 24 - Caso de uso: Depurar proyecto

### Desplegar Proyecto

| Nombre | Desplegar proyecto |
|---|---|
| **Descripción** | Permite al usuario desplegar el sistema de información en un destino previamente configurado. |
| **Parámetros de entrada** | Proyecto que desee desplegar |
| **Flujo** | • El usuario selecciona el comando desplegar.<br>• Selecciona el proyecto que desee desplegar y uno de los destinos que se encuentren configurados. |
| **Precondiciones** | El proyecto no debe contener errores. |
| **Postcondiciones** | El proyecto se despliega en el destino seleccionado. |

Tabla 25 - Caso de uso: Desplegar proyecto

# 5. DISEÑO

En este apartado veremos en detalle el diseño de nuestro software así como de la plataforma en la que está integrada.

En la actualidad no existe un software específico para elaborar sistemas de información debido en gran medida, a la gran cantidad de métodos y lenguajes que existen para la elaboración de los mismos además del carácter especial que tiene este tipo de software.

Esta clase de software debe clasificarse como software a medida debido en parte a que necesita una serie de requisitos previos bien definidos como es el caso del lenguaje, la tecnología, etc.

Una de las primeras preguntas que nos hicimos cuando planteamos el proyecto fue la de si partíamos de cero o buscábamos opciones para su desarrollo. Durante el proceso de análisis nos dimos cuenta que desarrollar este software desde cero conllevaría un esfuerzo y una dificultad excesiva para un sólo desarrollador. Debido a esto se optó por extender la funcionalidad de uno de los principales IDE de desarrollo del momento, Eclipse. Esta solución consiste en integrar nuestro entorno de modelado en Eclipse mediante un conjunto de plug-ins que extiendan la funcionalidad que ya nos ofrece esta herramienta. Las razones que nos han llevado a escoger este IDE para implementar nuestro software son las siguientes:

- Es uno de los entornos de desarrollo más usados en la actualidad.
- Cuenta con una arquitectura base que se ajusta a las necesidades de un software como el que se desarrolla en este trabajo.
- Dispone de una comunidad de usuarios muy extensa que nos permite solucionar las dudas que surjan durante la fase de desarrollo.
- Nos aporta las herramientas necesarias para conectar nuestro software con otras aplicaciones externas y que son usadas durante la elaboración de los sistemas de información.

## 5.1 DISEÑO ARQUITECTÓNICO

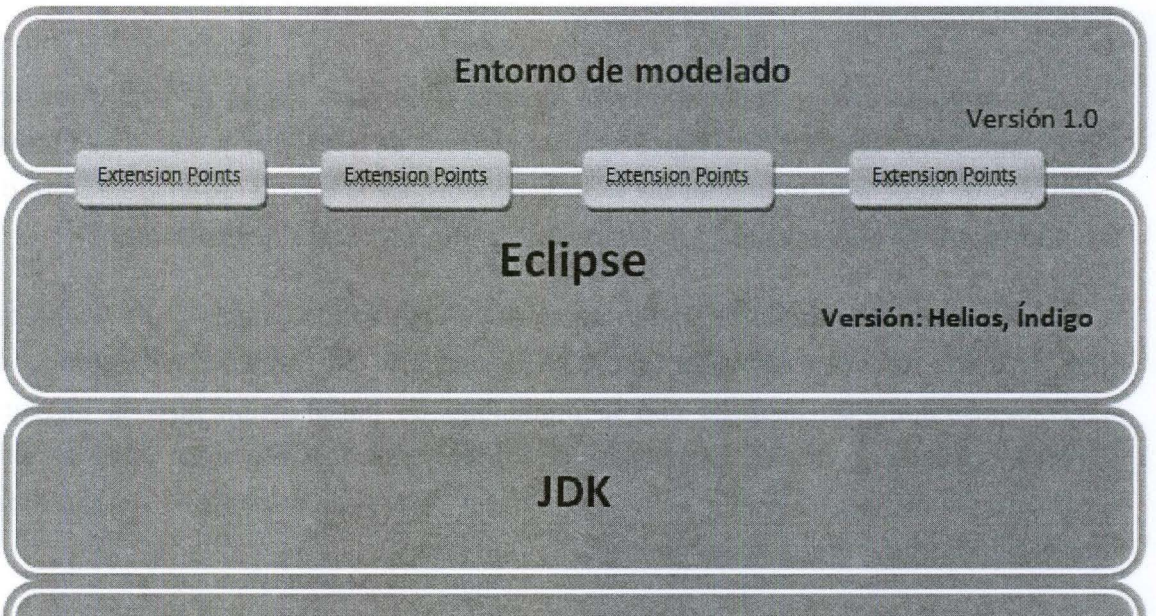


Ilustración 7 - Diseño Arquitectónico de la aplicación

Esta primera versión del entorno de modelado está construida como una extensión del IDE Eclipse a través de una serie de interfaces que nos facilita el entorno para este propósito. A lo largo del desarrollo de este proyecto se han publicado diferentes versiones de este IDE, por lo que el proyecto ha ido actualizándose a estas versiones para proporcionar la máxima compatibilidad con las versiones finales de Eclipse. A continuación veremos un poco más en profundidad en qué consisten las interfaces que nos proporciona Eclipse para llevar a cabo el desarrollo del entorno de modelado.

## 5.2 PUNTOS DE EXTENSIÓN

Eclipse nos ofrece una serie de interfaces para poder extender sus utilidades y herramientas. Estas interfaces se conocen como *puntos de extensión (*Ver ANEXO I). A continuación detallaremos aquellos puntos de extensión que serán usados para añadir todas las herramientas que componen el entorno de modelado.

### 5.2.1 Vistas

Muchos plug-ins añaden una vista de Eclipse nueva o mejoran una existente para proporcionar información al usuario.

Las vistas deben implementar la interfaz *IViewPart.* Por lo general, todas las vistas que desarrollemos en Eclipse serán subclases de *ViewPart* y, por este motivo, también son subclases indirectas de *WorkbenchPart,* de la que heredan gran parte del comportamiento necesario para implementar la interfaz (véase Ilustración 8).

El contenedor de vistas se denomina sitio de vistas (una instancia de *IViewSite),* que a su vez está contenido en una ficha de entorno de trabajo (una instancia de *IWorkbenchPage).* De acuerdo con la modalidad de inicialización *lazy,* la instancia *IWorkbenchPage* contiene instancias de *IViewReference* en lugar de la vista propiamente dicha para poder enumerar y hacer referencia a las vistas sin necesidad de cargar los plug-ins que la definan.

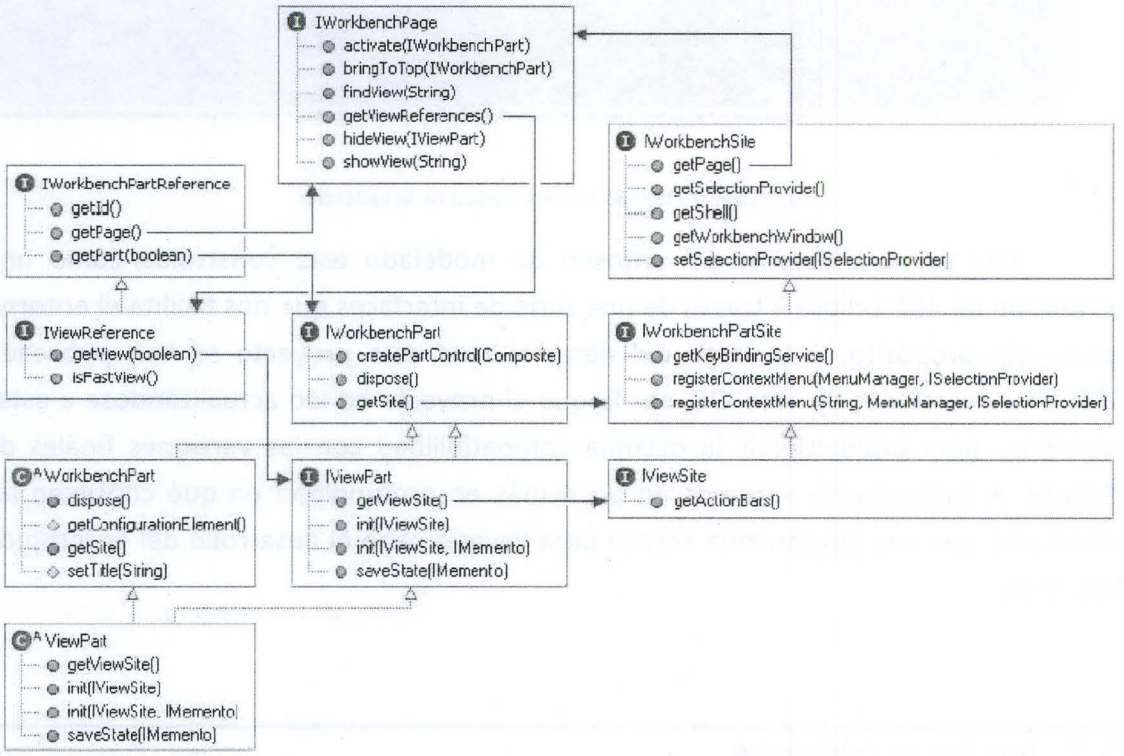


Ilustración 8 - Clases ViewPart

Las vistas comparten un conjunto de comportamientos comunes con los editores pero también tienen diferencias notables. Cualquier acción llevada a cabo en una vista debe afectar de inmediato al estado del entorno de trabajo y al recurso o recursos subyacentes, mientras que los editores siguen el paradigma clásico de abrir-

modificar-guardar. Los editores se muestran en un área de Eclipse, mientras que las vistas se organizan alrededor del área del editor.

Como el entorno de trabajo puede llegar a tener cientos de vistas, éstas se organizan por categorías.

### 5.2.2 Editores

Los editores son el mecanismo principal con el que los usuarios crean y modifican recursos, por ejemplo archivos. Eclipse dispone de algunos editores básicos como editores de código Java y de texto, así como otros editores con varias pestañas más complejos como el editor de manifiesto de plug-in.

Los editores deben implementar la interfaz *IEditorPart*. Por lo general, los editores son subclases de *EditorPart* y, por lo tanto también indirectamente de *WorkbenchPart,* así que heredan gran parte del comportamiento necesario para implementar la interfaz *IEditorPart (*véase Ilustración 9).

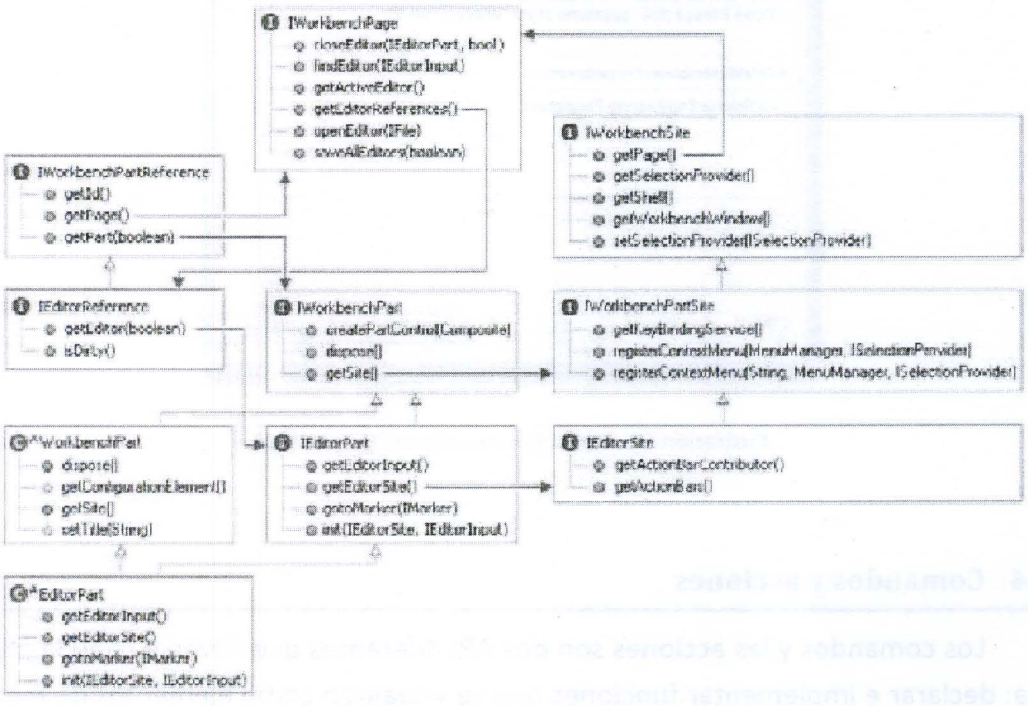


Ilustración 9 - Clases EditorPart

### 5.2.3 Perspectivas

Las perspectivas son una forma de agrupar vistas y comandos de Eclipse para una tarea particular como programar o depurar.

Las perspectivas deben implementar la interfaz *IPerspectiveFactory.* Esta interfaz define un método sólo, *createInitialLayout()*, que especifica el diseño de la ficha y el conjunto de acciones inicial de la perspectiva.

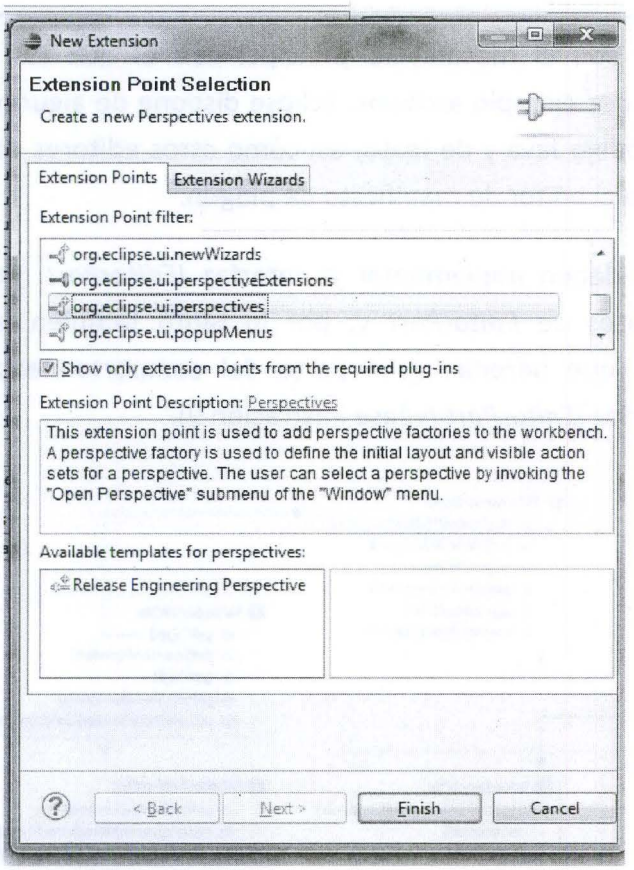


Ilustración 10 - Punto de extensión de perspectivas

### 5.2.4 Comandos y acciones

Los comandos y las acciones son dos API diferentes que llevan a cabo la misma tarea: declarar e implementar funciones que se visualizan como elementos de menú o botones de barra de herramientas. La API de acción forma parte de Eclipse desde la versión 3.0 y la API de comandos desde la versión 3.3 con algunas mejoras menores en Eclipse 3.4. Parece que en algún momento la API de acciones se quedará obsoleta, se trasladará a una capa de compatibilidad y se eliminará posteriormente, pero gran número de herramientas de Eclipse todavía dependen de la API de acciones, por no

mencionar las herramientas y los IDE de terceros que se sustentan en la infraestructura de Eclipse.

Al igual que el resto de elementos de Eclipse, los comandos y acciones se definen mediante puntos de extensión diferentes para que la nueva funcionalidad se pueda añadir fácilmente.

Para las acciones, existen varios puntos de extensión diferentes para cada una de las áreas de la UI (*User interface)* en las que pueden aparecer, pero sin separar la presentación de la implementación.

Por el contrario, la API de comandos separa presentación de implementación mediante tres puntos de extensión: uno para especificar el comando, otro para especificar el lugar de la UI en la que debe aparecer y un tercero para especificar la implementación. Esta posibilidad, junto con una sintaxis de expresión declarativa mucho más rica, otorga una mayor flexibilidad a la API de comandos que a la API de acciones.

### *5.2.4.1 Comandos*

Declarar e implementar un elemento de menú o de barra de herramientas con la API de comandos implica declarar un comando, al menos una contribución de menú para el comando y, como mínimo, un controlador para el comando.

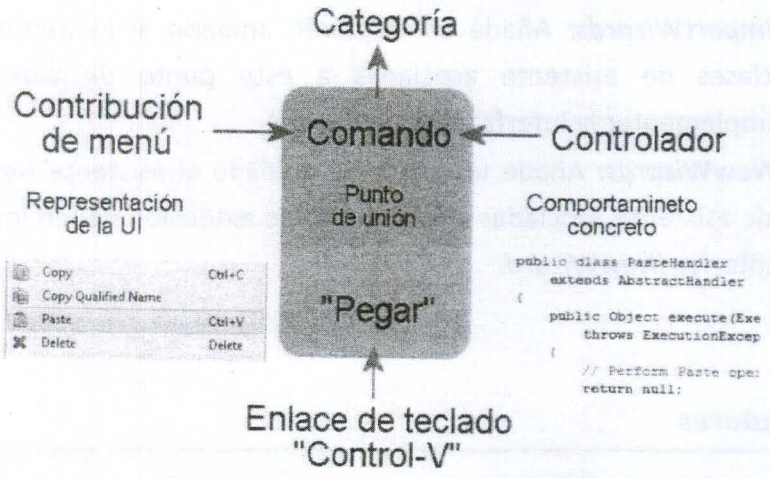


Ilustración 11 - Visión general de los comandos

La declaración de comando es el punto de unión abstracto que asocia una o más contribuciones de menú a uno o más controladores.

El primer paso es declarar el comando que representa el concepto de una función UI, por ejemplo "pegar", sin definir el lugar de la interfaz de usuario en el que

aparecerá ni qué ocurrirá cuando el usuario selecciona la función. Una declaración de controlador asocia el comando a una clase concreta que implementa el comportamiento del comando.

#### *5.2.4.2 Acciones*

Una acción de Eclipse está compuesta de varias partes, incluidas la declaración XML de la acción, la instancia del objeto *IAction* creada por la UI de Eclipse para representar la acción y el *IActionDelegate* que contiene el código para realizar la acción.

Esta división permite que Eclipse pueda representar la acción en un menú o barra de herramientas sin tener que cargar el plug-in que contiene la operación hasta que el usuario selecciona un elemento de menú específico o hace clic en la barra de herramientas.

### 5.2.5 Asistentes

Eclipse proporciona una serie de puntos de extensión para definir asistentes.

- ***ExportWizards:*** Añade un asistente anidado al asistente Export. Las clases de asistente asociadas a este punto de extensión deben implementar la interfaz *IExportWizard*
- ***ImportWizards:*** Añade un asistente anidado al asistente Import. Las clases de asistente asociadas a este punto de extensión deben implementar la interfaz *IImportWizard.*
- ***NewWizards:*** Añade un asistente anidado al asistente New. Las clases de asistente asociadas a este punto de extensión deben implementar la interfaz *INewWizard.*

### 5.2.6 Generadores

Los generadores (builder) de proyectos incrementales, se ejecutan de forma automática siempre que se modifica un recurso en un proyecto asociado. Por ejemplo, cuando se crea o modifica un archivo de código fuente Java, el compilador de Java incremental de Eclipse anota el archivo fuente y construye un archivo de clase. Los archivos de clase Java se conocen como recursos derivados porque el compilador los puede volver a construir a partir de archivos fuente Java.

Los generadores están limitados a un proyecto. Cuando uno o más recursos de un proyecto cambian, se notifica a los generadores asociados al proyecto. Si los cambios se han agrupado por lotes, el generador recibe una notificación que contiene una lista de todos los recursos modificados en lugar de notificaciones individuales de cada recurso. Los generadores procesan la lista de modificaciones y actualizan su estado de construcción para lo que vuelven a construir los recursos derivados necesarios, marcar recursos fuentes, etc. Los generadores reciben una notificación cuando un recurso cambia, como cuando un usuario guarda un archivo fuente modificado, y por lo tanto se ejecutan con bastante frecuencia. Por este motivo, un generador se debe ejecutar de forma incremental, es decir, sólo debe volver a construir aquellos recursos derivados que se hayan modificado.

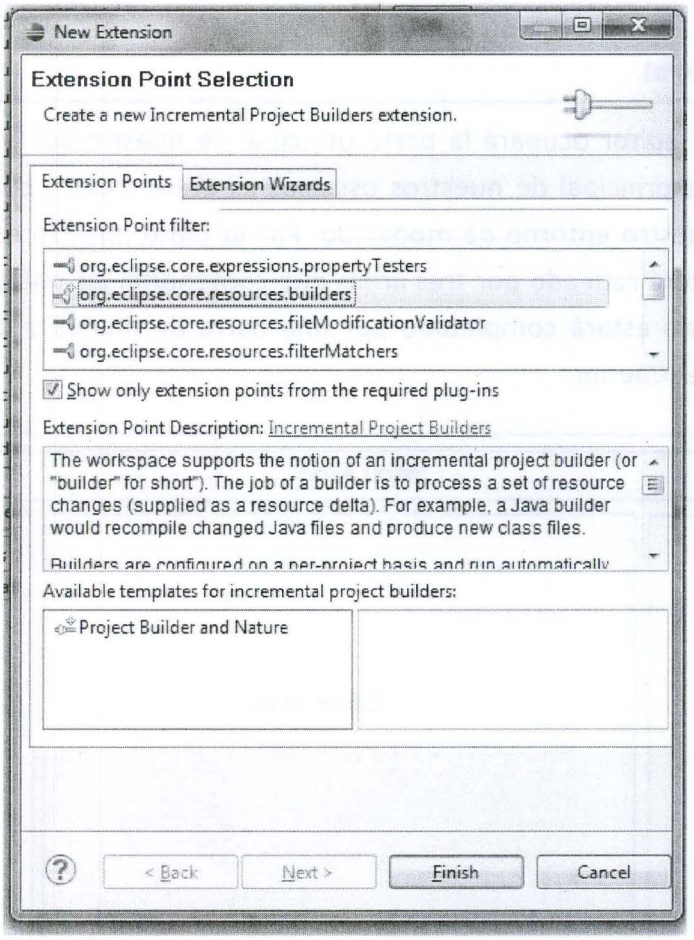


Ilustración 12 - Punto de extensión para los builders

La clase que contiene el comportamiento del builder, debe ser una subclase de *IncrementalProjectBuilder* y, como mínimo, debe implementar los métodos *build()* y *clean()*.

## 5.3 DISEÑO DE LA INTERFAZ DE USUARIO

Un aspecto importante a tener en cuenta durante el diseño de la aplicación es la interfaz con la que el usuario interactuará con la misma. Durante la elaboración de los distintos prototipos se ha ido puliendo las diferentes interfaces para conseguir una interfaz agradable y sencilla que permita realizar las tareas de manera rápida y productiva.

Aunque Eclipse nos proporciona una serie de vistas que podemos usar, es posible que el desarrollador necesite o modificarlas o generar nuevas vistas debido a requisitos de funcionalidad.

### 5.3.1 Vista general

El área del editor ocupará la parte principal de nuestro software, esto es así porque el objetivo principal de nuestros usuarios es el de crear y editar los ficheros que componen nuestro entorno de modelado. Por lo tanto nuestro editor ocupará la parte central y estará rodeado por tres áreas que contendrán las vistas principales del entorno. El entorno estará completado por una barra de menús y de herramientas situada en la parte superior.

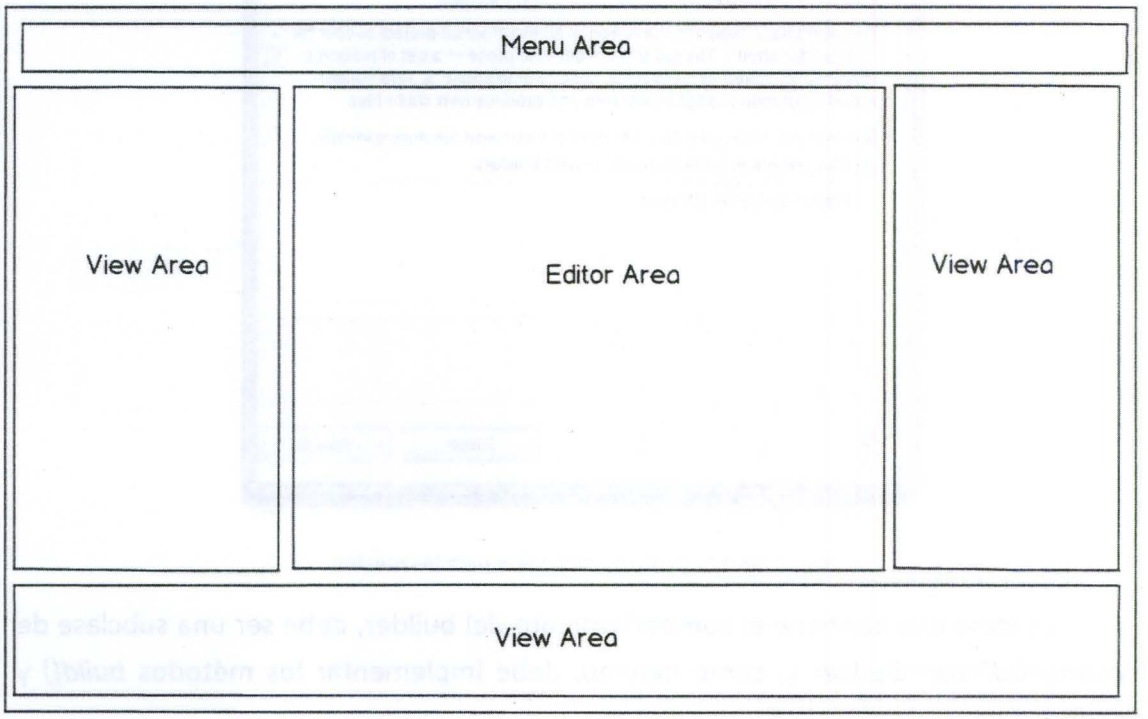


Ilustración 13 - Distribución general del entorno de modelado

### 5.3.2 Vista de explorador de proyectos

Esta es la vista que se va a encargar de mostrar los sistemas de información desarrollados por el usuario. Es una vista simple, tiene una barra de herramientas en la parte superior que facilita el acceso directo al usuario a las acciones más comunes como son la de expandir o contraer todo el contenido de la vista, aplicar o eliminar filtros al contenido, etc.

El contenido de la vista se mostrará en forma de árbol para que el usuario pueda acceder fácilmente al contenido del proyecto.

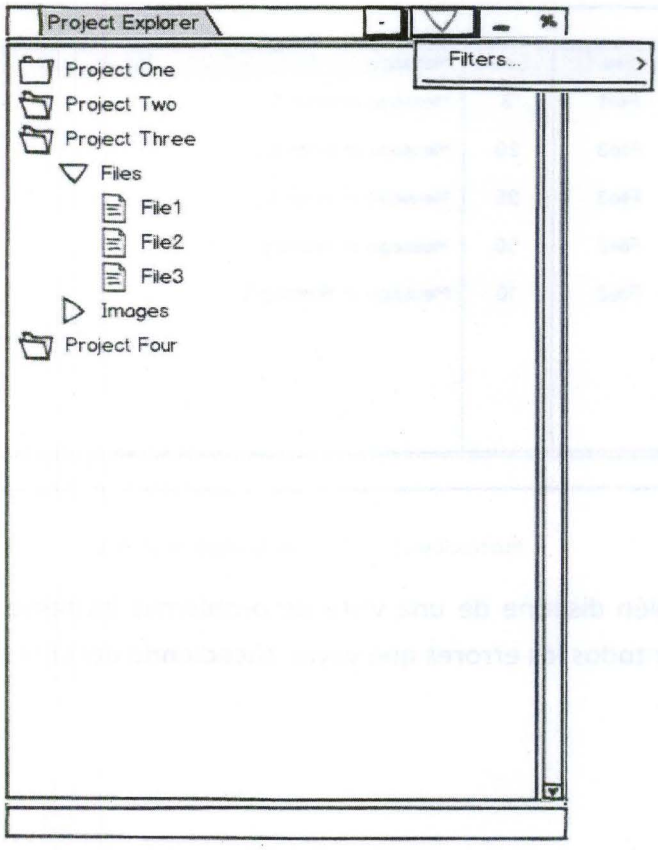


Ilustración 14 - Diseño del explorador de proyectos

Esta vista es muy parecida a la que viene por defecto en el entorno de Eclipse. El explorador de proyectos que nos ofrece Eclipse está preparada para mostrar las características de los proyectos Java, por esa razón, es una de las vistas que seguramente se tengan que modificar o sustituir para que se adapte mejor a las necesidades de los proyectos que se generen en el entorno de modelado.

### 5.3.3 Vista de errores

Esta es la vista encargada de mostrar los errores que se detecten en los proyectos gestionados por el usuario. Cuando el compilador procesa un determinado proyecto, este muestra los fallos encontrados en esta vista facilitándole al usuario la identificación de los errores y el lugar en el que se producen.

Entre la información que se muestra aparece: tipo de error producido (advertencia o error), fichero que contiene el error, línea en la que se encuentra, mensaje explicativo del error.

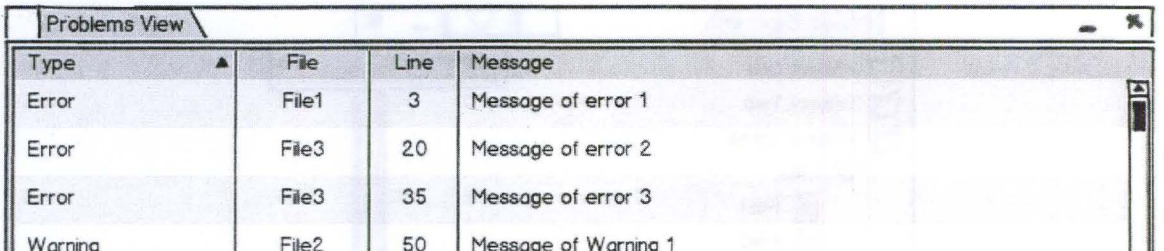


Ilustración 15 - Diseño de la vista de errores

Eclipse también dispone de una vista de problemas bastante completa y que nos permite mostrar todos los errores que vayan sucediendo durante la compilación.

### 5.3.4 Vista de tareas

Esta vista permite al usuario comprobar las tareas pendientes de los sistemas de información que está desarrollando. En un grupo de trabajo grande, este tipo de utilidades adquiere una importancia crucial durante el desarrollo de los sistemas de información y su posterior mantenimiento.

Esta vista permitiría:

- Sincronizar los trabajos de los analistas
- Informar de fallos encontrados durante la fase de pruebas
- Añadir nuevos requisitos
- Planificar el trabajo a realizar para los distintos sistemas de información
- Repartir tareas entre los distintos analistas.

La vista muestra las tareas en forma de árbol permitiendo al usuario organizarlas por categorías. Dispone de un buscador para localizar las tareas y filtros para simplificar la lista.

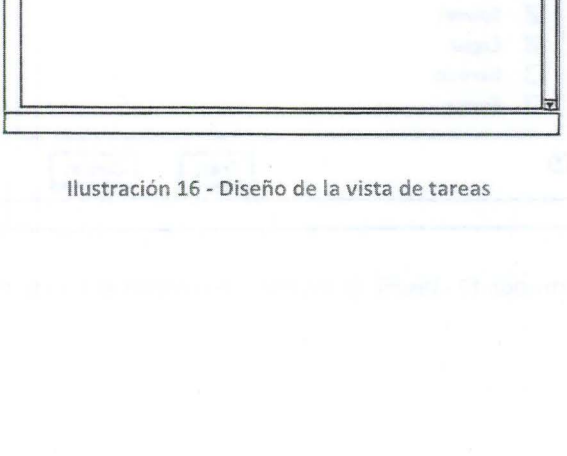


Ilustración 16 - Diseño de la vista de tareas

### 5.3.5 Asistente de creación de proyecto

Este tipo de asistentes depende muchas veces del lenguaje que se usará para desarrollar los sistemas de información. Los asistentes tienen el objetivo de facilitar la tarea de generar los recursos que utilizará el usuario en el entorno de modelado. El proyecto es el primer paso a la hora de generar un sistema de información independientemente del lenguaje que usemos.

En la siguiente ilustración se puede ver el diseño preliminar del asistente para la creación de proyectos que posteriormente se usó en el desarrollo del entorno de modelado.

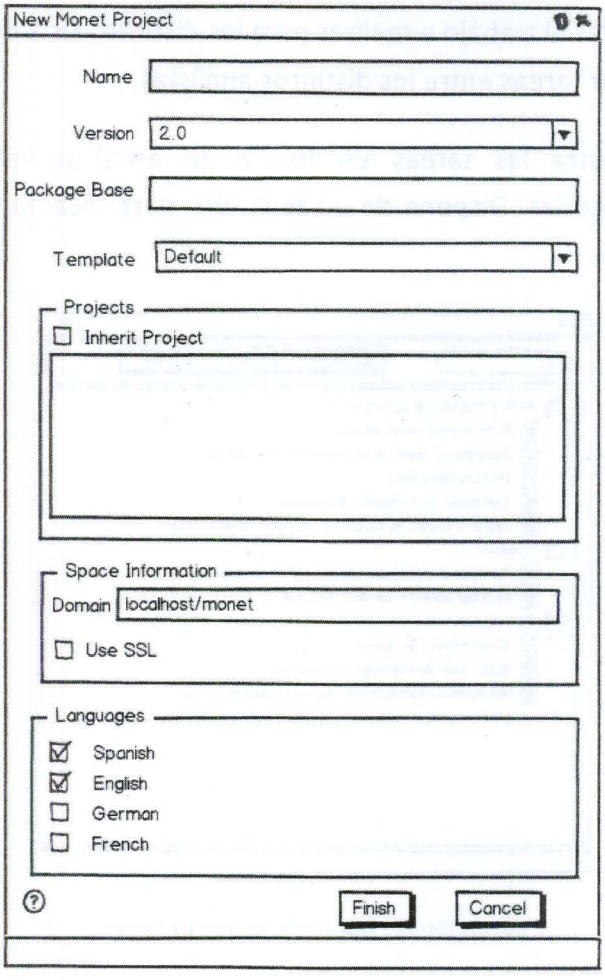


Ilustración 17 - Diseño de asistente de creación de proyectos

### 5.3.6 Asistente de creación de recursos

Al igual que el asistente anterior, el lenguaje que usemos para elaborar los sistemas de información es el que determinará la forma de este tipo de asistentes. Como ejemplo ilustrativo se muestra el caso de creación de un recurso usado en la etapa de desarrollo del entorno de modelado, que veremos en el siguiente apartado.

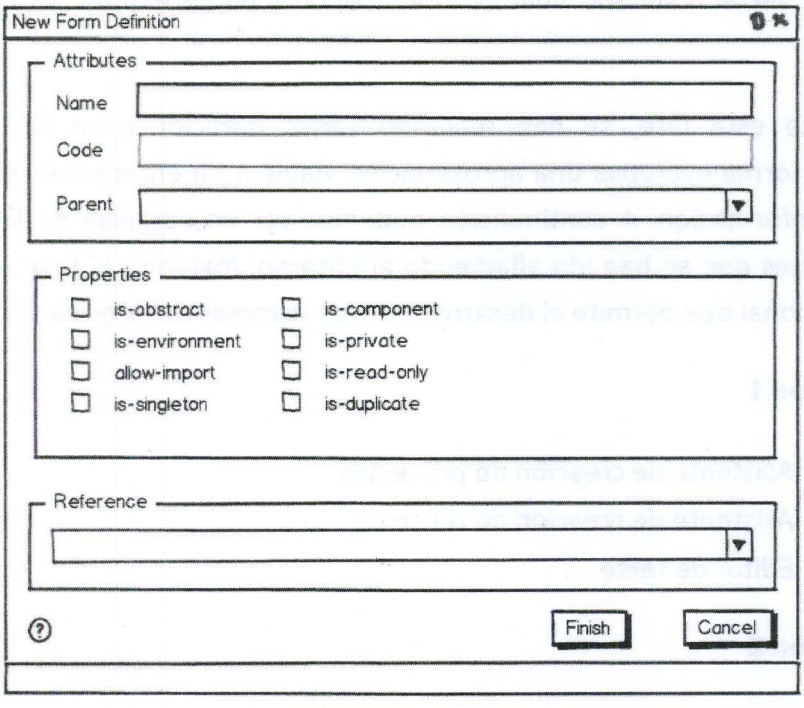


Ilustración 18 - Diseño del asistente de creación de recursos

# 6. DESARROLLO

Una vez estudiado la arquitectura de la plataforma de desarrollo que vamos a utilizar y teniendo claro los tipos de objeto que se pueden utilizar (descritos anteriormente), pasamos a la fase de desarrollo. En esta fase del trabajo el objetivo ha sido tratar de ver el resultado final de la aplicación y poder extraer las conclusiones finales.

Durante esta fase, se han realizado varias iteraciones en las que se ha elaborado de forma evolutiva una aproximación viable a un entorno de modelado de sistemas de información. A continuación podemos ver un desglose de las utilidades más importantes que se han ido añadiendo al entorno, logrando al final, una versión estable y funcional que permite el desarrollo de los sistemas de información.

**Iteración 1**

- Asistente de creación de proyectos
- Asistente de creación de recursos
- Editor de texto

**Iteración 2**

- Compilador "full-build" (Compilaba todo el proyecto)
- Sitio de actualización
- Creación de base de datos
- Vista de explorador de proyectos
- Añadir marcas de errores al editor

**Iteración 3**

- Compilador incremental
- Vista de dependencias
- Vista de plantillas
- Publicación de proyectos

**Iteración 4**

- Configurador de proyectos
- Herencia de proyectos
- Completar compilador

En los siguientes apartados detallaremos la manera de implementar los distintos plug-ins que fueron necesarios para la creación del entorno de modelado.

## 6.1 CREACIÓN DE PLUG-INS

Para construir un plug-in, los pasos básicos a realizar son:

- Decidir cómo va a estar integrado el plug-in en la plataforma.
- Identificar a qué punto de extensión hay que contribuir para integrar el plug-in en la plataforma.
- Implementar la extensión de acuerdo a las especificaciones del punto de extensión.
- Proveer de un archivo *manisfest* (*plugin.xml)* que describa la extensión con la que se está colaborando y el paquete con el código.

Como ejemplo se verá el desarrollo de un plug-in mínimo, "*Hello World*" que sirvió para familiarizarse con la herramienta y con el proceso que se seguirá para realizar el prototipo que se detalla en posteriores apartados.

El objetivo es extender un punto de extensión del *workbench* con una vista que incluya la etiqueta *"hello world"*. En el paquete *org.eclipse.ui* están incluidas las interfaces públicas que definen la interfaz de usuario del *workbench.* Algunas de estas interfaces tienen clases que las implementan por defecto y que pueden ser extendidas para introducir cambios. La interfaz que se necesita para el ejemplo es *IViewPart*, siendo la clase que la implementa por defecto *ViewPart.* Aunque las vistas del *workbench* suelen cambiar su contenido dependiendo del elemento que el usuario señale, en este caso, por simplificar, la vista permanecerá estática.

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.part.ViewPart;

public class HelloWorldView extends ViewPart {
    Label label;

    public HelloWorldView() {
    }

    public void createPartControl(Composite parent) {
        label = new Label(parent, SWT.WRAP);
        label.setText("Hello World");
    }

    public void setFocus() {}
}
```

Ilustración 19 - Detalle de la vista

El método *createPartControl* crea una etiqueta de la biblioteca SWT y escribe el texto "*Hello World*" en ella. En la siguiente imagen podemos ver el resultado de la vista creada.

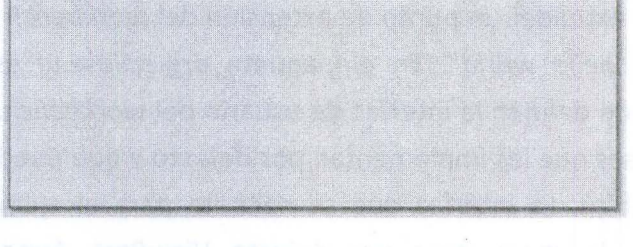


Ilustración 20 -Vista producida por la ejecución del plug-in "Hello World"

## 6.2 MONET

Tal y como se ha expuesto en apartados anteriores, este tipo de software tiene una serie de requisitos previos al desarrollo del mismo. Antes de comenzar con el desarrollo de la aplicación se debe conocer el lenguaje en el que se desarrollarán los sistemas de información. En nuestro caso y debido a que el alumno se encontraba integrado en un equipo de desarrollo de este tipo de software, se optó por realizar la herramienta para la plataforma en la que trabajaba diariamente.

Monet es una plataforma que pretende implementar la metodología de la ingeniería dirigida por modelos, si bien se diferencia de la línea seguida por MDA en que el sistema hace la interpretación de los modelos para generar un sistema plenamente operativo en lugar de la traducción o compilación de los modelos para generar un programa compilado. Los principales componentes funcionales que conforman la arquitectura de Monet son, por un lado, el entorno de modelado (cuya función es la edición y definición de modelos de negocio), y por otro lado, un entorno de ejecución (*Monet Execution Environment*) cuya función es el despliegue de espacios de negocio basados en un modelo de negocio para la configuración de un sistema de información en una unidad de negocio.

El modelo de negocio (*Business Model*) y el espacio de negocio (*Business Space*) son los elementos centrales de la arquitectura y representan la especificación del sistema de información y el sistema de información en sí respectivamente. Para conocer más detalles de esta plataforma, ver el ANEXO II.

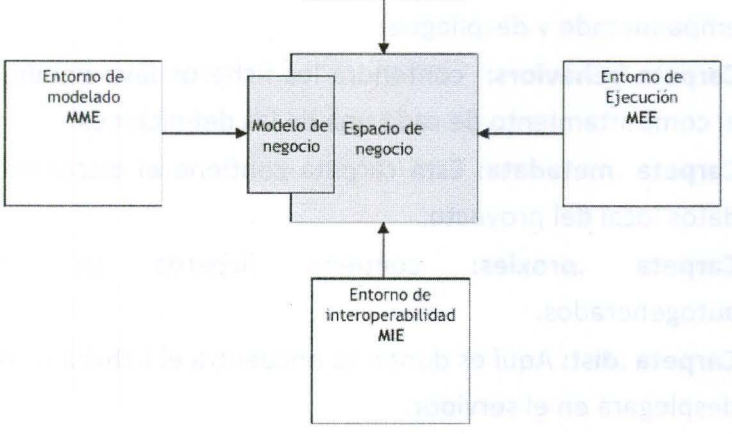


Ilustración 21 - Arquitectura de Monet

### 6.2.1 Estructura del proyecto

El primer paso en la construcción del entorno de modelado fue diseñar la estructura que tendría el proyecto de Monet, esto es, el conjunto de las carpetas que lo componen, ficheros obligatorios, etc. En la siguiente imagen podemos ver esta estructura.

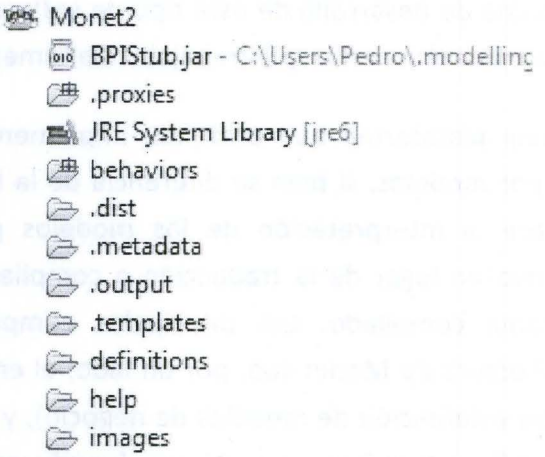


Ilustración 22 - Estructura de un proyecto de Monet

Cuando creamos un proyecto de este tipo se generan una serie de carpetas y ficheros automáticamente. Los más importantes serían:

- **Carpeta *definitions*:** almacenará todas las definiciones del sistema de información.
- **Carpeta .output**: contendrá la salida del builder de Monet que modificará los ficheros fuentes preparándolos para su posterior empaquetado y despliegue.
- **Carpeta behaviors:** contendrá los ficheros Java donde se desarrollará el comportamiento de cada una de las definiciones.
- **Carpeta .metadata:** Esta carpeta contiene el diccionario y la base de datos local del proyecto.
- **Carpeta .proxies:** contiene ficheros de comportamiento autogenerados.
- **Carpeta .dist:** Aquí es donde se encuentra el fichero comprimido que se desplegará en el servidor.
- **Fichero model.xml:** Este fichero contiene información sobre la versión de Monet que se está usando, la fecha de compilación, etc.

## 6.3 PERSPECTIVA

Las perspectivas son una forma de agrupar vistas y comandos de Eclipse para una tarea particular como programar y depurar. Las ampliaciones de mayor envergadura en Eclipse en las que participan varios plug-ins pueden ofrecer sus perspectivas propias. Mejoras menores en las que participan uno o dos plug-ins y proporcionan una o dos vistas de Eclipse nuevas normalmente mejoran perspectivas existentes en lugar de proporcionar otras nuevas.

Para nuestro entorno de modelado, hemos decidido crear una perspectiva propia que ordene las vistas nuevas que crearemos y la barra de herramientas personalizada. También es un paso para albergar nuevas funcionalidades que queramos introducir en el futuro.

La perspectiva quedaría distribuida de la siguiente manera:

- A la izquierda quedaría localizada la vista para visualizar los proyectos específicos de Monet.
- A la derecha estaría la vista de dependencias.
- En la parte inferior quedaría la vista de problemas y demás vistas que necesite el usuario.
- En la parte central se puede ver el área del editor.

Además se representa una barra de herramientas con los botones específicos que necesita el analista para poder publicar, crear y compilar los sistemas de información de Monet.

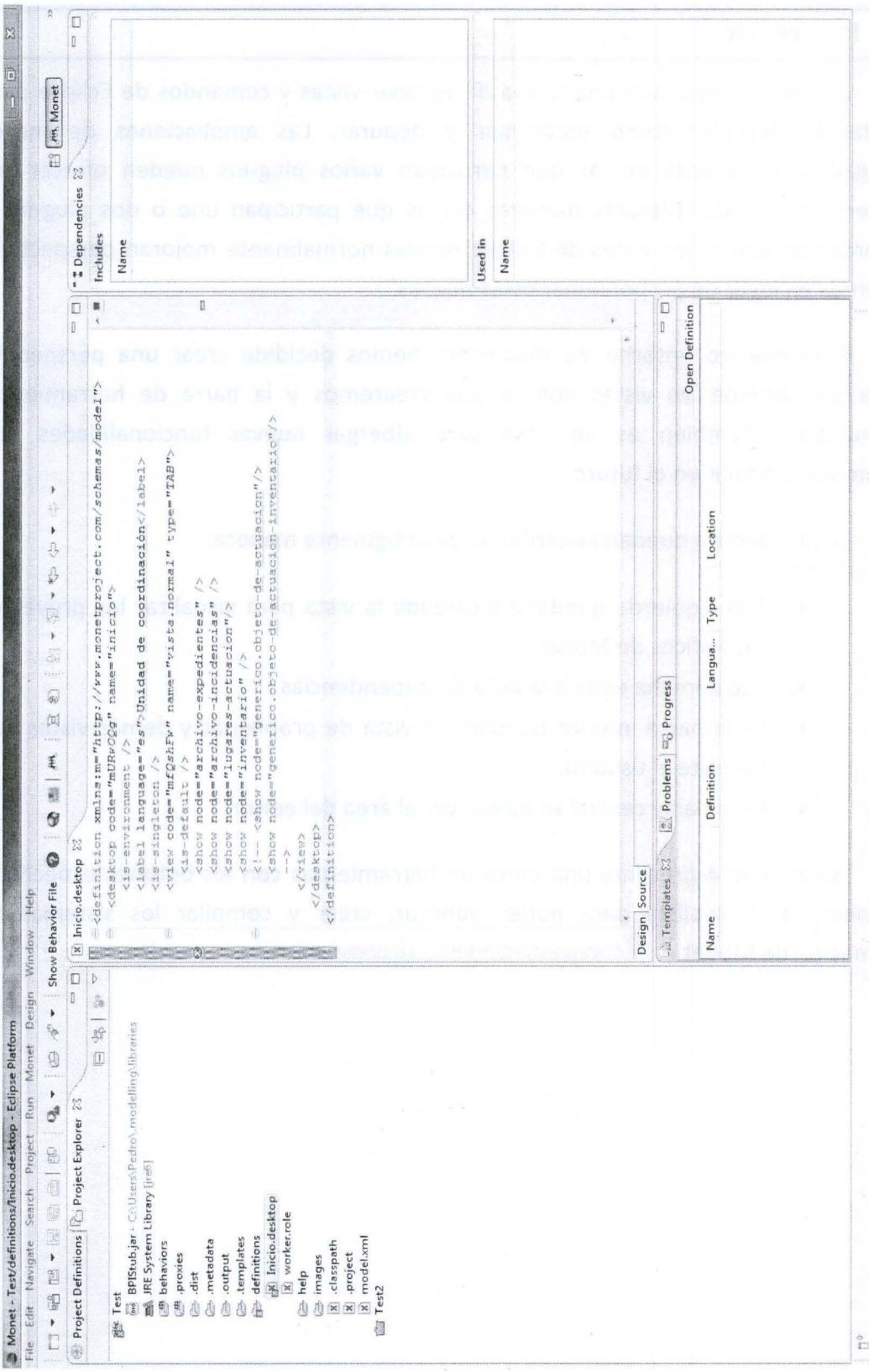


Ilustración 23 - Detalle de la perspectiva

## 6.4 EDITOR

Como vimos en el diseño de la arquitectura, una de las herramientas principales es el editor. Esta herramienta no solo sirve para crear o modificar recursos sino que facilita al ingeniero de sistemas la identificación y corrección de errores.

Eclipse dispone de algunos editores básicos como editores de código Java y de texto, así como otros editores con varias pestañas más complejos como el editor de manifiesto de plug-in.

Con los últimos frameworks de desarrollo como GEF (*Graphical Editing Framework)* se pueden realizar editores y vistas con componentes gráficos que permiten al usuario interactuar con ellos lo que a su vez modifica el modelo subyacente. Al igual que otras aplicaciones que muestran información gráfica, el entorno GEF está diseñado con la arquitectura Modelo-Vista-Controlador (MVC) que, como su nombre indica, consta de tres componentes principales: el modelo, la vista y el controlador.

En las primeras iteraciones del desarrollo y para que el analista o el ingeniero de sistemas pueda comenzar con el uso del entorno, le facilitaremos un editor de texto que soporte estructuras o marcas como es el caso del XML. Al ser un editor de texto simple no requiere mucha programación y le facilitamos una herramienta con la que crear y modificar los recursos del sistema de información.

Eclipse nos permite incorporar marcas a este tipo de editores con lo cual podemos identificar aquellos ficheros que contengan errores e incluso detectar la línea en la que se produce. En iteraciones más avanzadas del desarrollo podemos incorporar la auto-corrección de errores con el fin de mejorar la experiencia del usuario con la herramienta.

Inicio.desktop

```
<definition xmlns:m="http://www.monetproject.com/schemas/model">
  <desktop code="mURwOQg" name="inicio">
    <is-environment />
    <label language="es">Unidad de coordinación</label>
    <is-singleton />
    <view code="mfQshFw" name="vista.normal" type="TAB">
      <is-default />
      <show node="archivo-expedientes" />
      <show node="archivo-incidencias" />
      <show node="lugares-actuacion"/>
      <show node="inventario" />
      <!-- <show node="generico.objeto-de-actuacion"/>
      <show node="generico.objeto-de-actuacion-inventario"/>
      -->
    </view>
  </desktop>
</definition>
```

Ilustración 24 - Detalle del editor

### 6.4.1 Extensión del editor

Declarar un editor es un proceso de dos pasos:

- Definir el editor en el archivo del manifiesto
- Crear la parte del editor que contiene el código

El primer paso para crear un editor es definir el editor en el manifiesto de plug-in (*Añadir la extensión org.eclipse.ui.editors).*

El código que define el comportamiento del editor se encuentra en una clase que implementa la interfaz *IEditorPart*, que suele ser por lo general una subclase de una de estas clases concretas:

- ***EditorPart:*** Implementación base abstracta de la interfaz *IEditorPart.*
- ***MultipageEditorPart:*** Implementación base abstracta que amplía *EditorPart* para que admita editores con varias pestañas.
- ***FormEditor:*** Implementación base abstracta que amplía *MultiPageEditor* para que admita editores con varias pestañas que utilicen una o más fichas con formas y una ficha con código fuente de la entrada del editor.

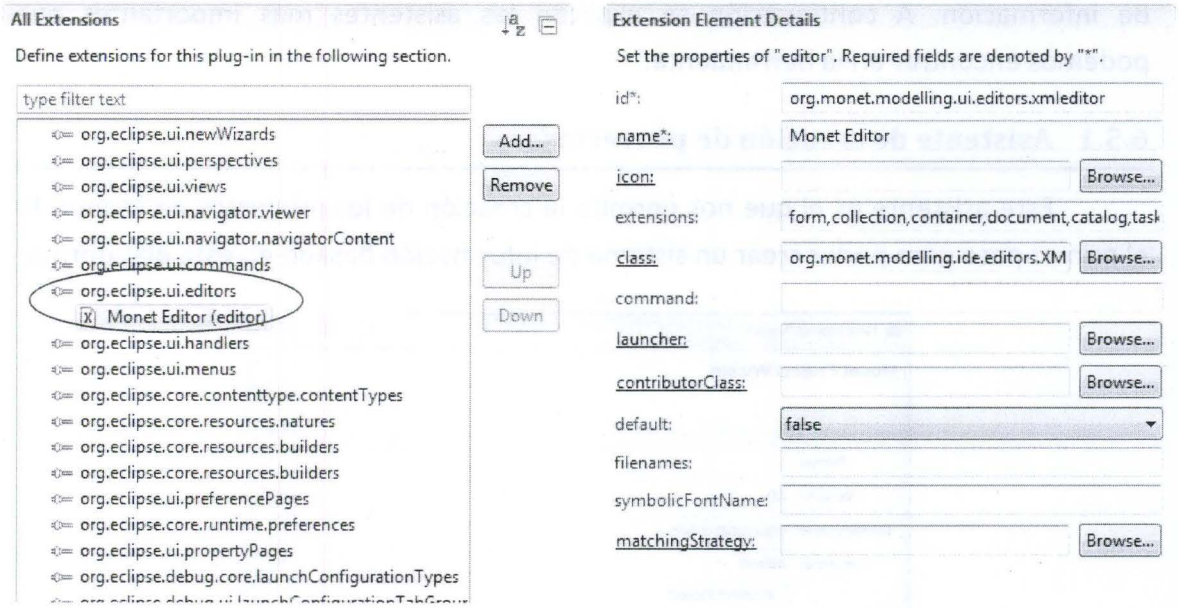


Ilustración 25 - Detalle de la extensión del editor y sus propiedades

En el caso del editor de Monet, hemos desarrollado un editor de tipo formulario (*FormEditor*) con el fin de que en futuras iteraciones se puedan incorporar nuevas funcionalidades como editores gráficos, etc. Actualmente, el editor muestra el código fuente en un editor de texto estructurado (*StructuredTextEditor*) que es perfecto para mostrar el código XML que utilizará el analista para desarrollar el sistema.

## 6.5 ASISTENTES Y DIÁLOGOS

Siempre que se solicita o muestra información de forma no modal al usuario, éste puede interactuar libremente con todos los recursos del entorno de trabajo. Las ventanas, fichas, editores y vistas son ejemplos de elementos UI no modales que no restringen el orden en el que el usuario interactúa con ellos. Los cuadros de diálogos y asistentes suelen ser modales, lo que sólo permite al usuario introducir la información solicitada o cancelar la operación. Estos elementos sólo deberían utilizarse cuando existen restricciones de programación que requieren recopilar o mostrar información antes de que se pueda realizar otra operación.

Crear un proyecto u otros recursos son un claro ejemplo de esta situación. En estos casos, la operación debe recopilar de forma secuencial toda la información necesaria antes de que se pueda llevar a cabo.

Para este software se han creado una serie de asistentes y diálogos que facilitan al usuario la tarea de crear los recursos necesarios para completar el sistema

de información. A continuación se explican los asistentes más importantes que podemos encontrar en la herramienta.

### 6.5.1 Asistente de creación de proyectos

Este asistente es el que nos permite la creación de los proyectos de Monet. Es el primer paso para poder crear un sistema de información basado en esta plataforma.

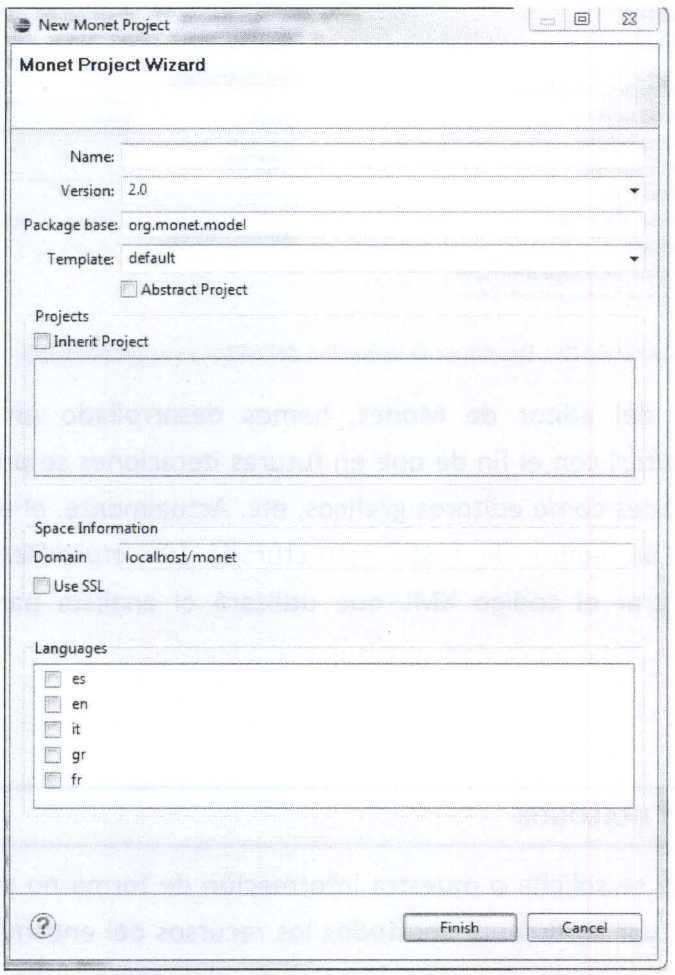


Ilustración 26 - Asistente de creación de proyectos

Para poder crear el proyecto es necesario que el usuario especifique una serie de parámetros esenciales como son:

- Nombre del proyecto
- Versión del lenguaje
- Paquete base para los ficheros de comportamiento
- Plantilla en la que se basará el proyecto. Esto es necesario ya que nos permite crear un proyecto con lo básico para poder publicarlo sin que

ocurra ningún error. También nos permitirá en un futuro añadir más plantillas que permitan agilizar la creación de los sistemas de información.

- Proyectos de los que hereda.
- Lenguajes que admite el proyecto.
- Configuración de despliegue. Esto permite configurar en el proyecto el servidor donde estará alojado el proyecto una vez esté desarrollado.

Una vez recolectada toda la información necesaria para la creación del proyecto, el entorno genera todo lo necesario para que el analista pueda comenzar a desarrollar el sistema. Entre las tareas que realiza el asistente está:

- Creación de carpetas del proyecto
- Asociar los builders al proyecto
- Crear la base de datos local.
- Crear el fichero model.xml

### 6.5.2 Asistentes de creación de recursos

Una vez creado el proyecto, los usuarios pueden generar las definiciones que compondrán el sistema de información. Para ello pueden ayudarse de una serie de asistentes que permiten la creación de la definición basándose en una plantilla. En la plataforma de Monet pueden crearse los siguientes recursos: catalogs, collections, containers, cubes, desktops, documents, exporters, forms, importers, references, roles, service-providers, services, tasks y thesaurus. Se ha creado un asistente para cada uno de estos recursos. En la Ilustración 27 podemos ver el asistente para la creación de formularios.

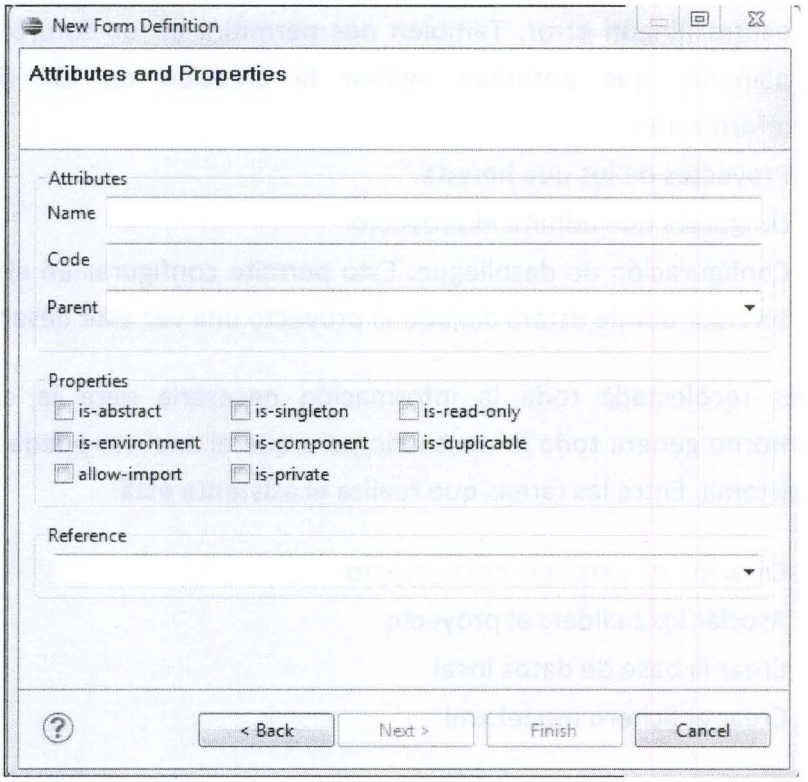


Ilustración 27 - Asistente de creación de formularios

Estos asistentes controlan que el usuario complete los campos que son obligatorios y además comprueban una serie de restricciones con el fin de evitar errores en el sistema de información.

A continuación podemos ver el resultado de la creación de un fichero cuando usamos el asistente.

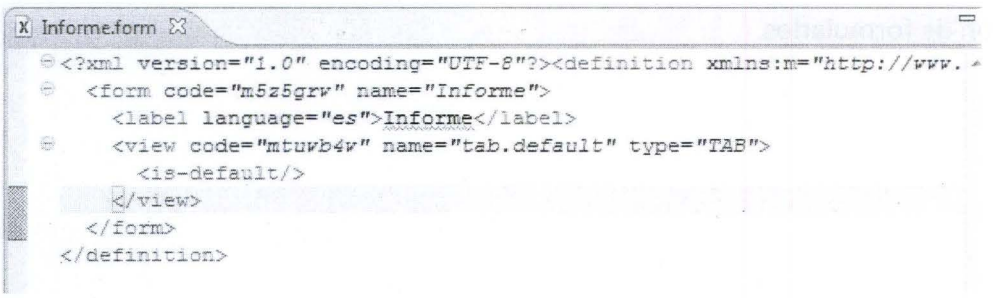


Ilustración 28 - Detalle de un fichero de tipo form

Como podemos apreciar en la imagen anterior se genera una serie de código extra que es obligatorio para este tipo de ficheros pero que no es necesario que el usuario rellene, con lo cual mejoramos la productividad y evitamos errores de compilación.

## 6.6 VISTAS

Muchos plug-ins añaden una vista de Eclipse nueva o mejoran una existente para proporcionar información al usuario.

Las vistas comparten un conjunto de comportamientos comunes con los editores pero también tienen diferencias notables. Cualquier acción llevada a cabo en una vista debe afectar de inmediato al estado del entorno de trabajo y al recurso o recursos subyacentes, mientras que los editores siguen el paradigma clásico de abrir-modificar-guardar. Los editores se muestran en un área de Eclipse, mientras que las vistas se organizan alrededor del área del editor.

Como el entorno de trabajo puede llegar a tener cientos de vistas, éstas se organizan en categorías.

A continuación pasamos a describir los pasos a seguir para declarar una vista en el entorno Eclipse y detallaremos las vistas que se han desarrollado para completar el entorno de modelado.

### 6.6.1 Extensión de la vista

Para crear una vista hay que seguir tres pasos:

1. Definir la categoría de la vista.
2. Definir la vista
3. Crear el comportamiento de la vista.

#### *6.6.1.1 Declarar la categoría de una vista*

Para definir una categoría de vista nueva, editamos el manifiesto del plug-in añadiendo la extensión *org.eclipse.ui.views.* Dentro de esta extensión añadimos la nueva categoría Monet, donde estarán alojadas todas las vistas del entorno de modelado.

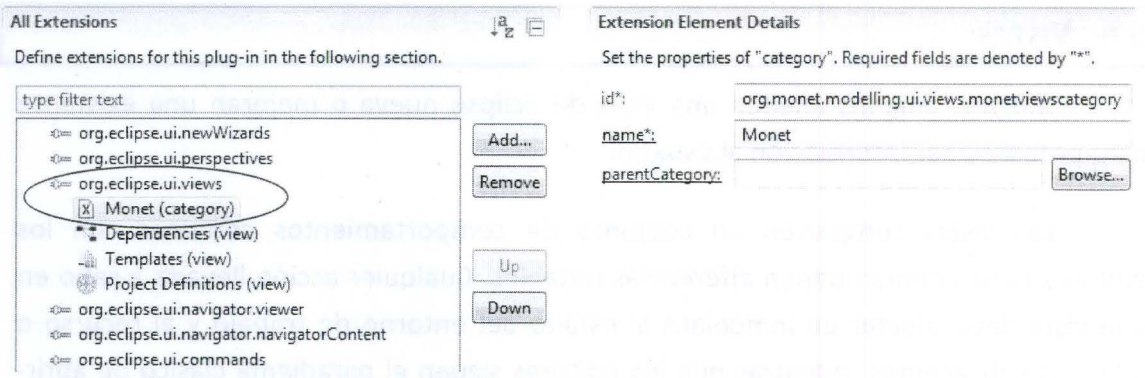


Ilustración 29 - Detalle de la declaración de la categoría Monet

### 6.6.1.2 Declarar una vista

Una vez definida la categoría, procedemos a añadir las vistas necesarias. En el entorno de modelado se crearán tres vistas: Dependencies, Templates y Project Definitions, que detallaremos en apartados posteriores.

### 6.6.1.3 Crear el comportamiento de la vista

El código que define el comportamiento de una ventana se suele encontrar en una clase que implementa la interfaz *IViewPart*, para lo que generalmente se crea una subclase de la clase abstracta *ViewPart*.

Las vistas suelen tener su propio modelo interno, que es el que usan para representar los objetos que se quieren visualizar en ellos. Cada una de las vistas del entorno de modelado tiene su propio modelo interno.

### 6.6.2 Vista de dependencias

Las definiciones de Monet pueden estar relacionadas entre sí de varias maneras: herencia, contenido, referencia, etc.

Este tipo de relaciones implica que un cambio en una definición puede afectar a todas las definiciones que dependan de la misma. Cuando disponemos de sistemas de información grandes o con muchas definiciones, el buscar estas definiciones dependientes puede provocar una pérdida constante de tiempo y esfuerzo por parte del ingeniero de sistemas. Este es el principal motivo por el cual se desarrolla esta vista

El objetivo de la misma es controlar que recurso está siendo usado por el analista y muestra el listado de las definiciones que de alguna manera están relacionadas con el recurso. Hay que tener en cuenta que para controlar este tipo de dependencias se hace uso de la base de datos del proyecto, que es la que contiene toda la información de interés de las definiciones. La base de datos se actualiza cada vez que se compila el proyecto por lo que debemos tener en cuenta que cuando se añaden o eliminan definiciones, la base de datos se mantendría en un estado obsoleto hasta que el compilador refleje los cambios efectuados en el proyecto.

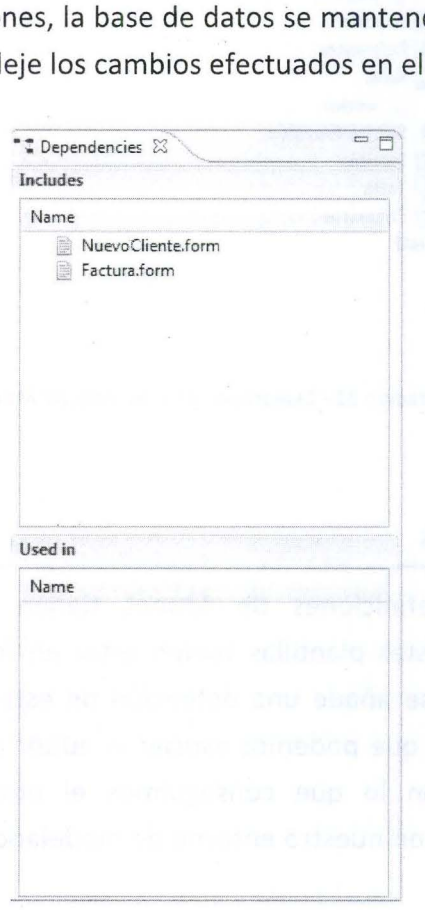


Ilustración 30 - Vista de dependencias

### 6.6.3 Explorador de proyectos

Eclipse dispone de diferentes vistas que permite visualizar los proyectos que actualmente se encuentran en el entorno de trabajo.

Nuestra idea es crear una vista que solo visualice los proyectos basados en la plataforma Monet. Esta vista además organiza los recursos del sistema de información de manera que el analista visualiza las definiciones ordenadas por tipo para que le resulte más fácil localizar los recursos que necesite.

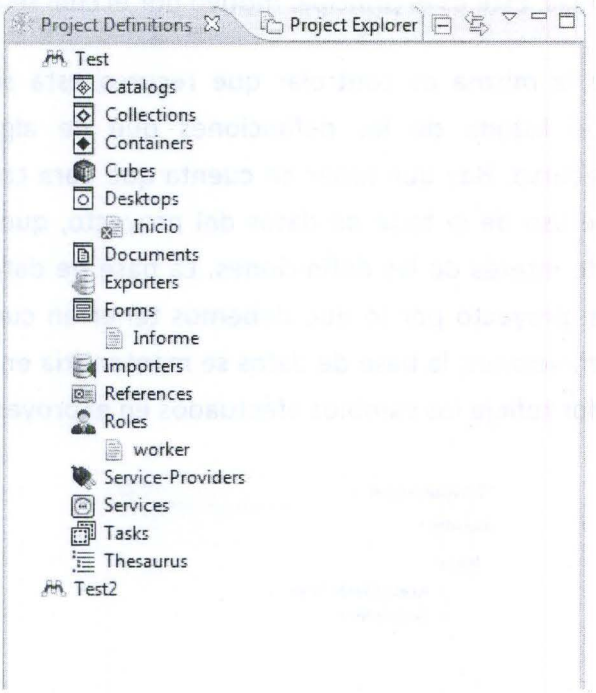


Ilustración 31 - Explorador de proyectos de Monet

### 6.6.4 Vista de templates

Algunas de las definiciones de Monet tienen asociados una serie de documentos o plantillas. Estas plantillas suelen estar en formato docx y se generan automáticamente cuando se añade una definición de este tipo. Una de las ventajas que nos permite Eclipse es que podemos asociar el editor con el cual queremos abrir esta serie de ficheros con lo que conseguimos el objetivo de tener todas las herramientas centralizadas en nuestro entorno de modelado.

La vista templates tiene como finalidad mostrar todas las plantillas que se encuentran asociadas a los proyectos de Monet y permitir al usuario abrir el editor que tiene configurado para este tipo de ficheros. Otra utilidad de la vista es que permite abrir la definición asociada a esta plantilla lo que permite mejorar la productividad del usuario a la hora de localizar los recursos que necesita.

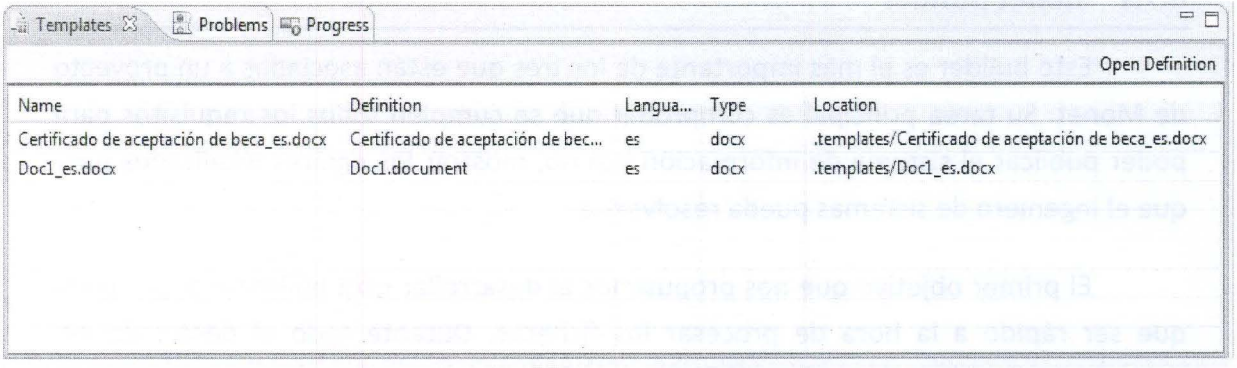


Ilustración 32 - Detalle de la vista de plantillas

## 6.7 GENERADORES

Los generadores (builder) de proyectos incrementales, se ejecutan de forma automática siempre que se modifica un recurso en un proyecto asociado. Por ejemplo, cuando se crea o modifica un archivo de código fuente Java, el compilador de Java incremental de Eclipse anota el archivo fuente y construye un archivo de clase. Los archivos de clase Java se conocen como recursos derivados porque el compilador los puede volver a construir a partir de archivos fuente Java.

Los generadores están limitados a un proyecto. Cuando uno o más recursos de un proyecto cambian, se notifica a los generadores asociados al proyecto. Si los cambios se han agrupado por lotes, el generador recibe una notificación que contiene una lista de todos los recursos modificados en lugar de notificaciones individuales de cada recurso. Los generadores procesan la lista de modificaciones y actualizan su estado de construcción para lo que vuelven a construir los recursos derivados necesarios, marcar recursos fuentes, etc. Los generadores reciben una notificación cuando un recurso cambia, como cuando un usuario guarda un archivo fuente modificado, y por lo tanto se ejecutan con bastante frecuencia. Por este motivo, un generador se debe ejecutar de forma incremental, es decir, sólo debe volver a construir aquellos recursos derivados que se hayan modificado.

Un proyecto de Monet tiene asociado 3 builders o generadores. Uno de ellos es el builder de Java que utilizamos en Monet para procesar los ficheros de

comportamiento del sistema. Los otros dos builders son para procesar los ficheros de definición del sistema, comprobar las restricciones, crear recursos derivados y empaquetar el resultado final siempre y cuando no se hayan producido errores.

### 6.7.1 Monet Builder

Este builder es el más importante de los tres que están asociados a un proyecto de Monet. Su tarea principal es comprobar que se cumplen todos los requisitos para poder publicar el sistema de información y si no, mostrar los errores localizados para que el ingeniero de sistemas pueda resolverlos.

El primer objetivo que nos propusimos al desarrollar este builder era que tenía que ser rápido a la hora de procesar los ficheros. Durante todo el desarrollo del prototipo se han llevado a cabo diferentes soluciones para conseguir agilizar el tiempo de ejecución de este proceso.

El builder está compuesto por un pipeline de 20 etapas, cada una con una funcionalidad diferente. En la imagen siguiente puede verse el conjunto de etapas que componen este builder.

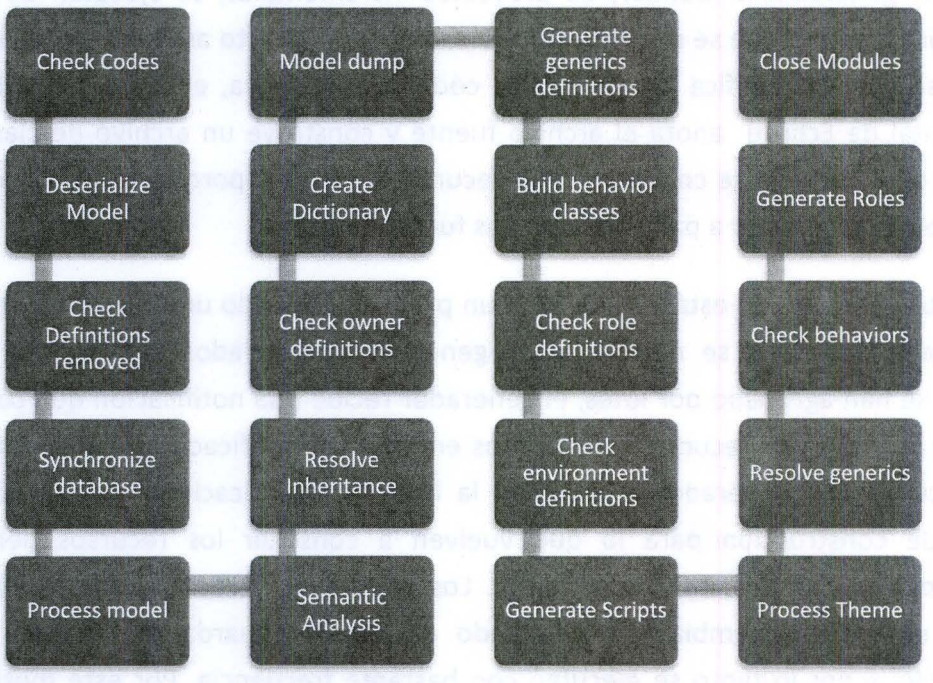


Ilustración 33 - Pipeline del builder de Monet

El objetivo de estas etapas es controlar todos los posibles errores que pueda cometer el ingeniero de sistemas.

Las tareas principales que se realizan durante estas etapas son las siguientes:

- Control de códigos de definición.
- Actualizar la base de datos local del proyecto.
- Crear el diccionario del proyecto.
- Chequear todas las restricciones semánticas.
- Resolver la herencia entre definiciones.
- Construir los comportamientos.
- Resolver las declaraciones genéricas.

El builder se ejecuta de manera incremental afectando solamente a los recursos que hayan sido modificados por el usuario y los recursos que tengan alguna relación de dependencia con los modificados por el usuario. Con esto conseguimos que proyectos sumamente grandes (del orden de 200 ficheros) se puedan compilar en un tiempo que no afecte de manera significativa a la productividad del ingeniero.

### 6.7.2 Packaging Builder

Este builder es el último en ejecutarse y solo lo hace si no se han localizado errores en los procesos de compilación anteriores. Su tarea principal es empaquetar los recursos necesarios para desplegar el proyecto en el servidor apropiado.

## 6.8 MARCADORES

Los marcadores se utilizan para anotar ubicaciones en un recurso. Por ejemplo, el compilador de Java de Eclipse no sólo crea archivos de clase a partir de archivos fuente, sino que también realiza anotaciones en los archivos de código fuente añadiendo marcadores para indicar errores de compilación, utilización de código obsoleto, etc.

En el caso que nos ocupa, usaremos los marcadores para visualizar errores que puedan cometerse en los ficheros xml y que detectan los builders que vimos en el apartado anterior.

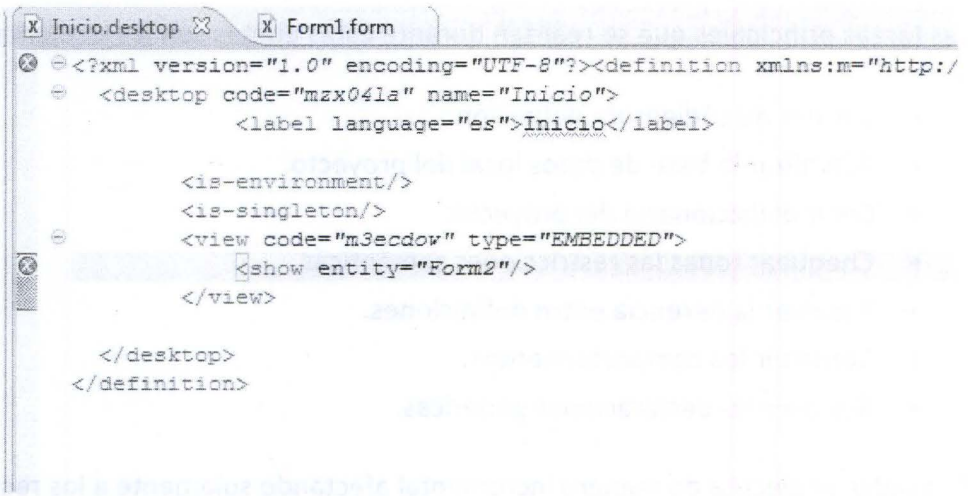


Ilustración 34 - Marcadores en el editor

Los errores detectados por el builder se podrán visualizar en la vista de problemas del Eclipse facilitando al ingeniero de sistemas la localización de los errores y mostrándole un mensaje para identificar la causa del problema.

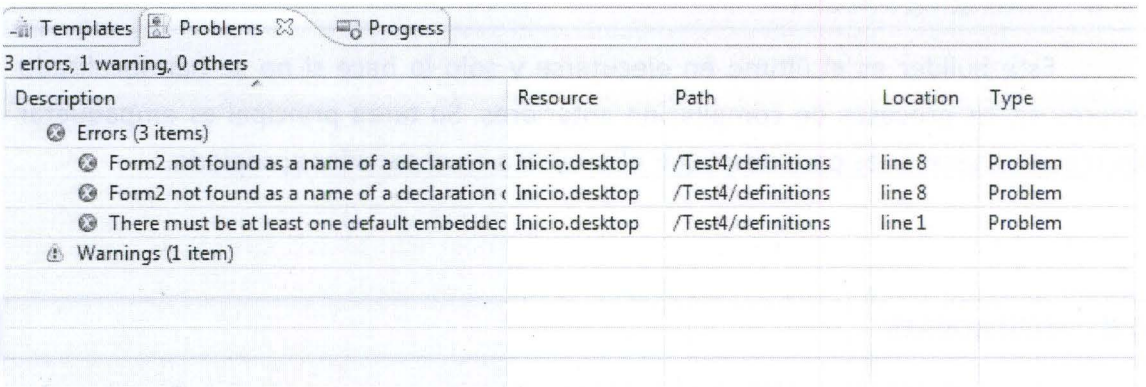


Ilustración 35 - Detalle de la vista de problemas

También se pueden detectar los errores en la vista del explorador de proyectos mediante un icono significativo.


Ilustración 36 - Detalle de error en la vista de proyectos

## 6.9 NATURALEZAS

Las naturalezas de proyectos se utilizan para asociar proyectos y generadores. La naturaleza Java de un proyecto lo convierte en un proyecto Java y asocia el compilador de Java incremental de Eclipse.

Un proyecto de Monet tendrá incluido dos naturalezas, la mencionada anteriormente de Java y una propia que asociará el proyecto Monet con los builders construidos específicamente para este tipo de proyectos. Gracias a esto conseguimos compilar los dos tipos de recursos que podemos tener en nuestros proyectos. En la siguiente imagen podemos ver el estado del fichero de proyecto con los builders y las naturalezas asociados.

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>Monet2</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>org.monet.modelling.builders.monetbuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.eclipse.jdt.core.javabuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.monet.modelling.builders.packagingbuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
    </buildSpec>
    <natures>
        <nature>org.monet.modelling.nature.monetnature</nature>
        <nature>org.eclipse.jdt.core.javanature</nature>
    </natures>
</projectDescription>
```

Ilustración 37 - Detalle del fichero de proyecto

Una naturaleza también se puede utilizar para determinar si una acción debería ser visible. Mientras que un marcador tiene una funcionalidad limitada pero se puede aplicar a cualquier recurso, una naturaleza está diseñada para contener una funcionalidad adicional que se puede aplicar a proyectos. Un marcador se aplica a sólo a un recurso en un espacio de trabajo, mientras que una naturaleza forma parte de un proyecto y, por lo tanto, varios desarrolladores la pueden compartir.

Gracias a esta funcionalidad podemos limitar los proyectos que podemos visualizar en la vista de proyectos para que solo se muestren los proyectos de Monet.

### 6.9.1 Declaración de la naturaleza

Para declarar una naturaleza tenemos que añadir al plug-in la extensión *org.eclipse.core.resources.natures*.

La declaración de naturaleza contiene el identificador local de la naturaleza. La naturaleza de Monet quedaría de la siguiente manera:

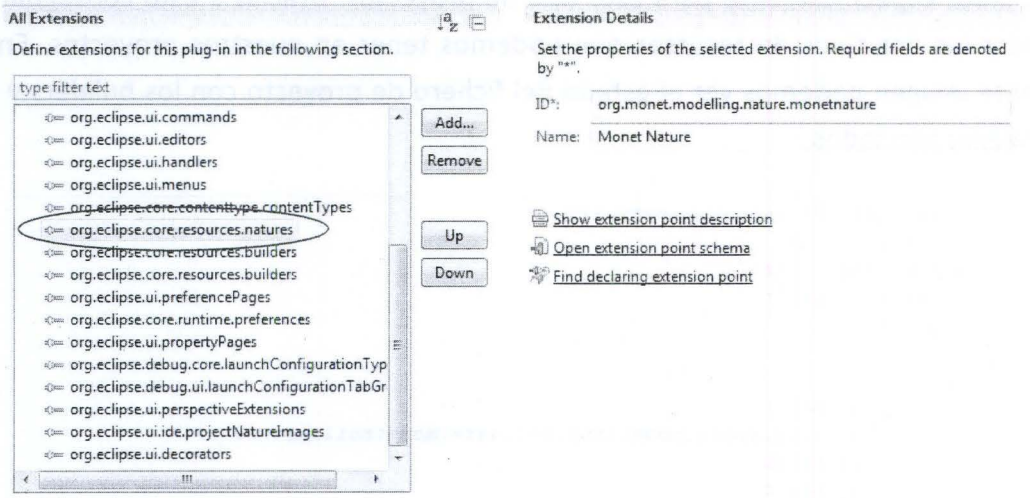


Ilustración 38 - Detalles de la naturaleza de Monet

### 6.9.2 Asociar generadores y naturalezas

Monet, como vimos anteriormente tiene asociados dos builders, que se encargan de compilar y construir el modelo para su posterior publicación. El desarrollar una naturaleza nos permite asociar a los proyectos los builders correspondientes. En la siguiente ilustración podemos apreciar como la naturaleza de Monet tiene asociado los builders correspondientes.

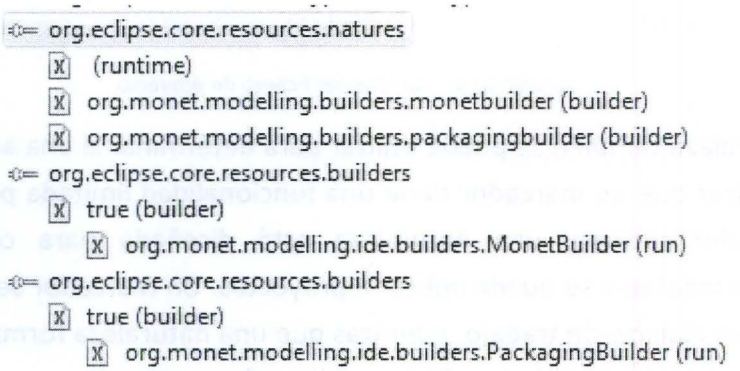


Ilustración 39 - Detalle de la naturaleza y los builders asociados

## 6.10 PÁGINA DE PROPIEDADES DEL PROYECTO

La página de propiedades es un diálogo especial que permite configurar determinados aspectos de un proyecto de Monet. La parte más importante de este diálogo es que nos facilita la tarea de configurar la ruta del servidor donde se publicará nuestro sistema de información.

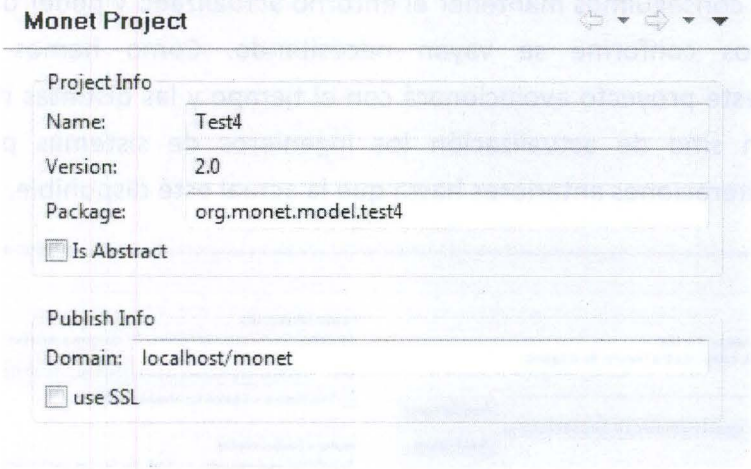


Ilustración 40 - Página de propiedades del proyecto

Esta página al igual que otras extensiones del proyecto está preparada para que en futuras iteraciones se pueda trabajar con diferentes versiones del lenguaje.

## 6.11 PUBLICACIÓN

Los proyectos de Monet se pueden configurar de manera que el analista pueda publicar los sistemas de información en servidores, ya sean locales o externos. Una de las herramientas que incorporamos fue la posibilidad de que el analista compilara y publicara los proyectos en la misma acción. Antes de publicar el proyecto, la aplicación debe comprobar que no se han producido errores durante la compilación.

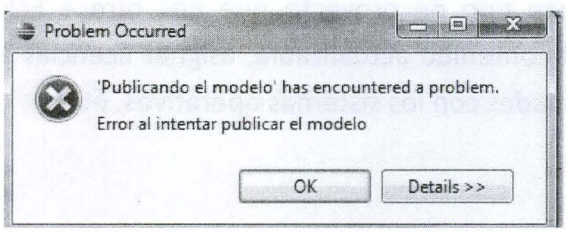


Ilustración 41 - Error durante la publicación del modelo

## 6.12 SITIO DE ACTUALIZACIÓN

Es posible agrupar uno o más plug-ins Eclipse en una característica de Eclipse. Una vez empaquetados como una característica, podremos cargar y descargar los plug-ins que componen el entorno de modelado como una unidad con el Update Manager de Eclipse.

Con esto conseguimos mantener el entorno actualizado y poder desplegarlo en distintos equipos conforme se vayan necesitando. Como hemos mencionado anteriormente este proyecto evolucionará con el tiempo y las distintas iteraciones. Al disponer de un sitio de actualización los ingenieros de sistemas podrán seguir trabajando con iteraciones anteriores hasta que la actual esté disponible.

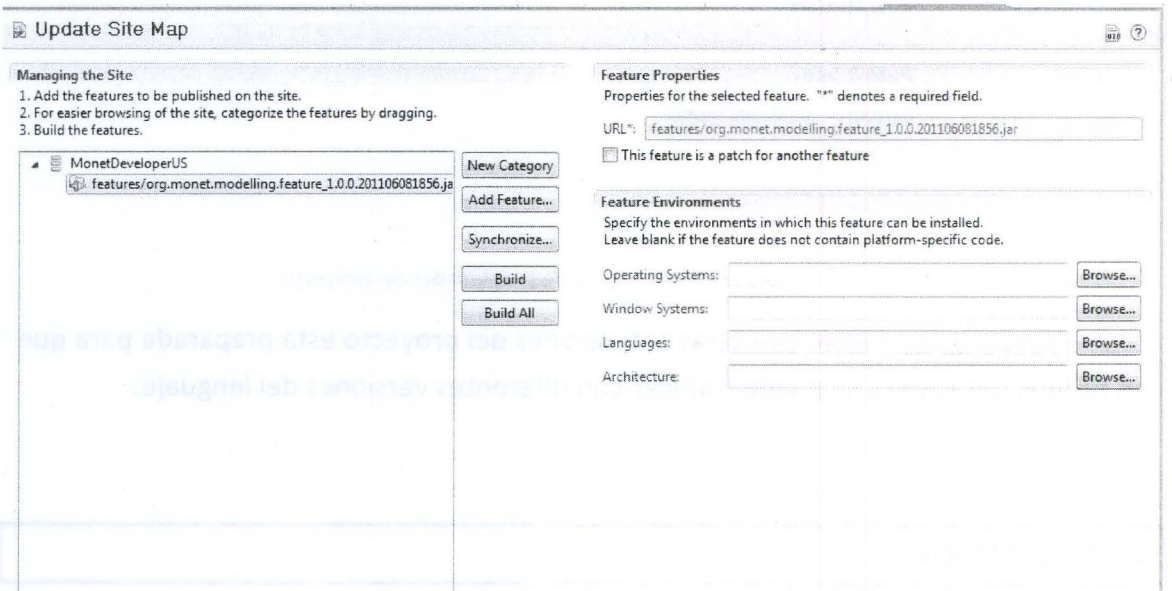


Ilustración 42 - Editor del sitio de actualización

Gracias a este tipo de proyecto que nos ofrece Eclipse se pueden generar páginas web con el contenido actualizable, asignar licencias al software desarrollado, rellenar compatibilidades con los sistemas operativos, etc.

El resultado final son una serie de archivos que conforman el producto que posteriormente se podrá descargar el usuario.

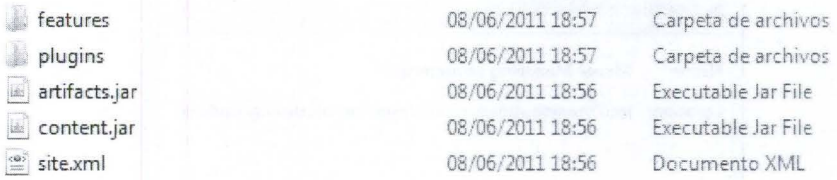


Ilustración 43 - Producto construido

## 6.13 PROCESO DE INSTALACIÓN DEL ENTORNO DE MODELADO

Como vimos en el apartado anterior, el entorno de modelado consta de un sitio de actualización desde el cual se puede descargar e instalar en cualquier entorno de Eclipse.

Para hacer la instalación, seleccionamos en el menú de Eclipse *Help --> Install New Software...*

Pulsamos sobre "*Available Software Sites*".

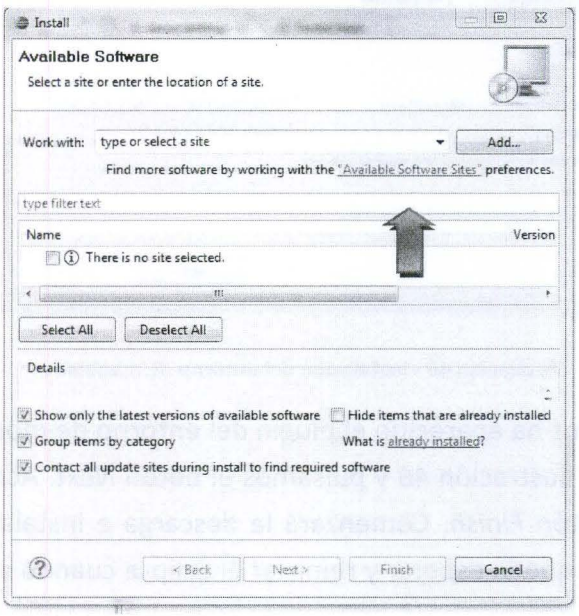


Ilustración 44 - Diálogo de instalación de nuevo software en Eclipse

Ahora sobre el botón *Add...,* y añadimos la URL donde se encuentra alojado el entorno de modelado.

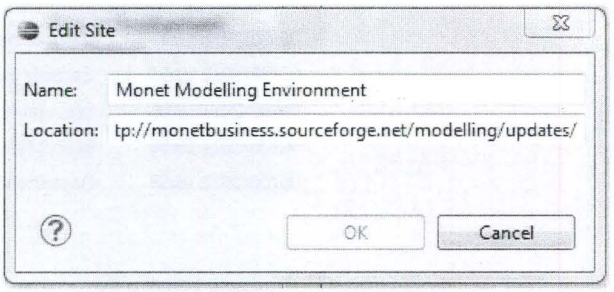


Ilustración 45 - Monet Modelling Environment URL

Damos al botón *Ok,* volviendo a la ventana anterior con el listado de todas las URL, debería aparecer el que acabamos de añadir, así que volvemos a dar sobre el botón *Ok.* Volviendo a la ventana de instalación, seleccionamos el sitio que acabamos de añadir en la pestaña de *Work with:*

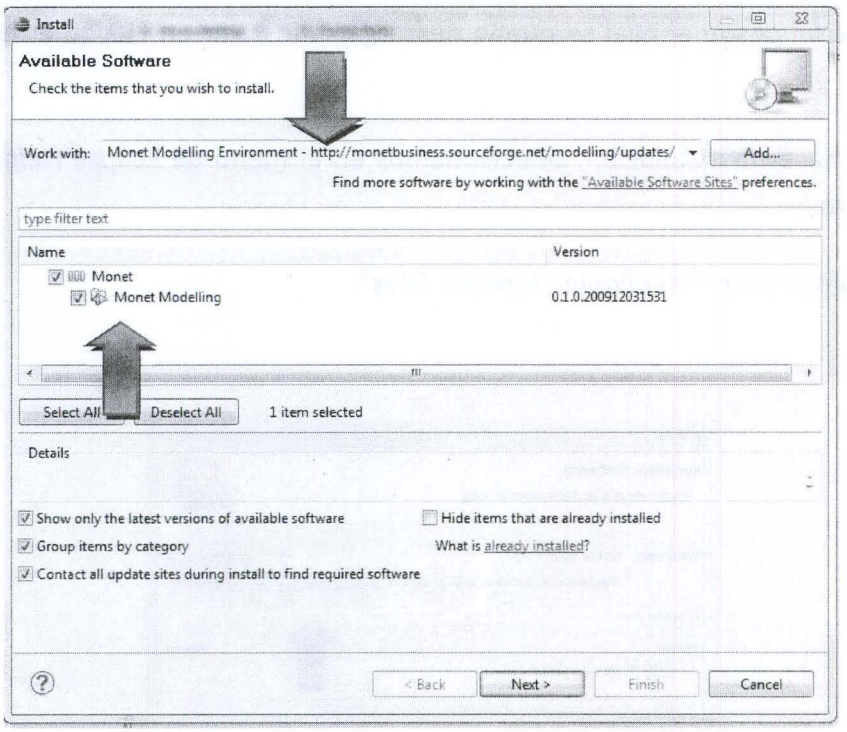


Ilustración 46 - Instalación del entorno de modelado

Vemos como nos ha aparecido el plugin del entorno de modelado, lo marcamos tal y como se ve en la Ilustración 46 y pulsamos el botón *Next.* Aceptamos la licencia y pulsamos sobre el botón *Finish.* Comenzará la descarga e instalación del entorno de modelado. Ahora sólo queda esperar y reiniciar el Eclipse cuando se nos indique.

# 7. CONCLUSIONES

Las conclusiones que puedo extraer del desarrollo de este trabajo son varias. La de mayor importancia es que gracias a este trabajo he podido comprender y compartir la visión del desarrollo dirigido por modelos, como una metodología de desarrollo muy potente.

Durante este desarrollo he podido comprobar que los requisitos que acompañan a este tipo de software son muy cambiantes. Más aún cuando los usuarios empiezan a experimentar con la aplicación. Estos cambios conllevan varias dificultades, ya que se deben realizar ajustes o modificaciones en diferentes partes de la aplicación , lo que puede introducir errores en la aplicación.

En mi corta experiencia como desarrollador de software a medida, he podido comprobar como cualquier cambio mínimo en los requisitos de la aplicación repercutía en la productividad de los ingenieros de sistemas. Cuando no se disponía de herramientas eficaces, los ingenieros de sistemas se veían obligados a soportar tiempos de espera enormes y a buscar errores a ciegas, lo que muchas veces implicaba la paralización completa del desarrollo del sistema.

Con el desarrollo del entorno se consiguió mejorar la productividad y la eficacia de todo el sistema. Entre las mejoras considerables que se consiguieron podemos destacar:

- Reducción del tiempo de compilación.
- Reducción del tiempo de publicación.
- Reducción de la localización y corrección de los errores.
- Mejora en el control y mantenimiento de las diferentes versiones.
- Mejora en el tiempo de desarrollo del modelo.
- Mejora en el trabajo colaborativo.

Estas mejoras supusieron un alivio y agilizaron de una manera sorprendente el desarrollo del sistema cumpliendo en la mayoría de las veces con los plazos establecidos.

No obstante, el principal inconveniente que podemos identificar en esta herramienta es el tiempo que se necesita para desarrollar este software. Se necesita un equipo de desarrollo que permita que este software sea productivo y mantenerlo

constantemente actualizado ya que necesitamos que el entorno avance paralelamente con el modelo.

Cualquier cambio que queramos introducir en el modelo, antes debe ser soportado por el entorno de modelado, lo que supone un tiempo para analizar esa mejora, desarrollarla y lo más importante probarla para asegurarnos de su correcto funcionamiento.

Como opinión personal, valoro enormemente la experiencia adquirida gracias al desarrollo de este proyecto y al grupo humano donde pude realizar las pruebas del entorno. Ha sido un trabajo duro y largo, lleno de altibajos, pero me siento muy satisfecho por haber conseguido un entorno funcional y útil que está siendo usado para llevar a cabo proyectos reales basados en la plataforma Monet.

También valoro muy positivamente el trabajar en una línea de investigación novedosa como es la ingeniería dirigida por modelos y en todo lo relacionado con el desarrollo de sistemas de información.

He podido aplicar a este proyecto todo lo aprendido a lo largo de la carrera. Considero que he mejorado mis habilidades de investigación así como las de desarrollo de software y planificación.

# 8. TRABAJO FUTURO

Este campo del desarrollo dirigido por modelos es sin duda un campo por el que se abrirán infinidad de líneas de investigación, desde la interoperabilidad con sistemas existentes, los mapas de trabajo o la automatización de ciertas tareas haciendo uso de agentes inteligentes que puedan realizar ciertas tareas dentro del sistema de información.

Desde un punto de vista del entorno de modelado quedan varias utilidades que se pueden incluir en futuras iteraciones e incluso mejorar las existentes. A continuación detallo algunas de ellas:

- Editor gráfico con asistente de contenido.
- Integración de la ayuda para los distintos tipos de objetos
- Multilenguaje
- Mejorar la publicación de los modelos proporcionando vistas para configurar los destinos.
- Mejorar el sistema de marcas en los editores y añadir la corrección automática en los que se pueda.
- Mejorar el compilador introduciendo procesamiento en paralelo de las etapas que lo permitan.
- Introducir el depurador

# 9. ANEXO I

## 9.1 ECLIPSE

Como es definida en la página web oficial: "*The Eclipse Platform is an IDE for everything and nothing in particular*" (la plataforma Eclipse es un IDE para todo y nada en particular). Es una poderosa herramienta que permite integrar diferentes aplicaciones para construir entornos de desarrollo integrado (IDEs) que pueden ser utilizados para la construcción de aplicaciones Web, Java, C/C++, etc., dando a los desarrolladores la libertad de elegir en un entorno multilenguaje y multiplataforma. Es un proyecto de desarrollo de software *open-source*, que está dividido en tres partes:

- ***The Eclipse Project:*** es un proyecto de desarrollo de software libre destinado a proporcionar una plataforma de desarrollo de herramientas integradas robusta, completa y comercial. Se subdivide, a su vez, en tres subproyectos:
  - la propia plataforma, que contiene las herramientas Eclipse.
  - JDT (*Java Development Toolkit)*: añade a la plataforma un IDE de Java completamente equipado, incluyendo: editor, compilador y depurador.
  - PDE (*Plug-in Development Environment):* es un conjunto de herramientas diseñadas para ayudar al desarrollador de Eclipse en las tareas de desarrollo, prueba, depuración, construcción y distribución de plug-ins.
- ***The Eclipse Tools Project:*** su misión es fomentar la creación de una gran variedad de herramientas, proporcionando un punto de coordinación entre los desarrolladores para minimizar la duplicación y promover la interoperabilidad entre los diversos tipos de herramientas.
- ***The Eclipse Technology Project:*** su misión es proporcionar nuevos canales de comunicación a desarrolladores, profesores y educadores para participar en la evolución de Eclipse. Está organizado en tres proyectos relacionados:
  - *Research:* investigación en dominios relacionados con Eclipse, tales como lenguajes de programación, herramientas y entornos de desarrollo.
  - *Incubators:* son pequeños proyectos que añaden capacidades al software base de Eclipse.
  - *Education:* estos proyectos se centran en el desarrollo de material de ayuda.

### 9.1.1 Objetivos de la Plataforma Eclipse

La plataforma Eclipse está diseñada para afrontar las siguientes necesidades:

- Soportar la construcción de gran variedad de herramientas de desarrollo.
- Soportar herramientas proporcionadas por diferentes fabricantes de software independientes (ISVs).
- Soportar herramientas que permitan manipular diferentes contenidos (HTML, Java, C/C++, JSP, XML...).
- Facilitar una integración transparente entre todas las herramientas y tipos de contenidos sin tener en cuenta al proveedor.
- Proporcionar entornos de desarrollo gráficos (GUI) y no gráficos (non-GUI).
- Compatibilidad con una gran variedad de sistemas operativos.
- Fomentar el uso de Java para el desarrollo de nuevos plug-ins.

El principal objetivo de la plataforma Eclipse es proporcionar mecanismos y reglas que permitan a los desarrolladores integrar de manera transparente sus herramientas. Estos mecanismos son proporcionados a través de APIs.

### 9.1.2 Arquitectura de la Plataforma Eclipse

Considerándola desde términos de diseño, la Plataforma Eclipse no ofrece una gran funcionalidad por sí sola, sino que su valor real yace en el modelo de plug-ins (unidades mínimas de funcionalidad, explicadas en apartados posteriores). Con lo cual, Eclipse está estructurada como un conjunto de subsistemas, los cuales son implementados en uno o más plug-ins que se ejecutan sobre una pequeña plataforma de ejecución (Ver Ilustración 47). Dichos subsistemas definen puntos de extensión para permitir agregar funcionalidad a la plataforma. A continuación se describen los principales componentes de la plataforma.

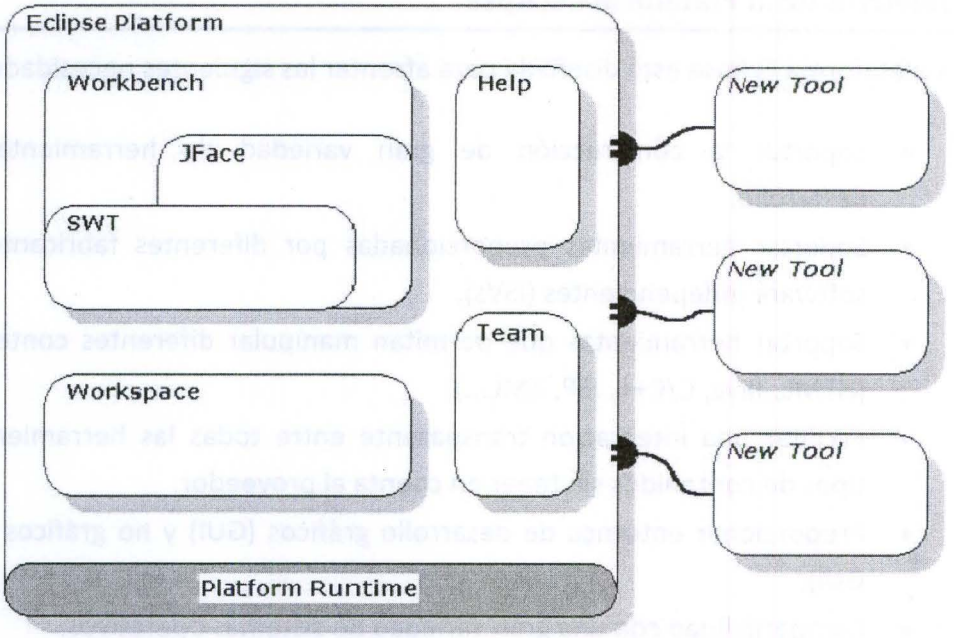


Ilustración 47 - Arquitectura de la plataforma Eclipse

### 9.1.2.1 *Plataforma de Ejecución (Platform Runtime)*

Se trata del único componente de Eclipse que no es un plug-in. Al iniciar la Plataforma de Ejecución se descubren de manera dinámica el conjunto de plug-ins disponibles, se leen sus archivos de manifiesto, y se construye en memoria un registro de plug-ins, que está disponible a través de la API de la plataforma. La plataforma mantiene el registro de aquellos plug-ins instalados así como de las funcionalidades que proveen. No podrán ser añadidos nuevos plug-ins después del inicio.

Para agregar nuevas funciones al sistema se usa un modelo de extensión común. Los "**puntos de extensión**" son lugares bien definidos dentro del sistema que permiten ser extendidos por los plug-ins. Cuando una herramienta contribuye con una implementación para un determinado punto de extensión se dice que agrega una "extensión" a la plataforma. A su vez, cada *plug-in* puede definir sus propios puntos de extensión de tal forma que puedan se extendidos por otros. Este mecanismo de extensión es la única manera de agregar funcionalidad a la plataforma. Las extensiones están típicamente escritas en Java utilizando la API de la plataforma. Sin embargo, algunos puntos de extensión tienen asociadas extensiones provistas de ejecutables o incluso extensiones que han sido desarrolladas mediante lenguajes *script*. De forma general, solo un subconjunto del total de la funcionalidad de la plataforma se encuentra disponible para extensiones no realizadas con Java.

Un objetivo muy importante del *Runtime* es que usuarios finales no sufran desventajas a causa del uso de memoria por aquellos plug-ins que, si bien están instalados, no están siendo usados. De esta manera, un plug-in puede ser instalado y agregado al registro pero el mismo no será activado a menos que se requiera mediante la actividad del usuario.

### *9.1.2.2 Workspace*

Se trata del bloque central, o espacio de trabajo, para los archivos regulares que son específicos de cada usuario, y sobre los que actúan las diferentes herramientas instaladas en la Plataforma.

El espacio de trabajo del usuario consta de uno o más proyectos donde cada uno se mapea en un directorio especificado por el usuario en el sistema de archivos. Cada proyecto contiene los archivos que son creados y manipulados por el usuario. Todos los archivos en el espacio de trabajo son directamente accesibles por programas estándar y herramientas del sistema operativo.

El conjunto de proyectos, archivos y carpetas que son generados por herramientas y almacenados en el sistema de archivos constituyen los **recursos.** Los recursos del workspace están organizados en una estructura de árbol, con los proyectos arriba y los archivos y carpetas por debajo. Existe un recurso especial, el *workspace root,* que funciona como raíz del árbol de recursos. Éste es creado internamente cuando el *workspace* es creado y existirá mientras éste exista.


Ilustración 48 - Árbol de recursos del Workspace

La Ilustración 48 representa la típica jerarquía de recursos en un *workspace.* La raíz implícita del árbol es el *workspace root* y cada uno de los proyectos es inmediatamente hijo suyo. Los proyectos pueden contener archivos y carpetas, pero

no otros proyectos. Los proyectos y carpetas funcionan como directorios de un sistema de archivos, es decir, si se borra un proyecto o una carpeta también serán borrados todos los recursos que contengan.

### *9.1.2.3 Workbench*

Implementa el aspecto visual que permite al usuario navegar por los recursos y utilizar las herramientas integradas. El *workbench* es simplemente un frame donde se presentan varias partes visuales. Estas partes se pueden dividir en dos categorías mayores: editores y vistas.

Los **editores** permiten al usuario abrir, editar y guardar objetos. El *workbench* incluye un editor sencillo de texto. A partir de puntos de extensión se pueden crear nuevos editores.

Las **vistas** proporcionan información sobre aquellos objetos con los que el usuario está trabajando en el *workbench.* Con lo cual, las vistas cambiarán su contenido al cambiar el objeto que seleccione el usuario.

Hasta ahora se ha descrito el *workbench* como: "esa ventana que se abre cuando inicias la Plataforma". pero en la Ilustración 49 se presenta una descripción un poco más concisa. A parte de las vistas y editores generales se encuentra el término **perspectiva.** En un *workbench* puede haber una o más perspectivas. Cada una consiste en una agrupación de vistas y editores que se presentan juntos en pantalla para que, desde el punto de vista del usuario, sea una ventana de *workbench* sencilla.

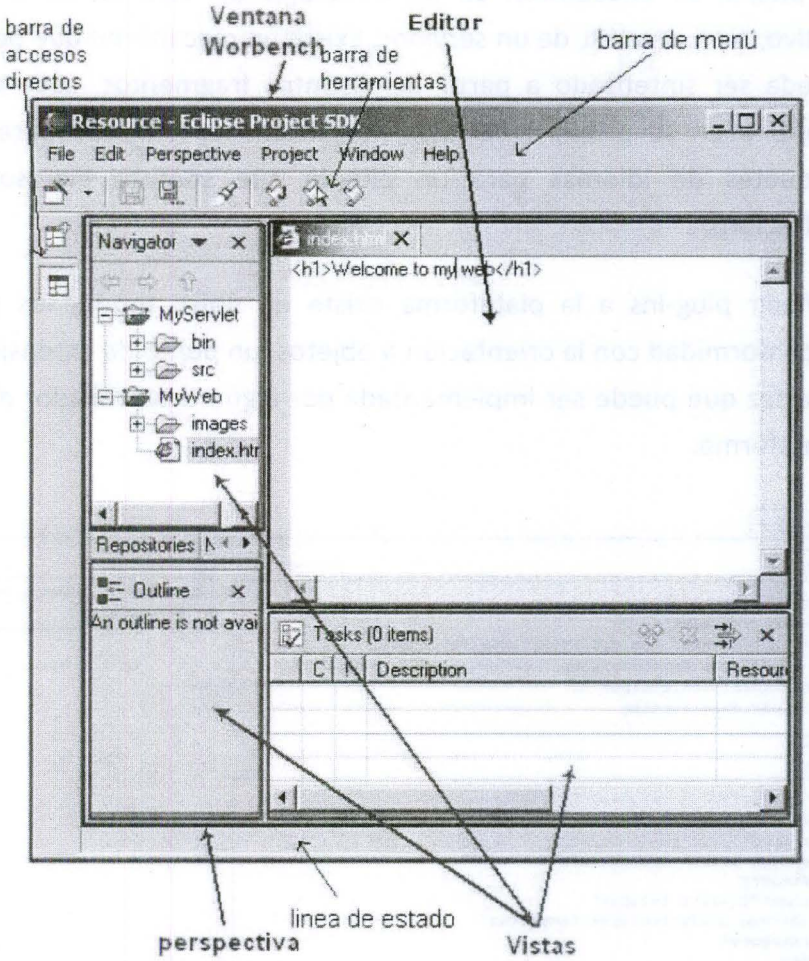


Ilustración 49 - Estructura de la ventana del Workbench

### 9.1.3 Arquitectura basada en plug-ins

#### *9.1.3.1 ¿Qué es un plug-in?*

Un plug-in es la unidad mínima de funcionalidad de Eclipse, que puede ser distribuida de manera separada. Herramientas pequeñas se escriben como un único plug-in, mientras que en las complejas la funcionalidad puede estar en varios plug-ins.

Los plug-ins, por lo general, están escritos en Java. Cada uno está formado por código Java, ficheros de lectura y otros recursos como imágenes, catálogos de mensajes, bibliotecas de código nativo, etc.

Existen algunos plug-ins que no contienen nada de código, por ejemplo: el plug-in que nos proporciona ayuda en forma de páginas HTML. Todos los recursos que

componen el plug-in se encuentran en un directorio del sistema de archivos del sistema operativo, o en una URL de un servidor. Existe un mecanismo que permite que un plug-in pueda ser sintetizado a partir de distintos fragmentos, cada uno en su propio directorio o en su propia URL. Este mecanismo es utilizado para distribuir diferentes paquetes de idiomas para un plug-in que soporte diversos idiomas **(internacionalización).**

Para añadir plug-ins a la plataforma existe un **único modo,** los puntos de extensión. En conformidad con la orientación a objetos, un punto de extensión no deja de ser una interfaz que puede ser implementada por algún desarrollador dispuesto a extender la plataforma.

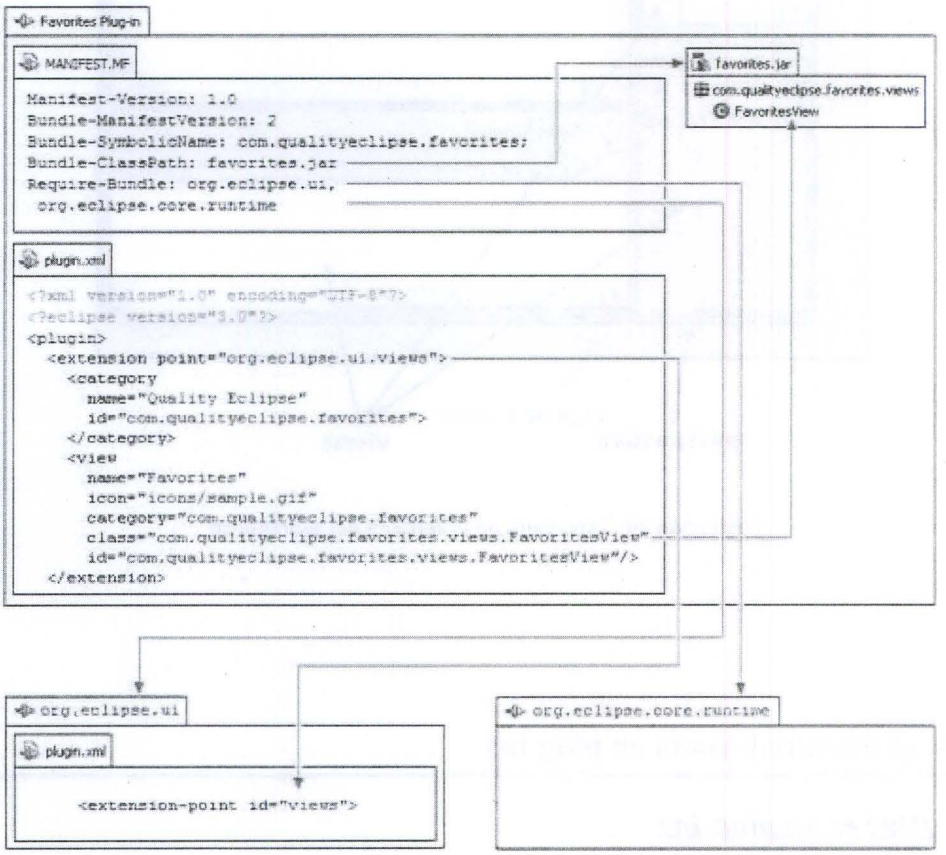


Ilustración 50 - Estructura básica de un plug-in.

### *9.1.3.2 Arquitectura y plug-in manifest*

Cada plug-in tiene un fichero denominado de manifiesto (*manifest*) en el cual se declaran sus interconexiones con otros plug-ins. La interconexión sigue un modelo muy simple: un plug-in declara un número de los denominados puntos de extensión, y un número de extensiones para uno o más puntos de extensión de otros plug-ins.

Los archivos de manifiesto de los plug-ins contienen XML. Un punto de extensión puede declarar tipos de elementos XML adicionales para ser utilizados en las extensiones. Esto permite el paso de datos entre plug-ins. Además, la información del archivo de manifiesto está disponible desde el registro de plug-ins sin haber activado los plug-ins o haber cargado algo de su código. Esto es fundamental para soportar un gran número de plug-ins además de marcar una diferencia notable, en cuanto a consumo de recursos, entre los IDEs comerciales y Eclipse. Hasta que el código del plug-in no es cargado, el efecto que tiene sobre el entorno de ejecución es nulo.

Los plug-ins no sólo extienden o amplían la plataforma base, también pueden extender, a su vez, otros plug-ins que hayan definido sus propios puntos de extensión. Es decir, un plug-in puede hacer interfaces públicas que otros plug-ins pueden implementar. Las implementaciones de las interfaces (llamadas extensiones) mostradas por los puntos de extensión se realizan típicamente en Java, aunque algunos puntos de extensión pueden acomodar extensiones proporcionadas por ficheros ejecutables nativos o componentes ActiveX; incluso pueden programarse en lenguajes script. El principal obstáculo con el cual se enfrentan las extensiones no realizadas en Java es la falta de acceso a la funcionalidad completa de la plataforma Eclipse.

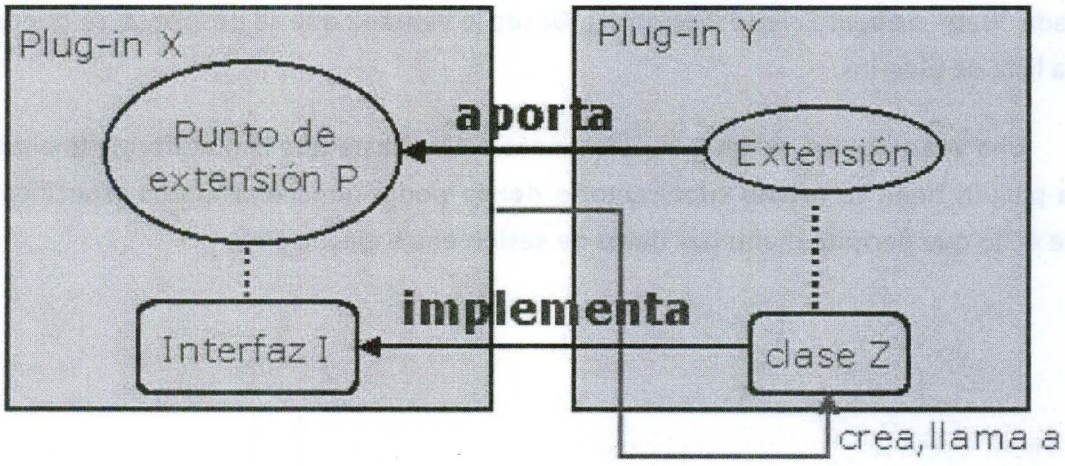


Ilustración 51 - Esquema de la extensión de un plug-in mediante otro plug-in

Como representa la ilustración anterior, la arquitectura de plug-ins en Eclipse viene marcada por los puntos de extensión y las aportaciones de los plug-ins a la plataforma así como entre ellos. En el ejemplo de la Ilustración 51, el *plug-in X* declara el punto de extensión P y la interfaz I asociada a P. Mientras que el *plug-in Y* implementa la interfaz I con su clase Z y por tanto, aporta la clase Z al punto de extensión P. Con lo cual, el *plug-in X* será el que instancia la clase Z y el encargado de llamar a sus I-métodos.

### *9.1.3.3 Activación de los plug-ins*

La plataforma se ejecuta en una única invocación de la máquina virtual Java (JVM). A cada plug-in se le asigna su propio cargador de clases (Java *classloader),* usado para cargar las clases del plug-in así como los recursos del mismo.

Un plug-in es activado cuando su código realmente necesita ser ejecutado. Una vez activado, un plug-in utiliza el registro de plug-ins para descubrir y acceder a las extensiones que contribuyen a sus puntos de extensión. Por ejemplo, el plug-in que declara el punto de acceso de preferencias de usuario puede descubrir todas las preferencias de usuario que contribuyen en Eclipse y mostrarlas para construir un cuadro de diálogo con todas ellas. Esto puede ser realizado, sin necesidad de cargar el código de esos plug-ins, simplemente consultando el registro de plug-ins. El plug-in será activado solo cuando el usuario seleccione la preferencia de la lista. Esto nos permite eliminar largos tiempos de arranque y es una solución de gran escalabilidad.

Es posible saber, qué plug-ins están activos en un momento determinado, para ello, es necesario lanzar la ayuda de Eclipse (desde el menú principal, seleccionando la entrada "*Help-->About Eclipse Platform*"). Desde la ventana que se despliega, se puede ver la lista de plug-ins.

Una vez activado, un plug-in permanece activo hasta que la plataforma finaliza. Cada plug-in tiene su propio subdirectorio donde poder almacenar datos específicos sobre él, lo que permite mantener datos de sesión entre ejecuciones.

# 10. ANEXO II

Uno de los requisitos que se necesitan para elaborar un entorno de modelado, es decidir el lenguaje que usarán los analistas para construir el sistema de información. Los sistemas de información que se podrán desarrollar en nuestro entorno de modelado están basados en la plataforma Monet. Esta plataforma está basada en el principio de ingeniería dirigida por modelos y permite el desarrollo de modelos de sistema de información y su ejecución en entornos web.

## 10.1 LENGUAJE DE MODELADO

Monet es una plataforma que pretende implementar la metodología de la ingeniería dirigida por modelos, si bien se diferencia de la línea seguida por MDA en que el sistema hace la interpretación de los modelos para generar un sistema plenamente operativo en lugar de la traducción o compilación de los modelos para generar un programa compilado.

En el ámbito de las organizaciones son más importantes atributos como la flexibilidad a los cambios y la agilidad en el desarrollo frente a la eficiencia computacional del código. Un programa compilado se ejecutará más deprisa que uno interpretado, pues el compilador ha producido previamente todo el código máquina necesario, y normalmente ha llevado a cabo un proceso de optimización. No obstante, la compilación tiene sentido cuando es necesario cuidar la eficiencia del código porque estamos tratando con sistemas que presentan una demanda significativa de recursos computacionales [16].

En el ámbito de las organizaciones, los recursos computacionales no son los críticos; es mucho más crítico que el sistema sea flexible para admitir cambios o que el desarrollo sea ágil.

### 10.1.1 Unidad de negocio y modelo de negocio

Una unidad de negocio en la arquitectura de la información propuesta por Monet es una entidad atómica dentro de la organización que cumple una determinada función o competencia, ofreciendo al exterior una serie de servicios cuyos clientes serán clientes finales u otras unidades de negocio. Para la consecución de sus objetivos, la unidad de negocio se apoyará en recursos humanos que colaboran en una

estructura horizontal. Cada persona tendrá una función y esta podría ser sustituida por un agente virtual, que delegaría esa función en otra unidad de negocio.

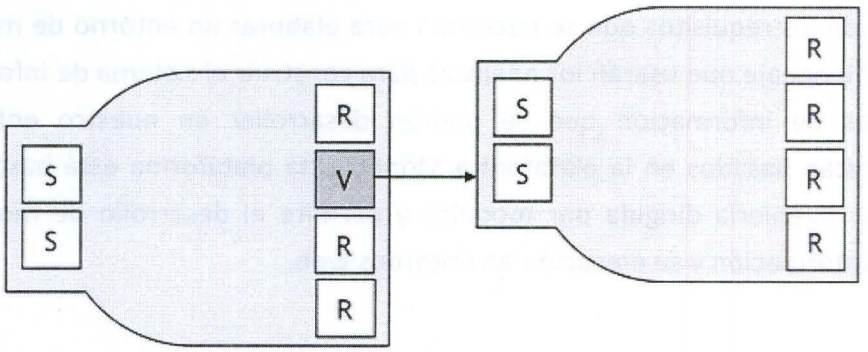


Ilustración 52 - Unidades de negocio dependientes

El modelo de negocio es el modelo que interpretará Monet y que describe completamente el funcionamiento de la unidad de negocio. Un modelo de negocio en Monet es un conjunto de definiciones, representados en XML, que definen cada uno de los aspectos estáticos de la unidad de negocio. Las definiciones representan entidades concretas del sistema de información como servicios, formularios, documentos, tareas, colecciones, catálogos o informes. Las definiciones forman parte de una arquitectura similar a la orientación a objetos donde cada definición puede tener una definición padre de la que hereda sus características. También existe la figura de definición abstracta, que impone además la restricción de que debe ser heredada por otra definición ya que no es posible la creación de un objeto a partir de ella.

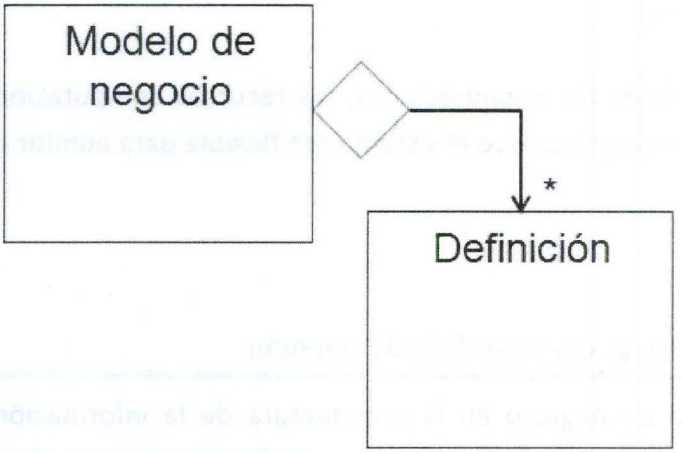


Ilustración 53 - Relación entre el modelo de negocio y las definiciones

Muchas de las entidades del sistema de información no son estáticas y tienen un cierto comportamiento. En Monet, existe un conjunto de definiciones a las que se les puede agregar un comportamiento dinámico, que permitirá describir la parte dinámica del sistema de información.

Las definiciones están a su vez supeditadas a un Meta-Modelo, donde se define la estructura de las definiciones en Monet, el cual a su vez está descrito mediante un Meta-Meta-Modelo que en este caso es el descrito por el lenguaje XML.

El modelo de negocio, una vez completamente definido, es proporcionado al llamado Espacio de negocio, que es el lugar creado por Monet a partir de las definiciones existentes en el modelo y donde convivirán todos los objetos de negocio creados a partir de las definiciones del modelo de negocio.

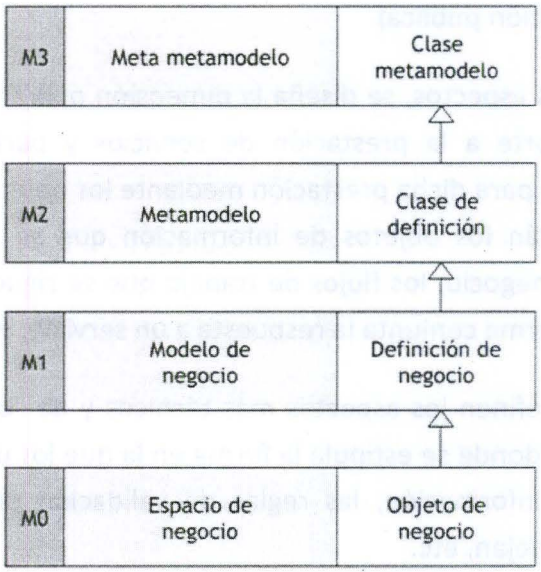


Ilustración 54 - Diferentes niveles de lenguajes en Monet

A la hora de ver un modelo de negocio, este es posible verlo desde tres dimensiones diferentes, la organizacional, la operacional y la tecnológica. Cada una de ellas describe aspectos de la unidad de negocio a un nivel de abstracción diferente.

- **Organizacional.** Representa la visión del negocio. En ella se definen los servicios que presta la unidad de negocio, la relación con el cliente y el cuadro de mando desde el que se podrá tener un seguimiento de la marcha del negocio.
- **Operacional.** Representa la arquitectura de la información y los procesos necesarios para proporcionar los servicios a los clientes. En esta dimensión se definen los objetos de información, las tareas a realizar y las reglas de asociación y/o recomendación de estas tareas a los trabajadores.

- **Tecnológica.** Representa cómo se gestiona la información tanto para facilitar la comprensión por parte de los usuarios como permitir la interoperabilidad entre sistemas. En esta dimensión se definen los comportamientos del sistema de información, las plantillas de documentos y los modos de visualización y representación de la información.

En el momento de comenzar el diseño y desarrollo de un modelo de negocio, es importante que este siga cada una de las dimensiones en el orden indicado, siendo este el más lógico desde un punto de vista externo a la unidad de negocio.

Lo importante en una unidad de negocio y lo primero que debe estar definido son los servicios que presta la unidad de negocio y que aspectos de la actividad regular de la unidad de negocio son claves para el éxito de la misma desde un punto de vista empresarial (económico en el ámbito privado o de satisfacción del ciudadano en el ámbito de la administración pública).

A partir de estos aspectos, se diseña la dimensión operacional de la unidad de negocio para dar soporte a la prestación de servicios y para registrar/tracear la actividad llevada a cabo para dicha prestación mediante los objetos de información. En este sentido se definirán los objetos de información que se manejarán de forma interna a la unidad de negocio, los flujos de trabajo que se siguen para completar las tareas que forman en forma conjunta la respuesta a un servicio, etc.

Por último, se definen los aspectos más técnicos y de "bajo nivel", esto es, la dimensión tecnológica, donde se estipula la forma en la que los usuarios interactuarán con estos objetos de información, las reglas de validación de estos, plantillas de documentos que se manejan, etc.

El error muchas veces cometido a la hora de plantear un sistema de información está en no seguir esta forma de trabajo "arriba-abajo", sino al revés. Esto se hace generalmente porque el cliente presiona para tener resultados rápidos, comenzando a definir la interfaz gráfica, los objetos de información de bajo nivel, etc. El problema viene cuando una vez todo construido, se va a comenzar a definir la capa de abstracción superior, esto es, la visión del negocio con el cuadro de mando. En este momento comienzan a salir datos que son necesarios para la construcción del cuadro de mando que no se han tenido en cuenta en las capas inferiores, cuando se tienen también muchos datos registrados que no se usan para nada, por lo que es necesario volver a cambiar todas las capas inferiores. Esto empeora cuando se usa una visión tradicional de construcción de software y estos cambios implican cambios en toda la aplicación, desde controles en interfaz gráfica, lógica de negocio, almacenamiento en base de datos, etc.

### 10.1.2 Arquitectura de la información

La arquitectura de la información en Monet se representa a partir de una serie de objetos de información que pueden usarse y que representan distintas relaciones entre los datos que almacenan. Podemos distinguir un gran grupo dentro de los objetos de información, que son todos los llamados nodos de información, que son los encargados de almacenar los datos del sistema de información. Luego existen otros tipos de objeto que complementan y/o enriquecen a estos objetos de tipo nodo.

Los diferentes tipos de nodos existentes en Monet son:

#### *10.1.2.1 Nodo Formulario*

Los nodos de tipo formulario son estructuras de información organizadas en campos, que pueden ser simples o múltiples. Existe un amplio abanico de tipos de campo que puede albergar un formulario:

- Campos básicos de texto, número, patrón de texto o código de texto autogenerado.
- Campos de selección, que no permiten una edición libre, sino que se debe seleccionar un valor de entre los posibles. Estos posibles valores se seleccionan a partir de un tesauro, otro objeto dentro de la arquitectura de la información que se describirá más adelante.
- Campos de tipo vínculo que permite establecer un enlace con otro nodo (siendo sus ciclos de vida independientes), así como campos de tipo contenedor, que permiten almacenar un nodo dentro del propio formulario (estando sus ciclos de vida enlazados, es decir, se crean y se destruyen a la vez que el formulario que los contiene).
- Campos avanzados como imagen, fichero o posición geográfica.

#### *10.1.2.2 Nodo Colección*

Los nodos de tipo colección permiten almacenar un conjunto de nodos, que si bien pueden ser de distinto tipo, todos ellos deben tener la característica común de implementar la misma referencia. La referencia es una definición que establece un conjunto de atributos que debe aportar un nodo, como si de una interfaz se tratase (en el contexto de la orientación a objetos).

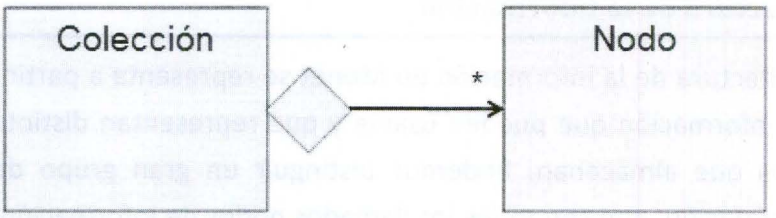


Ilustración 55 - Relación entre una colección y los nodos contenidos

Este tipo de nodo se caracteriza por representar una relación de agregación de uno a muchos. El tiempo de ida de los objetos que contiene no está determinado por el tiempo de vida de la colección en la creación. Cuando se crea la colección, esta está vacía y no contiene ningún elemento, estos se van agregando a posteriori. Sin embargo, cuando la colección se elimina, todos los nodos contenidos son eliminados también. Los nodos se visualizan en un listado, en donde se muestran los datos de referencia. Este listado se puede clasificar, ordenar y filtrar por cualquiera de los atributos definidos en la referencia.

### *10.1.2.3 Nodo contenedor*

Los nodos de tipo contenedor se basan en la relación de composición, es decir, están compuestos por uno o varios nodos de distinto tipo, que nada tienen en común. El tiempo de vida de sus componentes está determinado por el tiempo de vida del contenedor que los contiene, es decir, cuando se crea el contenedor, se crean sus componentes y cuando se elimina el contenedor, sus componentes también se eliminan.

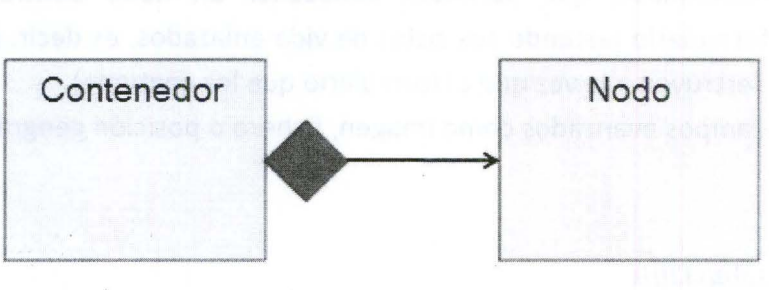


Ilustración 56 - Relación entre un contenedor y el nodo que contiene

#### *10.1.2.4 Nodo Catálogo*

Los nodos de tipo catálogo tienen como uso principalmente el hacer de envoltorio a una determinada colección, a la que se le aplican una serie de filtros, ordenación y/o agrupación por defecto. Estos catálogos se pueden usar luego para que los campos de tipo vínculo puedan mostrar al usuario los nodos que puede vincular con ese campo en particular.

#### *10.1.2.5 Nodo Documento*

Los nodos de tipo documento son, como su propio nombre indica, los que representan un documento. Estos nodos tienen un estado, en el que se indica si el documento es modificable o está consolidado. En función de este estado se pueden realizar operaciones diferentes. Si está en estado modificable, el documento pude recalcularse, es decir, el usuario no puede editar directamente el documento sino que este se rellena a partir de los datos de diferentes formularios, contenedores o colecciones del sistema de información. Una vez consolidado el documento, este ya no puede cambiar y a partir de ese momento se pueden realizar otro tipo de operaciones como su firma digital o su distribución a otras unidades de negocio. La interoperabilidad entre unidades de negocio se basa en el intercambio de documentos producidos por este tipo de nodos.

#### *10.1.2.6 Nodo Escritorio*

Los nodos de tipo escritorio pretenden ser un nodo especial donde el usuario pueda encontrar todo aquello que utiliza en su trabajo diario a mano, además de herramientas de groupware como videoconferencia o telefonía VOIP, notificaciones, notas, etc. Es un concepto que aún no está desarrollado en todo su potencial en Monet actualmente, estando en un estado aún muy primario.

### 10.1.3 Mapas de trabajo

Los mapas de trabajo en Monet son una forma dinámica de plantear el trabajo colaborativo en una organización. Su fundamento es que los flujos de trabajo en la vida real no son estáticos y no es abordable desde el punto de vista práctico generar un sistema que tenga en cuenta todos los posibles casos que se pueden llegar a dar en la realidad. No es abordable no solo porque la explosión de posibles combinaciones sea casi infinita, sino también porque no es posible siquiera conocer todos los posibles casos que se pueden dar.

La visión tradicional de los flujos de trabajo (WorkFlows), tienen el problema de una falta de dinámica, donde los usuarios puedan indicar un cambio en el flujo de trabajo en un momento determinado, debido a circunstancias no previstas en el momento de diseño del mismo [16].

En los mapas de trabajo se diseñan bajo la premisa de que los usuarios son responsables del trabajo que están haciendo y son ellos los que deben velar por que el trabajo se realice como se debe y no que sea el ordenador el que les ordene como deben trabajar, que pueden hacer o que cosas del mundo real pueden o no pueden pasar, limitándose el usuario a simplemente darle al botón "siguiente". En los mapas de trabajo, lo que se asegura es una trazabilidad de todo el trabajo realizado y todas las decisiones tomadas, de forma que siempre quede constancia de "quién hizo qué".

El paradigma en el que se basan los mapas de trabajo es un mapa de carreteras, donde tenemos una serie de lugares y una serie de carreteras que conectan cada uno de los lugares del mapa. En este mapa existe una serie de líneas de autobús que cubren todos los lugares, pero que pasan por todas las carreteras para ello. La equivalencia al caso de los mapas de trabajo es la siguiente. Los lugares son los llamados WorkPlaces y son como estados por los que pasa la tarea que realizamos. Las carreteras son los llamados WorkLines y representan un trabajo determinado que culmina en un determinado estado. Una vez se comienza un trabajo (subimos a una línea de autobús para recorrer una carretera), este pude finalizar en una serie de estados en función de cómo ha terminado este (lugar en el que nos bajamos del autobús). Sin embargo, como hemos comentado anteriormente no todas las carreteras están cubiertas por autobuses, por lo que no todos los posibles trabajos (carreteras) están representados en el mapa de trabajo, como ocurre en la realidad, no es posible tener en cuenta todas las posibles situaciones que se pueden dar.

Puede ocurrir que estando en un determinado lugar, queremos llegar a otro, pero en ese momento determinado no podemos/queremos tomar una línea de autobús y tomamos una forma alternativa de transporte por otra carretera. Esto es lo

que simboliza en los mapas de trabajo una situación excepcional no contemplada, por lo que podemos pasar de un lugar a otro sin pasar por ningún WorkLine.

Definido lo que es un mapa de trabajo, a continuación vamos a ver un ejemplo acompañado de la representación gráfica con la que se analizan y diseñan los mapas de trabajo. En esta representación los lugares (WorkPlaces) se representan en columnas y las líneas de trabajo (WorkLines) en horizontal. Cuando se cruzan, si existe una parada en ese lugar de esa línea, existe lo que denominamos un WorkStop.

Por último, queda definir el comportamiento del motor de mapas de trabajo. El motor tratará de completar el mapa de trabajo siguiendo los caminos disponibles a partir de un lugar inicial. Cuando se llega a una línea que tiene más de una parada, es necesario indicarle al motor en que para nos queremos bajar. Para tomar esta decisión necesitaremos información que puede estar ya disponible en el sistema de información, información de otra unidad de negocio, información por parte del usuario, información determinada por otra tarea o información que estará disponible pasado un determinado tiempo. Todas estas situaciones pueden no poderse concretar en el momento que estamos en la línea y será necesario bloquear la línea hasta que tenemos esos datos. Este concepto es el que denominamos WorkLock. Su función es bloquear el mapa de trabajo hasta que se obtenga el dato necesario. Pueden existir varios bloqueos en la línea y solo se podrá avanzar cuando alguno de ellos se libere. Existen los siguientes tipos de bloqueo:

- DesicionLock. Se le pide al usuario que tome de forma directa que parada debe tomarse de las disponibles en la línea.
- SyncLock. Bloquea la línea hasta que una determinada tarea termina su trabajo.
- BranchLock. Se bloquea dando el control al comportamiento definido por el analista, el cual de forma automática y en base a datos existentes en el sistema de información designará la parada a tomar.
- ServiceLock. Realiza una petición de servicio a otra unidad de negocio y espera a que esta responda. Una vez recibida la respuesta, se continua con el mapa.
- FormLock. Se muestra un formulario al usuario, el cual debe rellenar para poder continuar con el trabajo.
- TimerLock. Se desbloquea la línea hacia una determinada parada una vez ha pasado un determinado espacio de tiempo desde que se alcanzó el bloqueo.

### *10.1.3.1 Ejemplo de mapa de trabajo y diagrama de representación*

Para ejemplificar el funcionamiento y representación de un mapa de trabajo, vamos a plantear un caso sencillo, la resolución de incidencias relacionadas con un programa informático. Para simplificar el ejemplo, vamos a suponer que no existen más casos ante una incidencia sino que esta sea resuelta o no proceda.

En primer lugar, se notifica una incidencia y con ello se crea una tarea en Monet que tiene un mapa de trabajo definido como se muestra en la Ilustración 57, donde vemos que el mapa de trabajo está compuesto por dos líneas de trabajo (WorkLines), cuatro lugares (WorkPlaces), tres paradas (WorkStops) y dos bloqueos (WorkLocks).

En primer lugar vamos a describir los lugares:

- Notificado. La incidencia acaba de ser notificada. Este lugar es de tipo "event", lugar donde comienza el mapa de trabajo.
- Pendiente operación. El fallo indicado en la incidencia existe y está pendiente de estar corregido.
- Terminado. La incidencia se ha resuelto, estando el fallo notificado corregido. Este lugar es de tipo "goal", indicando que al llegar a este lugar se termina de forma satisfactoria el mapa de trabajo.
- Cancelado. El fallo al que hace referencia la incidencia no existe o no se ha podido reproducir, por lo que no se ha corregido nada. Este es un lugar de tipo "dead-end", indicando que si se llega a este lugar, se aborta el mapa de trabajo.

Cuando el mapa de trabajo comienza a ejecutarse, se encuentra en el lugar inicial de "Notificado". Se avanza por la línea de trabajo disponible: "Clasificando incidencia" y nos encontramos con un bloqueo de tipo "DesicionLock", en el que el usuario debe decidir en qué parada nos vamos a bajar de la línea de trabajo, existiendo dos paradas posibles: "Hay que corregirlo" y "No procedente".

Si el usuario escoge "No procedente", se abandona la línea por esa parada, encontrándonos en el lugar "Cancelado". Dado que este lugar es final de tipo "dead-end", aquí se aborta el mapa de trabajo.

Si el usuario escoge "Hay que corregirlo", se abandona la línea por esa parada, encontrándonos en el lugar "Pendiente operación". En ese lugar comienza otra línea, "Ejecutando", por la que sigue la ejecución del mapa de trabajo, encontrándose con otro bloqueo, también de tipo "DesicionLock". En esta ocasión, solo existe una posible opción, la parada "Corregido". Esto servirá para que el usuario pueda indicar el

momento en el que la incidencia ha sido corregida. Una vez llegado a esa parada, se abandona la línea, quedándose en el lugar "Terminado", que al ser un lugar de tipo "goal", hace que el mapa de trabajo termine.

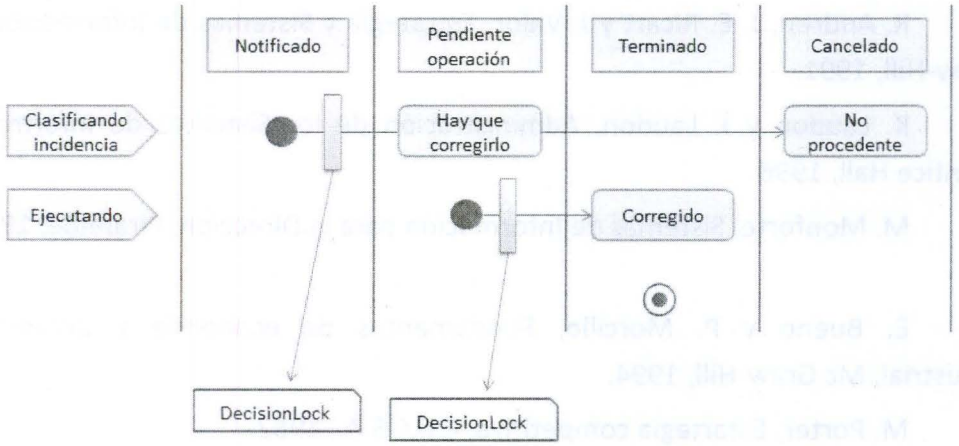


Ilustración 57 - Ejemplo de representación de un mapa de trabajo

Una vez hemos visto cómo funcionan los mapas de trabajo, imaginemos que nos encontramos durante la ejecución de una resolución de una incidencia y cuando vamos a corregirla vemos que el fallo ya ha sido corregido de forma indirecta, ya que se debía a otro error. El problema es que esta incidencia realmente no procedía, ya que era la misma que otra incidencia ya indicada y resuelta anteriormente, pero que por alguna razón se clasificó como que había que corregirla. El usuario ahora se vería obligado a responder que está corregida, ya que es la única opción disponible, cuando realmente no es lo que ha ocurrido, ya que no ha habido ningún tipo de acción asociado a esta incidencia que haya realizado.

Aquí es donde radica la potencia de los mapas de trabajo, el usuario podrá hacer un cambio de lugar, haciendo que el mapa de trabajo se sitúe en el lugar "Cancelado", abortando el mapa de trabajo, quedando registrado que esta incidencia no se resolvió debido a que ya estaba resuelta anteriormente.

# 11. BIBLIOGRAFÍA


1] R. Andreu, J. E. Ricart y J. Valor, Estrategia y Sistemas de Información, Mc Graw-Hill, 1991.

2] K. Laudon y J. Laudon, Administración de los Sistemas de Información, Prentice Hall, 1996.

3] M. Monforte, Sistemas de Información para la Dirección, Pirámide, 1994.

4] E. Bueno y P. Morcillo, Fundamentos de economía y organización industrial, Mc Graw-Hill, 1994.

5] M. Porter, Estartegia competitiva, C.E.C.S.A., 1982.

6] D. Schmidt, Model-driven engineering, IEEE computer, 2006.

7] J. Miller y J. Mukerji, Model driven architecture (mda)., Object Management Group, 2001.

8] C. Atkinson y T. Kuhne, Model-driven development: a metamodeling foundation, Sofware, IEEE, 2003.

9] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits y S. Neema, Developing applications using model-driven design environments., Computer, 2006.

10] S. Beydeda, M. Book y V. Gruhn, Model-driven software development., Springer Verlag, 2005.

11] L. Zamora Cortina, Conceptos Básicos sobre: Case y lenguajes de cuarta generación, 2003.

12] A. Amescua Seco y J. Lloréns Morillo, «Caminando hacia los 5GLs. Líneas de evolución de los entornos 4GL. Requerimientos de conocimiento para el empleo de 4GLs. Los nuevos entornos de desarrollo de software».

13] [En línea]. Available: http://en.wikipedia.org/wiki/Fourth-generation_programming_language.

14] J. J. Hernández Cabrera, «Implantación del ebussiness en pequeñas organizaciones con una orientación al modelado y la interoperabilidad,» 2011.

15] K. Laudon y J. Laudon, Sistemas de información Gerencial, Prentice Hall, 2004.

V. Fernández Alarcón, Desarrollo de sistemas de información. Una
16] metodología basada en el modelado, Ediciones UPC, 2006.

T. Gilb, «Principles of Software Engineering Management,» Addison-
17] Wesley, 1998.

B. Boehm, «A Spiral Model for Software Development Enhancement,»
18] *Computer,* vol. 21, nº 5, pp. 61-72, 1998.

B. Boehm, «The Spiral Model as a Tool for Evolutionary Software
19] Acquisition,» CrossTalk, Mayo 2001. [En línea]. Available: http://www.stsc.hill.af.mil/crosstalk/2001/05/boehm.html.

C. W. Dawson y G. Martín, El proyecto de fin de carrera en Ingeniería
20] Informática. Una guía para el estudiante, Prentince Hall, 2002.

E. Clayberg y D. Rubel, Eclipse plug-ins, Anaya multimedia, 2010.
21]

R. S. Pressman, Ingeniería del software. Un enfoque práctico, McGraw-Hill,
22] 2005.

J. Boodhoo, «Design patterns. Model View Presenter,» *MSDN Magazine-*
23] *Louisville,* pp. 91-100, 2006.

M. Fowler, Patterns of enterprise application architecture, Addison-Wesley
24] Professional, 2003.

G. Davis y M. Olson, Sistemas de Información Gerencial, Mc Graw-Hill,
25] 1985.

C. Edwars, J. Ward y A. Bytheway, Fundamentos de Sistemas de
26] Información, Prentice Hall, 1998.

D. García Bravo, Sistemas de Información en la Empresa, Pirámide, 2000.
27]

I. Gil Pechuan, Sistemas y tecnologías de la información para la gestión, Mc
28] Graw-Hill, 1996.