Desarrollo de prototipos para dar soporte a Flujos de Trabajo Móviles que integran Elementos Físicos en el ámbito de la Internet de las Cosas

José Luis Sepúlveda Socuéllamos

Supervisores:
Dr. Vicente Pelechano Ferragud
Dr. Pau Giner Blasco

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Julio 2011

Resumen

Hoy en día los dispositivos móviles están presentes de forma continua en nuestras vidas, imaginen un mundo en el que con nuestro teléfono móvil podamos prestar un libro de una biblioteca, o comprar un producto en un supermercado, suena bien, verdad?, ese mundo es posible, pues la tecnología actual nos lo permite. Para ello es necesario integrar los elementos físicos del mundo real en los Sistemas de Información (la llamada “Internet de las cosas”), esta integración se puede conseguir ya con tecnologías Auto-ID de identificación automática (como RFID).

Cuando estos objetos participan activamente en los procesos de negocio se puede reducir la participación del ser humano como transportador de información, lo cual reduce el número de errores y aumenta la eficiencia del proceso. La heterogeneidad de las tecnologías Auto-ID y los requisitos cambiantes de los procesos de negocio dificultan el desarrollo de este tipo de sistemas, de forma que el resultado final puede que no cumpla con las expectativas del usuario. Obtener prototipos del sistema en una fase temprana del desarrollo se presenta como clave para, en caso que fuera necesario, rediseñar el sistema antes de desarrollarlo.

Partiendo de la Ingeniería Dirigida por Modelos (MDE), esta tesis presenta un proceso de desarrollo para obtener de una forma sencilla y rápida prototipos para este tipo de sistemas. Para ello definiremos un escenario y mediante técnicas de generación de código obtendremos los prototipos.

Para la especificación del escenario se ha definico un lenguaje de modelado. A partir de esta especificación se pueden obtener los prototipos de una manera sistemática. Finalmente, se definió un caso de estudio para probar la aplicabilidad de la propuesta en dispositivos móviles reales.

Tabla de Contenido

1. INTRODUCCIÓN 1

1.1 MOTIVACIÓN 2
1.2 PRESENTACIÓN DEL PROBLEMA 4
1.3 SOLUCIÓN PROPUESTA 4
1.4 ESTRUCTURA DE LA TESIS 6

2. CONTEXTO TECNOLÓGICO 7

2.1 DISPOSITIVOS MÓVILES 8
2.1.1 Dispositivos 8
2.1.2 Aplicaciones 10
2.2 LA INTERNET DE LAS COSAS 11
2.3 PROCESOS DE NEGOCIO 13
2.3.1 Modelado de procesos de negocio 13
2.3.2 Ejecución de procesos de negocio 18
2.4 FLUJOS DE TRABAJO MÓVILES "PHYSICAL MOBILE WORKFLOWS" 19
2.4.1 Flujos de trabajo Inteligentes "smart workflows" 20
2.4.2 Interacción con dispositivos móviles "physical mobile interactions" 21
2.4.3 Procesos de negocio móviles "mobile business processes" 23
2.5 CONCLUSIONES 23

3. TÉCNICAS DE PROTOTIPADO EN EL ÁMBITO DE LOS SISTEMAS UBICUOS 24

3.1 TÉCNICAS Y TRABAJO RELACIONADO 24
3.2 CONCLUSIONES 29

4. TECNOLOGÍA Y HERRAMIENTAS USADAS EN EL DESARROLLO DE LA PROPUESTA 30

4.1 INGENIERÍA DIRIGIDA POR MODELOS: MDE 30
4.1.1 Metamodelo 31
4.1.2 Validación de un modelo 32
4.1.3 Generación de código 33
4.2 HERRAMIENTAS 36
4.2.1 Eclipse 37
4.2.2 Eclipse Modeling Project (EMP) 38
4.2.3 Eclipse Modeling Framework (EMF) 39
4.2.4 Graphical Modeling Framework (GMF) 41
4.2.5 Eugenia 43
4.2.6 OpenArchitectureWare (OAW) 45
4.3 CONCLUSIONES 46

5. DESARROLLO DE LA PROPUESTA 47

5.1 SISTEMAS QUE SOPORTEN FLUJOS DE TRABAJO MÓVILES 48
5.1.1 Método de desarrollo de sistemas que soporten flujos de trabajo móviles 48
5.1.2 Prototipado rápido para flujos de trabajo móviles 52
5.1.2.1 Pantallas mockup 52
5.1.2.2 Pasos a seguir 55
5.1.3 Conclusiones 57
5.2 MÉTODO DE DISEÑO PARA OBTENER PROTOTIPOS PARA FLUJOS DE TRABAJO MÓVILES 58
5.2.1 Conceptos a modelar 58
5.2.2 Definición del metamodelo 63
5.2.3 Definición del editor gráfico 69
5.2.4 Validación del modelo 75
5.2.5 Generación de código 78
5.2.5.1. Prototipo del usuario 80
5.2.5.2. Consola del operador 84
5.3 CONCLUSIONES 84

6. CASO DE ESTUDIO. APLICACIÓN DE LA PROPUESTA 85

6.1 DESCRIPCIÓN DEL HOTEL INTELIGENTE 85
6.1.1 Hotel Inteligente *85*
6.1.2. Escenario *85*
6.2 MODELADO DEL CASO DE ESTUDIO 88
6.2.1 Conceptos del caso de estudio a modelar *88*
6.2.2 Creación del escenario con el editor *91*
6.2.3 Transformación realizada *93*
6.2.3.1 Prototipo del usuario (*index.html*) 93
6.2.3.2 Consola del operador (*control.html*). 96
6.2.4 Prototipos obtenidos *96*
6.3 CONCLUSIONES 103

7. CONCLUSIONES 104

8. REFERENCIAS 105

APÉNDICES. I

A- METAMODELO I
B- METAMODELO CON ANOTACIONES EUGENIA. III
C- REGLAS DE VALIDACIÓN VI
D- PLANTILLA DE TRANSFORMACIÓN VIII
E- FUNCIONES DE APOYO XV
F- INSTALAR Y CONFIGURAR EL ENTORNO ECLIPSE DEL PRESENTE DESARROLLO XVI
G. DISEÑO DE LA INFRAESTRUCTURA PARA GENERAR LOS PROTOTIPOS. XVII

Lista de figuras

Figura 1.1. Los objetos del mundo real se integran en el mundo digital 2
Figura 1.2. Diferentes ejemplos de integración de objetos en el mundo digital 2
Figura 1.3. Nos dirigimos hacia la miniaturización de los dispositivos Auto-ID 3
Figura 1.4. Código QR 3
Figura 1.5. Aplicaciones que conectan el mundo real y el virtual 3
Figura 2.1. Dominio de los Flujos de Trabajo Móviles [Giner, 2010] 7
Figura 2.2. Iphone 4 y Blackberry 8
Figura 2.3. Mercado de las distintas plataformas 8
Figura 2.4. Sistemas operativos para smartphones 9
Figura 2.5. Notación BPMN - Elementos de flujo: Eventos 14
Figura 2.6. Notación BPMN - Elementos de flujo: Actividades 15
Figura 2.7. Notación BPMN - Elementos de flujo: Bifurcaciones 15
Figura 2.8. Notación BPMN - Elementos de conexión 16
Figura 2.9. Notación BPMN - Swimlanes 16
Figura 2.10. Notación BPMN - Artefactos 17
Figura 2.11. Notación BPMN - Ejemplo 17
Figura 2.12. Flujos de trabajo móviles. Áreas de investigación y su intersección [Giner, 2010] 19
Figura 2.13. Flujos de trabajo inteligentes [Wieland et al., 2008] 20
Figura 2.14. Flujos de trabajo inteligentes. Arquitectura propuesta por Wieland [Wieland et al., 2008] 21
Figura 2.15. Arquitectura genérica del marco de trabajo *Interacciones con Dispositivos Móviles* [Ruzkio, 2007] 22
Figura 3.1. Prototipos - Entorno de desarrollo de ActivityDesigner 25
Figura 3.2. Prototipos - CogTool. Ejemplo de interfaz donde se muestra un conductor realizando una tarea secundaria 26
Figura 3.3. Prototipos - Entorno de desarrollo de DART 26
Figura 3.4. Prototipos - Entorno de desarrollo de D.Tools 27
Figura 3.5. Prototipos - Uso de simulación Mago de Oz [Dow et al., 2005] 27
Figura 3.6. Prototipos - Técnica Mago de Oz usada en [Reilly et al., 2005] 28
Figura 4.1. Modelo e instancia del modelo 31
Figura 4.2. Modelos, lenguajes, metamodelos y metalenguajes [Kleppe, 2005] 32
Figura 4.3. Capas MOF para UML 32
Figura 4.4. Transformación de modelos 34
Figura 4.5. Arquitectura Eclipse 37
Figura 4.6. Arquitectura EMF 39
Figura 4.7. Jerarquía de clases del Metamodelo Ecore 40
Figura 4.8. Graphical Modeling Framework - Sus Dependencias 41
Figura 4.9. Graphical Modeling Framework - Resumen 42
Figura 4.10. GMF Dash Board 43
Figura 4.11. Arquitectura OAW 45
Figura 5.1. Estrategia de desarrollo para sistemas que soporten flujos de trabajo móviles [Giner, 2010] 48
Figura 5.2. Extracto del metamodelo Parkour [Giner, 2010] 49
Figura 5.3. Arquitectuta de Völter [Völter, 2005] 50
Figura 5.4. Presto - Componentes 50
Figura 5.5. Presto - Estrategias de ejecución 52

Figura 5.6. Fidelidad de los Mockups 53
Figura 5.7. Prototipos HTML generados..... 55
Figura 5.8. Prototipado rápido para flujos de trabajo móviles 55
Figura 5.9. Prototipo en HTML. Tras lanzar el operador un cambio en el contexto, el cliente del hotel recibe el mensaje de que el spa está libre..... 56
Figura 5.10. Parte automática y manual del desarrollo 58
Figura 5.11. Conceptos a modelar - Entorno físico. 59
Figura 5.12. Conceptos a modelar - Tareas (*WorkItem*)..... 61
Figura 5.13. Conceptos a modelar - Modo de operación..... 61
Figura 5.14. Conceptos a modelar - Ejemplos..... 62
Figura 5.15. Metamodelo ecore en forma de árbol..... 68
Figura 5.16. Editor gráfico - Paleta 69
Figura 5.17. Editor gráfico - Mostramos el nombre de la clase padre TypeElement..... 73
Figura 5.18. Editor gráfico - Etiquetas añadidas 73
Figura 5.19. Editor gráfico terminado 74
Figura 5.20. Check. Resultado de validar un modelo erróneo 77
Figura 5.21. Estructura *fichero index.html* 79
Figura 5.22. Estructura fichero *control.html* 79
Figura 5.23. Ficheros generados. 80
Figura 6.1. Caso de Estudio. Código QR entregado a Juan..... 86
Figura 6.2. Caso de Estudio. Diagrama BPMN para los servicios del hotel..... 88
Figura 6.3. Caso de Estudio. Diagrama BPMN para las funciones de la habitación 88
Figura 6.4. Caso de Estudio. Entorno físico..... 89
Figura 6.5. Caso de Estudio. Tareas a modelar 90
Figura 6.6. Caso de Estudio. SERVICIOS HOTEL..... 91
Figura 6.7. Caso de Estudio. FUNCIONES HABITACIÓN 91
Figura 6.8. Caso de Estudio. WorkItems para SERVICIOS HOTEL 92
Figura 6.9. Caso de Estudio. WorkItems para FUNCIONES HABITACIÓN..... 92
Figura 6.10. Caso de Estudio. Pantalla inicial (*index.html*) 96
Figura 6.11. Caso de Estudio. Consola del operador (*control.html*) 97

Lista de Tablas

Tabla 3.1. Problemas en el desarrollo de sistemas ubicuos 28
Tabla 4.1. Tipos de acción para las restricciones Check 33
Tabla 4.2. Principales instrucciones XPand 35
Tabla 4.3. Subproyectos EMP 38
Tabla 4.4. Anotaciones Eugenia 44
Tabla 5.1. Metamodelo - ScenarioModel 63
Tabla 5.2. Metamodelo - TypeElement 64
Tabla 5.3. Metamodelo - Element 64
Tabla 5.4. Metamodelo - State 65
Tabla 5.5. Metamodelo - Task 65
Tabla 5.6. Metamodelo - WorkItem 66
Tabla 5.7. Metamodelo - NamedElement 66
Tabla 5.8. Metamodelo - Info, InfoLine, InfoGroup 67
Tabla 5.9. Editor gráfico - TypeElement 70
Tabla 5.10. Editor gráfico - Element 70
Tabla 5.11. Editor gráfico - State 70
Tabla 5.12. Editor gráfico - Task 71
Tabla 5.13. Editor gráfico - WorkItem 71
Tabla 5.14. Editor gráfico - InfoGroup 72
Tabla 5.15. Editor gráfico - InfoLine 72
Tabla 5.16. Check. La clase padre debe tener alguna clase hija 76
Tabla 5.17. Check. Evitar nombres de clase con valor *null* 76
Tabla 5.18. Check. Debe existir al menos una clase referenciada 76
Tabla 5.19. Check. Evitar repetir nombres de clase 76
Tabla 5.20. Check. No mezclar objetos de distintos *TypeElement* 76
Tabla 5.21. Plantilla xpand. Pantalla inicial 81
Tabla 5.22. Plantilla xpand. Pantalla para cada elemento detectado 81
Tabla 5.23. Plantilla xpand. Workitem de evento (tarea pendiente) 82
Tabla 5.24. Plantilla xpand. Workitem normal 83
Tabla 5.25. Plantilla xpand. Simular detección de un objeto físico 84
Tabla 5.26. Plantilla xpand. Simular evento 84

1. Introducción

Hoy en día estamos preparados tecnológicamente para acercarnos al mundo imaginado por Weiser [Weiser, 1991], un mundo en el que los objetos físicos se integren en el mundo digital de una forma natural, sin tener que hacer grandes esfuerzos para interactuar con ellos, a este concepto se le conoce como computación ubicua o pervasiva [Hansmann et al, 2001].

Dentro de la computación ubicua el presente trabajo se centra en el reto de la obtención de prototipos para poder validar el sistema antes de proceder a su desarrollo. Para obtener estos prototipos utilizaremos herramientas de desarrollo dirigidas por modelos (MDE), lo que nos permitirá abstraernos tanto de la tecnología usada para el desarrollo final, como de la usada para dotar de información digital a los objetos físicos.

Los prototipos creados nos servirán para evaluar sistemas ubicuos en el ámbito de los flujos de trabajo móviles (physical mobile workflows), estos sistemas aprovechan las actuales capacidades de los dispositivos móviles para detectar objetos del mundo real e integrarlos en el proceso de negocio, por ejemplo a la hora de prestar un libro (con información digital) de una biblioteca mediante un dispositivo móvil, existe un flujo de información que será gestionado y observado a través de los prototipos creados para este escenario.

1.1 Motivación

En el mundo imaginado por Weiser [Weiser, 1991] se hace uso de la tecnología de una forma natural, de forma que todos los objetos físicos tienen una identidad digital, estando todos interconectados entre sí, este concepto es conocido como Internet de las Cosas (figura 1.1) [Gershenfeld et al., 2004]. Actualmente las técnicas de identificación automática (Auto-ID) nos permiten dotar a los objetos de esta identidad digital, un ejemplo de ella es Identificación por Radio Frecuencia (RFID).

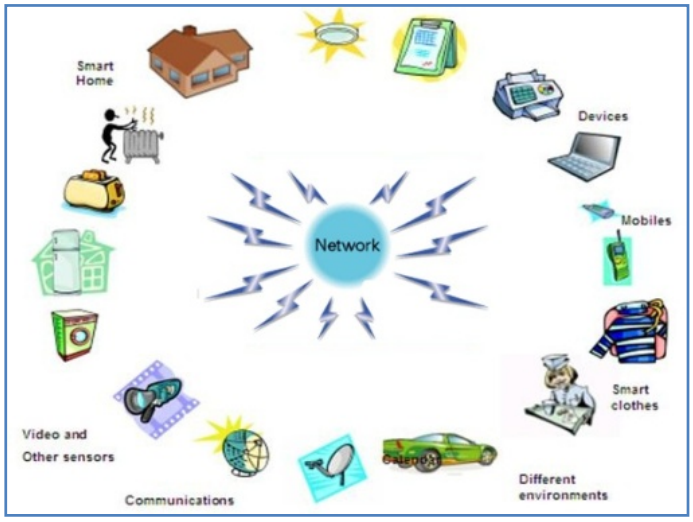


Figura 1.1. Los objetos del mundo real se integran en el mundo digital

En la figura 1.2 se pueden ver algunos ejemplos en diferentes entornos en los que se integran los objetos reales en el mundo virtual. En la figura, arriba a la izquierda podemos ver un ejemplo de *wearables* u ordenadores pegados a la piel, a la derecha, asistencia sanitaria, debajo a la izquierda, realidad aumentada y a su lado, displays interactivos.

Figura 1.2. Diferentes ejemplos de integración de objetos en el mundo digital

Esta integración se está haciendo realidad debido a dos factores:

1. Los dispositivos de identificación digital son cada vez más pequeños (figura 1.3), más potentes y más económico su desarrollo. Incluso se pueden identificar los objetos digitales con códigos QR impresos en un papel (figura 1.4).
2. Todo el mundo dispone de un dispositivo móvil (teléfono móvil, pda, ipad, etc). Estos dispositivos móviles cada vez tienen mayores prestaciones, mayores capacidades de interactuar con estos objetos digitales.

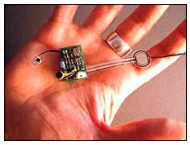


Figura 1.3. Nos dirigimos hacia la miniaturización de los dispositivos Auto-ID

Figura 1.4. Código QR

Estos factores permiten que vayan apareciendo cada vez más aplicaciones que conectan el mundo virtual con el real. En la figura 1.5 podemos ver algunos ejemplos de aplicaciones que están funcionando hoy.

Figura 1.5. Aplicaciones que conectan el mundo real y el virtual

1.2 Presentación del problema

El principal problema al que nos enfrentamos a la hora de desarrollar una aplicación para sistemas ubicuos es, por un lado, el coste económico y temporal:

1. El coste de etiquetar los objetos físicos con algún tipo de tecnología (QR-Codes, RFID, etc).
2. El coste de incorporar en los dispositivos móviles la tecnología para interactuar con los objetos físicos.
3. El coste del desarrollo de la aplicación en una tecnología concreta.

Todo lo anterior supone un alto coste previo al despliegue de la aplicación, que es muy dependiente de las tecnologías a utilizar. Por otro lado, no existe garantía de que el resultado final del desarrollo satisfaga las necesidades del usuario. Por tanto es muy importante evaluar primero la utilidad y aceptación por parte del usuario del sistema final antes de proceder a realizar toda la inversión tecnológica y económica.

Es por ello que nos planteamos la siguiente pregunta: ¿cómo podemos aproximarnos a la aceptación del sistema por parte del usuario antes de realizar grandes esfuerzos en su desarrollo?.

1.3 Solución Propuesta

Para responder a la pregunta planteada en la sección anterior, esta tesis tiene como objetivo principal obtener prototipos lo más cercanos posibles a la aplicación final para sistemas ubicuos dentro del ámbito de los flujos de trabajo móviles, de forma que el usuario tenga la sensación de estar usando la versión final del sistema. Esto implica una serie de desafíos:

1. Conseguir los prototipos de una forma **rápida**.
2. **Simular** la integración de los objetos físicos en el mundo digital. Hay que tener en cuenta que evaluar sistemas ubicuos puede ser muy complejo debido a las condiciones que hay que simular [Neely et al., 2008]. En nuestra propuesta usaremos técnicas de Mago de Oz [Dahlbäck et al., 1993], en las que un operador introducirá en el sistema la detección e información suministrada por los objetos físicos.
3. Conseguir un alto grado de **abstracción e independencia de la tecnología**.

Para llevar a cabo estos desafíos este trabajo propone, partiendo de la ingeniería dirigida por modelos (MDE), los siguientes pasos:

1. **Modelar los conceptos que forman parte de un escenario**. Consideramos un escenario como la instancia de una serie de flujos de trabajo de uno o más procesos de negocio.
2. **Diseñar un escenario** o modelo concreto mediante los conceptos modelados en el paso anterior. Construiremos un editor gráfico para realizar esta tarea de una forma más intuitiva.
3. **Validar el modelo** o escenario obtenido de forma automática.
4. **Generar de forma automática código** a partir del modelo validado, obteniendo como resultado final los prototipos.

El seguir una aproximación MDE nos permite obtener un alto grado de abstracción y ser independientes de la tecnología que se usará en el desarrollo final.

A modo ilustrativo, se han obtenido prototipos para un caso de estudio de un hotel (Hotel Inteligente), aunque nuestra propuesta podríamos aplicarla a cualquier otro escenario que hayamos modelado.

Los prototipos obtenidos serán pantallas HTML con un aspecto similar a las pantallas del popular iPhone, lo cual dará al resultado un mayor realismo. En concreto consistirán en:

1. Pantalla HTML inicial del usuario.
2. Pantalla HTML para cada actividad de los flujos de trabajo involucrados en el escenario.
3. Pantalla HTML para introducir en el sistema la detección de objetos físicos.

Mediante pantallas HTML conseguimos de una forma rápida y sencilla la simulación del sistema final, que puede ser ajustado posteriormente con herramientas estándar. Los requisitos tecnológicos para hacer uso de estos prototipos son bajos, solo necesitaremos un servidor web donde alojar las ficheros HTML obtenidos, un dispositivo móvil para que el usuario pruebe el sistema, y otro para que un operador lance los eventos de detección de objetos físicos. Los dispositivos deben disponer de conectividad y un navegador web, no es necesario que dispongan de ninguna tecnología de identificación automática.

1.4 Estructura de la Tesis

Esta tesis está organizada en siete capítulos. El capítulo 2 muestra el contexto tecnológico en el cual se desarrolla el presente trabajo. En el capítulo 3 comentaremos algunos trabajos sobre prototipado para sistemas ubicuos. El capítulo 4 presenta la tecnología usada en el desarrollo, para ello se ha utilizado la plataforma Eclipse con los proyectos Eclipse Modeling Project (EMP)[1] para el modelado y OpenArchitectureWare (OAW)[2] para la validación y generación de código. El capítulo 5 aborda el método de diseño seguido, partiendo de un escenario formado por flujos de trabajo móviles obtendremos un modelo del mismo, y tras ser validado generamos el código de los prototipos. En el capítulo 6 aplicaremos nuestra propuesta a un caso de estudio: Hotel Inteligente. El último capítulo presenta las conclusiones.

[1] http://www.eclipse.org/modeling
[2] http//www.openarchitectureware.org

2. Contexto Tecnológico

En este capítulo se describen las tecnologías y conceptos que se aplican en el presente trabajo. En primer lugar introduciremos los dispositivos móviles actuales y sus aplicaciones. La sección 2.2 explica con más detalle la llamada Internet de las Cosas y el soporte existente en términos de tecnologías. La sección 2.3 introduce la definición de Procesos de Negocio. En la sección 2.4 incidiremos en el ámbito de aplicación del presente trabajo dentro de la intersección de la Internet de las Cosas, las Aplicaciones para dispositivos Móviles y los Procesos de Negocio: los Flujos de Trabajo Móviles (figura 2.1).

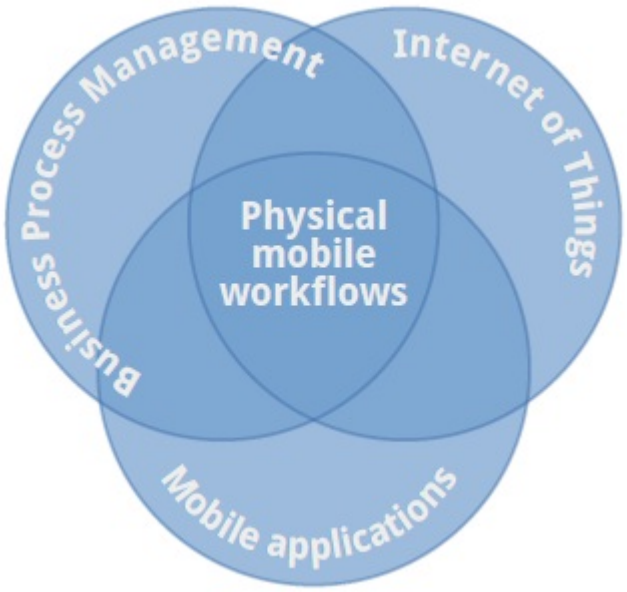


Figura 2.1. Dominio de los Flujos de Trabajo Móviles [Giner, 2010]

2.1 Dispositivos Móviles

2.1.1 Dispositivos

Hoy en día los dispositivos móviles incorporan cada vez mayores funciones, dejando de ser solo dispositivos pensados para comunicarse de forma verbal, como lo fueron en sus orígenes. Entre estas funciones tenemos: navegación web, servicio de gps, cámara de fotos, interpretación de qr-codes, acceso a redes sociales, mensajería instantánea, etc.

Estamos ante los denominados teléfonos inteligentes o smartphones, los cuales nos permiten interactuar cada vez más con nuestro entorno, gracias a sus mayores capacidades de cómputo, a su mayor conectividad y al cada vez mayor auge de aplicaciones diseñadas para sacar partido de sus características y su capacidad de movilidad.

Entre los dispositivos más populares están la plataforma Blackberry de RIM y la plataforma iPhone de Apple (figura 2.2). Actualmente están usándose cada vez más, dispositivos que usen los sistemas operativos open source Symbian (Nokia) y Android (HTC, Samsung, Motorola, ARM, Intel).

Figura 2.2. Iphone 4 y Blackberry

En el gráfico de la figura 2.3 se puede ver el mercado de estos dispositivos[3], y en la figura 2.4 los distintos sistemas operativos que se usan[4].

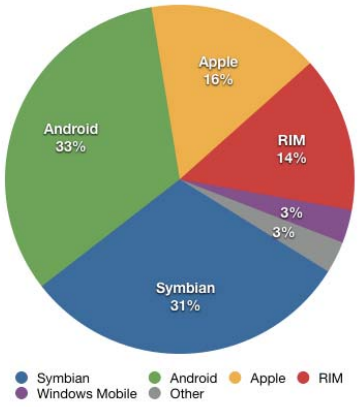


Figura 2.3. Mercado de las distintas plataformas

[3] http://en.wikipedia.org/wiki/Smartphone
[4] http://en.wikipedia.org/wiki/Mobile_operating_system

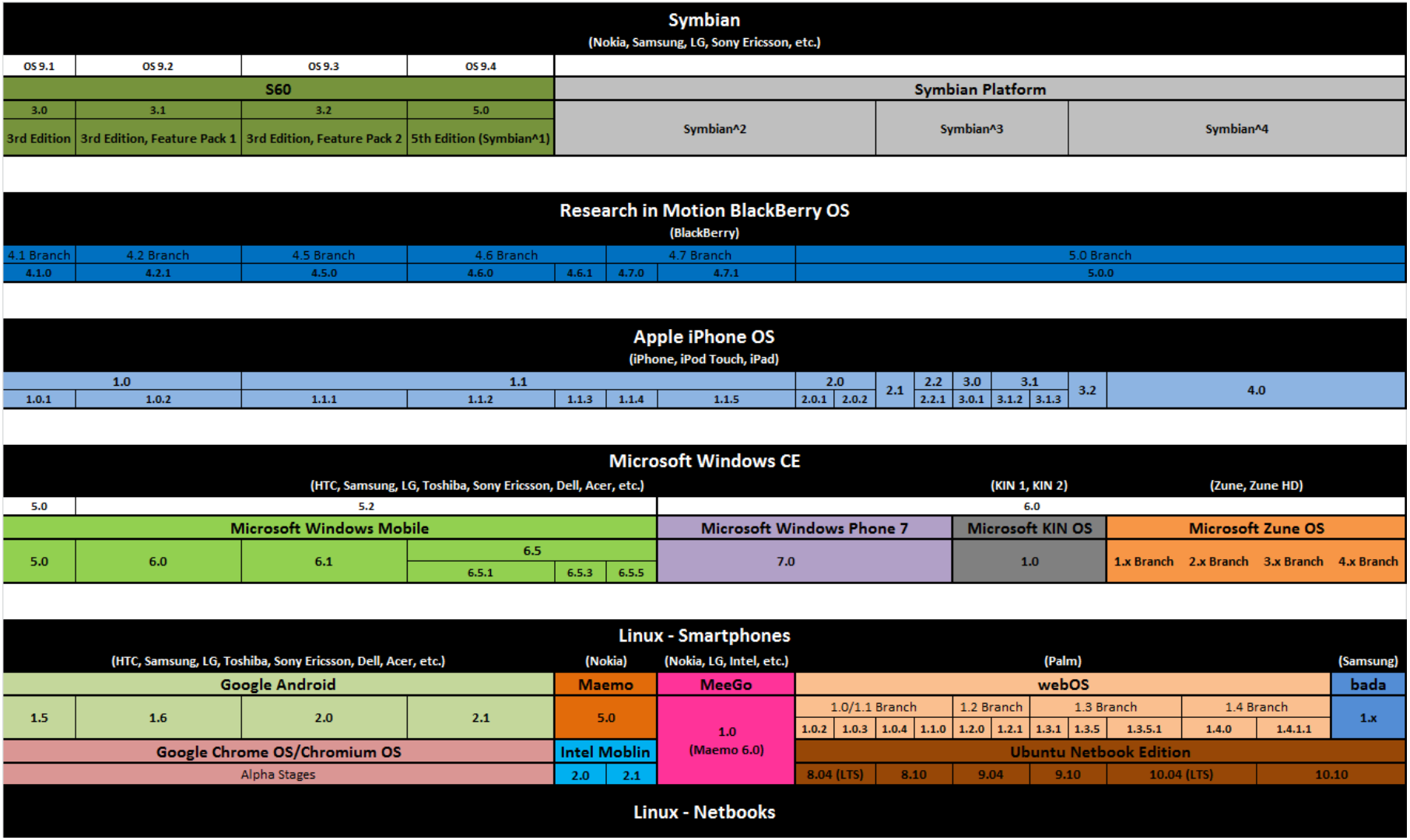


Figura 2.4. Sistemas operativos para smartphones

2.1.2 Aplicaciones

Una característica importante de casi todos los teléfonos inteligentes es que permiten la instalación de nuevas aplicaciones para incrementar el procesamiento de datos y la conectividad.

Los smarphones tienen un uso cada vez mayor por parte de los usuarios, pero sobre todo es interesante su uso corporativo, pues sus facilidades y movilidad son cada vez más valorados y utilizados dentro de las corporaciones: lectura de correo, gestión de procesos, aplicaciones específicas, etc. Conscientes de las ventajas que trae consigo la movilidad para el sector corporativo, tanto fabricantes como desarrolladores de sistemas operativos móviles, se esfuerzan por incorporar en sus plataformas aplicaciones orientadas a apoyar a las empresas en la ejecución de procesos de negocio en un ambiente de movilidad

Desafortunadamente, las aplicaciones móviles plantean varios desafíos importantes:

1. Los dispositivos móviles típicos tienen su hardware severamente restringido por lo cual requieren una infraestructura muy ligera de software.
2. En ausencia de una conexión estable a Internet, puede ser imposible, impracticable, o muy costoso que los dispositivos móviles contacten servidores centralizados.
3. Los enlaces a redes inalámbricas en los dispositivos móviles pueden ser interrumpidos frecuentemente y de forma impredecible.

Adicionalmente y para su uso corporativo es necesario que los dispositivos móviles cumplan ciertos requisitos :

- Protocolos de comunicación (interna y externa): SMS, MMS, Bluetooth, email, y HTTP
- Lenguajes de ejecución de procesos: WS-BPEL, WSDL, XML, y XPath
- Formatos de mensaje: SOAP y SOAP con adjuntos
- Lenguajes de interfaz de usuario: tecnologías web tales como XHTML
- Integración a aplicaciones nativas del teléfono: mensajería, navegador, mapas, lanzador, agenda y calendario
- Integración al ambiente móvil de la ejecución: tiempo e información de la zona horaria del usuario, cobertura de la red, nivel de batería, localización, velocidad, distancia, temperatura, y otros sensores externos y datos ambientales
- Garantías de calidad: limitaciones en la memoria y ancho de banda de la red.

Actualmente existen plataformas de código abierto como Android[5], con lo que se tiende a reducir estos problemas. Android es un sistema operativo móvil iniciado por Google pero ahora administrado por el Open Handset Alliance. Este consorcio incluye prestigiosos fabricantes tales como HTC, Motorola, Samsung o Sony Ericsson y grandes portadoras tales como Sprint, T-Mobile o Vodaphone.

Android es un sistema operativo basado en Java que se ejecuta sobre Linux. El sistema es muy ligero y está completamente equipado. Las aplicaciones Android se desarrollan usando el lenguaje de programación Java y pueden ser llevadas fácilmente a una nueva plataforma, con lo que cada vez existen más aplicaciones para esta plataforma.

No obstante y como se ha podido ver, sigue siendo muy costoso realizar el desarrollo de una aplicación móvil, lo cual hace más interesante nuestra propuesta de obtener rápidos y sencillos prototipos para evaluar el sistema antes de proceder a su desarrollo.

2.2 La Internet de las Cosas

La visión de la Internet de las Cosas propone añadir a los objetos del mundo real una identidad digital, logrando así la integración de los mundos real y virtual. Se trata de etiquetar los objetos con un único identificador para hacerlos reconocibles a los sistemas computacionales, de forma que los servicios que los Sistemas de Información ofrecen puedan alcanzar el mundo real [Giner, 2010]. Esta idea fue bien ilustrada por Bruce Sterling en su charla en The Emerging Technology Conference en 2006:

> "Estamos hablando del objeto diario, el más barato, la cosa más obvia que uno puede comprar o usar, que tiene una identidad digital única, de manera que llega a ser rastreado, clasificado, organizado, en el espacio y en el tiempo."

Los avances en las tecnologías de Identificación Automática (Auto-ID) han acercado esta visión de la Internet de las Cosas a la realidad. Auto-ID permite que objetos del mundo real sean detectados automáticamente por un sistema software, haciendo que los objetos no dependan ya de los humanos. Gracias a la Auto-ID, la gente, los lugares y las cosas pueden ser identificados en una variedad de diferentes modos. Identificación por Radio Frecuencia (RFID), Smartcards, códigos de barras, bandas magnéticas, y memorias de contacto, son algunas de las tecnologías Auto-ID disponibles.

[5] http://www.android.com/

Los objetos automáticamente identificables reciben diferentes nombres, tales como Spimes (objetos que son rastreables en espacio y tiempo), UFOs (Ubiquitous Findable Objects) o EKOs (Evocative Knowledge Objects). Esta heterogeneidad en terminología muestra que la Internet de las Cosas está todavía bajo construcción.

Desde las diferentes aplicaciones emergentes para la Internet de las Cosas, el presente trabajo está particularmente interesado en la integración de objetos del mundo real en los procesos de negocio o flujos de trabajo móviles. El paradigma de la Internet de las Cosas puede proporcionar numerosos beneficios en este campo, conduciendo a interesantes desafíos y oportunidades en diferentes áreas de negocio tales como mantenimiento y reparación, seguridad y responsabilidad, marketing, entre otras.

Hay un creciente interés en las tecnologías de la Internet de las Cosas desde la academia y la industria. El incremento en el nivel de madurez de la Internet de las Cosas se demuestra por la presencia de Auto-ID en muchos sistemas en producción real. Auto-ID está presente en llaves de coche, en las tarjetas de los empleados, en los tickets para controlar el acceso a algún evento, en las tarjetas de transporte, etc.

Actualmente existen seis formas principales de identificación automática (Auto-ID) (Jamali et al., 2007):

Códigos de barras. Consisten en una representación óptica leible por una máquina. Su uso requiere una línea de visión directa entre la máquina lectora y la etiqueta, que suele requerir la intervención del usuario. Es la tecnología dominante, pues se usa en cualquier supermercado al pasar la compra por la caja.

Los códigos de barras bidimensionales han aparecido recientemente. Estas tecnologías codifican la información en una imagen bidimensional, en el caso de los Códigos QR, las imágenes están formadas por una matriz de cuadrados blancos y negros.

Memorias de contacto. Recipiente de acero inoxidable en forma de moneda que encapsula una memoria. Esta memoria es accedida cuando entra en contacto con el objeto, por ejemplo una huella digital.

Bandas magnéticas. Una banda de material magnético en una tarjeta. El uso más común son las tarjetas de crédito.

Bandas ópticas. Se añade a una tarjeta un panel de material sensible al láser, similar al de CD ROMs o DVDs.

Identificación por radiofrecuencia (RFID). Consiste en la transmisión de la identidad de un objeto mediante ondas de radio. Una etiqueta RFID está formada por

un chip y una antena. Como no se necesita una línea de visión directa entre las etiquetas y los lectores, proporciona un alto nivel de automatismo.

Smartcards. Son tarjetas que incluyen circuitos integrados capaces de procesar información.

2.3 Procesos de Negocio

En el ámbito empresarial, un proceso de negocio (Business Process) se define como un conjunto de tareas que siguen un orden lógico establecido para lograr un resultado real y palpable a las organizaciones, en el contexto de su negocio.

La gestión de los procesos de negocio, Business Process Management (BPM), es una disciplina formada por un conjunto de técnicas y tecnologías que permiten a la organización administrar y optimizar de forma continua sus actividades y procesos de negocio.

2.3.1 Modelado de procesos de negocio

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

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

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

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

BPMN proporciona cuatro categorías de elementos de modelado:

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

4. Artefactos: Grupo, Objeto de datos y Asociaciones

Elementos de flujo

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

- **Evento**, sirve para representar algo que ocurre en el proceso. Este 'algo' tiene una causa (disparador de alguna acción) y un efecto (un resultado). Existen tres tipologías de eventos, según su representación en el proceso:
- Eventos de inicio (Start), para representar flujos de entrada.
- Eventos intermedios (Intermediate), para representar flujos tanto de entrada como de salida.
- Eventos de fin (End), para representar flujos de salida.

La Figura 2.5 muestra los eventos completos que están definidos en la notación:

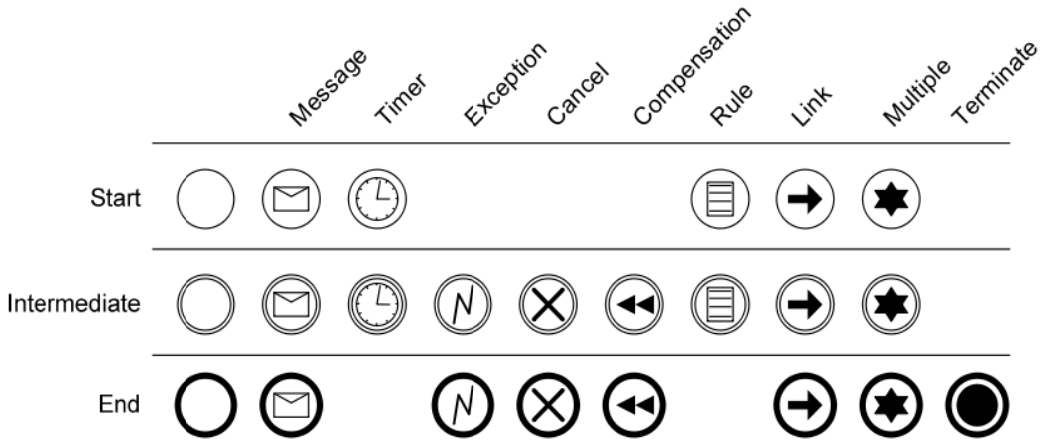


Figura 2.5. Notación BPMN - Elementos de flujo: Eventos

- **Actividad**, es un término genérico para denominar una acción hecha por un participante de un proceso. Las Actividades pueden ser atómicas (task) o no atómicas (sub-process). Además, se incluyen atributos en la notación para indicar si la actividad se realiza una sola vez o se repite, detallando si las repeticiones se hacen de manera secuencial (looping) o en paralelo (instancias múltiples). La Figura 2.6 muestra los tipos de Actividades definidas:

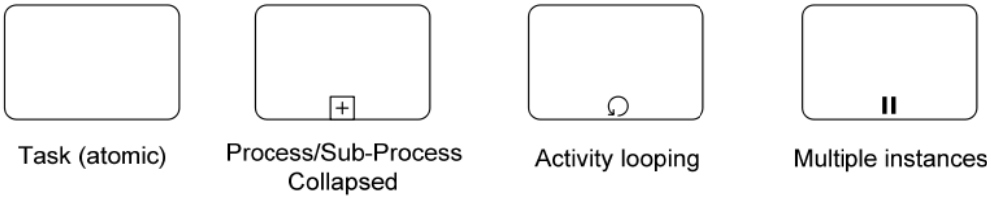


Figura 2.6. Notación BPMN - Elementos de flujo: Actividades

- **Bifurcaciones (Gateways)**, es usado para controlar la convergencia y/o divergencia en la secuencia del flujo de proceso. Estas bifurcaciones determinan divisiones, uniones y combinaciones. La Figura 2.7 muestra los tipos de Bifurcaciones definidas:

Figura 2.7. Notación BPMN - Elementos de flujo: Bifurcaciones

Elementos de conexión

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

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

La Figura 2.8 muestra los diferentes tipos de elementos de conexión.

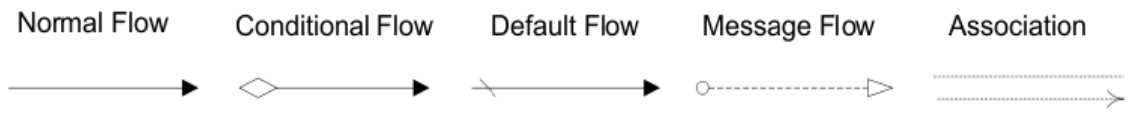


Figura 2.8. Notación BPMN - Elementos de conexión

Swimlanes

Estos elementos representan la responsabilidad en los diagramas: "quién hace qué". Son rectángulos que definen el proceso y los participantes. Hay dos tipos de elementos de Swimlanes:

- Pool, representa un participante dentro del proceso.
- Lane, subdivide los elementos Pool en entidades más lógicas. Por ejemplo, un Pool puede ser una empresa y los Lanes puede ser los departamentos en que se divide.

La Figura 2.9 muestra las franjas de responsabilidad:

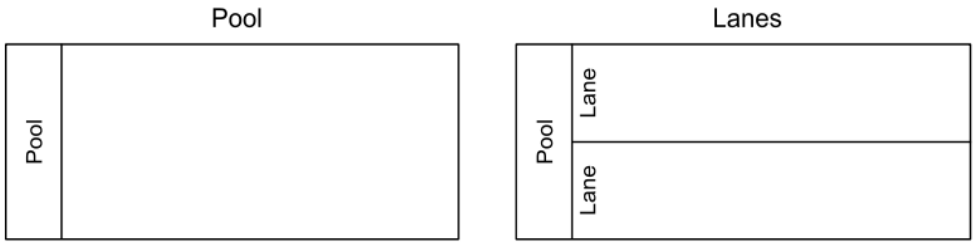


Figura 2.9. Notación BPMN - Swimlanes

Artefactos

Son elementos que no influyen en el cambio del flujo de ejecución, pero su objetivo es mejorar la comprensión del proceso. La notación BPMN define los siguientes artefactos:

- **Grupo**, permite agrupar actividades, como propósito de documentación.
- **Objeto de datos**, representa la información que una actividad requiere o su resultado.
- **Anotaciones**, permite adicionar información, para clarificar la lectura del diagrama.

La representación de estos elementos es mostrada a continuación, en la Figura 2.10:

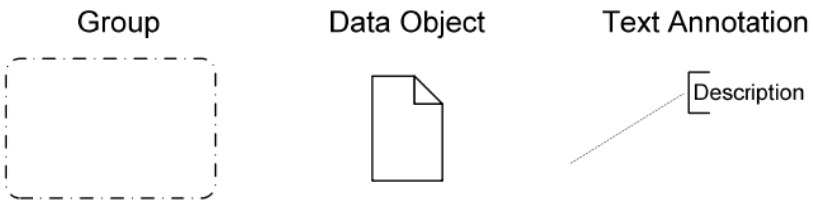


Figura 2.10. Notación BPMN - Artefactos

Además de estas cuatro categorías, la notación BPMN maneja conceptos avanzados de modelado tales como: manejo de excepciones, transacciones y compensación. Podemos ver un ejemplo en la figura 2.11.

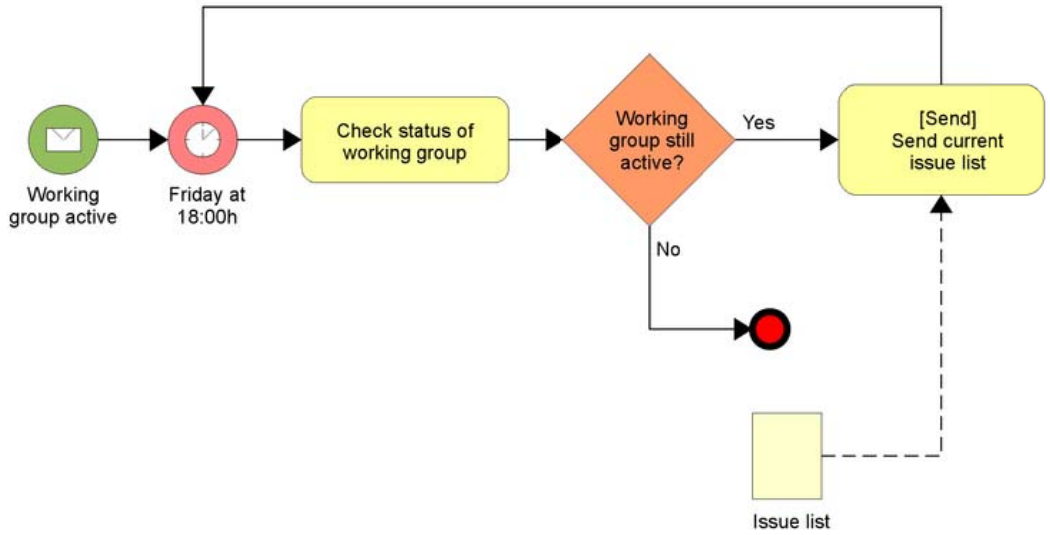


Figura 2.11. Notación BPMN - Ejemplo

Recientemente ha aparecido la especificación BPMN 2.0 [OMG, 2011], que aporta cambios en la notación y cambios técnicos. Los principales cambios son:

- BPMN 2.0 es extendido hacia un modelo y una notación que incluye un metamodelo.
- Nuevos modelos y diagramas: Conversación y Coreografía.
- Extensión del modelo de Colaboración: Múltiples participantes y nuevo objeto de mensajes.

- Extensión de la tipología de actividades, eventos y gateways: muchos nuevos tipos.
- Definición de un metamodelo de intercambio: basado en diagramas de clases de UML.
- Reglas para la ejecución de diagramas y mapeo hacia BPEL.

En http://www.bpmb.de/images/BPMN2_0_Poster_ES.pdf se pueden ver un resumen con todas las novedades en cuanto a notación.

2.3.2 Ejecución de procesos de negocio

El modelado de procesos de negocio es muy útil para capturar los requisitos, pero también es necesario tener definiciones ejecutables del proceso de negocio que permitan una rápida actualización de los procesos de los sistemas de información.

Existen diferentes lenguajes que permiten la definición de la ejecución de procesos de negocio.

- XML Process Definition Language (XPDL)
- Yet Another Workflow Language (YAWL)
- Web Service Business Process Execution Language (WS-BPEL) [Alves et al., 2007]. Lenguaje basado en XML para la coordinación de Servicios Web.

WS-BPEL es uno de los lenguajes más usados en el área de la ejecución de procesos de negocio, existiendo muchas soluciones tanto comerciales como de código abierto. Existen transformaciones que permiten trasladar de forma automática los procesos de negocio modelados con BPMN a una definición ejecutable WS-BPEL [Giner et al., 2007a].

Con la definición de BPMN 2.0 han aparecido herramientas que permiten la ejecución de procesos BPMN. Algunas de estas herramientas son Intalio BPMS[6] y Oracle BPM[7].

[6] http://www.intalio.com/bpms
[7] http://www.oracle.com/us/technologies/bpm/index.html

2.4 Flujos de Trabajo móviles “Physical Mobile Workflows”

En primer lugar vamos a definir las propiedades que debe cumplir un sistema que soporte flujos de trabajo móviles:

1. **Un proceso de negocio bien definido**. Las actividades involucradas deben ser claramente definidas y coordinadas.
2. **Identificación explícita**. Para integrar los elementos físicos en el proceso de negocio se utilizan tecnologías Auto-ID.
3. **Participación del usuario**. La participación del usuario en el flujo de trabajo se considera básica.

Los flujos de trabajo móviles son un tipo específico de sistemas que se pueden considerar en la intersección de la Gestión de los Procesos de Negocio (BPM), Aplicaciones para Dispositivos Móviles y la Internet de las Cosas En las secciones anteriores hemos visto de forma general, conceptos de estas áreas.

En esta sección vamos a concretar más y comentaremos (figura 2.12) las tres áreas de investigación que cumplen la mayoría de las propiedades comentadas arriba: flujos de trabajo inteligentes (*smartworkflows*), interacción con dispositivos móviles (*physical mobile interactions*) y procesos de negocio móviles (*mobile business processes*).

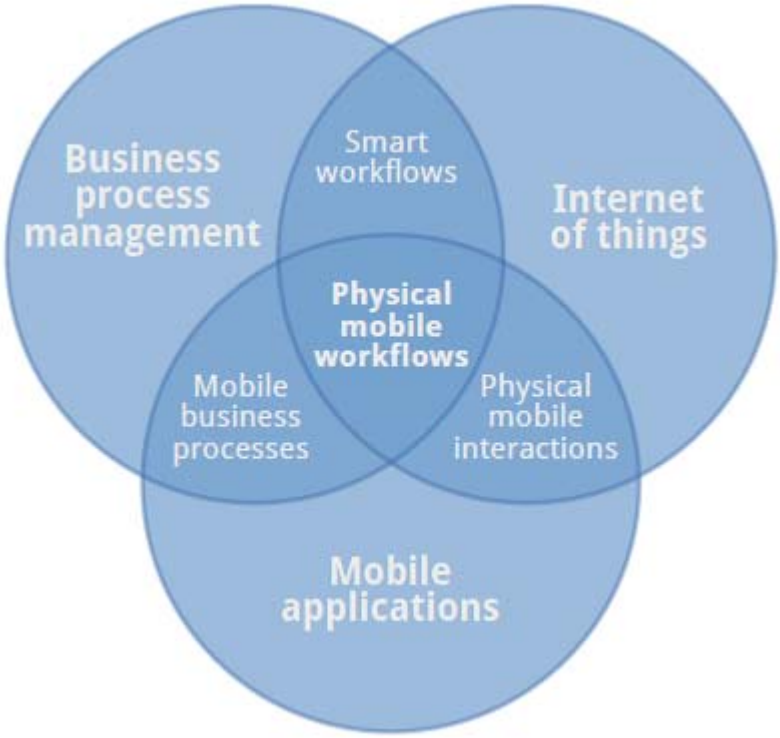


Figura 2.12. Flujos de trabajo móviles. Áreas de investigación y su intersección [Giner, 2010]

2.4.1 Flujos de trabajo Inteligentes “smart workflows”

Los Flujos de trabajo Inteligentes [Wieland et al., 2008] son procesos de negocio que cruzan la frontera entre el mundo físico y el digital. Los flujos de trabajo móviles pueden ser considerados un tipo específico de Flujos de trabajo Inteligentes que son accedidos mediante un dispositivo móvil.

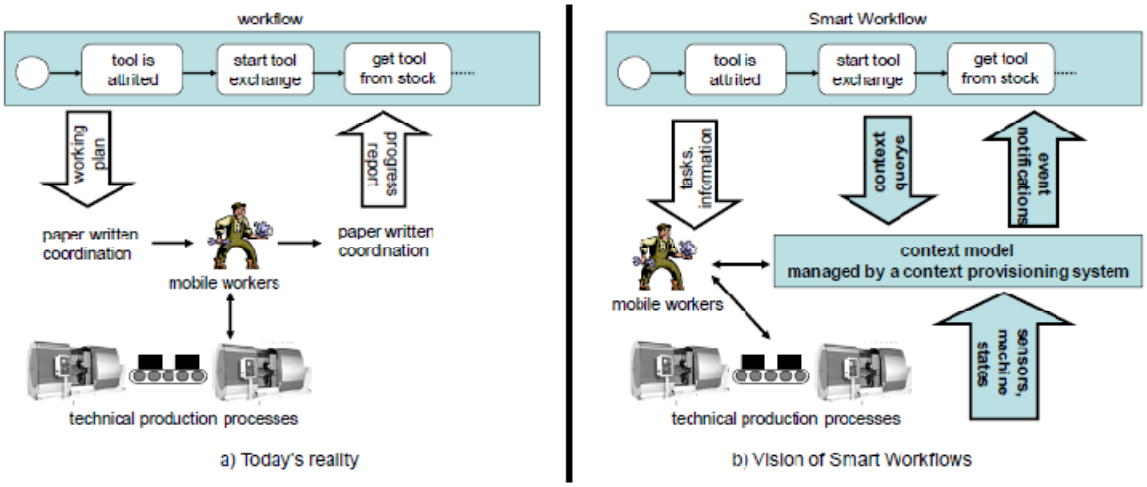


Figura 2.13. Flujos de trabajo inteligentes [Wieland et al., 2008]

Dentro de esta área destacan las siguientes propuestas:

Flujos de trabajo sensibles al contexto. Wieland define los Flujos de Trabajo Inteligentes y proporciona una arquitectura para transformar la información de bajo nivel capturada mediante sensores (no solo mediante Auto-ID), en información del nivel de negocio. Propone un modelo de arquitectura formado por tres capas basado en la plataforma Nexus[8]. La figura 2.14 muestra la arquitectura de un sistema que soporta el proceso productivo de una fábrica.

[8] http://www.nexus.uni-stuttgart.de

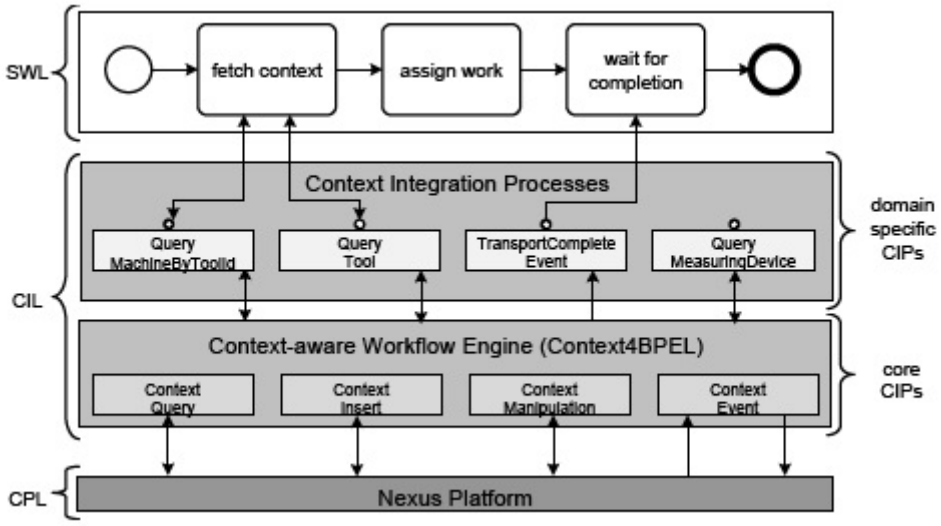


Figura 2.14. Flujos de trabajo inteligentes. Arquitectura propuesta por Wieland [Wieland et al., 2008]

Para soportar la definición flujos de trabajo se definió el lenguaje Context4BPEL, un lenguaje de modelado para flujos de trabajo sensibles al contexto.

PerFlows. Se definen como flujos de trabajo personales [Urbanski et al., 2009]. Los sistemas PerFlow tienen como objetivo la interoperabilidad entre los dispositivos y las aplicaciones para soportar las tareas personales del usuario.

DecoFlows. Un marco de trabajo basado en tareas para enlazar las tareas del usuario con los servicios del sistema [Loke, 2009].

Infraestructura Auto-ID definida mediante SAP. [Bornhövd et al., 2004] Tiene como principal objetivo convertir la información capturada mediante sensores en información del proceso de negocio mediante reglas de mapeado y metadatos. Se centra en la tecnología RFID para Auto-ID.

2.4.2 Interacción con dispositivos móviles “physical mobile interactions”

La Interacción con dispositivos móviles [Ruzkio et al., 2006] son interacciones entre un usuario y un objeto del mundo real dotado de información digital mediante un dispositivo móvil. Los flujos de trabajo móviles pueden ser considerados un tipo específico de sistema basado en interacciones con dispositivos móviles que soportan un proceso de negocio bien definido.

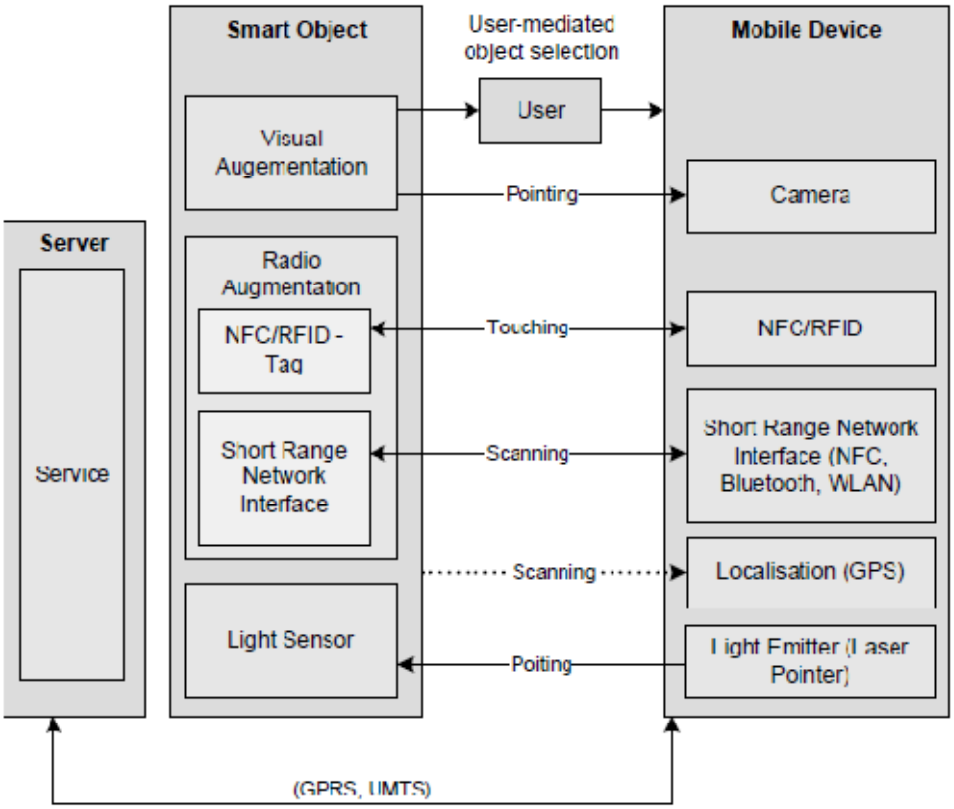


Figura 2.15. Arquitectura genérica del marco de trabajo *Interacciones con Dispositivos Móviles* [Ruzkio, 2007]

Dentro de esta área destacan las siguientes propuestas:

Marco de trabajo para Interacciones con Dispositivos Móviles (PMIF). Este marco de trabajo fue desarrollado para soportar un rápido desarrollo de aplicaciones para dispositivos móviles [Ruzkio et al., 2005b]. En la figura 2.15 se pueden ver los elementos involucrados en esta arquitectura.

Perfil de Interfaz de Usuario sensible al Contexto (CUP). Definida por [den Borgh & Coninx, 2005], es una notación basada en UML que permite la especificación de requisitos para integrar el contexto dentro de aplicaciones interactivas.

Perfil Físico de Interfaz de Usuario (PUIP). PUIP [Ruzkio et al., 2005a] soporta el diseño de diferentes aspectos de la interacción con dispositivos físicos. Extiende CUP y también se basa en UML.

2.4.3 Procesos de negocio móviles “mobile business processes”

Los procesos de negocio móviles permiten a los usuarios interactuar con los servicios de los procesos de negocio de una forma cercana al lugar donde son necesarios. Los flujos de trabajo móviles pueden ser considerados un tipo específico de procesos de negocio móviles en los que los elementos físicos están integrados en el proceso mediante Auto-ID.

Dentro de esta área destacan las siguientes propuestas:

PerCollab. PerCollab [Chakraborty & Lei, 2004] usa una version extendida de WS-BPEL (xBPEL) para definir formalmente procesos de negocio con la participación humana haciendo uso del contexto dinámico del usuario para conseguir resolver problemas de sistemas móviles.

Jardinería de Procesos Móviles (PML). PML [Köhler & Grhun, 2004] es un método para descomponer procesos de negocio en diferentes niveles de detalle para identificar subprocesos móviles.

Sliver. Sliver [Hackmann et al., 2006] es un motor de ejecución para dispositivos móviles que soporta SOAP y WS-BPEL.

Motor de flujos de trabajo móviles. Este motor es presentado [Pajunen & Chande, 2007] como una plataforma capaz de sincronizar aplicaciones locales con servicios del sistema.

2.5 Conclusiones

En este capítulo hemos visto como el presente trabajo se enmarca dentro del ámbito de los flujos de trabajo móviles, los cuales se pueden considerar en la intersección de tres dominios: La Internet de Las Cosas, Procesos de Negocio y Aplicaciones Móviles. Posteriormente hemos concretado aún más las áreas de investigación de estos dominios: *Procesos de Negocio Móviles, Flujos de Trabajo Inteligentes* e *Interacciones con Dispositivos Móviles*. Cualquier sistema que soporte flujos de trabajo móviles estará por tanto, en la intersección de estas tres áreas.

3. Técnicas de prototipado en el ámbito de los sistemas ubicuos

Un prototipo es un modelo conceptual que captura la intención o la idea de un diseño. A menudo los prototipos están incompletos, pues lo que se quiere es que el usuario capte la idea antes de proceder al desarrollo. Los beneficios de usar prototipos son:

1- Comunicar el concepto del diseño. Los prototipos tienen la capacidad de mostrar la idea o el concepto funcional de la aplicación.
2- Tomar decisiones de diseño. En muchas ocasiones, un mismo problema tiene varias soluciones de diseño posibles. Elegir la correcta puede ser muy difícil. Crear un rápido prototipo de cada posible solución puede ayudar a elegir la solución de diseño correcta. No es necesario construir elaborados y funcionales prototipos, pueden ser diseños en papel, presentaciones powerpoint, animaciones flash, páginas HTML, etc.
3- Testear los conceptos de diseño. Una vez se ha tomado una decisión y se ha elegido un diseño, antes de proceder a realizar el desarrollo, es necesario testear estos conceptos con el usuario. En este caso el prototipo debe ser lo más cercano posible a la aplicación final, para que el usuario evalúe su utilidad y facilidad de uso.

Si a todo lo anterior añadimos lo complicado y costoso que puede llegar a ser el desarrollo de una aplicación móvil corporativa que soporte procesos de negocio (ver sección 2.1.2), se enfatiza aún más la importancia de obtener prototipos en este contexto. En este capítulo vamos a comentar algunas de las técnicas de prototipado que se pueden seguir en el contexto de los sistemas ubicuos, así como trabajo relacionado, terminaremos con las conclusiones.

3.1 Técnicas y trabajo relacionado

Dentro del marco específico del prototipado en la computación ubicua todos los autores coinciden en su importancia y en su dificultad. Importancia por la reducción de costes y tiempo en el desarrollo que se consigue con el uso de prototipos, y dificultad por lo complicado de simular el entorno del usuario.

Hay herramientas específicas para desarrollar prototipos para sistemas ubicuos, como

- “Activity Designer” [Li et al., 2008]. Obtiene prototipos para procesos basados en actividades. Permite a los diseñadores modelar actividades basadas en escenarios concretos de su vida diaria. En la figura 3.1 se puede ver su entorno de desarrollo.
- “CogTool” [Bonnie et al., 2005]. CogTool al igual que el presente trabajo genera pantallas en html. En este caso para crear un guión gráfico o *storyboard*. Podemos ver un ejemplo en la figura 3.2.
- “Dart” [Dow, 2005]. Se integra dentro de Macromedia Director, un entorno de desarrollo multimedia ampliamente utilizado. Los prototipos se centran en especificar relaciones complejas entre el mundo real y virtual. En la figura 3.3 se puede ver su entorno de desarrollo.
- “D.Tools” [Bohlen et al., 2005]. Permite crear prototipos de forma rápida. Estos prototipos permiten que se les conecte componentes hardware de forma sencilla, para recrear así la interacción con el usuario. En la figura 3.4 se puede ver su entorno de desarrollo.

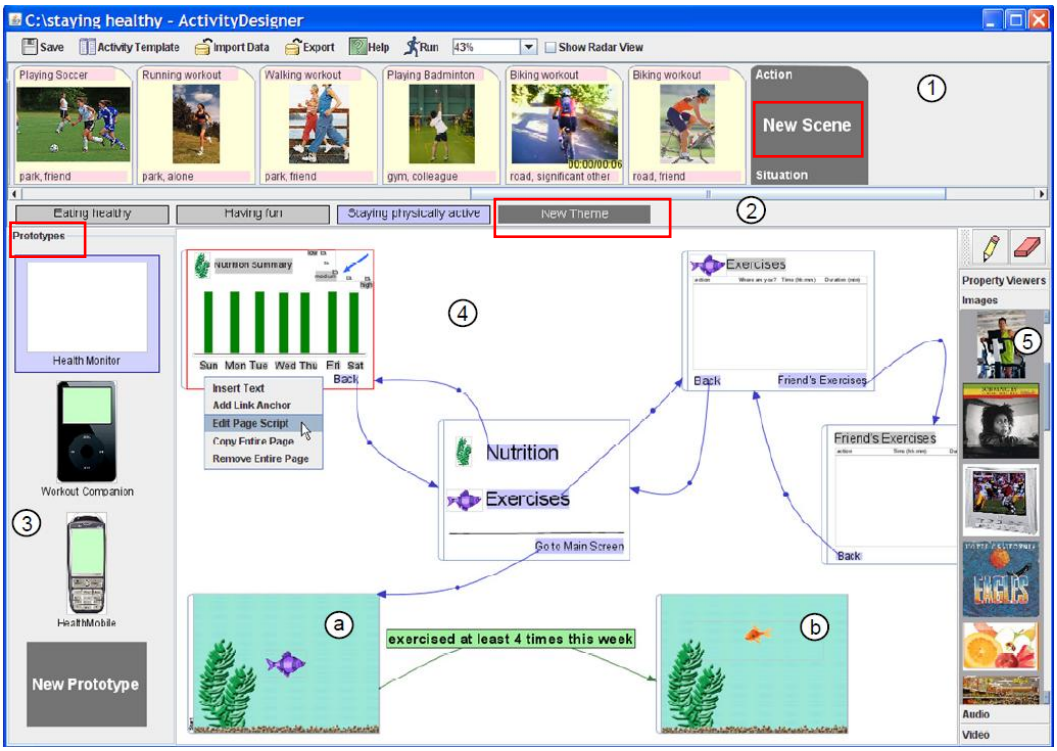


Figura 3.1. Prototipos - Entorno de desarrollo de ActivityDesigner

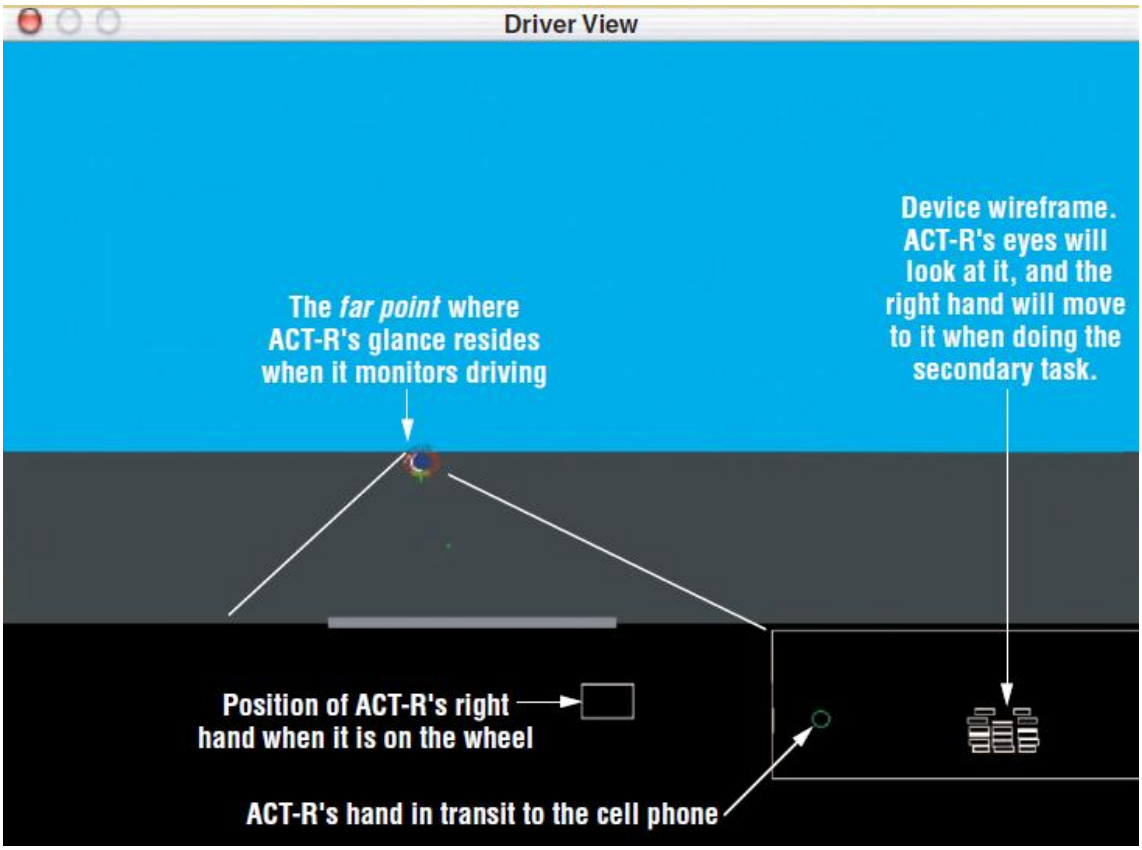


Figura 3.2. Prototipos - CogTool. Ejemplo de interfaz donde se muestra un conductor realizando una tarea secundaria

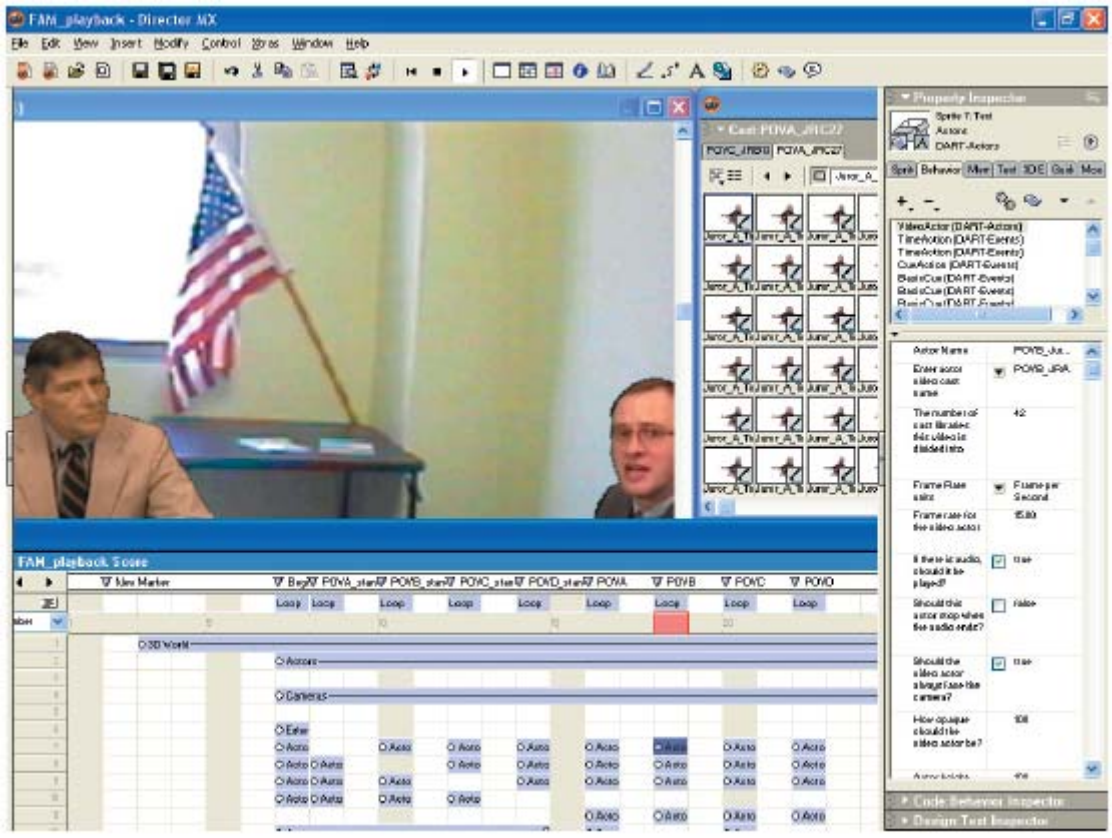


Figura 3.3. Prototipos - Entorno de desarrollo de DART

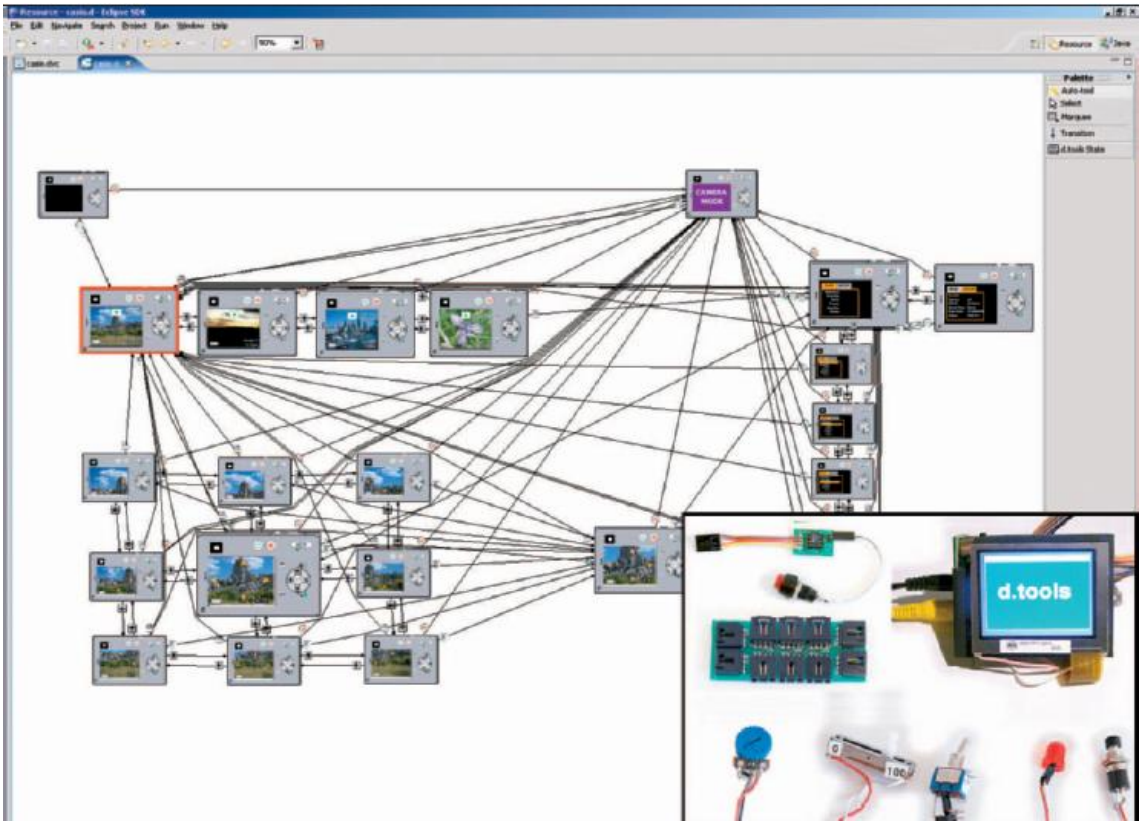


Figura 3.4. Prototipos - Entorno de desarrollo de D.Tools

El uso de técnicas "Mago de Oz" para simular la tecnología sin tener que implementarla utilizada en nuestra propuesta, es utilizada en los trabajos de [Stringer et al., 2005], [Dow et al., 2005] figura 3.5, [Reilly et al., 2005] figura 3.6, entre otros. Reilly comenta que el uso de este tipo de técnicas nos aleja del entorno real del usuario, Abowd también expone que no se pueden usar estas técnicas en el desarrollo de "paratypes" (prototipos evaluados en entornos reales) [Abowd et al., 2005].

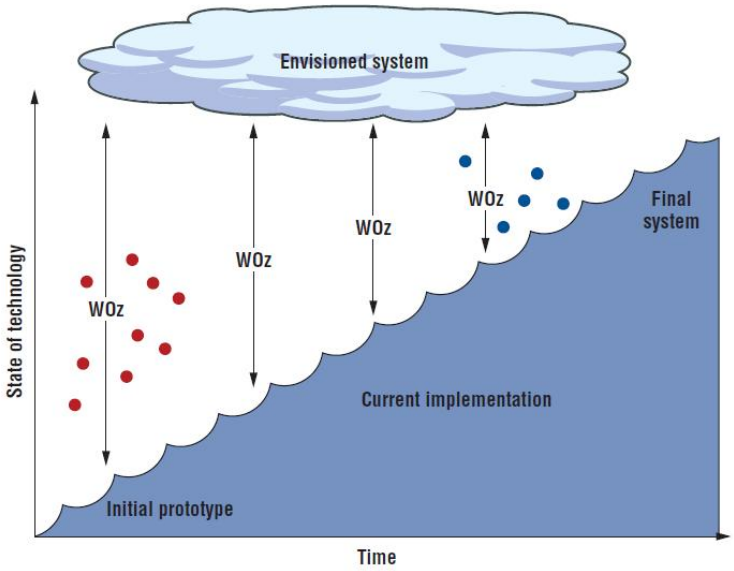


Figura 3.5. Prototipos - Uso de simulación Mago de Oz [Dow et al., 2005]

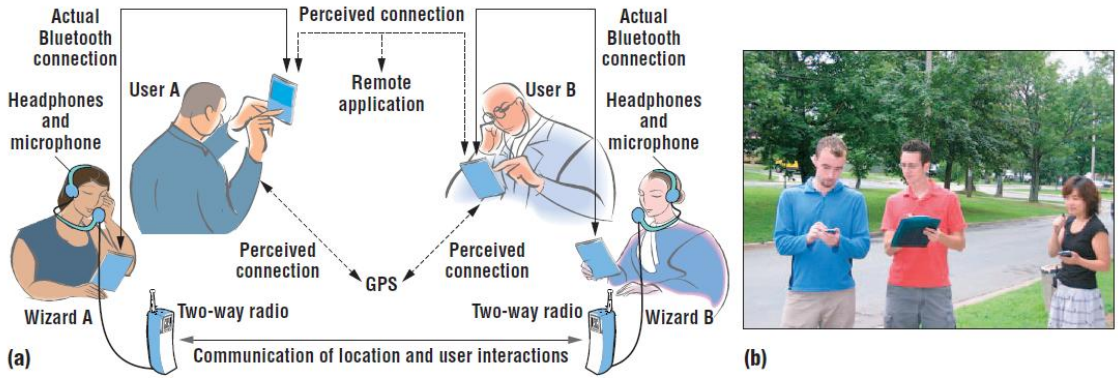


Figura 3.6. Prototipos - Técnica Mago de Oz usada en [Reilly et al., 2005]

Siek en el artículo "Advances in Evaluating Mobile and Ubiquitous Systems" [Siek et al., 2009] presenta una serie de papers relevantes en computación ubicua y comenta la falta de una metodología para evaluar los sistemas ubicuos (tabla 3.1).

Objetivo	**Problemas**
Estudiar la usabilidad en ambientes reales	Falta de metodología Analizar datos recogidos Ética Naturaleza multi-disciplinar
Evaluar los sistemas ubicuos con usuarios	Contexto Medir y detectar el uso o no del sistema por parte de los usuarios
Meta final	***"advancing the state of the art towards a consistent set of methods and guidelines"***

Tabla 3.1. Problemas en el desarrollo de sistemas ubicuos

En el marco de *flujos de trabajo móviles,* tenemos la tesis de Pau Giner "Automating the development of Physical Mobile Workflows" [Giner, 2010], en cuyo trabajo se expone una metodología para desarrollar un sistema en este marco, un aparte del desarrollo comenta el prototipado rápido mediante html mock-ups y "Mago de Oz", esta parte es la que se ha desarrollado y complementado en el presente trabajo. Otros trabajos sobre *flujos de trabajo móviles* son [Giner et al., 2009] donde se presenta la plataforma Presto para modelar procesos de negocio móviles y el trabajo de [Torres et al., 2007] para modelar procesos de negocio.

Como se comentó en la sección 1.3 nuestra propuesta parte de la ingeniería dirigida por modelos (MDE) para obtener los prototipos, En esta línea de usar tecnologías MDE en sistemas ubicuos tenemos de nuevo el trabajo de [Torres et al., 2007], también [Giner & Torres, 2007] y [Giner et al., 2007b], estos tres trabajos están centrados en los

procesos de negocio, y en el ámbito más general de la Internet de las Cosas tenemos [Giner et al., 2008].

3.2 Conclusiones

Hemos visto diferentes trabajos en prototipado para sistemas ubicuos, así como diverso trabajo relacionado, tanto con los flujos de trabajo móviles, como con aproximaciones MDE dentro de la computación ubicua. En el siguiente capítulo vamos a ver las herramientas y tecnología que hemos usado en el presente trabajo, para pasar a ver en el capítulo cinco el desarrollo de nuestra propuesta.

4. Tecnología y herramientas usadas en el desarrollo de la propuesta

En el capítulo 2 hemos visto el contexto tecnológico en el que se enmarcan los flujos de trabajo móviles. En el capítulo 3 hemos visto la importancia de prototipo, así como técnicas y trabajo relacionado. Empezaremos este capítulo hablando de MDE y cómo encaja en el presente desarrollo, pasando en la siguiente sección a ver las herramientas usadas dentro de la plataforma Eclipse[9] para desarrollar la propuesta.

4.1 Ingeniería dirigida por modelos: MDE

MDE se basa en el desarrollo de un sistema a partir de un modelo, el cual es una descripción abstracta del sistema e independiente de la tecnología a utilizar. Para ello es necesario una definición inambigua y precisa de los conceptos y relaciones que forman parte del modelo, lo cual permite procesar el modelo bien para transformarlo en otro modelo o bien para generar código de una forma automática.

En nuestro desarrollo utilizaremos los siguientes conceptos de la ingeniería dirigida por modelos:

1. Metamodelo.
2. Validación basada en modelos.
3. Generación de código.
4. Automatización.

Para la definición del metamodelo usaremos Ecore dentro del marco de trabajo Eclipse Modeling Project (EMP), y para la validación del modelo, generación del código y automatización usaremos la herramienta openArchitectureWare (OAW), la cual lee un metamodelo y lo usa para validar los modelos creados, y no solo para validar el modelo, también para generar código, y todo de forma automática. En el capítulo siguiente veremos con más detalle estas herramientas

Veamos ahora con más detalle los principales conceptos de la ingeniería dirigida por modelos que sigue nuestra propuesta: metamodelo, validación y generación de código.

[9] http://www.eclipse.org

4.1.1 Metamodelo

MDE propone el uso de metamodelos para definir un sistema formalizando conceptos (objetos) y sus relaciones, de forma que podemos definir un metamodelo como la especificación de un lenguaje (DSL-Domain Specific Language) para definir modelos. Usando metamodelos, la descripción de un sistema mediante un modelo resulta no ambigua a nivel sintáctico, lo que la hace procesable para, mediante técnicas de transformación de modelos, convertirlo en otro modelo o generar código. Lo más interesante de la definición del metamodelo es que es independiente de la tecnología que se va a usar en el desarrollo, de forma que si se cambia la tecnología el metamodelo sigue sirviendo.

En la figura 4.1 podemos ver en un ejemplo como se ha definido el modelo *Perro*, el cual se ha construido en base a un metamodelo, que podríamos considerar como el conjunto de trazos en dos dimensiones. Luego tenemos a Lazy, que es una instancia de *Perro*, un perro en concreto.

Figura 4.1. Modelo e instancia del modelo

El Object Management Group (OMG) definió Model Driven Architecture (MDA) [Miller & Mukerji, 2003] para dar soporte a estas ideas mediante estándares para metamodelado (lenguaje para definir el modelo) y transformación de modelos. En la figura 4.2 se puede ver gráficamente los lenguajes y metalenguajes necesarios para definir un modelo. Un estándar para definir metamodelos definido por OMG es MetaObject Facility (MOF)[10].

[10] http://www.omg.org/mof/

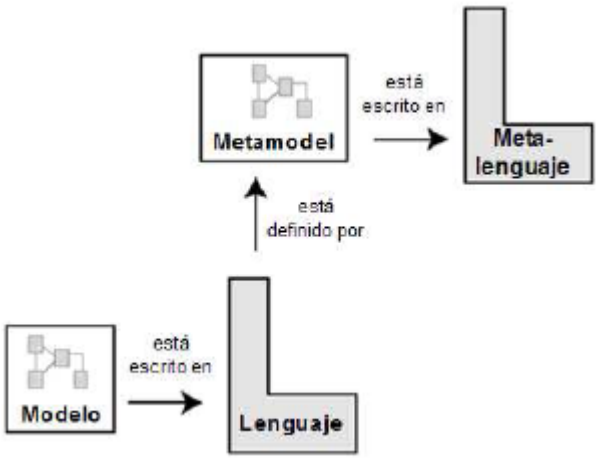


Figura 4.2. Modelos, lenguajes, metamodelos y metalenguajes [Kleppe, 2005]

Como ejemplo, en la figura 4.5 podemos ver la arquitectura de cuatro capas definida por MOF del conocido lenguaje de modelado UML.

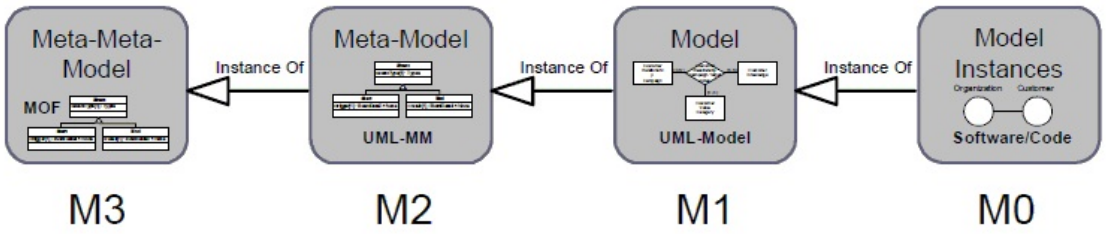


Figura 4.3. Capas MOF para UML

4.1.2 Validación de un modelo

Una vez hemos definido un metamodelo, estamos en condiciones de modelar los conceptos capturados en el mismo y obtener un modelo. No obstante, pueden haber algunos conceptos o reglas que no se puedan expresar en el metamodelo, por ejemplo que haya dos instancias de metaelementos que tengan el mismo nombre. Para evitar estas inconsistenciase existen diferentes lenguajes para validar el modelo obtenido.

MDE enfatiza las técnicas de validación mediante lenguajes de reglas. Normalmente estas reglas se especifican dentro del dominio específico del metamodelo. Uno de los beneficios de usar DSL es que podemos validar los modelos con unas reglas de validación más estrictas, las reglas específicas del dominio más las reglas genéricas de modelado.

La principal referencia a la hora de construir estas reglas es Object Constraint Language [11] (OCL), el cual ha sido aceptado como el estándar en modelos UML. No obstante es un lenguaje difícil de comprender, en el presente desarrollo usaremos el lenguaje *Check*[12] dentro de openArchitectureWare, el cual nos permite simplificar la definición de las reglas.

Este lenguaje es bastante sencillo de usar y entender como se puede apreciar en la tabla 4.1.

Listado 4.1

```
import scenario;
context Element ERROR
    "Element: must have a Initial State!" :
    this.initial_state!=null
```

Veamos el significado del listado 4.1:

1. En primer lugar se importa el metamodelo a validar.
2. Se especifica el contexto dentro del metamodelo (metaclase) al que se aplica la restricción, seguido de ERROR o WARNING, estas palabras indican el tipo de acción a tomar en el caso de que la restricción falle (tabla 4.1)

WARNING	Si la restricción falla se muestra el mensaje y se continúa la ejecución.
ERROR	Si la restricción falla se muestra el mensaje y se detiene el proceso.

Tabla 4.1. Tipos de acción para las restricciones Check

3. Después el mensaje a mostrar si la restricción falla. En este mensaje se pueden acceder a los atributos o a funciones para que sea más descriptivo.
4. Finalmente se pone la condición a evaluar.

4.1.3 Generación de código

En MDE la metodología de desarrollo se trata de una vez obtenido el modelo, realizar una transformación de este modelo, bien en otro modelo, o bien en código (lo cual también puede ser considerado un modelo), y hacerlo de forma automática.

[11] http://www.omg.org/spec/OCL/2.2/
[12] http://www.openarchitectureware.org/

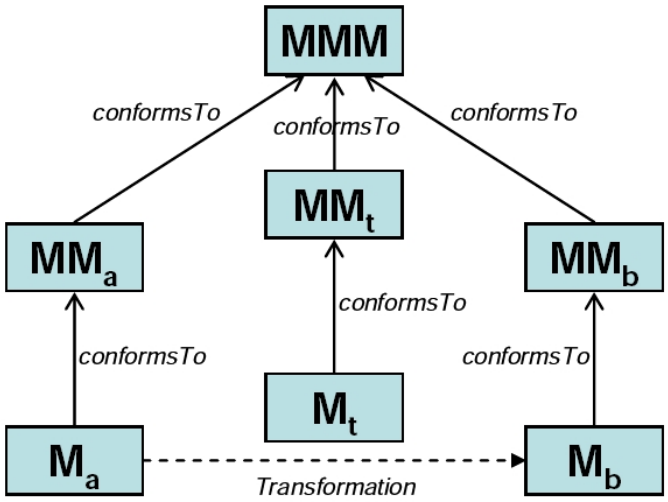


Figura 4.4. Transformación de modelos

Como se puede ver en la figura 4.4, se define una transformación de modelos entre dos metamodelos MMa y MMb, esta transformación permite transformar instancias Ma de MMa en instancias Mb de MMb. Normalmente tendremos un metamodelo de transformación MMt, por ejemplo un lenguaje de transformación de modelos, en el que estará escrita nuestra transformación Mt.

Las transformaciones entre modelos se conocen con el acrónimo M2M (model to model). Entre los lenguajes de transformación M2M tenemos QVT[13], ATL[14], XTend[15], etc.

Como el objetivo final de un desarrollo es la obtención de código, existe una transformación del modelo en texto, estas transformaciones se conocen con el acrónimo M2T (model to text), aunque el código también podría ser considerado un modelo cuyo metamodelo sería el lenguaje de programación.

Las transformaciones M2T son realizadas normalmente por lenguajes basados en plantillas, como Velocity[16], MTL[17], XPand[18], etc. En el presente desarrollo utilizaremos *Xpand* de openArchitectureWare.

Xpand tiene como principales características:

1. Soporte de polimorfismo.

[13] http://www.omg.org/spec/QVT/1.0/
[14] http://www.eclipse.org/m2m/atl
[15] http://www.openarchitectureware.org/
[16] http://velocity.apache.org/
[17] http://www.eclipse.org/modeling/m2t/
[18] http://www.openarchitectureware.org/

2. Soporte de ficheros de extensiones que incluyen expresiones complejas escritas en Xtend (lenguaje de transformación de modelos que forma parte de OAW).
3. Soporte de tipos básicos (String, Boolean, Integer y Real) y colecciones (List y Set), así como los conceptos definidos en el metamodelo.
4. Soporte de recursión

En la tabla 4.2 podemos ver las principales instrucciones del lenguaje, las cuales van siempre entre los símbolos « y » , el resto de texto se genera tal cual está escrito.

Statement	Explicación
import	Hace conceptos del metamodelo visibles a la plantilla
Define	Define una nueva plantilla
File	Abre un fichero y dirige la salida hacia él
Expand	Llama a otra plantilla para uno o varios elementos
Foreach	Itera sobre cada elemento
if	Condición
Extensión	Importa un fichero de extensiones que incluye expresiones complejas
Error	Reporta un error
Let	Define una variable temporal
Rem	Comentarios

Tabla 4.2. Principales instrucciones XPand

Una vez introducidos los conceptos MDE, pasemos en la siguiente sección a ver las herramientas utilizadas dentro de la plataforma Eclipse.

4.2 Herramientas

Las herramientas MDE que usaremos para desarrollar los conceptos MDE de la sección anterior se enmarcan dentro de la plataforma Eclipse, que es un entorno de desarrollo de código abierto multiplataforma. Estas herramientas nos permitirán definir el metamodelo, crear un editor gráfico para el modelado de escenarios, validar los modelos y transformar modelos, en nuestro caso para obtener código.

Empezaremos presentando Eclipse, continuaremos con el proyecto Eclipse Modeling Project (EMP), seguiremos con la herramienta de modelado Eclipse Modeling Framework[19] (EMF), posteriormente veremos Graphical Modeling Framework[20] (GMF) y Eugenia, que nos permitirán crear un editor gráfico para obtener un modelo a partir del metamodelo y terminaremos con OpenArchitectureWare (OAW), herramienta que integraremos dentro de Eclipse para validación y generación de código.

[19] http://www.eclipse.org/modeling/emf/
[20] http://www.eclipse.org/modeling/gmp/

4.2.1 Eclipse

La plataforma de desarrollo que se ha usado para implementar los proyectos, es Eclipse. Este framework nos proporciona un completo entorno de trabajo con el que podemos desarrollar fácilmente nuestros proyectos. Eclipse es una herramienta de código libre, que se puede descargar gratuitamente desde su página Web.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

Eclipse se basa en el concepto de plug-ins, que son los encargados de darle funcionalidad a Eclipse. Cada uno de ellos le aporta algo nuevo, de modo que el conjunto de todos ellos forma una potente herramienta.

El lenguaje de programación principal que maneja Eclipse, es Java, lo que ofrece a nuestros proyectos una completa portabilidad, ya que este lenguaje es ejecutable en muchos sistemas operativos, como Windows y Linux.

El trabajo de Eclipse se basa en un proyecto central, el cual incluye un framework genérico para la integración de las herramientas, y un entorno de desarrollo Java construido utilizando el primero. Otros proyectos extienden el framework central para soportar clases específicas de herramientas y entornos de desarrollo, en la figura 4.5 podemos ver su arquitectura.

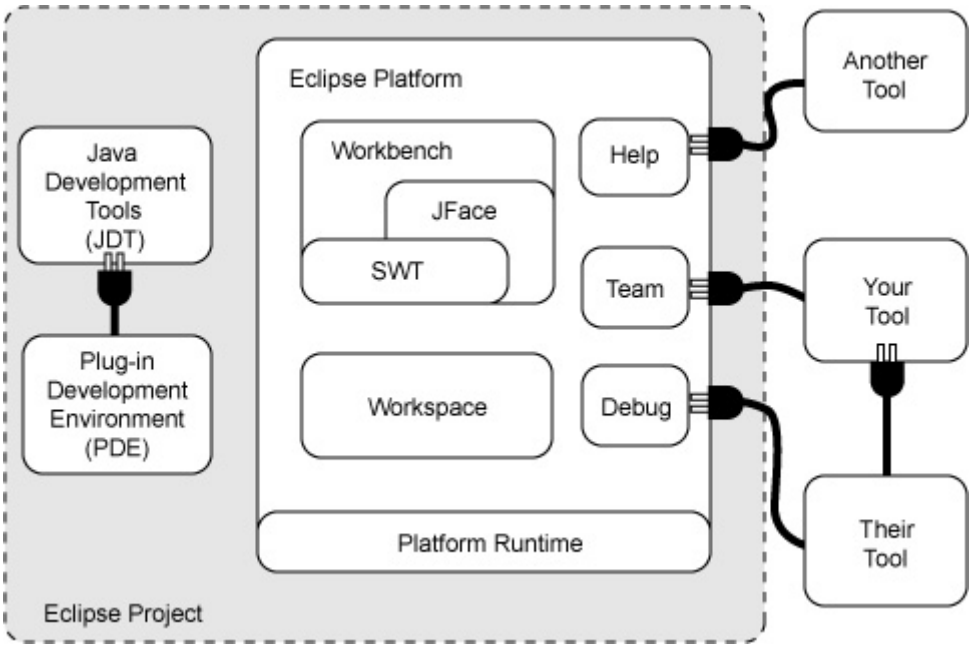


Figura 4.5. Arquitectura Eclipse

4.2.2 Eclipse Modeling Project (EMP)

Dentro de la comunidad Eclipse, EMP se centra en el desarrollo dirigido por modelos, proporcionando un conjunto unificado de marcos de trabajo de modelado, de herramientas y de estándares de implementación. En la tabla 4.3 se pueden ver los diferentes subproyectos, remarcando en color rojo los que hemos usado.

Subproyecto	Descripción
Eclipse Modeling Framework (**EMF**)	Modelado y generación de código.
Eclipse Modeling Framework Technologies (**EMFT**)	Nuevas tecnologías que extienden o complementan EMF
Modeling Amalgamation Project (Amalgam)	Proporciona un empaquetado mejorado, integración y facilidad de uso de los componentes del proyecto de modelado.
Model to Model (M2M)	Transformación modelo a modelo con una implementación del núcleo de QVT
Model To Text (M2T - including JET, XPand, ...)	Transformación de modelos a texto.
Graphical Modeling Framework (GMF)	Desarrollo de editores gráficos basados en EMF y GEF.
Generative Modeling Technologies (GMT - including AM3, AMW, **Epsilon**, GEMS, MoDisco, MOFScript, oAW, OMCW, TCS, UMLX, and VIATRA2)	Orientado a la producción de prototipos en el área de MDE.
Model Development Tools (MDT)	Implementación de los estándares en metamodelado.
Textual Modeling Framework (TMF)	Para desarrollar sintaxis modo texto y sus correspondientes editores basados en EMF.

Tabla 4.3. Subproyectos EMP

EMP soporta los siguientes estándares o los soportará en el futuro

- Estándares Object Management Group (OMG)
  - Meta-Object Facility (MOF)
  - Unified Modeling Language (UML) and UML Profiles.
  - Model-Driven Architecture (MDA) related specifications
  - Query, View, Transformation (QVT)
  - MOF to Text (MOF2T)

    - Diagram Interchange Specification (DIS)
    - XML Metadata Interchange (XMI)
- Business Process Modeling Notation (BPMN)
- Business Process Definition Metamodel (BPDM)
- XML Schema Definition (XSD)

4.2.3 Eclipse Modeling Framework (EMF)

EMF es uno de los subproyectos de EMP, proporciona un marco de modelado y de generación de código para aplicaciones desarrolladas en la plataforma Eclipse basadas en la definición de un metamodelo. EMF genera una serie de clases Java para manipular el modelo y un editor básico en forma de árbol para crear instancias del modelo. En la figura 4.6 podemos ver la arquitectura EMF.

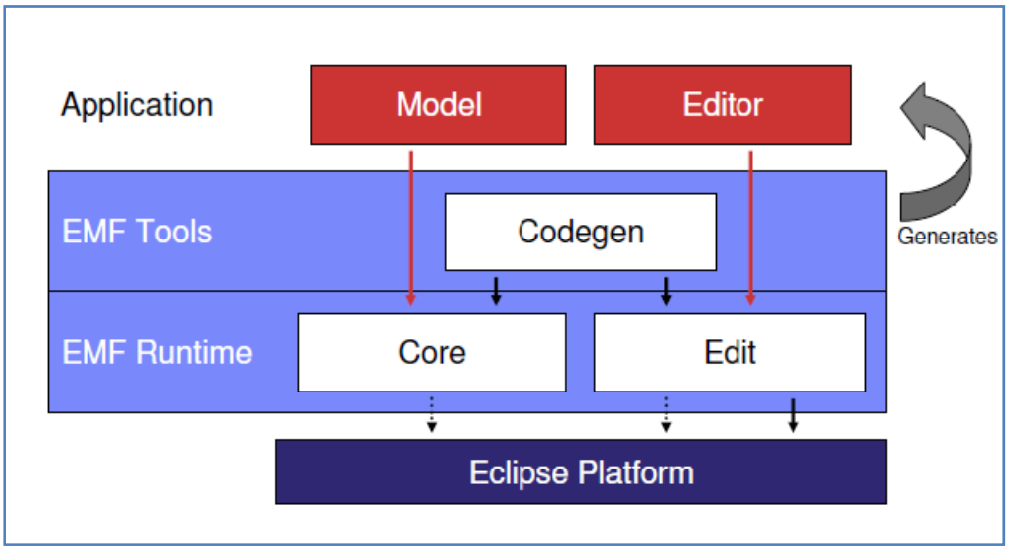


Figura 4.6. Arquitectura EMF

EMF comenzó siendo una implementación de MOF (Meta Object Facility), estándar de OMG (Object Management Group). Actualmente ha evolucionado hacia una eficiente implementación del núcleo (core) de la API de MOF en Java, pasando a llamarse **Ecore** para evitar confusiones. En el actual estándar MOF 2.0 se ha definido un core similar que se ha llamado EMOF (Essential MOF). Aunque hay pequeñas diferencias entre Ecore y EMOF, EMF puede leer y escribir serializaciones de EMOF, siendo por tanto compatible con este estándar. EMF proporciona herramientas para definir el metamodelo y para la creación de modelos a partir del metamodelo definido.

Ecore está formado por los elementos de modelado que se pueden observar en la figura 4.7 (sin atributos) como son EClass para definir clases, EAttribute para definir

sus atributos y EReference para las relaciones. EClasses pueden agruparse en EPackages, que a su vez se pueden estructurar en subpaquetes. Cada elemento de modelado puede tener anotaciones EAnnotation. También hay algunas clases abstractas para estructurar mejor el modelo Ecore como ENamedElement, ETypedElement, etc

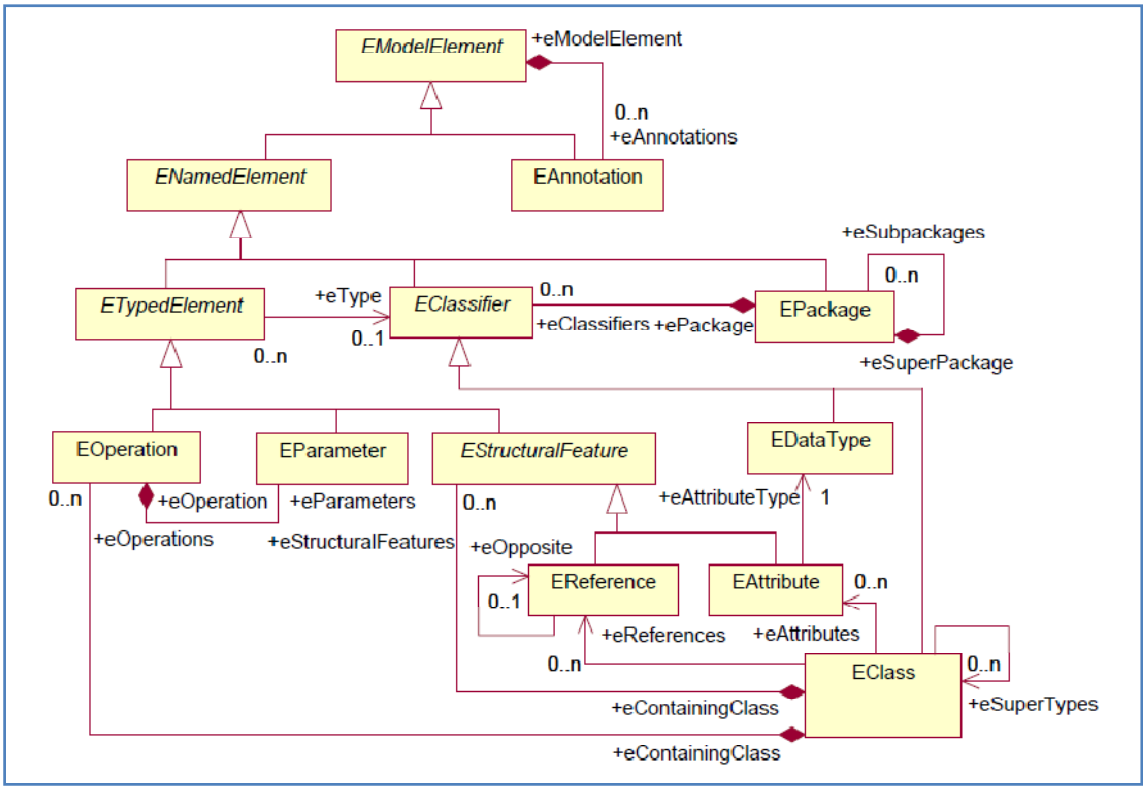


Figura 4.7. Jerarquía de clases del Metamodelo Ecore

EMF usa XMI (XML Metadata Interchange) como su forma canónica para definir un metamodelo. Existen varias formas para que nuestro metamodelo tenga formato XMI:

1. Crear directamente el documento XMI usando un editor de texto o de XML.
2. Exportar el documento XMI desde una herramienta gráfica de modelado.
3. Usar anotaciones en las interfaces de java con propiedades del modelo
4. Usar el XML Schema para describir una serialización del modelo.

Una vez especificado el metamodelo EMF, el generador EMF crea las correspondientes clases Java, aunque no vamos a entrar a comentarlas. Lo deseable es usar, como indica la segunda opción, una herramienta gráfica por ser más intuitiva y sencilla de utilizar, pues solo se necesitan un subconjunto de las características de modelado de UML, en concreto definición de clases, sus atributos y relaciones, por lo que no es necesaria una avanzada herramienta de modelado.

Otra forma de representar nuestro metamodelo es Emfatic[21], un sencillo lenguaje diseñado para representar modelos EMF mediante texto. Esta representación es la que tenemos que usar para añadir las anotaciones necesarias para crear el editor gráfico (ver Eugenia en el punto 4.2.4). El generador de Emfatic puede convertir modelos EMF al formato de texto emfatic y viceversa.

Hemos visto EMF, la herramienta usada para crear un metamodelo ecore desde la plataforma Eclipse. Vamos ahora a ver las herramientas que nos permitirán crear un editor gráfico para obtener un modelo o escenario a partir del metamodelo definido, estas son GMF y Eugenia.

4.2.4 Graphical Modeling Framework (GMF)

Dentro de EMP es posible crear de forma manual un editor gráfico mediante el Eclipse Graphical Editor Framework [22](GEF), pero lo que nos interesa es generar de forma automática un editor, para ello disponemos del subproyecto Graphical Modeling Framework (GMF). GMF parte de la definición del metamodelo para generar el editor, haciendo de puente entre EMF y GEF. En la figura 4.8 se pueden ver sus dependencias.

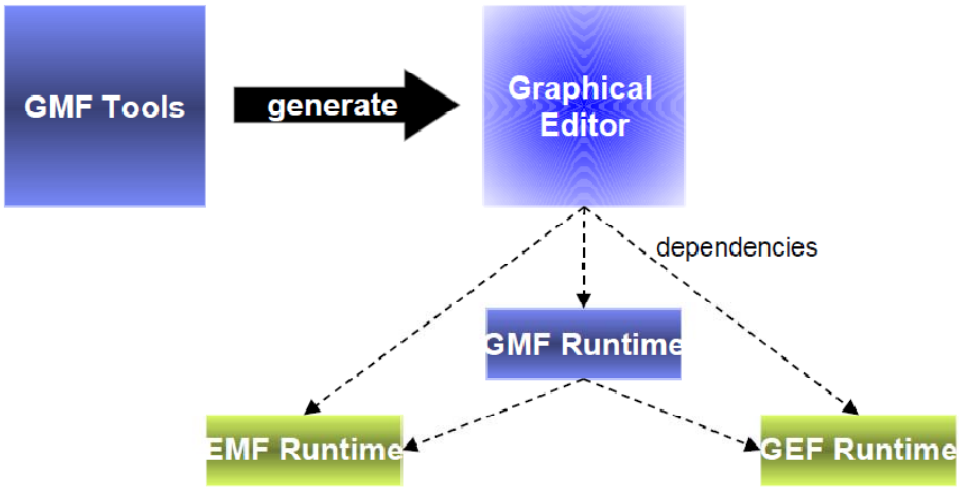


Figura 4.8. Graphical Modeling Framework - Sus Dependencias

Para construir el editor GMF partimos del metamodelo ecore y del modelo *.genmodel*. El *.genmodel* es el modelo usado como entrada por el generador de código,

[21] http://wiki.eclipse.org/Emfatic
[22] http://www.eclipse.org/gef/

EMF lo usa para crear las clases del metamodelo y el plugin *.edit*. GMF genera los siguientes modelos adicionales:

1. **.gmfgraph** – define la notación gráfica: diseño de figuras, nodos gráficos y conexiones entre ellos.
2. **.gmftool** – define la paleta de dibujo y otras herramientas.
3. **.gmfmap** – asocia los dos modelos anteriores con el modelo ecore.

En la figura 4.9 se puede ver como encajan todos los modelos usados. Finalmente se obtiene el plugin con el editor gráfico. En la figura 4.10 se puede ver el GMF Dashboard o asistente que nos guía en la generación del plugin dentro de Eclipse.

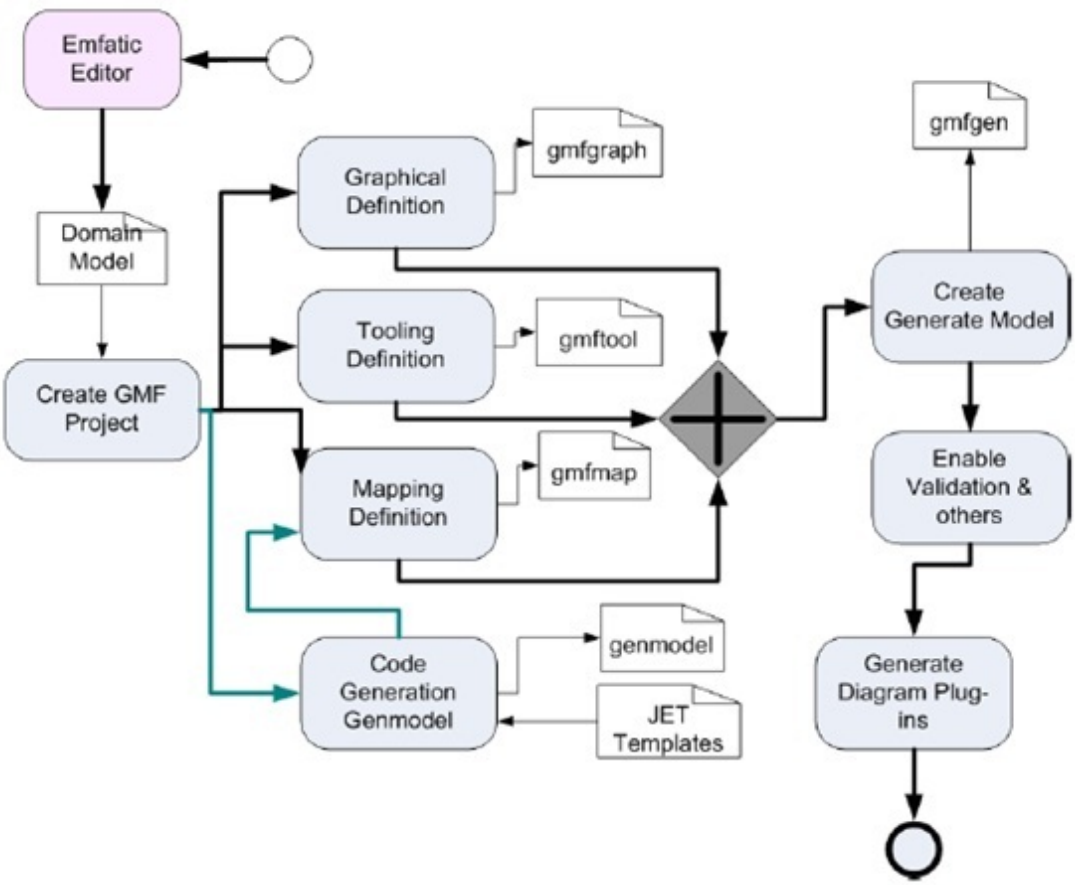


Figura 4.9. Graphical Modeling Framework - Resumen

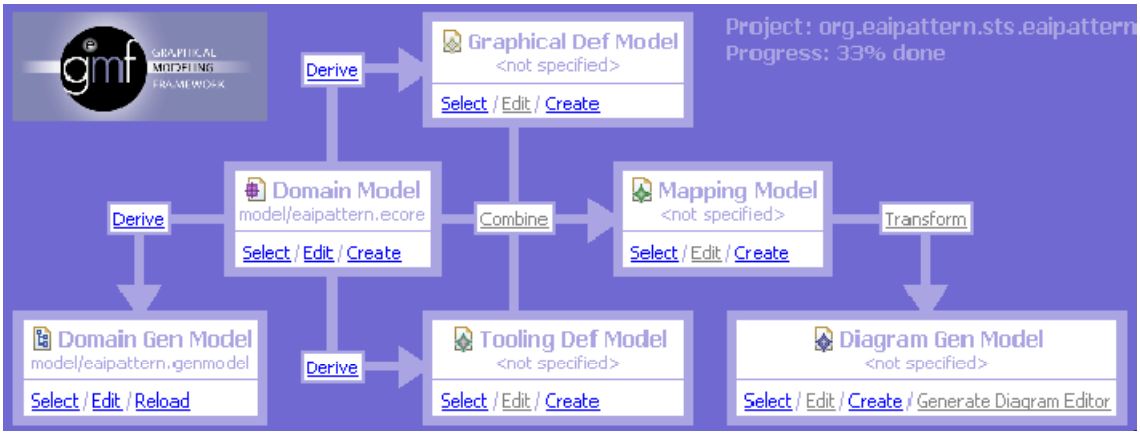


Figura 4.10. GMF Dash Board

4.2.5 Eugenia

Epsilon[23] es una familia de lenguajes de programación que se usan para interactuar con metamodelos EMF para llevar a cabo tareas MDE como generación de código, transformación modelo a modelo, validación de modelos, etc. Estos lenguajes usan todos una forma basada en OCL[24] para acceder al modelo y Epsilon Object Language (EOL[25]) para manipular el modelo.

Epsilon también contiene varias herramientas y utilidades que complementan EMF/GMF, entre ellas la que nos interesa es **Eugenia**, un asistente para generar editores GMF.

Eugenia es una herramienta que automáticamente genera los modelos . *gmfgraph*, *.gmftool y .gmfmap* necesarios para implementar el modelo GMF. Para ello parte de un metamodelo ECore que lleva anotaciones (EAnnotation en figura 4.7) que son interpretadas por Eugenia para crear un primer editor gráfico de una forma muy sencilla, evitando así la complejidad de GMF. Estas anotaciones las escribiremos en el metamodelo en formato emfatic mediante un editor de texto, y posteriormente obtendremos el fichero ecore para empezar a generar de forma automática los modelos.

EuGENia soporta las anotaciones para los elementos Ecore de la tabla siguiente. Los atributos que soportan cada una se pueden ver en http://www.eclipse.org/gmt/epsilon/doc/eugenia/ .

[23] http://www.eclipse.org/gmt/epsilon/
[24] http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL
[25] http://www.eclipse.org/gmt/epsilon/doc/eol/

EPackage	
gmf	Se coloca al principio del fichero e indica que se esperan anotaciones Eugenia
EClass	
gmf.diagram	Denota el objeto raíz del metamodelo. Solo se puede poner en una clase y debe ser no abstracta.
gmf.node	Denota que el objeto debe aparecer en el diagrama como un nodo.
gmf.link	Denota las clases que en el diagrama deberán aparecer como enlaces.
EReference	
gmf.link	Para non-containment EReference, aparece como enlace en el diagram.
gmf.compartment	Para containment EReference, creará un compartimento donde irán los elementos contenidos por el nodo contenedor.
gmf.affixed	Para containment EReference, los nodos contenidos irán pegados al nodo contenedor.
EAttribute	
gmf.label	Define etiquetas adicionales para la EClass que contiene el atributo.

Tabla 4.4. Anotaciones Eugenia

Eugenia también utiliza los siguientes ficheros de transformaciones escritos en EOL para acceder y modificar los modelos implicados de forma automática:

1. ECore2GMF.eol: permite acceder al metamodelo ECore, y a los modelos *GmfTool, GmfGraph y GmfMap*.
2. FixGMFGen.eol: permite acceder al metamodelo ECore y al modelo *GmfGen*.

Como resultado final obtenemos un proyecto eclipse que al ejecutarlo como una nueva instancia de Eclipse pasa a formar parte del entorno, permitiéndonos usar el editor gráfico para crear el modelo.

Pasemos ahora a ver OAW, herramienta que usaremos para validar y generar código.

4.2.6 OpenArchitectureWare (OAW)

Para la generación de código, la validación semántica del modelo obtenido con el editor gráfico y la automatización se ha utilizado la herramienta de código libre OpenArchitectureWare 4.3.1 [Efftinge, 2008], la cual hay que incorporar dentro del entorno Eclipse. OAW utiliza los lenguajes *XPand2* para generar código, *Check* para validar semánticamente el modelo y *Xtend* para extender las metaclases. Todos estos lenguajes han pasado a formar parte del subproyecto Model to Text[26] (M2T) de EMP. En la figura 4.11 podemos ver la arquitectura de OAW.

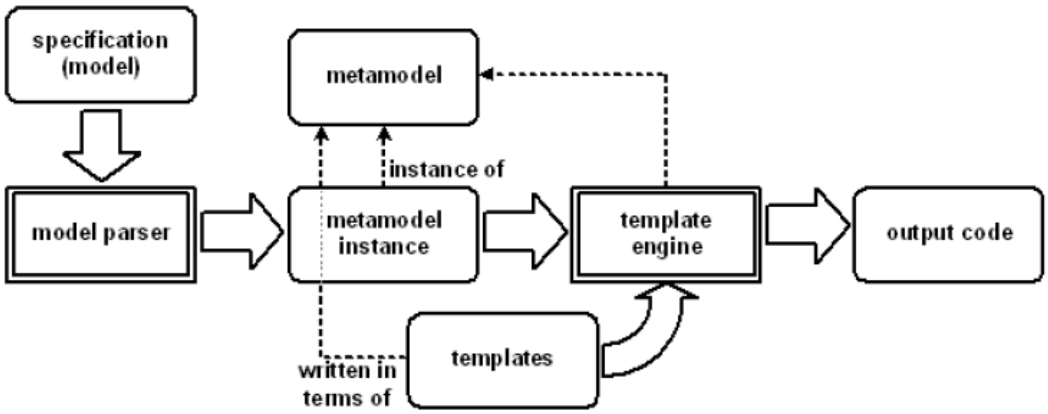


Figura 4.11. Arquitectura OAW

OAW también incorpora un motor MWE (Modeling Workflow Engine) para gestionar todo el proceso de validación y generación de código, para ello se configura el fichero *workflow.oaw* del proyecto. Dentro del fichero *workflow.properties* se ponen valores de las variables usadas dentro del fichero oaw, como *modelFile*, que nos indica el nombre del fichero que guarda la instancia del modelo, o *checkFile*, que nos indica donde está el fichero con las reglas de validación. De este modo las transformaciones utilizadas son independientes de una configuración específica, aumentando su reutilización.

Los pasos que se siguen en el fichero workflow.oaw a la hora de validar y generar código son los siguientes:

1. En primer lugar cargamos la definición del metamodelo.
2. Validamos la instancia del modelo.
3. Vaciamos el directorio en el que se va a generar el código.

[26] http://www.eclipse.org/modeling/m2t/

4. Generamos el código aplicando la plantilla xpand definida y dejamos el resultado en el directorio que previamente habíamos borrado.

OAW no incorpora herramientas específicas para crear los ficheros necesarios, con un simple editor de texto es suficiente.

4.3 Conclusiones

En este capítulo hemos visto los conceptos MDE utilizados en nuestro desarrollo, así como las herramientas utilizadas dentro de la plataforma Eclipse. En el apéndice F se explica cómo instalar Eclipse y todos los plugins necesarios, y en el apéndice G hemos puesto un paso a paso sobre cómo crear todos los proyectos y ficheros necesarios para recrear el presente trabajo.

Como resumen los pasos que seguiremos son:

1. Crearemos el metamodelo (ecore, emfatic) que nos permitirá diseñar escenarios que soportan flujos de trabajo móviles .
2. Crearemos un editor gráfico (GMF y Eugenia) que nos permita de una forma más intuitiva diseñar el escenario.
3. La instancia del modelo obtenida con el editor la pasaremos al entorno OAW para la validación y generación de código.

En el capítulo siguiente pasaremos a ver el desarrollo de nuestra propuesta.

5. Desarrollo de la propuesta

Una vez hemos visto la tecnología y las herramientas que vamos a utilizar, pasaremos a ver en este capítulo el desarrollo de nuestra propuesta.

Como el presente trabajo se enmarca dentro del desarrollo de prototipos para sistemas que soporten flujos de trabajo móviles, empezaremos viendo el método de desarrollo seguido en el trabajo de [Giner, 2010], y su propuesta general de prototipado rápido, viendo cómo son los prototipos obtenidos y los pasos generales para obtenerlos. De esta forma nos será más fácil comprender cómo deben funcionar los prototipos que vamos a obtener para este tipo de sistemas.

En la sección 5.2 veremos el método de diseño que vamos a seguir en el desarrollo de nuestra propuesta para obtener estos prototipos. En la sección 5.2.1 haremos una descripción de los conceptos a modelar, que serán formalizados en la sección 5.2.2 en un metamodelo. En la sección 5.2.3 veremos el editor gráfico creado para definir una instancia o escenario. Finalmente en la sección 5.2.4 se presentan las reglas de validación del escenario y en la 5.2.5 la plantilla utilizada para la generación de código.

5.1 Sistemas que soporten flujos de trabajo móviles

Para comprender mejor cómo son los prototipos a obtener, empezaremos viendo el método de desarrollo seguido en la tesis de Pau Giner, *Automating the development of Physical Mobile Workflows* [Giner, 2010]. Posteriormente veremos cómo son los prototipos de prueba que obtiene para estos sistemas, prototipos cuyo aspecto y funcionamiento son similares a los obtenidos en el presente trabajo. Continuaremos viendo su propuesta general de prototipado rápido para estos sistemas. Terminaremos con las conclusiones, donde veremos cómo encaja nuestra propuesta dentro de su trabajo.

5.1.1 Método de desarrollo de sistemas que soporten flujos de trabajo móviles

[Giner, 2010] presenta un método de desarrollo de sistemas que soporten flujos de trabajo móviles partiendo de la ingeniería dirigida por modelos, buscando como punto de partida capturar en un modelo los conceptos que caracterizan el enlace físico-virtual. De esta forma se logra partir desde un alto grado de abstracción hasta llegar a la implementación final en una tecnología concreta.

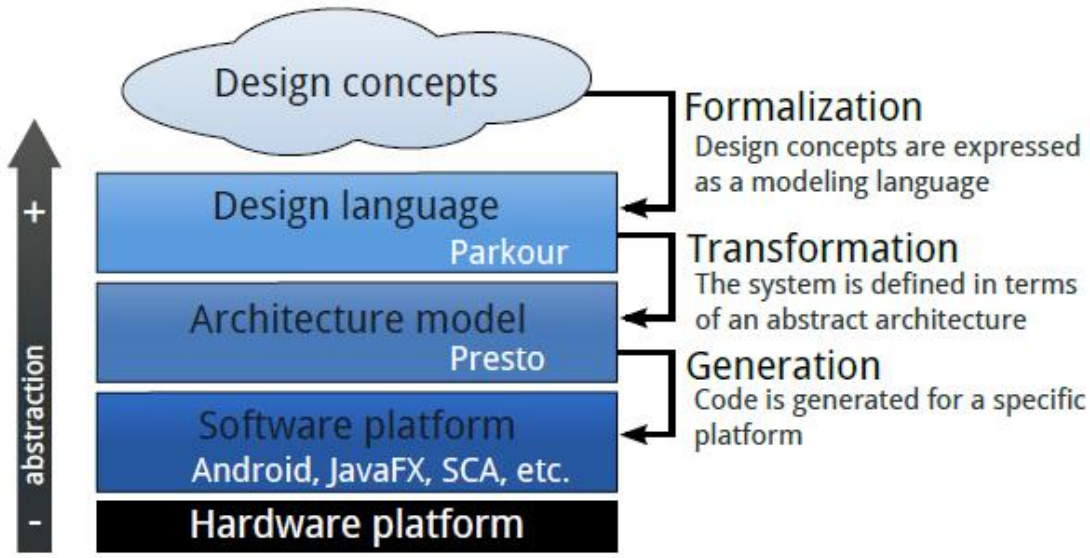


Figura 5.1. Estrategia de desarrollo para sistemas que soporten flujos de trabajo móviles [Giner, 2010]

Como se puede ver en la figura 5.1 se parte del metamodelo Parkour, que captura los conceptos de diseño de los flujos de trabajo móviles, pasando mediante una transformación de modelos, a un modelo Presto. Presto es una arquitectura definida para

soportar aplicaciones en el dominio de los flujos de trabajo móviles de una forma independiente de la tecnología. Finalmente se usan técnicas de generación de código para pasar de esta arquitectura genérica en componentes de una plataforma concreta de desarrollo.

Para modelar flujos de trabajo móviles, se diseñó el metamodelo Parkour. Este metamodelo extiende el metamodelo BPMN, extendiendo *Data Object*, y *Activity*. En Parkour se capturan todos los requisitos de los flujos de trabajo móviles de forma independiente de la tecnología usada. En la figura 5.2 se puede ver un extracto del metamodelo Parkour.

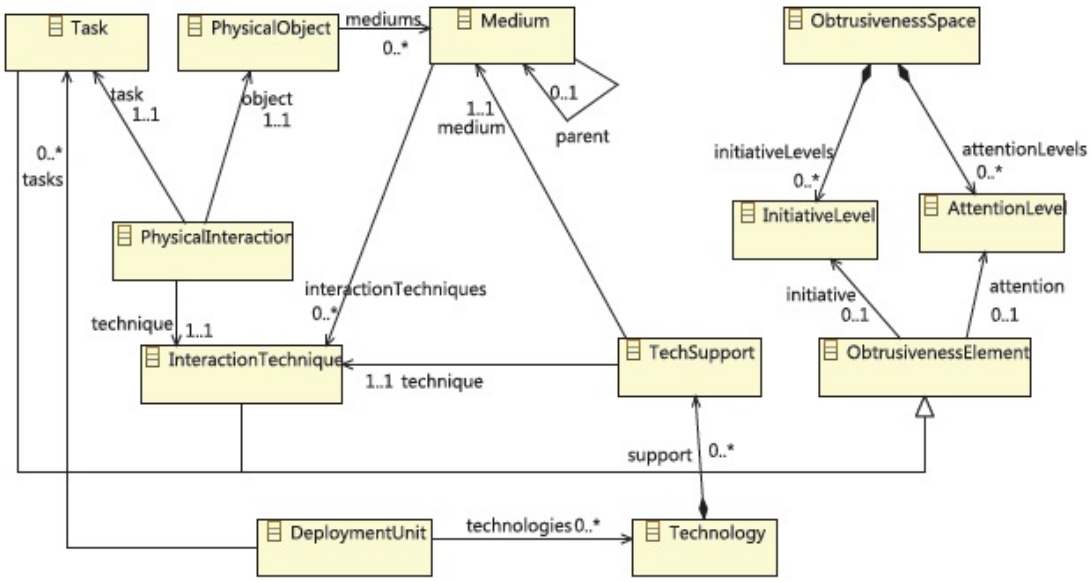


Figura 5.2. Extracto del metamodelo Parkour [Giner, 2010]

Se usa por tanto BPMN (ver sección 2.3.1 para más detalles) para capturar las actividades o tareas del usuario involucradas en el proceso, sus dependencias temporales, los eventos que pueden ocurrir y los distintos puntos de decisión. Parkour se usa para caracterizar el enlace físico-virtual del proceso: objetos físicos que participan en el proceso, medios de interacción, tecnología, nivel de obstrusividad para cada tarea.

Una vez obtenido el modelo Parkour, mediante técnicas de transformación de modelos, es transformado en un modelo Presto. Presto sigue la arquitectura de Völter [Völter, 2005]. Dentro de esta arquitectura, nuestra propuesta desarrolla la fase *Vertical Prototype*.

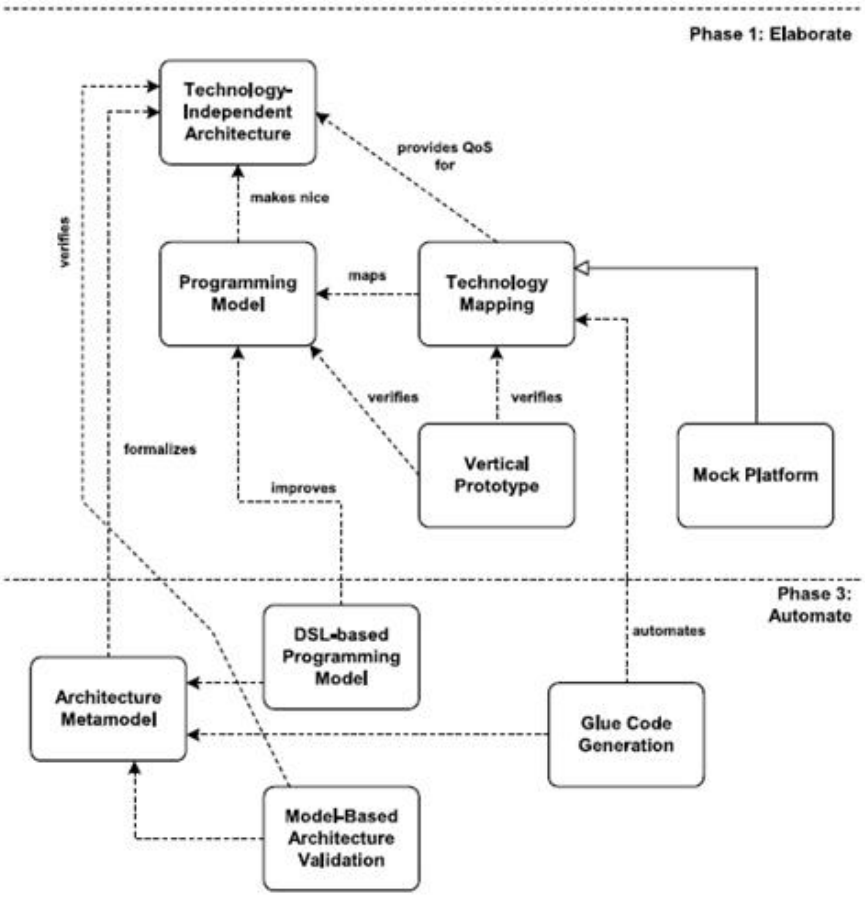


Figura 5.3. Arquitectuta de Völter [Völter, 2005]

Presto soporta flujos de trabajo móviles de una forma modular y ampliable, en la figura 5.4 se pueden ver los componentes de su arquitectura.

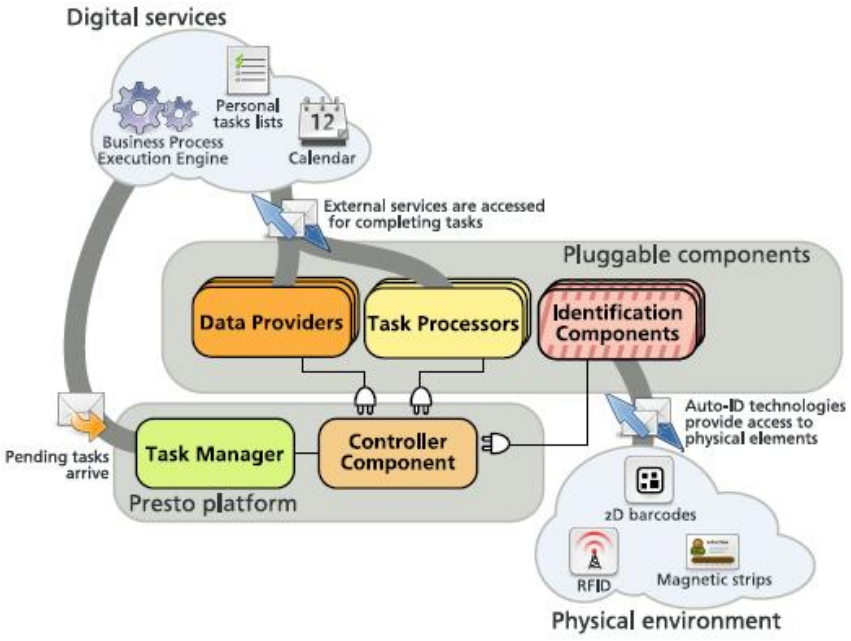


Figura 5.4. Presto - Componentes

La arquitectura de Presto está formada por un *Task Manager*, un *Controller Component* y varios *Task Processors*, *Identification Components* y *Data Providers*. El desarrollo de una solución específica supone crear ciertos plug-ins e incorporarlos a la plataforma para extender su funcionalidad, con soporte para nuevas tareas, tecnologías de identificación y fuentes de datos. Veamos una descripción de los componentes, y qué información suministran al usuario:

1. El **Task Manager** actúa como un buffer almacenando trabajo pendiente de ser procesado. En un ambiente móvil nos interesa que el usuario acceda a sus tareas pendientes desde un único punto, tareas que pueden ser de distintos procesos. Task Manager hace uso de la metáfora lista dinámica to-do, la cual representa las tareas pendientes del usuario y las posibles acciones a realizar. Esta lista será la información a mostrar en la pantalla inicial del prototipo del usuario.
2. Los **Task Processors** soportan la extensión de la plataforma en términos de funcionalidad. Son las piezas básicas para construir una interfaz basada en tareas que será presentada al usuario según el elemento físico detectado y el estado del flujo de trabajo.
3. Los **Identification Components** proporcionan mecanismos para acceder al entorno físico y transferir identificadores entre el espacio físico y el virtual. Este comportamiento será simulado en los prototipos.
4. Los **Data Provider** transforman los identificadores en información que es relevante para el usuario. Este comportamiento también será simulado en los prototipos.
5. El **Controller Component** coordina todos los componentes. El modo en que la información es transferida depende del modo de operación considerado. Presto soporta dos modos de operación general: modo dirigido por el objeto (object-driven) y modo dirigido por la tarea (task-driven)
    - Object-driven: primero se identifica el contexto físico y entonces se muestran las tareas al usuario.
    - Task-driven: es el usuario el que selecciona la tarea, y después se accede al contexto físico.

Nuestros prototipos seguirán un modo de funcionamiento object-driven. En la figura 5.5 se puede ver en un diagrama BPMN el funcionamiento de ambos modos.

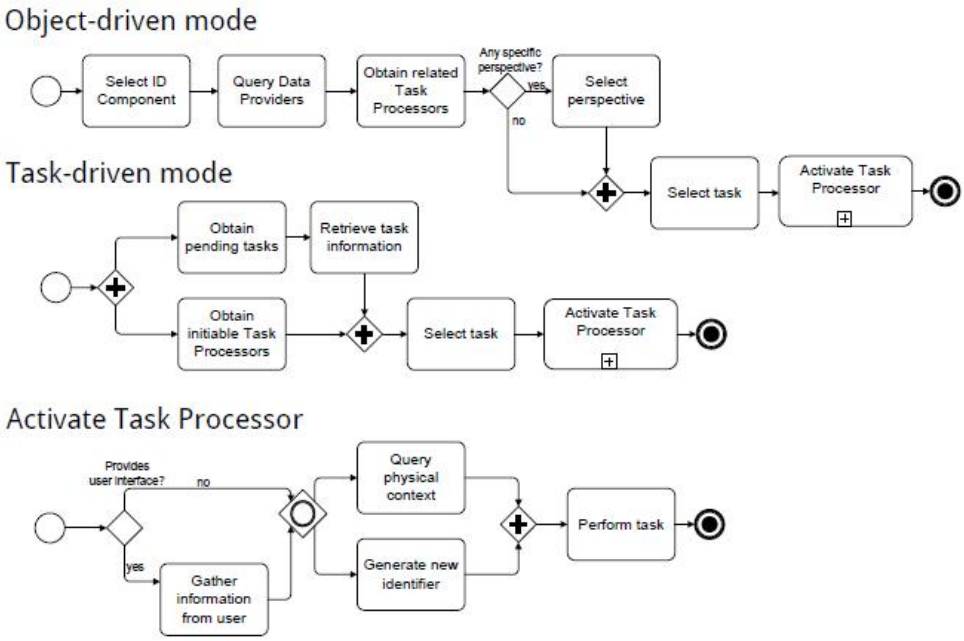


Figura 5.5. Presto - Estrategias de ejecución

5.1.2 Prototipado rápido para flujos de trabajo móviles

En esta sección veremos en primer lugar cómo son los prototipos diseñados en el trabajo de [Giner, 2010] en cuanto a su formato y funcionalidad, prototipos que son iguales a los que vamos a generar en nuestra propuesta: *pantallas mockup*, prototipos rápidos y fáciles de implementar. Continuaremos viendo los pasos generales propuestos para obtener prototipos en el ámbito de los flujos de trabajo móviles.

5.1.2.1 Pantallas mockup

Las pantallas mock-ups son muy usadas para diseñar la interacción con el usuario, se caracterizan principalmente por ser diseños de pantalla realizados habitualmente en html, de forma que son fáciles de implementar y presentar al usuario, en la figura 5.6 se puede apreciar el grado de fidelidad y de desarrollo en tiempo de los distintos tipos de prototipos que se usan a lo largo de un desarrollo [27].

[27] http://www.uxbooth.com/blog/concerning-fidelity-and-design

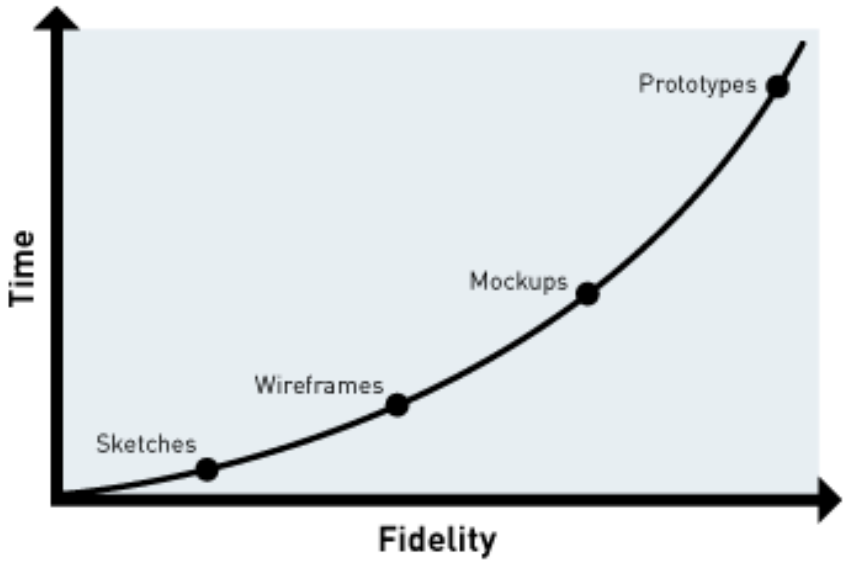


Figura 5.6. Fidelidad de los Mockups

Para el desarrollo de mockups en HTML son necesarios conseguir dos objetivos contradictorios:

1. Queremos unos mockups realistas.
2. Queremos unos mockups muy fáciles de desarrollar.

Para cumplir con estos objetivos hemos considerado usar un conjunto de utilidades HTML para el desarrollo de aplicaciones web para móviles. En concreto hemos usado el marco de trabajo iUI[28].

La Interfaz de usuario Framework (iUI) es libre para descargar y usar, ya que es de código abierto, soporta javascript, CSS e imágenes y se usa para desarrollar aplicaciones móviles para iphone o dispositivos compatibles. iUI tiene las siguientes características:

1. Crear menús navegables e interfaces al estilo iPhone mediante HTML estándar.
2. No es necesario conocimientos de Javascript.
3. Capacidad de manejar cambios de orientación del teléfono.
4. Crear aplicaciones web con un aspecto iPhone.
5. iUI permite el desarrollo de sitios web que aparecen y funcionan como aplicaciones para el iPhone nativas[29] .

[28] http://code.google.com/p/iui/
[29] http://code.google.com/p/iui/wiki/Introduction

En el listado 5.1 podemos ver un extracto de un prototipo HTML creado mediante iUI.

Listado 5.1

```
<ul id="home" title="Tasks" selected="true" hideBackButton="true" >
    <li id="pending_group" class="group">Pending tasks</li>
    <li id="ret-link"><a href="#pending"></a></li>
    <div id="p-task"></div>

  <li id="action-Liberar-Masaje"><a href="#decode" >Liberar SERVICIOS HOTEL Masaje</a></li>

  <li id="action-Habilitar/Arreglar-Clima"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Clima</a></li>
  <li id="action-Habilitar/Arreglar-Limpieza"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Limpieza</a></li>
  <li id="action-Habilitar/Arreglar-Alimentacion"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Alimentacion</a></li>
    <li id="actions_group" class="group">Actions</li>

  <li><div class="annotation" >SERVICIOS HOTEL</div></li>
  <li><a href="#decode" onclick="setAction('Asignar');">Asignar</a></li>
  <li><a href="#decode" onclick="setAction('Esperar');">Esperar</a></li>

  <li><div class="annotation" >FUNCIONES HABITACION</div></li>
  <li><a href="#decode" onclick="setAction('Activar');">Activar</a></li>
  <li><a href="#decode" onclick="setAction('Desactivar');">Desactivar</a></li>
</ul>
<ul id="Spa" title="Details" >
    <li><h2>Spa</h2> <div class="annotation" id="Spa-status">Disponible</div></li>
   <li id="actions-Spa" class="group">Actions</li>
   <li id="action-Asignar-Spa"><a href="#Asignar-Spa">Asignar</a></li>
</ul>
```

El código de arriba contiene dos pantallas mockups. iUI muestra solo un fragmento HTML a la vez de los muchos que puede tener definidos en un fichero HTML. Según la navegación del usuario, iUI proporciona las adecuadas transiciones entre pantallas. Dos fragmentos están definidos en el listado. El primero muestra las tareas pendientes y las

acciones, el segundo muestra las tareas asociadas al elemento físico Spa, definido en nuestro caso de estudio Hotel Inteligente. La figura 5.7 muestra el resultado de estos mockups HTML.

Figura 5.7. Prototipos HTML generados

5.1.2.2 Pasos a seguir

El enfoque global para el prototipado rápido de flujos de trabajo móviles se puede ver en la figura 5.8 [Giner, 2010]. El objetivo es sumergir al usuario en un entorno que le haga sentir que está usando el sistema final.

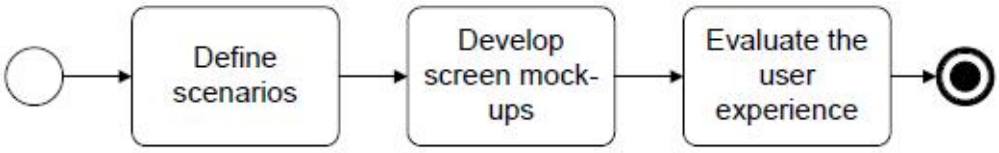


Figura 5.8. Prototipado rápido para flujos de trabajo móviles

La interacción con el mundo real es simulada usando técnicas Mago de Oz. Un operador proporciona el actual contexto físico usando otro dispositivo móvil (figura 5.9), de forma que el usuario se sumerge en un entorno similar a un sistema con capacidades Auto-ID, pero mucho más fácil de producir.

Figura 5.9. Prototipo en HTML. Tras lanzar el operador un cambio en el contexto, el cliente del hotel recibe el mensaje de que el spa está libre

Los pasos seguidos de una forma general son los siguientes:

1- **Definir un escenario acorde a la definición del proceso**. Una instancia específica del flujo de trabajo es definida para ilustrar un caso de uso relevante.
2- **Detectar los objetos físicos que participan en el escenario**. Una interfaz debe ser suministrada al operador que le muestre los posibles eventos de detección.
3- **Definir las pantallas mockup para cada paso del proceso**. La pantalla a mostrar depende del contexto físico y de la acción a realizar.
4- **Proporcionar opciones complementarias y mensajes de error**. Para proporcionar un mayor realismo, enlaces a funciones adicionales pueden ser añadidas.

5.1.3 Conclusiones

En la sección 5.1.1 hemos visto un resumen de la propuesta de [Giner, 2010] para desarrollar sistemas que soporten flujos de trabajo móviles, y en la sección 5.1.2 cómo son los prototipos obtenidos y los pasos generales para obtenerlos. Veamos qué conceptos vistos en estas secciones desarrolla o complementa nuestra propuesta:

1- Nuestra propuesta consiste en **desarrollar la fase *Vertical Prototype*** dentro de la arquitectura de Völter del desarrollo de Presto.
2- En nuestro desarrollo seguiremos una **aproximación MDE**, **y capturaremos en un metamodelo aquellos conceptos necesarios para obtener los prototipos**. Este metamodelo nos permitirá:
    a. Definir un escenario acorde a la definición del proceso.
    b. Integrar en el escenario los objetos físicos que participan en el proceso.
3- Los **prototipos** que obtendremos tendrán las siguientes características:
    a. Los prototipos serán pantallas mockup HTML acorde a lo visto en la sección 5.1.2.1.
    b. Tras ver la arquitectura Presto, nuestros prototipos deben mostrar la siguiente información al usuario:
        i. Una pantalla inicial que muestre las tareas pendientes y la lista de acciones de todos los procesos que el usuario puede realizar.
        ii. Una pantalla para la detección de cada elemento físico, que nos muestre la lista de tareas disponibles para ese objeto en un estado concreto (modo de ejecución object-driven).
        iii. Una pantalla para cada tarea seleccionada de la lista de acciones posibles del apartado anterior (Actívate Task Processor en figura 5.5).
        iv. Una pantalla por cada tarea pendiente finalizada.
    c. Deberemos simular, por tanto:
        i. Detección de un elemento físico.
        ii. Resolución de una tarea pendiente.

5.2 Método de diseño para obtener prototipos para flujos de trabajo móviles

Una vez hemos visto en la sección anterior cómo son y en qué consisten los prototipos que vamos a generar, pasaremos a ver el método de diseño seguido. Nuestro método de diseño sigue una aproximación MDE, lo que implica que los conceptos capturados en un modelo acorde a un metamodelo guian todo el proceso de desarrollo. Hemos seguido los siguientes pasos:

1- Definir los conceptos a modelar en un metamodelo.
2- A partir de este metamodelo, obtendremos un modelo, que representará un escenario que soporta flujos de trabajo móviles.
    a. Para obtener el modelo crearemos un editor gráfico.
3- El modelo obtenido lo validaremos con reglas de validación Check.
4- Al modelo validado le aplicaremos una plantilla XPand que obtendrá los prototipos en HTML descritos en la sección anterior. Estos prototipos mostraran la información al usuario acorde a Presto. También obtendremos un prototipo para simular la detección de elementos físicos y resolución de tareas pendientes.

Los pasos 3 y 4 se realizarán de forma automática. En la figura siguiente se puede ver cómo se sitúan las herramientas utilizadas con los pasos de nuestro método.

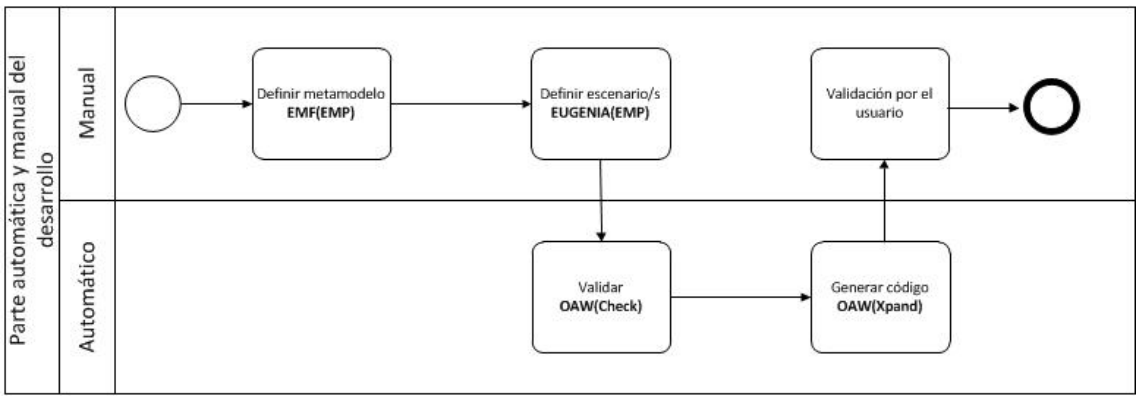


Figura 5.10. Parte automática y manual del desarrollo

5.2.1 Conceptos a modelar

El metamodelo Parkour es la base para desarrollar sistemas que soporten flujos de trabajo móviles, pero en nuestro caso, queremos obtener prototipos, y aunque podríamos

usar este metamodelo para modelar los escenarios, no vamos a necesitar muchas de sus clases, por lo que se decidió hacer uno nuevo, mucho más simple, enfocado a la obtención de prototipos. En el caso de los prototipos lo que se describe es un escenario concreto, no el funcionamiento detallado de la aplicación y todas sus posibles variantes.

Nos centraremos en la información que queremos mostrar al usuario.

Vamos a considerar tres grupos de conceptos:

1. Por un lado el **entorno físico** que participa en el flujo de trabajo.
2. Las **tareas activadas** para cada elemento.
3. **Información**.

Entorno físico

En este grupo queremos modelar los objetos físicos que intervienen en el proceso. Para aplicar la generalidad de nuestra propuesta vamos a considerar a los objetos físicos como parte de un grupo de objetos del mismo tipo, por ejemplo el tipo de objeto Book, tendrá asociadas una serie de propiedades que compartirán todos los elementos de su grupo. Por ejemplo *prestar libro*, *devolver libro*, etc., serán tareas aplicables a los elementos *El Quijote*, *Matrix* del grupo *Books*.

Figura 5.11. Conceptos a modelar - Entorno físico.

Según se ve en la figura 5.11 consideraremos que cada grupo de elementos puede tener asociadas una lista de tareas, de posibles tareas pendientes y una serie de estados en los que podrá estar un miembro del grupo. Las tareas que un tipo de elemento podrá realizar dependerá del estado en que se encuentre, por ejemplo si un libro está prestado, no puede tener asociada la tarea *Prestar*.

Con la captura de estos conceptos podemos obtener:

1. Para el usuario:
    - Pantalla inicial con la lista de tareas pendientes y un listado de todas las tareas que se pueden realizar con un determinado tipo de elemento.
    - Pantalla con la lista de tareas asociada a cada elemento físico detectado.
2. Para el operador:
    - Posibilidad de lanzar la simulación de la detección de cada elemento físico.

Tareas a activar.

Conforme vayamos añadiendo elementos y tareas al entorno físico obtendremos una gran cantidad de tareas a modelar. Por ejemplo, en un entorno con cinco elementos de un mismo tipo, y con tres posibles tareas, tendríamos quince pantallas a obtener para cada una de las tareas asociadas a cada elemento. Una de las ventajas de nuestra aproximación, es que no es necesario definir todas estas combinaciones, ya que describiendo el escenario, estas se generan automáticamente.

Como esto puede sobrecargar excesivamente el prototipo resultante, se decidió que en el diseño del escenario se pudiese decidir qué tareas asociadas a un objeto concreto serían modeladas para obtener su correspondiente pantalla en el prototipo

Esto nos podría servir para dejar solo activas aquellas tareas de aquellos elementos que queremos que el usuario evalúe, y que no se pierda con el resto de tareas. De esta forma podemos centrarnos en evaluar una parte del proceso u otra.

A este concepto le hemos llamado *WorkItem*. Como se puede ver en la figura 5.12 un *WorkItem* tendrá asociado un objeto físico, una tarea, y un posible cambio de estado del objeto.

Con la captura de estos conceptos podremos obtener:

1. Para el usuario:
    - Pantalla asociada a cada tarea de un elemento físico para el que hayamos definido un *WorkItem*.
    - Si el *WorkItem* está formado por una tarea pendiente, pantalla informativa de resolución de una tarea pendiente
2. Para el operador:

- Si el *WorkItem* está formado por una tarea pendiente, se creará en la consola del operador una entrada que le permita simular la resolución de la misma.

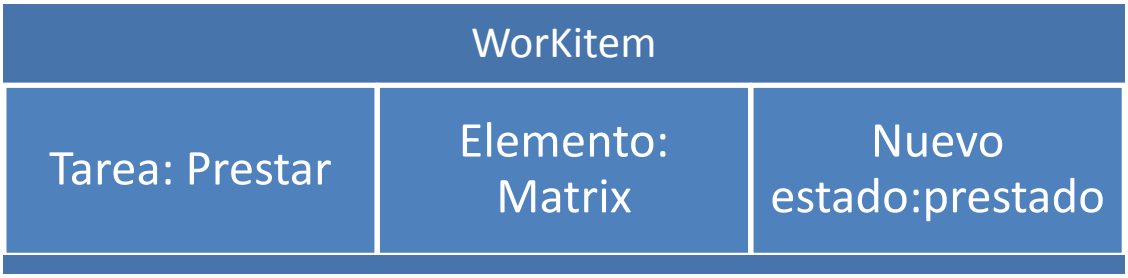


Figura 5.12. Conceptos a modelar - Tareas (*WorkItem*)

No hemos usado BPMN para modelar las tareas, aunque sí podemos usarlo como referencia, pues queremos tener todo integrado en un mismo modelo, tanto el entorno físico, como las tareas a modelar, y poder así decidir qué tareas modelamos.

Información.

Se trata de la información a mostrar al usuario. Esta información estará asociada a un *WorkItem*, o bien a una tarea.

Se ha considerado la información como texto, aunque podría cambiarse para mostrar imágenes u otro tipo de información.

En la figura 5.13 se puede ver el modo de operación seguido a la hora de gestionar la información que será mostrada al usuario.

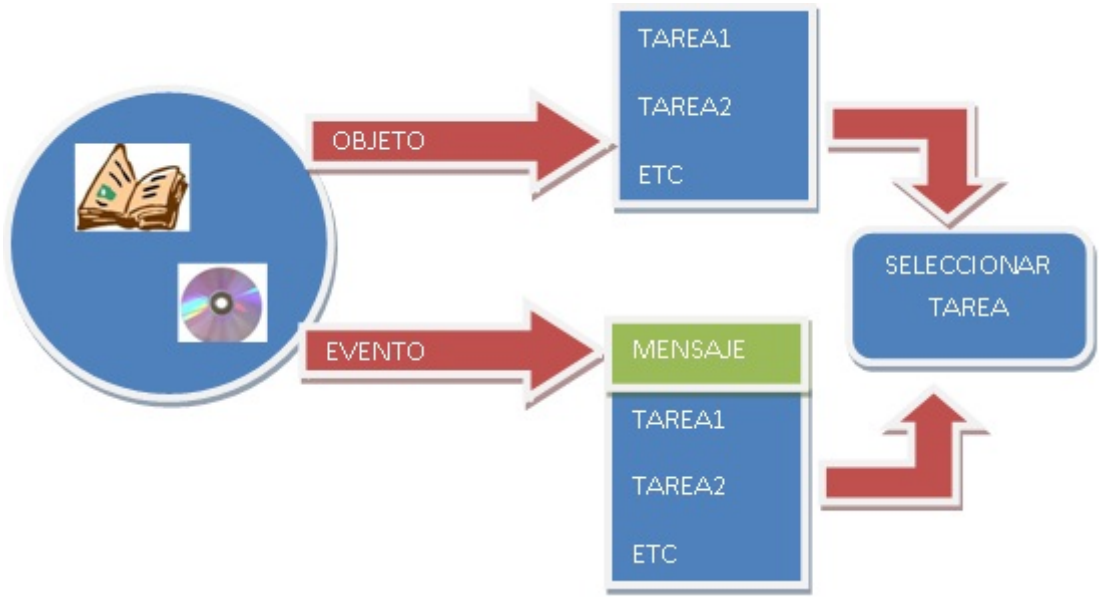


Figura 5.13. Conceptos a modelar - Modo de operación

Una vez se le ofrece al usuario la lista de tareas asociada a ese elemento físico o a ese evento producido por un elemento físico, este puede seleccionar una tarea de la lista, lo que podrá implicar un cambio de estado del elemento y la posible creación de una tarea pendiente. Los eventos siempre estarán asociados a tareas pendientes e implicarán un cambio de estado del elemento y la eliminación de una tarea pendiente. Para clarificar estos conceptos veamos las siguientes figuras.

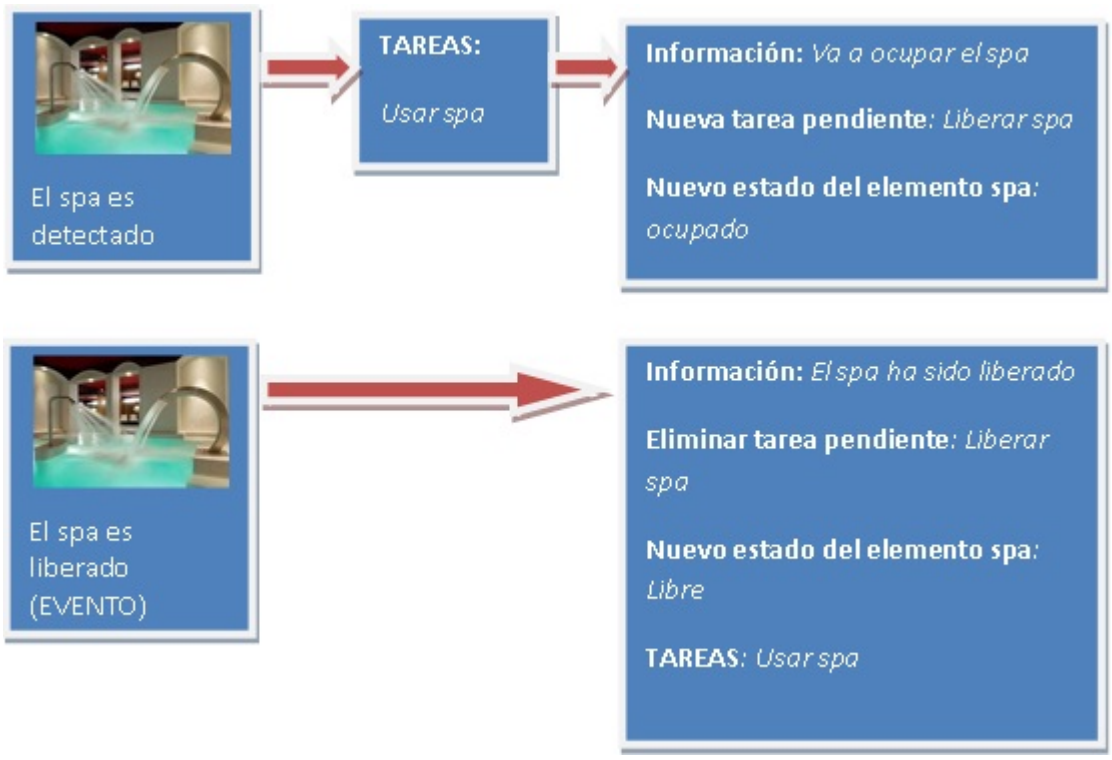


Figura 5.14. Conceptos a modelar - Ejemplos

Tanto la detección del spa, como la liberación del spa serán simulados, pues intervienen tecnologías Auto-ID para detectar elementos físicos que no van a ser implementadas. Hemos visto que un elemento físico tiene asociadas una lista de tareas en un determinado estado, que seleccionar una tarea puede producir un cambio de estado y agregar una tarea pendiente, y que un evento produce un cambio de estado y la eliminación de una tarea pendiente.

Haciendo un pequeño resumen de estos conceptos, tendremos por un lado los elementos del mundo físico, los cuales tienen una lista de tareas asociadas a cada estado posible, incluyendo tareas pendientes, y por otro lado tendremos los workitems o tareas a modelar, que tendrán asociados un elemento físico, una tarea (pendiente o no) y un posible cambio de estado.

Veamos en la siguiente sección cómo modelar estos conceptos.

5.2.2 Definición del metamodelo

Como ya se presentó en la sección 4.1, MDE propone el uso de metamodelos para formalizar conceptos y sus relaciones. Un metamodelo define las construcciones que pueden ser usadas para definir un sistema, y cómo se pueden combinar de forma no ambigua a nivel sintáctico, lo que hace procesable estas definiciones.

Nuestro metamodelo captura los conceptos definidos en la sección anterior. En primer lugar definimos *ScenarioModel*, que representa el modelo o escenario en sí, que estará formado por los elementos físicos y los workitems o tareas a modelar.

Para definir los elementos físicos hemos definido la clase *TypeElement*, que sería la clase genérica de un tipo de elemento (por ejemplo Servicios Hotel), la cual estaría formada por tareas, estados y los elementos físicos (por ejemplo spa, gimnasio, etc). Cada tarea tiene asociada información a mostrar al usuario.

Para definir los tareas a modelar hemos definido la clase *WorkItem*, la cual está formada por un elemento, una tarea y un posible cambio de estado. Cada *workitem* tiene una información asociada a mostrar al usuario.

Tal y como se vió en las herramientas, el metamodelo ha sido formalizado usando Ecore dentro del proyecto EMF de la plataforma Eclipse. Veamos con detalle cada una de las clases definidas y sus relaciones. En el apéndice A se puede ver el metamodelo completo.

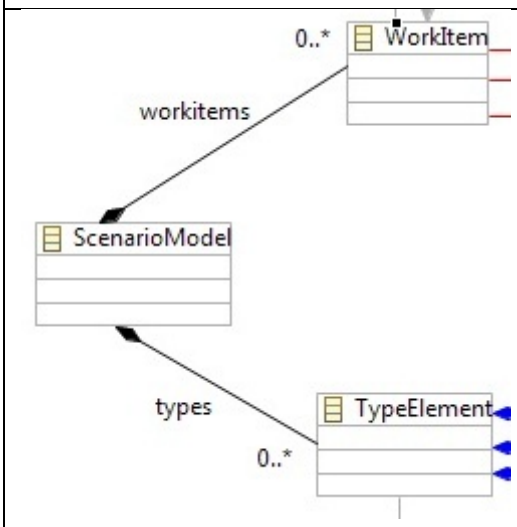
CLASE: ScenarioModel

Significado: Es la raíz a partir de la cual creamos el escenario a representar. Está compuesta de WorkItems (tareas a modelar) y TypeElements (tipos de objetos).
Clases extendidas: NamedElement

Tabla 5.1. Metamodelo - ScenarioModel

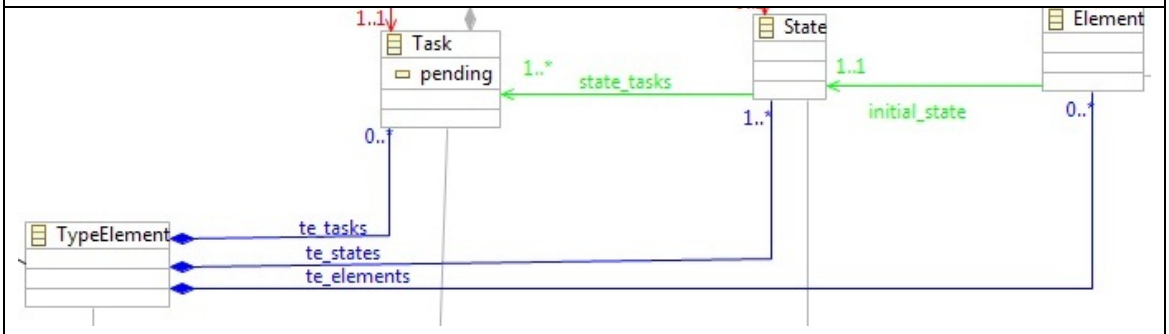
CLASE: TypeElement

Significado: Esta clase viene a representar los tipos posibles de elementos que puede haber en el mundo físico y que van a tener una identidad digital. Está formada por Tasks (tareas), States (estados) y Elements (objetos físicos del mundo real).
Clases extendidas: NamedElement
Ejemplo: DVD

Tabla 5.2. Metamodelo - TypeElement

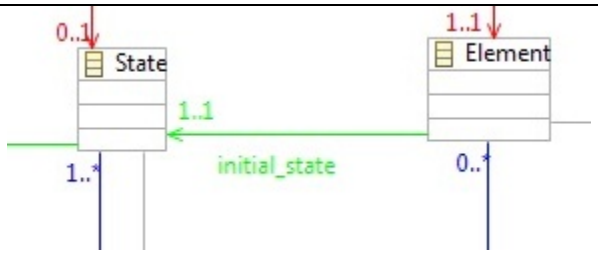
Clase: Element

Significado: Representa los distintos elementos u objetos físicos de un determinado TypeElement. Debe tener asociado un State. Todo objeto físico se encuentra en un estado concreto. En los prototipos simularemos la detección de cada Element.
Clases extendidas: NamedElement
Ejemplo: “Matrix”->State(“Prestado”) “El señor de los anillos”->State(“Libre”)

Tabla 5.3. Metamodelo - Element

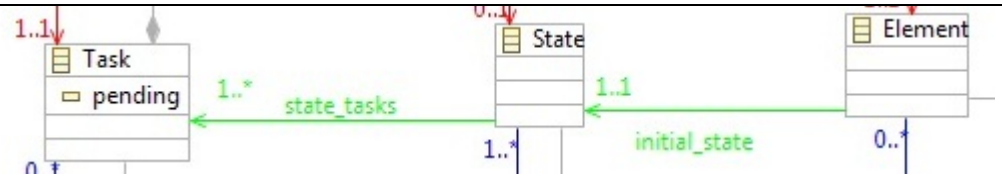
Clase: State

Significado: Representa los estados posibles en los que se puede encontrar un determinado Element (objeto físico), así mismo un estado tiene asociado una serie de Tasks, una lista de tareas que se pueden realizar con un objeto en ese estado.
Clases extendidas: NamedElement
Ejemplo: “Prestado”->Tasks(“Devolver”) “Reservado”->Tasks(“Liberar”) “Libre”->Tasks(“Prestar”,”Reservar”)

Tabla 5.4. Metamodelo - State

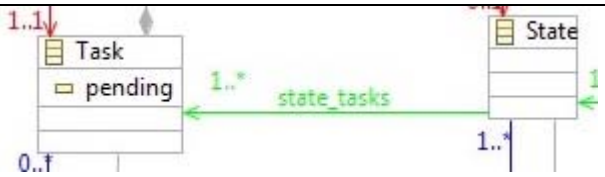
Clase: Task

Significado: Representa las posibles tareas que se pueden realizar con un Element (objeto físico) en un determinado State (estado). Tiene el atributo booleano “pending”, que cuando es true indica que se trata de una tarea pendiente, en este caso es necesario un evento para que se realice la tarea. El usuario no tendrá disponible esta tarea, pero sí el operador.
Clases extendidas: NamedElement
Ejemplo: “Devolver”, pending=true (requiere un evento, el usuario debe devolver el DVD) “Prestar”, pending=false “Liberar”,pending=true (requiere un evento, el usuario debe liberar el DVD)

Tabla 5.5. Metamodelo - Task

Esta parte del metamodelo representa el mundo físico, los objetos o elementos de la Internet de las Cosas que forman parte de nuestro escenario, con sus posibles estados y sus tareas. Continuemos viendo el metamodelo.

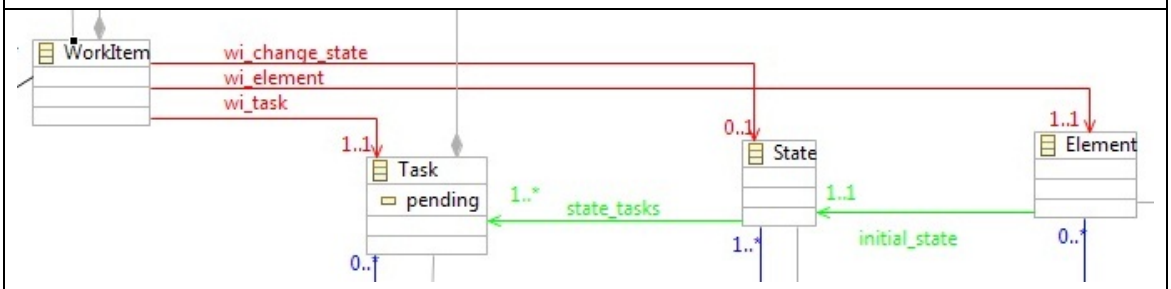
Clase: WorkItem

Significado: Esta clase agrupa a un elemento con una tarea, y a un posible cambio de estado. Nos va a indicar una tarea a modelar del escenario. Si la tarea tiene el atributo pending=true, esto nos indicará que es necesario simular en nuestro prototipo la ocurrencia de un evento, por ejemplo simular que el usuario devuelve un dvd prestado.
Clases extendidas: NamedElement
Ejemplo: Task(“Devolver”, pending=true), Element(”Matrix”), State(“Libre”) La tarea “Devolver” nos hará simular el evento en el que el usuario devuelve el dvd “Matrix”, pasando este a un nuevo State. Task(“Prestar”,false),Element(“El señor de los anillos”), State(“Prestado”)

Tabla 5.6. Metamodelo - WorkItem

Mediante *workitems* decidiremos qué tareas involucradas en el proceso de negocio modelamos, lo que implicará que estarán accesibles en los prototipos generados.

Veamos ahora como hemos representado la información a mostrar al usuario.

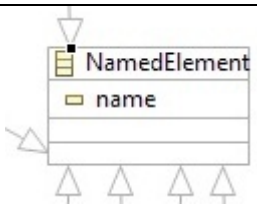
Clase: NamedElement

Significado: Esta clase es extendida por el resto, de forma que todas heredan el atributo “name”. Así ponemos nombre a cada clase.

Tabla 5.7. Metamodelo - NamedElement

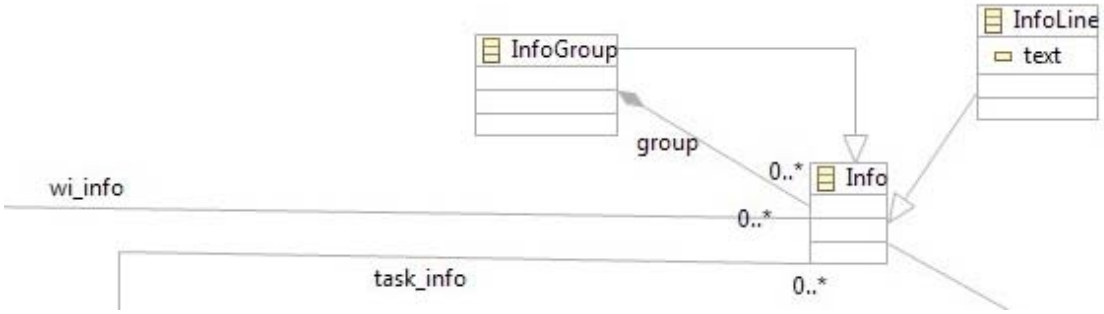
Clases: Info, InfoLine, InfoGroup

Significado: Representa líneas de texto o grupos de líneas de texto, que se pueden anidar. Las referencias Workitem.wi_info y Task.task_info son las que tienen esta información asociada.
Ejemplo: WorkItem(Task(“Devolver”, pending=true), Element(”Matrix”), State(“Libre”)).wi_info(“*El DVD ha sido devuelto*”). Task(“Prestar”, pending=false).task_info(“*Desea alquilar este dvd??”*)

Tabla 5.8. Metamodelo - Info, InfoLine, InfoGroup

En la siguiente figura podemos ver el metamodelo definido en la plataforma Eclipse.

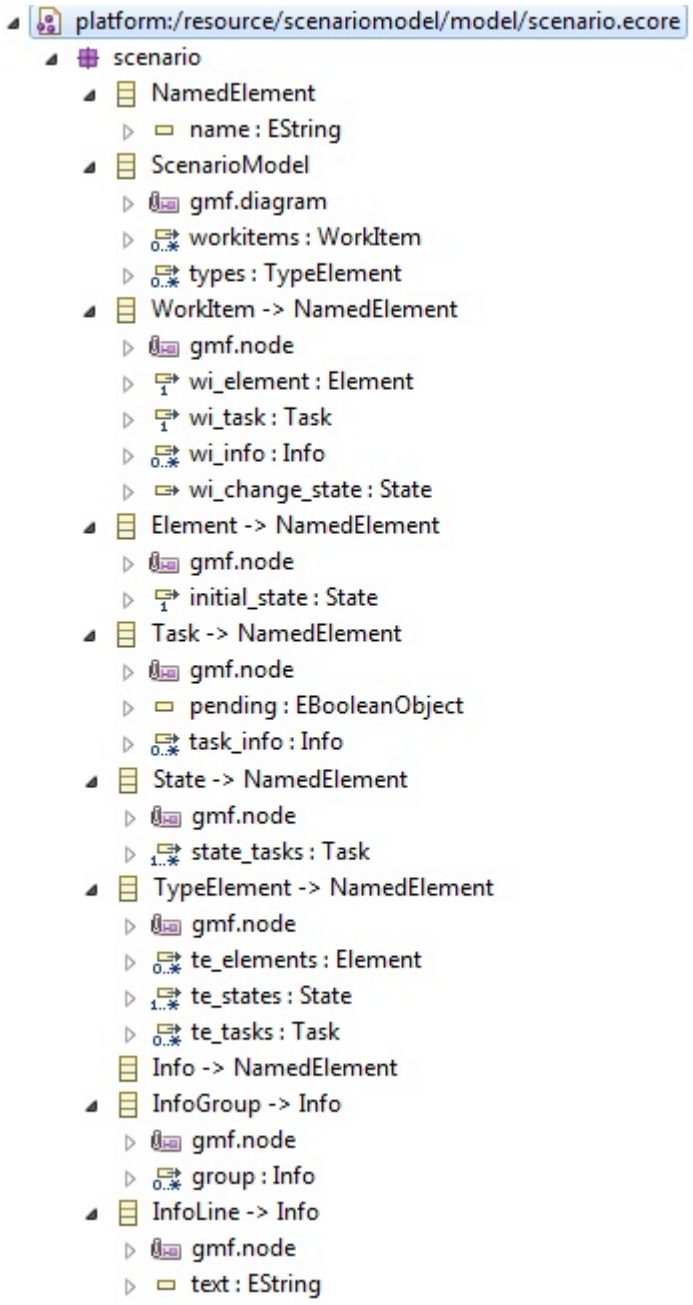


Figura 5.15. Metamodelo ecore en forma de árbol

Hemos visto el metamodelo, ahora vamos a ver cómo creamos un intuitivo editor gráfico que nos permita diseñar una instancia de este metamodelo, a partir de la cual generaremos el código o prototipo entendible por el usuario.

5.2.3 Definición del editor gráfico

En el punto anterior hemos visto las metaclases que forman nuestro metamodelo y que nos han hecho entender el significado de los modelos que obtendremos a partir de él. Lo que vamos a ver ahora es cómo utilizar este metamodelo para que mediante un sencillo e intuitivo editor gráfico nos sea más sencillo construir esa instancia.

Para crear el editor gráfico ampliaremos el metamodelo ecore creado en la sección anterior con anotaciones Eugenia (apéndice B). En la figura 5.6 podemos ver la paleta resultante. Seleccionaremos la clase que necesitemos de la paleta y la insertaremos en el escenario que estemos diseñando.

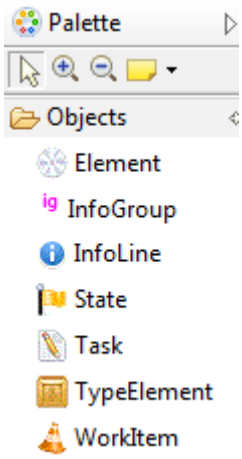
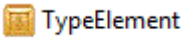


Figura 5.16. Editor gráfico - Paleta

Vamos a ir viendo en una serie de tablas el resultado producido por cada elemento de la paleta y las anotaciones Eugenia que han sido necesarias añadir al metamodelo ecore. En primer lugar pondremos la anotación Eugenia utilizada para producir la imagen de la paleta y en segundo la anotación Eugenia usada para producir el resultado al dibujar.

Imagen paleta: 

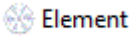
Anotación Eugenia:

```
@gmf.node(label="name",     color="255,255,196",     border.width="3",
tool.small.bundle="scenariomodel",
tool.small.path="model/icons/te.gif",
tool.large.bundle="scenariomodel",
tool.large.path="model/icons/tt.gif", figure="rectangle")
```

Resultado al dibujar: 

Anotación Eugenia: `@gmf.compartment(foo="bar")`

Tabla 5.9. Editor gráfico - TypeElement

Imagen paleta: 

Anotación Eugenia:

```
@gmf.node(label="name", tool.small.bundle="scenariomodel",
tool.small.path="model/icons/element.gif")
```

Resultado al dibujar: 

Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list")`

Tabla 5.10. Editor gráfico - Element

Imagen paleta: 

Anotación Eugenia:

```
@gmf.node(label="name",                              color="197,250,254",
tool.small.bundle="scenariomodel",
tool.small.path="model/icons/state.gif")
```

Resultado al dibujar: 

Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list`

Tabla 5.11. Editor gráfico - State

Imagen paleta: 

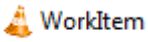
Anotación Eugenia:

```
@gmf.node(label="name", color="255,196,254",
tool.small.bundle="scenariomodel",
tool.small.path="model/icons/task.gif")
```

Resultado al dibujar: 

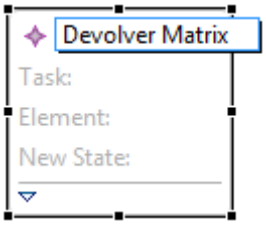
Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list")`

Tabla 5.12. Editor gráfico - Task

Imagen paleta: 

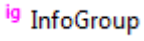
Anotación Eugenia:

```
@gmf.node(label="name", tool.small.bundle="scenariomodel",
tool.small.path="model/icons/workitem.gif")
```

Resultado al dibujar: 

Anotación Eugenia: `@gmf.compartment(foo="bar")`

Tabla 5.13. Editor gráfico - WorkItem

Imagen paleta: 

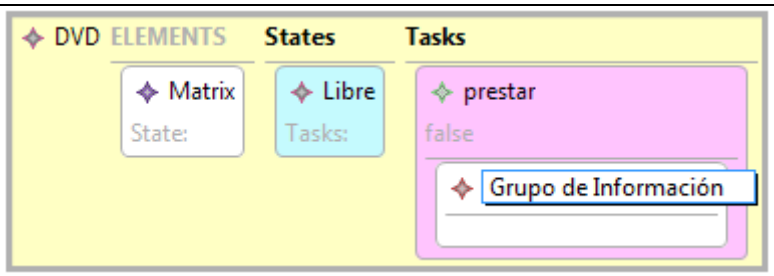
Anotación Eugenia:

```
@gmf.node(label="name", tool.small.bundle="scenariomodel",
tool.small.path="model/icons/infogroup.gif")
```

Resultado al dibujar: 

Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list")`

Resultado al dibujar:



Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list")`

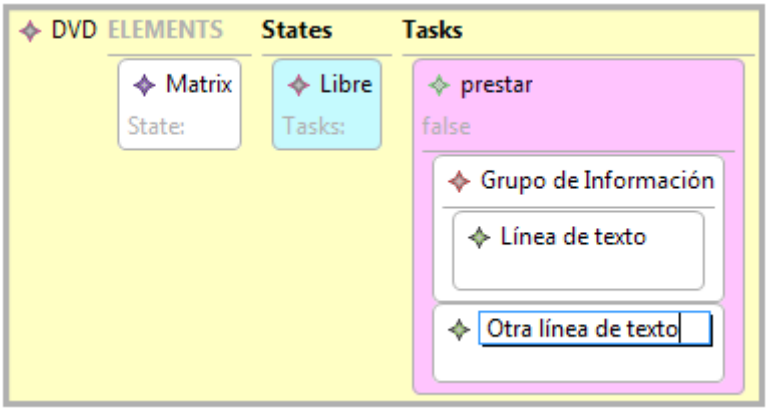
Tabla 5.14. Editor gráfico - InfoGroup

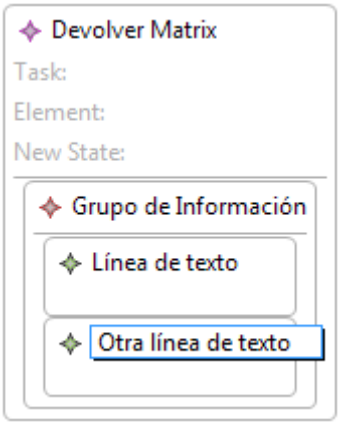
Imagen paleta: InfoLine

Anotación Eugenia:

```
@gmf.node(label="text", tool.small.bundle="scenariomodel",
tool.small.path="model/icons/infoline.gif")
```

Resultado al dibujar:





La línea de texto puede ir dentro de un grupo de información o sola

Anotación Eugenia: `@gmf.compartment(foo="bar", layout="list")`

Tabla 5.15. Editor gráfico - InfoLine

Hemos usado el fichero *Ecore2GMF.eol*[30], en el cual damos formato a las figuras que obtendremos en el editor gráfico. También hemos hecho cambios en las clases java generadas. Con estos cambios lo que conseguimos en el editor es sacar el nombre de la clase padre *TypeElement* a la hora de seleccionar un *Element*, *State* o *Task*, para no mezclar de distintos *TypeElements* (figura 5.17), y también añadir unas etiquetas a *WorkItem*, *Task*, *State* y *Element* con información de sus propiedades (figura 5.18).

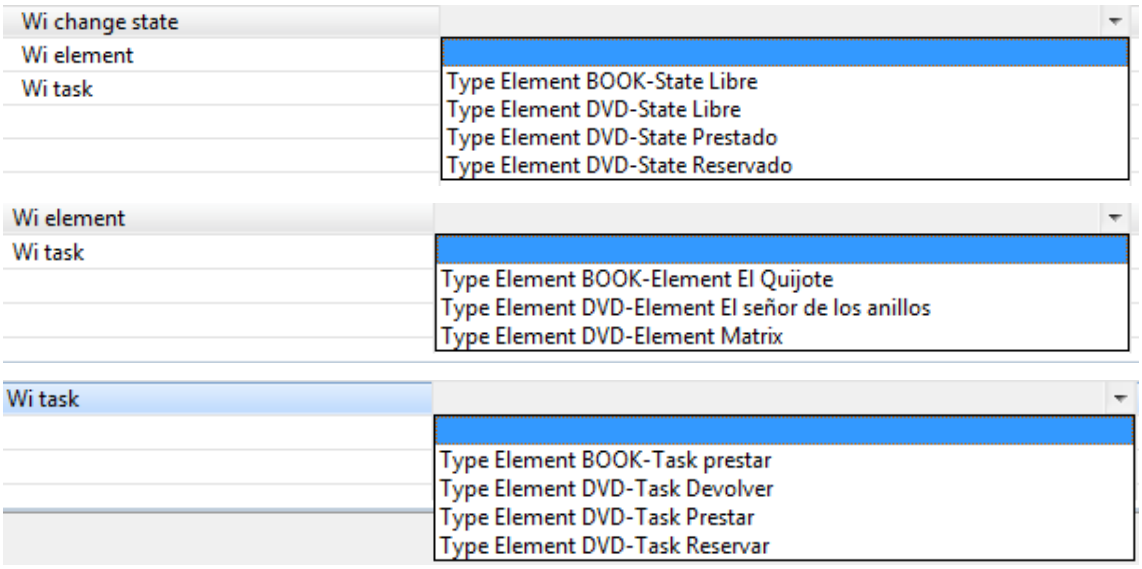


Figura 5.17. Editor gráfico - Mostramos el nombre de la clase padre TypeElement

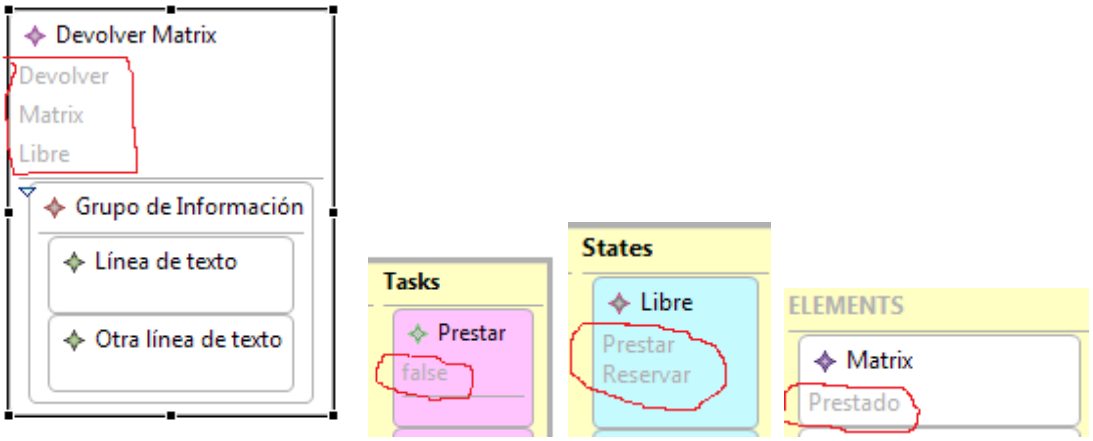


Figura 5.18. Editor gráfico - Etiquetas añadidas

30 http://sourceforge.net/apps/mediawiki/epsilonlabs/index.php?title=EuGENia_Customization

En la figura 5.19 podemos ver el editor gráfico terminado dentro del entorno Eclipse. Se han insertado un tipo de elemento y un workitem, ambos funcionan a modo de contenedores, que deberemos ir rellenando mediante los iconos de la paleta y las propiedades asociadas a cada clase.

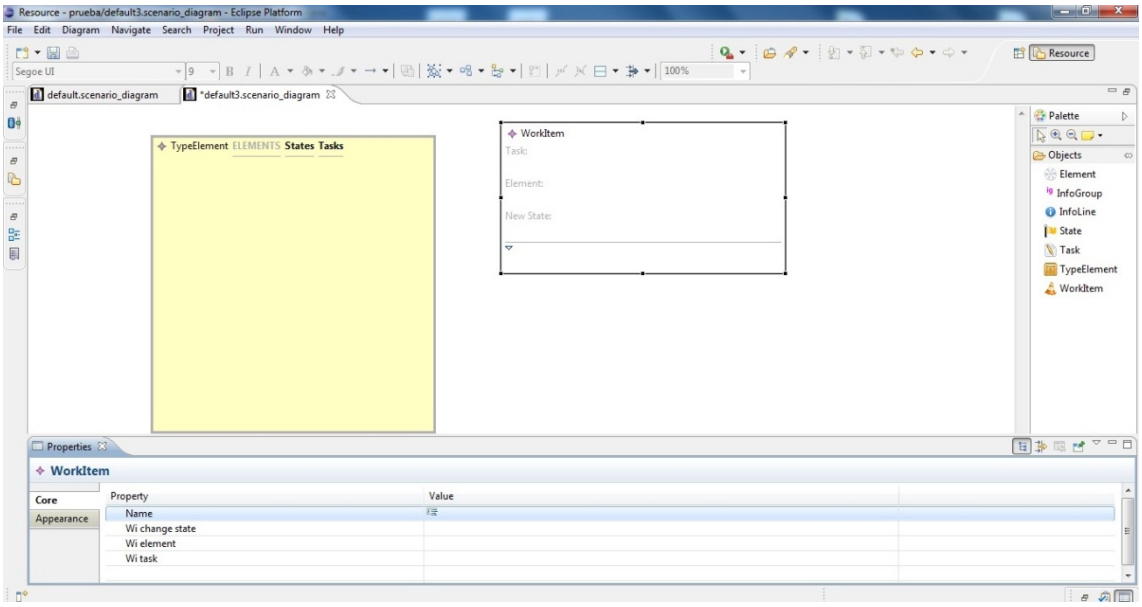


Figura 5.19. Editor gráfico terminado

Una vez obtenido el editor gráfico, lo usaremos para diseñar un escenario, obteniendo como resultado un fichero llamado *default.scenario*, que será el que contendrá la instancia del metamodelo. Este fichero será el que pasaremos a OAW para validarlo y si es correcto, generar los prototipos. Pasemos a verlo en la siguiente sección.

5.2.4 Validación del modelo

En las secciones anteriores hemos definido el metamodelo y el editor gráfico que nos permitirá modelar escenarios. El siguiente paso de nuestro desarrollo es validar el escenario dibujado. Para ello comprobaremos que cumple todas las reglas de validación que hemos definido. Haciendo un pequeño resumen, las reglas de validación serán las siguientes:

1. Deben existir algún *TypeElement* (tipo de objeto).
2. Los nombres de *TypeElement* deben ser únicos.
3. Los nombres de *TypeElement*, *State*, *Task* y *Element* no pueden ser nulos.
4. Dentro de un *TypeElement*:
    a. Los nombres de cada *State*, *Task* y *Element* deben ser únicos.
    b. Debe tener al menos un *State* y una *Task* (un estado y una tarea)
    c. Un *State* debe tener al menos una *Task* (un estado de un tipo de objeto debe tener al menos una tarea asociada).
    d. Cada *Task* asociada a un *State* deben ser del mismo *TypeElement* (no mezclar de distintos tipos de objetos).
    e. Un *Element* debe tener un *State* inicial (todo objeto físico se debe encontrar en un estado inicial de los existentes para ese tipo de objeto).
    f. El *State* inicial de un *Element* debe ser del mismo *TypeElement* (no mezclar de distintos tipos de objetos).
5. Un *WorkItem* debe tener asociado un *Element* y una *Task* (las tareas a modelar son sobre un objeto físico).
6. El *Element*, *Task* y posible nuevo *State* de un *Workitem* deben ser del mismo *TypeElement* (no mezclar de distintos tipos de objetos).

Para esta parte del desarrollo pasaríamos a usar OAW, en concreto el fichero *Checks.chk* será el que incorpore todas las reglas de validación. El fichero *workflow.oaw* gestiona todo el flujo de validación y generación de código, de forma que en primer lugar una vez cargada la instancia del metamodelo, esta se evalúa semánticamente, comprobando que se cumplen todas las reglas del archivo *Checks.chk*. Veamos estas reglas de validación en las tablas siguientes. Las reglas parecidas no se ponen para todas las clases involucradas, es suficiente con poner una para comprender la estructura de la regla.

Regla: La clase padre debe tener alguna clase hija.
Clases: ScenarioModel, TypeElement

`context ScenarioModel ERROR`
`  "ScenarioModel: types can't be null" :`
`  types.size != 0;`

Tabla 5.16. Check. La clase padre debe tener alguna clase hija

Regla: Evitar nombres de clases con valor *null*
Clases: TypeElement, Element, Task, State

`context TypeElement ERROR`
`  "TypeElement: must have a name!" :`
`  this.name != null;`

Tabla 5.17. Check. Evitar nombres de clase con valor *null*

Regla: Debe existir al menos una clase referenciada
Clases: State, Element, WorkItem

`context State ERROR`
`  "State: must have 1 or more tasks!" :`
`  this.state_tasks!= null;`

Tabla 5.18. Check. Debe existir al menos una clase referenciada

Regla: Evitar repetir nombres de clases
Clases: ScenarioModel, TypeElement

`context ScenarioModel ERROR`
`  "ScenarioModel: Names of TypeElements must be unique" :`
`  types.forAll(a1 | types.notExists(a2 | a1 != a2 &&`
`a1.name.toUpperCase() == a2.name.toUpperCase() ));`

Tabla 5.19. Check. Evitar repetir nombres de clase

Regla: No mezclar objetos de distintos TypeElements
Clases: Element, State, WorkItem

`context Element if this.initial_state != null ERROR`
`  "Element: Initial state must be from same TypeElement!" :`
`  this.eContainer == this.initial_state.eContainer;`
`context State if state_tasks.size!=0 ERROR`
`  "State: Tasks must be from same TypeElement" :`
`  state_tasks.forAll(a1 | a1.eContainer == this.eContainer);`
`context WorkItem if (wi_element.eContainer != null) && (wi_task.eContainer != null) ERROR`
`  "WorkItem: Element, Task and State must be from same TypeElement!" :`
`  (wi_change_state == null && (wi_element.eContainer == wi_task.eContainer)) ||`
`(wi_change_state != null && (wi_element.eContainer == wi_task.eContainer) &&`
`(wi_task.eContainer == wi_change_state.eContainer));`

Tabla 5.20. Check. No mezclar objetos de distintos *TypeElement*

En la figura 5.20 se puede ver el resultado de evaluar un modelo con errores. El total de las reglas se pueden consultar en el apéndice C.

```
0    INFO  WorkflowRunner     - ----------------------------------------------------------------------------
0    INFO  WorkflowRunner     - openArchitectureWare 4.3.1, Build 20090107-2000PRD
0    INFO  WorkflowRunner     - (c) 2005-2008 openarchitectureware.org and contributors
0    INFO  WorkflowRunner     - ----------------------------------------------------------------------------
0    INFO  WorkflowRunner     - running workflow: workflow/Workflow.oaw
0    INFO  WorkflowRunner     -
873  INFO  CompositeComponent - XmiReader(xmiParser): file 'default.scenario' => slot 'model'
1108 INFO  CompositeComponent - CheckComponent: slot model check file(s): metamodel/Checks
1374 ERROR WorkflowRunner     - Workflow interrupted. Reason: Errors during validation.
1374 ERROR WorkflowRunner     - TypeElement: Names of Elements must be unique
[scenario.impl.TypeElementImpl@141b571 (name: SERVICIOS HOTEL)]
```

Figura 5.20. Check. Resultado de validar un modelo erróneo

5.2.5 Generación de código

A partir de la descripción de un escenario obtenida mediante el editor gráfico visto en la sección anterior y basada en el metamodelo definido, y una vez validado correctamente, podemos pasar a generar código mediante técnicas de transformación M2T. OAW y una plantilla xpand nos permitirán recorrer el modelo o escenario obtenido y generar el código asociado. La aplicación de plantillas a modelos funcionan de manera similar a la forma en la que se usan para generar páginas web dinámicas en el área de desarrollo de aplicaciones web.

La plantilla xpand contiene partes estáticas y dinámicas. Las partes estáticas son pasadas automáticamente al código generado, para las partes dinámicas los elementos del modelo pueden ser iterados y trozos de código son generados instanciándolos con valores obtenidos del modelo.

OAW nos permite extender el metamodelo con funciones (apéndice D), de forma que xpand puede hacer uso de las mismas cuando recorra la instancia del metamodelo. Hemos extendido el metamodelo con dos funciones: (1) *onclick*, que extiende a *WorkItem*, devolviendo las funciones javascript a ejecutar cuando el usuario valida una tarea, y (2) *actions*, función que usa onclick para obtener un listado de tareas asociados al elemento del *WorkItem*.

Como resultado final obtenemos los ficheros *index.html* y *control.html*. El primero simula la interfaz del usuario y el segundo lo usará un operador para simular la detección de elementos físicos o la resolución de una tarea (técnica Mago de Oz). Para la simulación solo necesitamos un servidor web donde alojar las páginas y un dispositivo conectado a la red para visualizarlas.

Como vimos en la sección 5.1.2, los ficheros generados se enmarcan en el framework IUI, usado para crear aplicaciones web con aspecto similar a las del popular iPhone. Las diferentes pantallas generadas para el usuario (figura 5.20) estarán en el fichero *index.html*, y estarán delimitadas por los marcadores *
* y *
*, también usaremos los marcadores * * y * * para las pantallas en las que queramos hacer más énfasis, por ejemplo para la resolución de una tarea pendiente. Cada pantalla tendrá un identificador *id*, de forma que podremos usar funciones javascript para activar aquella que nos interese. También usaremos funciones para:

1. Cambiar el estado de un elemento en su pantalla asociada.
2. Eliminar una tarea pendiente de la pantalla inicial.

3. Eliminar la pantalla de un elemento que ha cambiado de estado.
4. Crear una pantalla nueva para un elemento que ha cambiado de estado.
5. Añadir a la pantalla creada las acciones que se pueden realizar en su nuevo estado.

La consola (figura 5.21) generada para el operador (fichero *control.html*) consistirá en una única pantalla desde la que podrá:

1. Lanzar eventos de detección para cada elemento físico.
2. Lanzar eventos de fin de tarea pendiente.

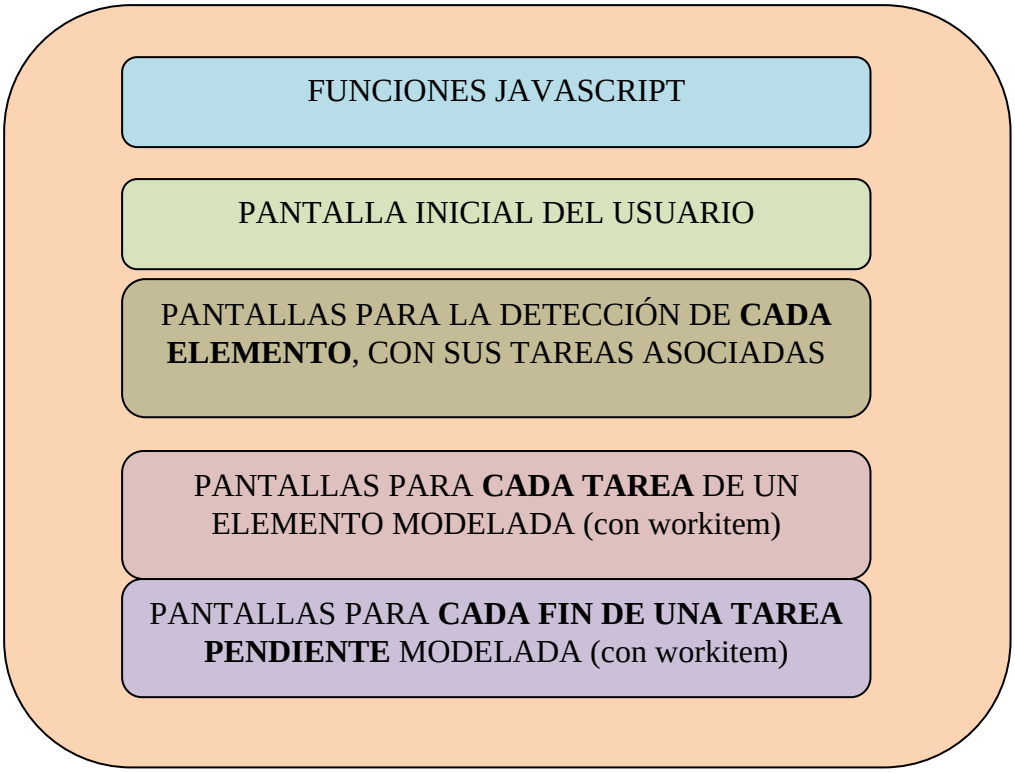


Figura 5.21. Estructura *fichero index.html*

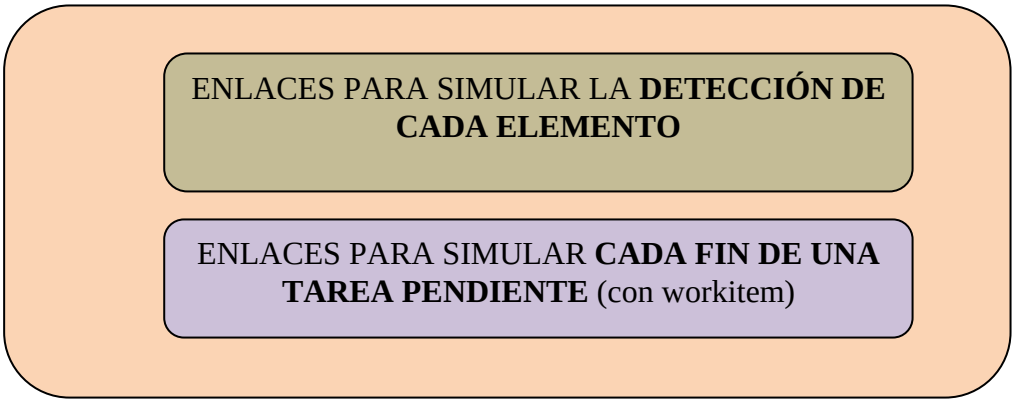


Figura 5.22. Estructura fichero *control.html*

Figura 5.23. Ficheros generados.

Para obtener el resultado de la figura 5.23 y siguiendo con el flujo del proceso regulado por el fichero *workflow.oaw*, se carga el fichero *Template.xpt*, el cual tiene la plantilla *xpand* con las transformaciones a realizar. Esta plantilla incluye un punto de inicio (*Root*) que es el que se llama desde *workflow.oaw* ("template::*Template::Root* FOR model"). Vamos a ver de forma general las transformaciones realizadas. La plantilla entera se puede consultar en el apéndice C.

5.2.5.1. Prototipo del usuario

En primer lugar se genera el fichero *index.html*:

1. Obtenemos la pantalla inicial que será mostrada al usuario (tabla 5.21):
2. Para sacar la lista de tareas pendientes:
    a. Recorreremos todos los elementos de cada tipo, y sacaremos las tareas pendientes asociadas al estado inicial del elemento.
3. Para sacar el listado informativo de todas las tareas que se pueden realizar:
    a. Recorremos todos los tipos de elementos y sacamos la lista de tareas con el atributo pending a false.
    b. Pondremos antes de la lista de tareas a qué tipo de elemento pertenecen.

```
«REM»
     CODIGO PARA CADA CADA TAREA PENDIENTE, pantalla inicial
     «ENDREM»
        «FOREACH ty.te_elements AS e-»
          «FOREACH e.initial_state.state_tasks AS t-»
            «IF t.pending==true-»
   <li id="action-«t.name.replaceAll(' ','')»-«e.name.replaceAll('
',''')»"><a href="#decode" >«t.name» «ty.name» «e.name»</a></li>
            «ENDIF-»
          «ENDFOREACH-»
        «ENDFOREACH-»                                                    2a
     «ENDFOREACH-»
         <li id="actions_group" class="group">Actions</li>
     «FOREACH types AS ty-»
     «REM»
     CODIGO PARA CADA CADA TAREA, pantalla inicial
     «ENDREM»                                                            3b
   <li><div class="annotation" >«ty.name»</div></li>
        «FOREACH ty.te_tasks AS t-»
          «IF t.pending!=true-»
   <li><a href="#decode" onclick="setAction('«t.name.replaceAll('
',''')»');">«t.name»</a></li>
          «ENDIF-»
        «ENDFOREACH-»                                                    3a
     «ENDFOREACH-»
```

Tabla 5.21. Plantilla xpand. Pantalla inicial

4. Obtenemos la pantalla que será mostrada cuando sea detectado un elemento físico (tabla 5.22):
    a. Recorremos todos los tipos de elementos, para acceder a todos sus elementos.
    b. Mostramos el estado del elemento.
    c. Mostramos la lista de tareas con el atributo pending a false, asociadas a ese elemento en su estado actual.

```
«REM»
     CODIGO PARA CADA ELEMENTO
     «ENDREM»
     «FOREACH types.te_elements AS e-»
<ul id="«e.name.replaceAll(' ','')»" title="Details" >
  <li><h2>«e.name»</h2> <div class="annotation"
id="«e.name.replaceAll(' ','')»-
status">«e.initial_state.name»</div></li>
    <li id="actions-«e.name.replaceAll(' ','')»"                         4b
class="group">Actions</li>
          «FOREACH e.initial_state.state_tasks AS t-»
          «IF t.pending!=true-»
    <li id="action-«t.name.replaceAll(' ','')»-«e.name.replaceAll('
',''')»"><a href="#«t.name.replaceAll(' ','')»-«e.name.replaceAll('
',''')»">«t.name»</a></li>
          «ENDIF-»                                                       4c
          «ENDFOREACH-»
</ul>                                                                    4a
     «ENDFOREACH-»
```

Tabla 5.22. Plantilla xpand. Pantalla para cada elemento detectado

5. Obtenemos la pantalla para el fin de cada tarea pendiente, un *Workitem* con Task.pending igual a true, (tabla 5.23):
    a. Tras validar la tarea, si la tarea pendiente no produce un cambio de estado, la borramos de la pantalla inicial.
    b. Tras validar la tarea, si produce un cambio de estado, borramoa la tarea pendiente de la pantalla inicial y llamamos a la función onclick que nos devuelve llamadas a funciones javascript definidas en el fichero *index.html*:
        i. Eliminamos la pantalla del elemento
        ii. Volvemos a crear la pantalla del elemento con su nuevo estado y las tareas asociadas.
    c. Mostramos la información genérica asociada a la tarea.
    d. Mostramos la información particular asociada al workitem.

```
<form id="«w.wi_task.name.replaceAll(' ','')»-
«w.wi_element.name.replaceAll(' ','')»" class="dialog" >
        <fieldset>
          <h1>Evento</h1>
            «IF w.wi_change_state!=null-»
   <a class="button blueButton"  type="cancel"
onclick="remove('action-«w.wi_task.name.replaceAll(' ','')»-
«w.wi_element.name.replaceAll(' ','')»');«w.onclick()»">Ok</a>
            «ELSE-»
   <a class="button blueButton" type="cancel"
onclick="remove('action-«w.wi_task.name.replaceAll(' ','')»-
«w.wi_element.name.replaceAll(' ','')»');">Ok</a>
            «ENDIF-»
        </fieldset>
        <div  class="panel">
         <fieldset>
            <div class="row">
          <ul>
          <li class="linklike"><h2> «w.wi_element.name» </h2></li>
          «REM»
          CODIGO PARA LA INFO ASOCIADA A LA TAREA
          «ENDREM»
          «FOREACH w.wi_task.task_info AS i-»
           «EXPAND informas FOR i-»
          «ENDFOREACH-»
          «REM»
          CODIGO PARA LA INFO ASOCIADA AL WORKITEM
          «ENDREM»
          «FOREACH w.wi_info AS i-»
           «EXPAND informas FOR i-»
          «ENDFOREACH-»
       </ul>
   </div>
   </div>
</form>
```

5b 5a 5c 5d

Tabla 5.23. Plantilla xpand. Workitem de evento (tarea pendiente)

6. Obtenemos las pantallas para cada tarea modelada, *WorkItem* con *task.pending* igual a false (tabla 5.24):
    a. Mostramos la información genérica asociada a la tarea.
    b. Mostramos la información particular asociada al workitem.
    c. Tras validar la tarea, si no produce un cambio de estado no hacemos nada.
    d. Tras validar la tarea, si produce un cambio de estado, llamamos a la función onclick que nos devuelve llamadas a funciones javascript definidas en el fichero *index.html*:
        i. Eliminamos la pantalla del elemento
        ii. Volvemos a crear la pantalla del elemento con su nuevo estado y las tareas asociadas.

```
<ul id="«w.wi_task.name.replaceAll(' ','')»-
«w.wi_element.name.replaceAll(' ','')»" title="«w.wi_task.name»" >
        <li><h2>«w.wi_element.name»</h2></li>
        <li class="group">«w.wi_task.name»</li>
            «REM»
          CODIGO PARA LA INFO ASOCIADA A LA TAREA
          «ENDREM»
          «FOREACH w.wi_task.task_info AS i-»
           «EXPAND informas FOR i-»
          «ENDFOREACH-»                                          6a
          «REM»
          CODIGO PARA LA INFO ASOCIADA AL WORKITEM
          «ENDREM»
          «FOREACH w.wi_info AS i-»
           «EXPAND informas FOR i-»
          «ENDFOREACH-»                                          6b
          «IF w.wi_change_state!=null-»
   <li><a href="#«w.wi_element.name.replaceAll(' ','')»"
onclick="«w.onclick()»">Ok</a></li>                              6d
          «ELSE-»
   <li><a href="#«w.wi_element.name.replaceAll(' ','')»">Ok</a></
          «ENDIF-»                                               6c
</ul>
```

Tabla 5.24. Plantilla xpand. Workitem normal

En los pasos 5 y 6 hemos recorrido todas las tareas modeladas (workitems), y generado pantallas según sean fin de tarea pendiente o no.

```
«FOREACH workitems AS w-»
       «IF w.wi_task.pending-»
          Código tabla 5.5
       «ENDIF»
          Código tabla 5.6
       «ENDIF»
«ENDFOREACH»
```

5.2.5.2. Consola del operador

En segundo lugar creamos el fichero *control.html*, que contendrá la pantalla html que usará un operador para simular la detección de elementos y el fin de tareas pendientes.

7. Recorremos todos los elementos de todos los tipos e insertamos un enlace para su detección.

```
    «FOREACH types.te_elements AS e-»
  <li><a href="/set/«e.name.replaceAll(' ','')»">«((TypeElement)
e.eContainer).name»-«e.name»</a></li>
    «ENDFOREACH»
```

Tabla 5.25. Plantilla xpand. Simular detección de un objeto físico

8. Para simular un evento, recorremos todos los *workitems* y ponemos el enlace para aquellos que tengan asociada una tarea pendiente.

```
«FOREACH workitems AS w-»
    «IF w.wi_task.pending==true-»
  <li><a href="/set/«w.wi_task.name.replaceAll(' ','')»-
«w.wi_element.name.replaceAll(' ','')»">«((TypeElement)
w.wi_element.eContainer).name»-«w.wi_task.name»
«w.wi_element.name»</a></li>
    «ENDIF-»
    «ENDFOREACH-»
```

Tabla 5.26. Plantilla xpand. Simular evento

La forma de activar las pantallas correspondientes para el usuario es llamando a la página /set/id-de-la-pantalla-a-mostrar, id que ya se creó en cada pantalla del fichero index.html.

5.3 Conclusiones

En este punto hemos visto cómo hemos desarrollado nuestra propuesta usando la tecnología y herramientas explicadas en el capítulo 4, obteniendo como resultado unos prototipos adecuados para la simulación de sistemas que soporten flujos de trabajo móviles, y todo ello de una forma sencilla y rápida. Para clarificar más los conceptos vamos a pasar a ver en el punto siguiente un caso de estudio.

6. Caso de estudio. Aplicación de la propuesta

En este capítulo vamos a ver la aplicación práctica de nuestra propuesta mediante un sencillo caso de estudio de un hotel. En primer lugar describiremos el caso de estudio. Pasaremos a ver los conceptos del caso de estudio que queremos modelar, y los definiremos mediante el editor gráfico creado para tal propósito. Continuaremos viendo el resultado de aplicar las transformaciones al modelo, comentando parte del código generado. Finalmente veremos los prototipos obtenidos, tanto el prototipo que verá el usuario, como la consola usada por el operador para simular la detección de elementos del mundo físico y eventos producidos por estos.

6.1 Descripción del Hotel Inteligente

6.1.1 Hotel Inteligente

Las habitaciones del hotel tienen una consola desde la que se pueden manejar funciones de la habitación como la iluminación, el clima, etc.

Estas funciones pueden ser activadas desde la consola de la habitación o bien desde una aplicación móvil desarrollada para iPhone. Para ello al cliente se le facilita una hoja con una serie de códigos QR, uno es la dirección web de la aplicación, y el resto de códigos corresponde cada uno a una función de la habitación.

Esto permite al cliente activar las funciones desde el sofá, la cama, o incluso desde fuera de la habitación.

El hotel también les ofrece a sus clientes servicios de masaje y spa. Estos servicios son detectados automáticamente mediante RFID cuando el cliente pasa cerca de los locales destinados a ofrecerlos. Si el cliente tiene la aplicación cargada en su iPhone podrá solicitar el servicio o bien ponerse en lista de espera.

Para comprobar la generalidad de nuestra propuesta hemos considerado la existencia en el hotel de dos tipos de elementos físicos: (1) las funciones de la habitación y (2) los servicios del hotel.

6.1.2. Escenario

La familia Pérez ha reservado su estancia en nuestro hotel, cuando hizo la reserva online respondió a una serie de preguntas sobre la temperatura y la iluminación de la

habitación, para que esta se encuentre como ellos esperan a su entrada. Los Pérez esperan poder disfrutar del relax del spa y los demás servicios que ofrece el hotel.

Cuando Juan Pérez llega a la recepción del hotel recibe una tarjeta para abrir la habitación, y le dan en un papel un listado de códigos QR. Le informan de que el primer código es dirección web de la aplicación que le permitirá manejar la habitación, y el resto, corresponde cada uno a una función de la habitación. También le proporcionan las claves para entrar en la aplicación. Como Juan no tiene ningún dispositivo para conectarse a la wifi del hotel, éste le cede un ipod previo pago de una fianza. Le informan que en la habitación hay una consola para manejar las funciones de la habitación, funciones que se pueden activar previamente desde el iPhone, aunque no esté físicamente en la habitación.

Figura 6.1. Caso de Estudio. Código QR entregado a Juan

Al llegar a la habitación introduce la tarjeta y entra en la habitación, en la consola le aparece un mensaje de bienvenida. Comprueba que la habitación está no muy caldeada y con una iluminación no muy fuerte que es lo que había solicitado por internet.

Como es la hora de bañar a la niña su mujer le pide que suba la temperatura de la habitación, para ello lee el código QR con su móvil y accede a la dirección web. Lo primero que ve son las tareas pendientes y entre ellas hay una que es *Arreglar/Habilitar SERVICIOS HOTEL clima*, "vaya contrariedad", piensa Juan, pero al momento recibe un aviso en el ipod de que la avería está solucionada, Juan activa el servicio, y ya desde la consola sube dos grados la temperatura. Almacena en los favoritos la dirección web, de forma que no tenga que volver a leer con el móvil el código QR.

Cuando están terminando de arreglar a la niña reciben en el ipod un aviso de que está habilitada la función alimentación, como ya es muy tarde y están muy cansados, deciden coger de nuevo el ipod, leen el código QR asociado, y solicitan la cena en la habitación, para ello al momento les llaman por teléfono y les indican el menú disponible.

Después de cenar Juan acerca el ipod a la zona de códigos QR *Iluminación* y activa desde el ipod el servicio, selecciona la iluminación nocturna, la cual deja una iluminación muy tenue, y de esta forma consiguen dormir a la niña, lo cual les deja

tiempo para ver una película, como no les gusta la parrilla, activan el pay-per-view, y al instante tienen disponible una serie de películas en la pantalla del televisor.

Los Pérez dejan la habitación y van al restaurante a desayunar, de camino pasan por delante de la sala de masaje, al instante el ipod les detecta el servicio, así que Juan decide hacer uso del mismo, comprueba que el servicio está ocupado, pero que puede añadirse a la lista de espera, Juan confirma y espera el aviso de que pueda recibir el masaje. Mientras están terminando de desayunar, Juan recibe el mensaje en el ipod de que el servicio de masaje está disponible, así que solicita el servicio.

El resto de la familia se acercan al spa, tras ser detectado el servicio deciden solicitarlo, el cual sí está disponible.

Mientras disfrutan del relax del hotel reciben un nuevo aviso en el ipod informándoles que el servicio Limpieza ya está habilitado, así que deciden solicitarlo.

“Esto sí son vacaciones” piensa Juan mientras recibe su masaje cómodamente tumbado…..

6.2 Modelado del caso de estudio

Una vez expuesto el caso de estudio, pasaremos en esta sección a aplicar nuestra propuesta: primero identificaremos los conceptos a modelar, pasaremos a realizar el modelado con el editor, seguiremos viendo el resultado de aplicar la plantilla de transformación y finalmente veremos los resultados obtenidos.

Como punto de partida podemos usar los procesos de negocio asociados a los dos tipos de elementos detectados “Servicios HOTEL” y “FUNCIONES HABITACIÓN”.

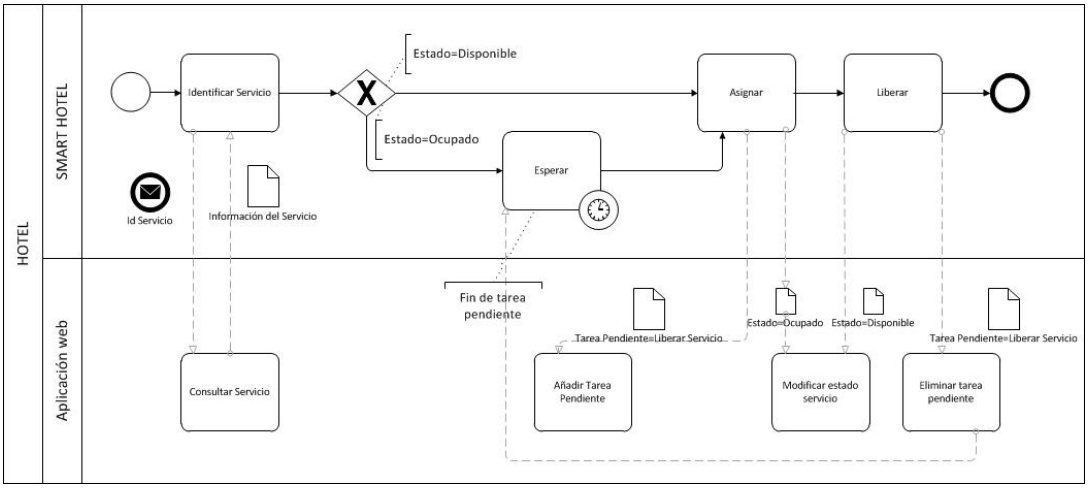


Figura 6.2. Caso de Estudio. Diagrama BPMN para los servicios del hotel

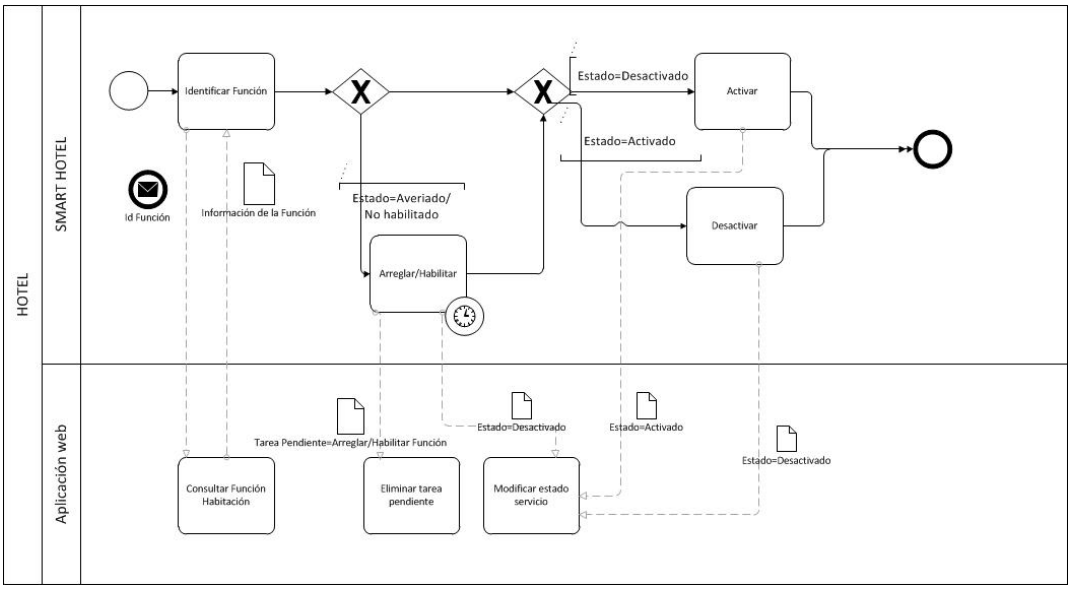


Figura 6.3. Caso de Estudio. Diagrama BPMN para las funciones de la habitación

6.2.1 Conceptos del caso de estudio a modelar

De la lectura del caso de estudio y de los procesos de negocio asociados podemos obtener los conceptos del mundo físico y las tareas a modelar.

Entorno Físico

Se observan dos tipos de elementos, (1) los servicios que nos ofrece el hotel, como el spa y el masaje, y (2) las funciones que nos ofrece la habitación, como el clima o la iluminación.

Un servicio del hotel se puede usar, liberar o esperar si está ocupado. Una función de la habitación se puede activar o desactivar, y arreglar o habilitar en caso de que esté estropeada o fuera de horario (como las comidas o la limpieza).

Estos conceptos los podemos resumir en la figura 6.4.

Figura 6.4. Caso de Estudio. Entorno físico

Tareas a modelar.

Lo siguiente sería ver las tareas que se producen en el caso de estudio, pues no vamos a modelar todas las combinaciones posibles. De la lectura del caso de estudio obtenemos las tareas de la figura 6.5 como participantes de nuestro escenario.

Figura 6.5. Caso de Estudio. Tareas a modelar

Una vez tenemos los conceptos que queremos modelar pasaremos a usar el editor y a definirlos mediante nuestro metamodelo.

6.2.2 Creación del escenario con el editor

En primer lugar debemos dibujar con el editor gráfico (figuras 6.6 y 6.7) los tipos de objetos (*TypeElements*) SERVICIOS HOTEL y FUNCIONES HABITACIÓN, junto con sus objetos físicos (*Elements*), tareas (*Tasks*) y estados (*States*), tal y como se vió en la figura 6.4. Añadiremos información dentro de cada *Task*. De esta forma modelamos el entorno físico del escenario.

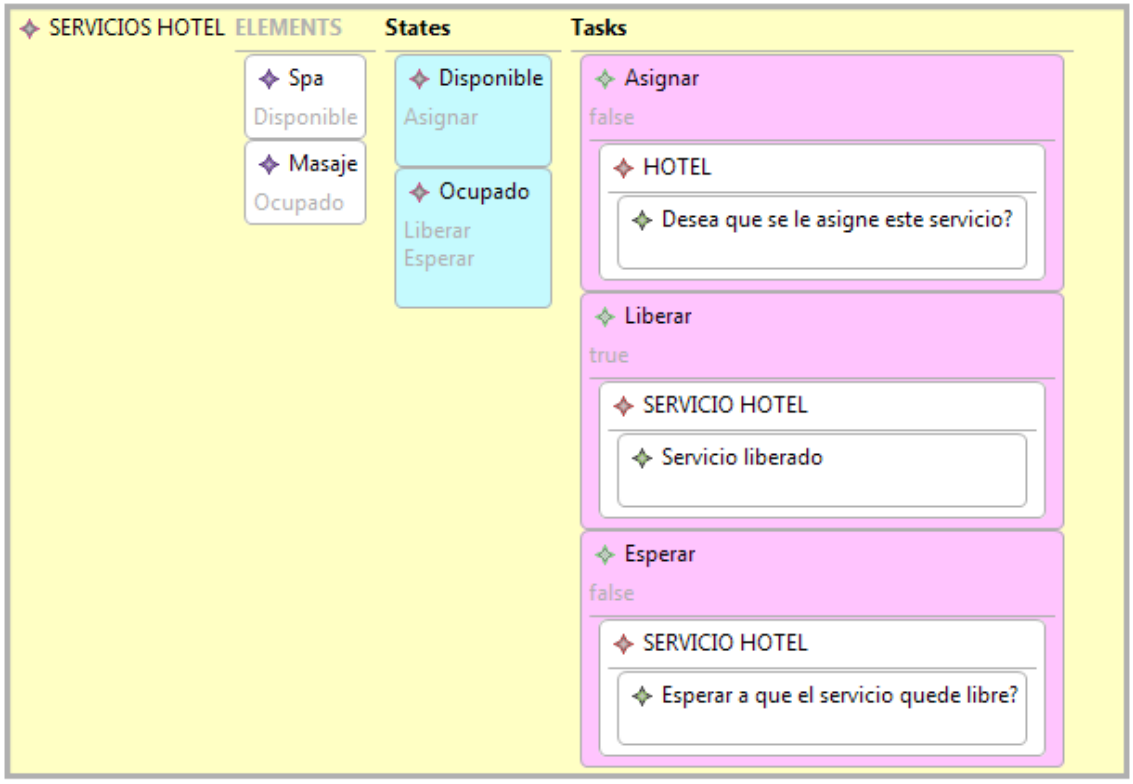


Figura 6.6. Caso de Estudio. SERVICIOS HOTEL

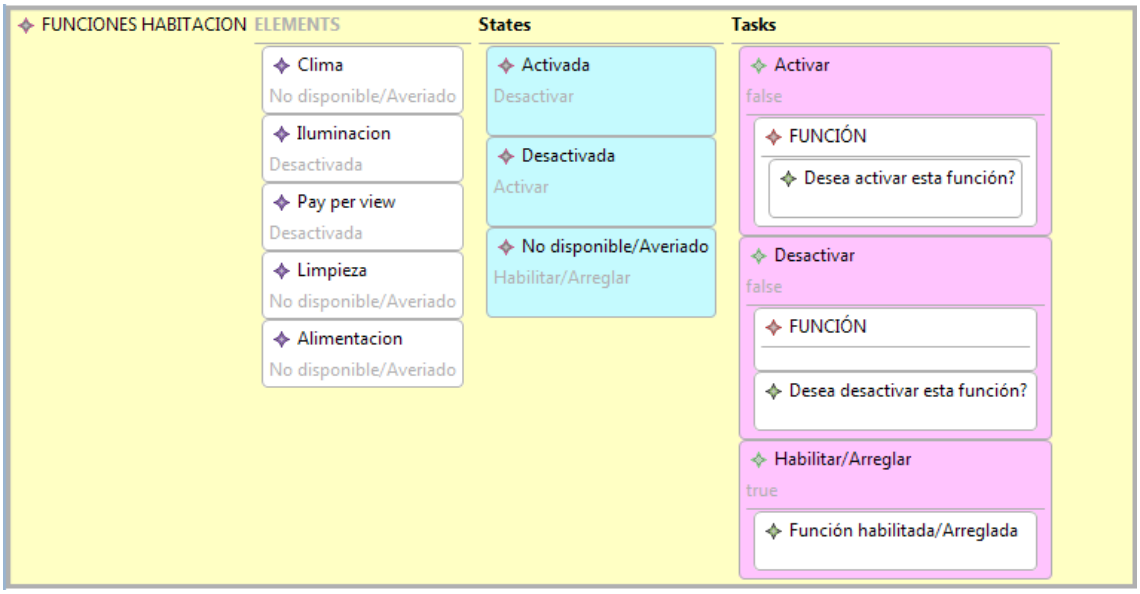


Figura 6.7. Caso de Estudio. FUNCIONES HABITACIÓN

En segundo lugar hay que dibujar las tareas a modelar (*Workitems*) de nuestro escenario (figuras 6.8 y 6.9), tal y cómo se vió en la figura 6.5. Recordemos que las tareas a modelar están formadas por un elemento, una tarea y un posible cambio de estado. Si la tarea que forma el *Workitem* es una tarea pendiente estaremos ante un evento, que creará una entrada en la consola del operador para su simulación. Añadiremos información dentro de cada *Workitem*.

Figura 6.8. Caso de Estudio. WorkItems para SERVICIOS HOTEL

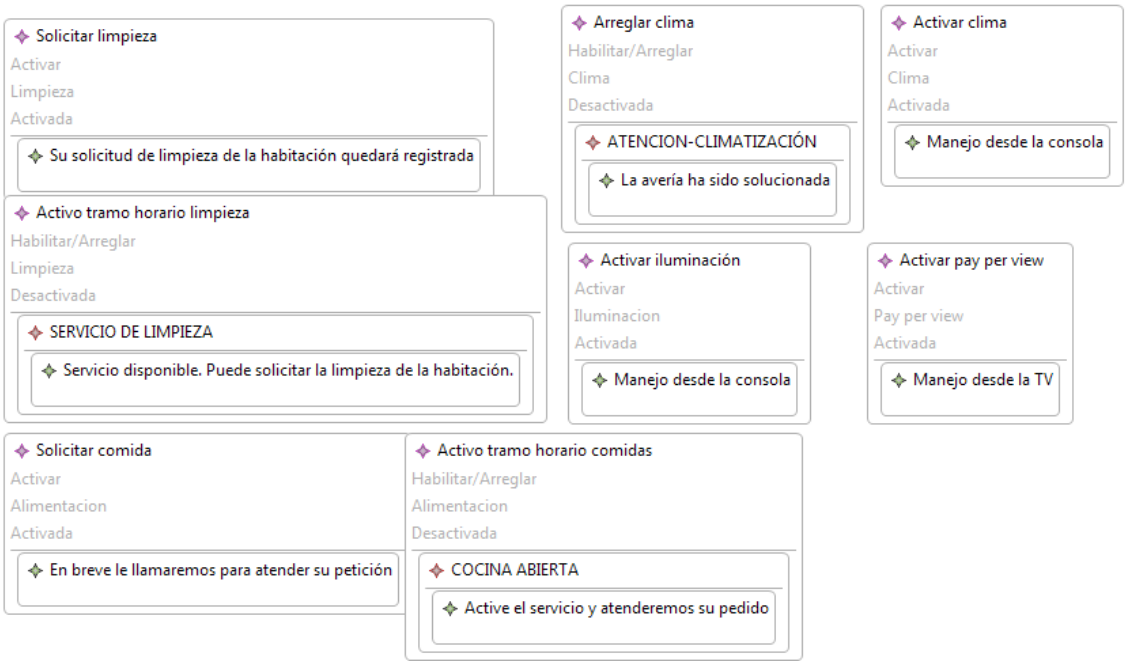


Figura 6.9. Caso de Estudio. WorkItems para FUNCIONES HABITACIÓN

Una vez dibujado el escenario, obtenemos el resultado en el fichero *default.scenario*, el cual pasamos al generador oaw, y si todo es correcto obtenemos los prototipos en html. En el punto siguiente vamos a ver los resultados obtenidos.

6.2.3 Transformación realizada

Una vez validado correctamente el escenario guardado en el fichero *default.scenario* y tras aplicarle la plantilla *Xpand*, obtenemos los ficheros *index.html* para el usuario y *control.html* para el operador.

Veamos los resultados de las transformaciones

6.2.3.1 Prototipo del usuario (*index.html*)

Recuadraremos en color morado el identificador de cada pantalla, se usará para activarla.

Código de la pantalla inicial

1. Listado de Tareas pendientes
2. Listado de Tareas
    a. Tareas para SERVICIOS HOTEL
    b. Tareas para FUNCIONES HABITACIÓN


```
<ul id="home" title="Tasks" selected="true" hideBackButton="true" >
    <li id="pending_group" class="group">Pending tasks</li>
    <li id="ret-link"><a href="#pending"></a></li>
    <div id="p-task"></div>

    <li id="action-Liberar-Masaje"><a href="#decode" >Liberar SERVICIOS HOTEL Masaje</a></li>
                                                                                        1
    <li id="action-Habilitar/Arreglar-Clima"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Clima</a></li>
    <li id="action-Habilitar/Arreglar-Limpieza"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Limpieza</a></li>
    <li id="action-Habilitar/Arreglar-Alimentacion"><a href="#decode" >Habilitar/Arreglar FUNCIONES HABITACION Alimentacion</a></li>
    <li id="actions_group" class="group">Actions</li>

        <li><div class="annotation" >SERVICIOS HOTEL</div></li>
        <li><a href="#decode" onclick="setAction('Asignar');">Asignar</a></li>
        <li><a href="#decode" onclick="setAction('Esperar');">Esperar</a></li>
                                                                                        2a
        <li><div class="annotation" >FUNCIONES HABITACION</div></li>
        <li><a href="#decode" onclick="setAction('Activar');">Activar</a></li>
        <li><a href="#decode" onclick="setAction('Desactivar');">Desactivar</a></li>
                                                                                        2b
                                                                                        2
</ul>
```

Código de la pantalla asociada a un elemento

1- Estado del elemento.

2- Listado de tareas en su estado actual.

```
<ul id="Spa" title="Details" >
    <li><h2>Spa</h2> <div class="annotation" id="Spa-status">Disponible</div></li>                    1
   <li id="actions-Spa" class="group">Actions</li>
   <li id="action-Asignar-Spa"><a href="#Asignar-Spa">Asignar</a></li>                                2
</ul>
```

Código de la pantalla asociada a una tarea de un elemento (workitem)

1- Información asociada a la tarea

2- Información asociada al workitem.

3- Al validar la tarea, se realiza una llamada a funciones javascript, que realizan lo siguiente:

   a. *Remove("Spa")*: elimina la pantalla Spa

   b. *Add_element("Spa","Spa","Ocupado")*: crea de nuevo la pantalla Spa con su nuevo estado.

   c. *Add_action("action-Esperar-Spa","Esperar",'actions-Spa','Esperar-spa')*: Añade la tarea "Esperar" a la pantalla Spa. Si hubiera más tareas asociadas al nuevo estado se añadirían.

   d. *Add("action-Liberar-Spa", "Libera Spa","pending-group","decode")*: añade la tarea pendiente "Liberar Spa" a la pantalla inicial del usuario.


```
<ul id="Asignar-Spa" title="Asignar" >
    <li><h2>Spa</h2></li>
    <li class="group">Asignar</li>

        <ul><p style="font-size: 20px;font-weight: bold;"> HOTEL </p>
        <li style="list-style: none"><p> Desea que se le asigne este servicio? </p></li>          1
        </ul>

        <li style="list-style: none"><p> Pase por recepción a recoger la llave. </p></li>          2
    <li><a href="#Spa" onclick="remove('Spa');add_element('Spa','Spa','Ocupado');add('action-
Esperar-Spa','Esperar','actions-Spa','Esperar-Spa');add('action-Liberar-Spa','Liberar
Spa','pending_group','decode');">Ok</a></li>                                                      3
</ul>
```

Código de la pantalla asociada al fin de una tarea pendiente (workitem)

1- Al validar el evento, se realiza una llamada a funciones javascript, que realizan lo siguiente:
   a. *remove('action-Liberar-Masaje')*: elimina la tarea pendiente de la pantalla inicial.
   b. *remove('Masaje')*:elimina la pantalla Masaje
   c. *add_element('Masaje','Masaje','Disponible')*:crea de nuevo la pantalla Masaje con su nuevo estado.
   d. *add('action-Asignar-Masaje','Asignar','actions-Masaje','Asignar-Masaje')*: añade la tarea "Asignar" a la pantalla Masaje. Si hubiera más tareas asociadas al nuevo estado se añadirían.
2- Información asociada a la tarea.
3- Información asociada al workitem.


```
<form id="Liberar-Masaje" class="dialog" >
      <fieldset>
        <h1>Evento</h1>
      <a class="button blueButton" type="cancel" onclick="remove('action-Liberar-
Masaje');remove('Masaje');add_element('Masaje','Masaje','Disponible');add('action-Asignar-
Masaje','Asignar','actions-Masaje','Asignar-Masaje');">Ok</a>                    1
      </fieldset>
      <div  class="panel">
       <fieldset>
         <div class="row">
         <ul>
         <li class="linklike"><h2> Masaje </h2></li>

            <ul><p style="font-size: 20px;font-weight: bold;"> SERVICIO HOTEL </p>
            <li style="list-style: none"><p> Servicio liberado </p></li>          2
            </ul>

            <li style="list-style: none"><p> Masaje disponible </p></li>          3
            </ul>
   </div>
   </div>
</form>
```

6.2.3.2 Consola del operador (*control.html*).

La consola del operador está formada por enlaces que activan bien las pantallas de los elementos detectados, o bien las pantallas de los eventos producidos, fin de tareas pendientes (figura 6.11).

Código para lanzar la detección de elementos.

```
<li><a href="/set/Spa">SERVICIOS HOTEL-Spa</a></li>
<li><a href="/set/Masaje">SERVICIOS HOTEL-Masaje</a></li>
<li><a href="/set/Clima">FUNCIONES HABITACION-Clima</a></li>
<li><a href="/set/Iluminacion">FUNCIONES HABITACION-Iluminacion</a></li>
<li><a href="/set/Payperview">FUNCIONES HABITACION-Pay per view</a></li>
<li><a href="/set/Limpieza">FUNCIONES HABITACION-Limpieza</a></li>
<li><a href="/set/Alimentacion">FUNCIONES HABITACION-Alimentacion</a></li>
```

Código para lanzar el fin de una tarea pendiente.

```
<li><a href="/set/Liberar-Masaje">SERVICIOS HOTEL-Liberar Masaje</a></li>
<li><a href="/set/Habilitar/Arreglar-Clima">FUNCIONES HABITACION-Habilitar/Arreglar Clima</a></li>
<li><a href="/set/Habilitar/Arreglar-Limpieza">FUNCIONES HABITACION-Habilitar/Arreglar Limpieza</a></li>
<li><a href="/set/Habilitar/Arreglar-Alimentacion">FUNCIONES HABITACION-Habilitar/Arreglar Alimentacion</a></li>
<li><a href="/set/null">Cancel</a></li>
```

6.2.4 Prototipos obtenidos

En primer lugar veamos las pantallas iniciales tanto del usuario (figura 6.10) como la que usará el operador (figura 6.11) para simular la detección de funciones/servicios del hotel, y la resolución de tareas pendientes (eventos).

Figura 6.10. Caso de Estudio. Pantalla inicial (*index.html*)

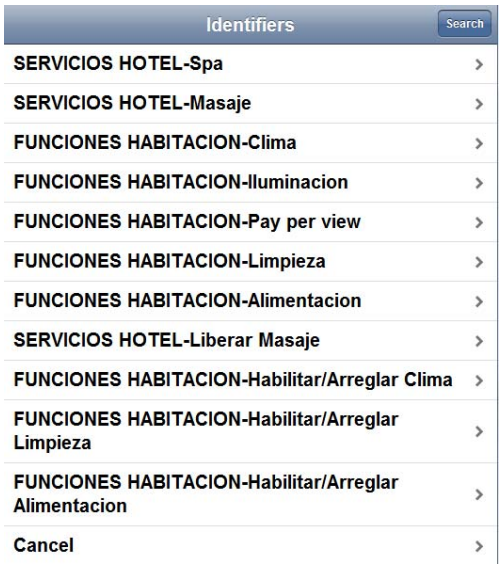


Figura 6.11. Caso de Estudio. Consola del operador (*control.html*)

Como podemos ver en la figura 6.11, el operador tiene la posibilidad de simular la detección de cada elemento físico, en primer lugar aparecen los elementos de SERVICIOS HOTEL, y después los de FUNCIONES HABITACIÓN. Le siguen la simulación de eventos, *WorkItems* que tienen una tarea pendiente (*Task* con atributo pending igual a *true*).

A partir de los procesos de negocio asociados a cada tipo de elementos, se presentan las siguientes situaciones:

1- FUNCIONES HABITACION
    a. Se detecta la función de una habitación:
        i. Estado *Activada*: Tarea disponible *Desactivar*.
        ii. Estado *Desactivada*: Tarea disponible: *Activar*.
    b. Se recibe el evento *Habilitar/Arreglar función*:
        i. Mensaje informativo.
        ii. Eliminar tarea pendiente.
        iii. Pasos de 1.a.
2- SERVICIOS HOTEL
    a. Se detecta un servicio del hotel.
        i. Estado *Ocupado*: Tarea disponible *Esperar*.
        ii. Estado *Disponible*: Tarea disponible: *Asignar*.
    b. Se recibe el evento *Liberar Servicio*:
        i. Mensaje informativo.
        ii. Eliminar tarea pendiente.
        iii. Pasos de 2.a.

Muchas de estas situaciones son repetitivas, así vamos a mostrar solo dos:

1- Avería clima solucionada (Evento) + Activar clima: 1.b + 1.a.ii.
2- Esperar masaje + Liberar masaje (Evento) + Asignar masaje: 2.a.i + 2.b + 2.a.ii

En primer lugar pondremos la opción que selecciona el operador de la figura 6.11 desde su consola en caso de ser necesario, y después las pantallas que va viendo el usuario.

Avería clima solucionada (Evento) + Activar clima.

Operador: FUNCIONES HABITACION-Habilitar/Arreglar Clima

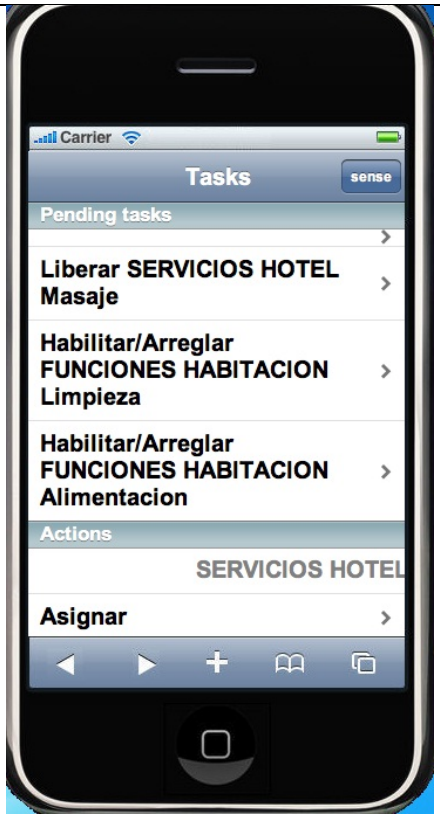
Usuario: 

1- El usuario recibe un evento que le informa de que se ha reparado la avería de climatización.

2- El usuario ve que tiene disponible la opción *Activar* para la climatización.

3- Tras seleccionar *Activar*, el usuario ve una pantalla de confirmación, mostrándole información al respecto. Hace click en *OK* para continuar.

4-Tras activar el clima, el usuario ve de nuevo la pantalla asociada al Clima, ahora con la opción de *Desactivar*.



5- Como el clima ya está arreglado, desaparece como tarea pendiente de la pantalla inicial del usuario.

Esperar masaje + Liberar masaje (Evento) + Asignar masaje.

Operador: SERVICIOS HOTEL-Masaje

Usuario: 

1- Tras detectar el masaje, el usuario ve que está ocupado, y que tiene disponible la opción de *Esperar*.

2- Tras seleccionar *Esperar*, el usuario ve una pantalla de confirmación, mostrándole información al respecto. Hace click en *OK* para continuar.

Operador: SERVICIOS HOTEL-Liberar Masaje

Usuario: 

3- El usuario recibe un evento que le informa que el servicio de masaje está disponible. (*Liberar masaje* se ha eliminado de la lista de tareas pendientes)

4- El usuario ve que tiene disponible la opción *Asignar* para el masaje.

5- Tras seleccionar *Asignar*, el usuario ve una pantalla de confirmación al respecto. Hace click en *OK* para continuar.

6- Le aparece de nuevo la pantalla asociada al masaje, ahora con la opción *Esperar* activa.

7- En la pantalla inicial del usuario se ha generado la tarea pendiente *Liberar masaje*.

6.3 Conclusiones

Hemos visto la aplicación de nuestra propuesta a un caso de estudio. Nos hemos centrado en los conceptos a modelar, en cómo capturar estos conceptos mediante el editor y finalmente en los prototipos o pantallas resultantes. Hemos mostrado cómo funcionan estas pantallas a través de los flujos de trabajo asociados al escenario modelado y cómo se simula la detección de elementos y eventos desde la consola del operador.

7. Conclusiones

En el presente trabajo hemos llegado a las siguientes conclusiones:

1- En cuanto a usar una aproximación **MDE**:
   a. El alto grado de **abstracción** que se consigue al utilizar metamodelos en un desarrollo MDE como es el presente trabajo.
   b. Lo complicado y difícil que es definir el **metamodelo** usado para capturar los conceptos usados para obtener los prototipos. De hecho podemos asegurar que la parte más complicada del presente trabajo fue la definición del metamodelo.
   c. Una vez obtenido el metamodelo es mucho más intuitivo usar un **editor gráfico** para obtener instancias del metamodelo.
2- En cuanto al uso de **prototipos**:
   a. La importancia de prototipar para tomar las decisiones de diseño **correctas** antes de realizar el desarrollo.
   b. Lo importante que es prototipar en desarrollos dentro del marco de la Internet de las cosas, en vías de **reducir** costes y tiempos de desarrollo.
   c. La importancia de usar técnicas ***Mago de Oz*** [Dahlbäck et al., 1993], en el marco de la Internet de las Cosas para simular la dotación de identidad digital a los elementos físicos sin tener que implementar la tecnología necesaria, con el consiguiente ahorro de costes.

8. Referencias

Abowd Gregory D., Hayes Gillian R., Iachello Giovanni, Kientz Julie A., Patel Shwetak N., Stevens Molly M. (2005). *Prototypes and Paratypes: Designing Mobile and Ubiquitous Computing Applications*. PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 67-73).

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B.,Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C. K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., & Yiu, A. (2007). *Web services business process execution language version 2.0. OASIS Specification.*

Böhlen Marc, Fabian Jesse, Pfeifer Dirk, Rinker JT (2005). *Prototypes in Pervasive Computing*. PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 78-80)

Bonnie E. John, Dario D. Salvucci (2005). *Multipurpose Prototypes for Assessing User Interfaces in Pervasive Computing Systems*. PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 27-34).

Bornhövd, C., Lin, T., Haller, S., & Schaper, J. (2004). *Integrating automatic data acquisition with business processes experiences with sap's auto-id infrastructure*. In vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases, (pp. 1182-1188). VLDB Endowment.

Chakraborty, D., & Lei, H. (2004). *Pervasive enablement of business processes*. In PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), (p. 87). Washington, DC, USA: IEEE Computer Society.

Dahlbäck, N., Jönsson, A., & Ahrenberg, L. (1993). *Wizard of oz studies: why and how*. In IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces, (pp. 193-200). New York, NY, USA: ACM.

den Bergh, J. V., & Coninx, K. (2005). *Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (cup)*. In SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization, (pp. 87-94). New York, NY, USA: ACM Press.

Dow Steven, MacIntyre Blair, Lee Jaemin, Oezbek Christopher, Jay David Bolter, Gandy Maribeth (2005). Wizard of Oz Support throughout an Iterative Design Process. PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 18-26).

Dumas, M., & ter Hofstede, A. H. M. (2001). *Uml activity diagrams as a workflow specification language*. In UML'01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, (p. 76–90). London, UK: Springer-Verlag.

Efftinge Sven, Friese Peter, Haase Arno, Hübner Dennis, Kadura Clemens, Kolb Bernd, Köhnlein Jan, Moroff Dieter, Thoms Karsten, Markus Völter, Schönbach Patrick, Eysholdt Moritz, Hübner Dennis, Reinisch Steven. (2008) *openArchitectureWare User Guide Version 4.3.1*.

Gershenfeld, N., Krikorian, R., & Cohen, D. (2004). *The internet of things*. Scientific American, 291 (4), 46-51.

Giner, P., & Torres, V. (2007). *Una propuesta basada en modelos para la construcción de sistemas ubicuos que den soporte a procesos de negocio*. In F. Losavio, G. H. Travassos, V. Pelechano, I. Diaz, & A.Matteo (Eds.) X Work shop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software. Isla Margarita (Venezuela).

Giner, P., Torres, V., & Pelechano, V. (2007a). *Building ubiquitous business process following an mdd approach*. In XII Jornadas de Ingeniería del Software y Bases de Datos. Zaragoza.

Giner, P., Torres, V., & Pelechano, V. (2007b). *Building ubiquitous business process following an mdd approach*. In XII Jornadas de Ingeniería del Software y Bases de Datos. Zaragoza.

Giner, P., Fons, J., & Pelechano, V. (2008). A domain specific language for the internet of things. V Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM'08).

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2009). *Presto: A pluggable platform for supporting user participation in smart workows*. In Proceedings of the Sixth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous). Toronto, Canada.

Giner, Pau. (2010). .Tesis. *Automating the development of Physical Mobile Workflows*. Centro de Investigación en Métodos de Producción de Software. Universidad Politécnica de Valencia.

Hackmann, G., Haitjema, M., Gill, C. D., & Roman, G.-C. (2006). *Sliver: A bpel workow process execution engine for mobile devices.* In ICSOC, (pp. 503-508).

Hansmann, U., Nicklous, M. S.,& Stober, T. (2001). *Pervasive computing handbook*. New York, NY, USA: Springer-Verlag New York, Inc.

Hofreiter, B., Huemer, C., & Klas, W. (2002). *ebxml: Status, research issues, and obstacles*. In Proc. Of 12th Int. Workshop on Research Issues on Data Engineering (RIDE02), (p. 7–16).

Jamali, B., Sharp, E., Thorne, A., & Cole, P. (2007). *Technology selection for identification applications*. Tech. rep., Auto-ID Labs.

Köhler, A., & Gruhn, V. (2004). *Analysis of mobile business processes for the design of mobile information systems*. In ECWeb, (pp. 238-247).

Li, Y., & Landay, J. (2008). Activity-based prototyping of ubicomp applications for longlived, everyday human activities. In ACM Conference on Human Factors in Computing Systems (HCI'08), (pp. 1303 - 1312).

Loke, S. W. (2009). *Building taskable spaces over ubiquitous services*. IEEE Pervasive Computing, 8 (4), 72-78.

Mayer, R. J., Painter, M. K., & DeWitte, P. (1992). *Idef family of methods for concurrent engineering and business re-engineering applications*. College Station, TX: Knowledge Based Systems, Inc.

Miller, J., & Mukerji, J. (2003). MDA Guide Version 1.0.1. Tech. rep., ObjectManagement Group (OMG).

Neely, S., Stevenson, G., Kray, C., Mulder, I., Connelly, K., & Siek, K. A. (2008). *Evaluating pervasive and ubiquitous systems*. IEEE Pervasive Computing, 7 (3), 85-88.

OMG (2006). *Business Process Modeling Notation (BPMN) Specification*. OMG Final Adopted Specification. dtc/06-02-01.

OMG (2011). *Business Process Model and Notation (BPMN) version 2.0*. OMG Final Adopted Specification. formal/2011-01-03.

Pajunen, L., & Chande, S. (2007). Developing workflow engine for mobile devices. In EDOC'07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, (p. 279). Washington, DC, USA: IEEE Computer Society.

Reilly Derek, Dearman David, Welsman-Dinelle Michael, Inkpen Kori (2005) .Evaluating Early Prototypes in Context: Trade-offs, Challenges, and Successes .PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 42-50).

Rukzio, E., Pleuss, A., & Terrenghi, L. (2005a). *The physical user interface profile (puip): modelling mobile interactions with the real world*. In TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams, (pp. 95-102). New York, NY, USA: ACM.

Rukzio, E., Wetzstein, S., & Schmidt, A. (2005b*). A framework for mobile interactions with the physical world*. In Wireless Personal Multimedia Communication (WPMC) conference. Aalborg, Denmark.

Rukzio, E., Leichtenstern, K., & Callaghan, V. (2006). *An experimental comparison of physical mobile interaction techniques: Touching, pointing and scanning*. In 8th International Conference on Ubiquitous Comput ing, UbiComp 2006 . Orange County, California.

Rukzio, E. (2007). *Physical Mobile Interactions: Mobile Devices as Pervasive Mediators for Interactions with the Real World*. Ph.D. thesis, Faculty for Mathematics, Computer Science and Statistics.

Siek Katie A., Neely Steve, Stevenson Graeme, Kray Christian, Mulder Ingrid (2009). *Advances in Evaluating Mobile and Ubiquitous Systems*. Guest Editorial Preface

Stringer Mark, A.Rode Jennifer, F.Toye Eleanor, F.B (2005).*The Webkit Tangible User Interface: A Case Study of Iterative Prototyping*. PERVASIVEComputing, OCTOBER-DECEMBER 2005 (pp. 35-41)

Urbanski, S., Huber, E., Wieland, M., Leymann, F., & Nicklas, D. (2009). *Perflows for the computers of the 21st century*. Pervasive Computing and Communications, IEEE International Conference on, 0 , 1-6.

Torres, V., Giner, P., & Pelechano, V. (2007). *Modeling ubiquitous business process driven applications*. In J. Eder, S. L. Tomassen, A. L. Opdahl, & G. Sindre (Eds.) CAiSE Forum. ISSN: 1503-416X.

Völter, M. (2005). *Software architecture patterns - a pattern language for building sustainable software architectures.*

Weiser, M. (1991). *The computer for the 21st century*. Scientific American, 265 (3), 66-75.

Wieland, M., Kaczmarczyk, P., & Nicklas, D. (2008). *Context integration for smart workflows*. In PerCom, (pp. 239-242).)

.

APÉNDICES.

A- METAMODELO

En este apéndice vamos a describir el metamodelo usado para modelar escenarios que soporten flujos de trabajo móviles. El metamodelo ha sido definido mediante Ecore, que es una implementación de la especificación Essential MOF ampliamente usada dentro de la comunidad Eclipse. Mediante herramientas gráficas de Eclipse crearemos un editor que nos permitirá hacer uso de este metamodelo para definir los escenarios a modelar. En la figura A.1 se pueden ver los metaelementos definidos y sus relaciones mediante un diagrama de clases. Estos elementos son descritos a continuación.

NamedElement. Es un metaelemento abstracto que es extendido por metaelementos con nombre.

ScenarioModel. Representa el escenario que vamos a modelar.

TypeElement. Representa un tipo de elementos físicos, por ejemplo DVD. Está formado por *Elements*, *Tasks* y *States*, que corresponden a objetos físicos, tareas y estados, respectivamente.

Workitem. Representa una tarea a modelar. Está formado por un *Element*, una *Task* y un posible cambio de *State*. Incorpora información *Info*.

Task. Representa las tareas posibles que se pueden realizar con un tipo de elemento.

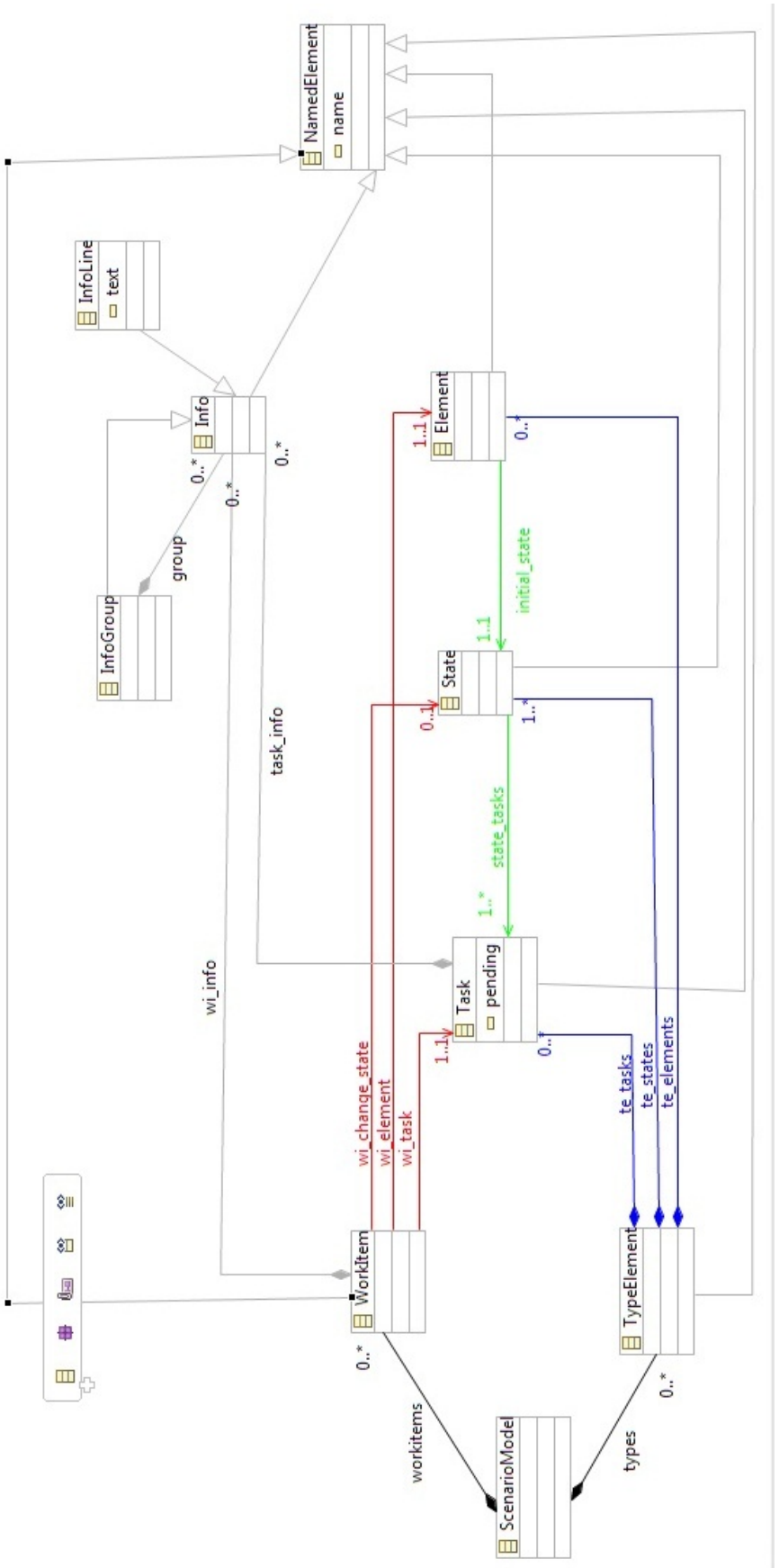
State. Representa los distintos estados en los que pueden estar los elementos de un determinado tipo. Cada estado tiene una serie de tareas asociadas, que representan las acciones que se pueden realizar en ese estado.

Element. Representa los elementos físicos concretos de un determinado tipo de elementos. Deben tener un estado inicial.

Info. Representa información.

Infogroup. Representa un grupo de información.

Infoline. Representa una línea de texto informativa.


NamedElement
name
InfoLine
text
Info
InfoGroup
group
Element
initial_state
State
task_info
state_tasks
Task
pending
wi_info
wi_change_state
wi_element
wi_task
te_tasks
te_states
te_elements
WorkItem
TypeElement
workitems
types
ScenarioModel
0..*
1..1
1..*
0..1

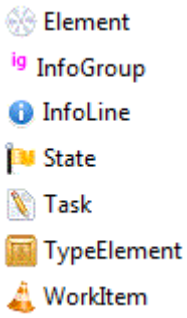
B- METAMODELO CON ANOTACIONES EUGENIA.

En este apéndice vamos a ver el metamodelo en formato emfatic y con las anotaciones Eugenia necesarias para crear el editor gráfico. A partir de este fichero crearemos el metamodelo ecore.

Las anotaciones Eugenia van precedidas de @gmf, tenemos las siguientes anotaciones según el elemento Ecore al que se apliquen:

EClass:

- *@gmf.diagram*: Denota el objeto raíz del metamodelo. Solo se puede poner en una clase y debe ser no abstracta. En nuestro caso es la clase *ScenarioModel*.
- *@gmf.node*: Denota que el objeto debe aparecer en el diagrama como un nodo. Uno de sus atributos indica el icomo que aparecerá en el editor. En nuestro caso tenemos los siguientes nodos: *Workitem, Element, Task, State, Infogroup, Infoline y TypeElement*. En la siguiente figura se pueden ver la paleta con los iconos asociados a cada uno



Ereference:

- *@gmf.compartment*: Para containment EReference, creará un compartimento donde irán los elementos contenidos por el nodo contenedor. En nuestro caso tenemos los siguientes compartimentos:
  - *Wi_info*, dentro de la clase Workitem, contiene elementos de la clase Info.
  - *Task_info*, dentro de la clase Task, contiene elementos de la clase Info.
  - *Te_elements, te_states y te_tasks*, dentro de la clase TypeElement, contienen respectivamente, elementos de la clase Element, State y Task.
  - *Group*, dentro de la clase InfoGroup, contiene elementos de la clase Info.

Los atributos que soportan cada una se pueden ver en http://www.eclipse.org/gmt/epsilon/doc/eugenia/

En el listado siguiente el nombre de cada clase corresponde al metaelemento con el mismo nombre del apéndice A. Las anotaciones Eugenias se han recuadrado y sombreado.

```
1  @namespace(uri="http://www.openarchitectureware.org/scenariomodel",
2  prefix="scenario")
3  package scenario;
4
5  abstract class NamedElement {
6    attr String name;
7  }
8
9  @gmf.diagram(foo="bar")
10 class ScenarioModel {
11
12   @gmf.compartment(foo="bar")
13   val WorkItem[*] workitems;
14
15   @gmf.compartment(foo="bar")
16   val TypeElement[*] types;
17 }
18
19 @gmf.node(label="name", tool.small.bundle="scenariomodel",
20 tool.small.path="model/icons/workitem.gif")
21 class WorkItem extends NamedElement {
22   ref Element[1] wi_element;
23   ref Task[1] wi_task;
24
25   @gmf.compartment(foo="bar", layout="list")
26   val Info[*] wi_info;
27   ref State wi_change_state;
28 }
29
30 @gmf.node(label="name", tool.small.bundle="scenariomodel",
31 tool.small.path="model/icons/element.gif")
32 class Element extends NamedElement {
33   ref State[1] initial_state;
34 }
35
36 @gmf.node(label="name", color="255,196,254",
37 tool.small.bundle="scenariomodel",
38 tool.small.path="model/icons/task.gif")
39 class Task extends NamedElement {
40   attr Boolean pending = "false";
41
42   @gmf.compartment(foo="bar", layout="list")
43   val Info[*] task_info;
44 }
45
46 @gmf.node(label="name", color="197,250,254",
47 tool.small.bundle="scenariomodel",
48 tool.small.path="model/icons/state.gif")
49 class State extends NamedElement {
50   ref Task[+] state_tasks;
51 }
```

```
52
53  @gmf.node(label="name", color="255,255,196", border.width="3",
54  tool.small.bundle="scenariomodel",
55  tool.small.path="model/icons/te.gif",
56  tool.large.bundle="scenariomodel",
57  tool.large.path="model/icons/tt.gif", figure="rectangle")
58  class TypeElement extends NamedElement {
59
60    @gmf.compartment(foo="bar", layout="list")
61    val Element[*] te_elements;
62
63    @gmf.compartment(foo="bar", layout="list")
64    val State[+] te_states;
65
66    @gmf.compartment(foo="bar", layout="list")
67    val Task[*] te_tasks;
68  }
69
70  class Info extends NamedElement {
71  }
72
73  @gmf.node(label="name", tool.small.bundle="scenariomodel",
74  tool.small.path="model/icons/infogroup.gif")
75  class InfoGroup extends Info {
76
77    @gmf.compartment(foo="bar", layout="list")
78    val Info[*] group;
79  }
80
81  @gmf.node(label="text", tool.small.bundle="scenariomodel",
82  tool.small.path="model/icons/infoline.gif")
83  class InfoLine extends Info {
84    attr String text;
85  }
```

C- REGLAS DE VALIDACIÓN

En este apéndice incluimos todas las reglas de validación que debe cumplir el modelo o escenario obtenido mediante el editor gráfico, para que pueda ser considerado semánticamente válido. Se encuentran incluidas dentro del fichero checks.chk.

Veamos el significado del fichero de reglas:

1. En primer lugar se importa el metamodelo a validar.
2. Después vienen las reglas con el siguiente formato:
    a. Se especifica el contexto dentro del metamodelo (metaclase) al que se aplica la restricción, seguido de ERROR o WARNING, estas palabras indican el tipo de acción a tomar en el caso de que la restricción falle

WARNING	Si la restricción falla se muestra el mensaje y se continúa la ejecución.
ERROR	Si la restricción falla se muestra el mensaje y se detiene el proceso.

    b. Después el mensaje a mostrar si la restricción falla. En este mensaje se pueden acceder a los atributos o a funciones para que sea más descriptivo.
    c. Finalmente se pone la condición a evaluar.

Veamos el listado de las reglas, junto con una breve descripción de su significado.

```
1  import scenario;
2
3  context ScenarioModel ERROR
4      "ScenarioModel: types can't be null" :
5      types.size != 0;
6  context TypeElement ERROR
7      "TypeElement: must have a name!" :
8      this.name != null;
9  context Element ERROR
10     "Element: must have a name!" :
11     this.name != null;
12 context Task ERROR
13     "Task: must have a name!" :
14     this.name != null;
15 context State ERROR
16     "State: must have a name!" :
17     this.name != null;
18 context State ERROR
```

La raíz debe contener algún elemento TypeElement.

Deben tener un nombre.

Un estado debe tener al menos una tarea.

```
19      "State: must have 1 or more tasks!" :
20      this.state_tasks != null;
21  context ScenarioModel ERROR
22      "ScenarioModel: Names of TypeElements must be unique" :
23      types.forAll(a1 | types.notExists(a2 | a1 != a2 && a1.name.toUpperCase() ==
24  a2.name.toUpperCase() ));
25  context TypeElement ERROR
26      "TypeElement: Names of Elements must be unique" :
27      te_elements.forAll(a1 | te_elements.notExists(a2 | a1 != a2 && a1.name.toUpperCase() ==
28  a2.name.toUpperCase() ));
29  context TypeElement ERROR
30      "TypeElement: Names of Tasks must be unique" :
31      te_tasks.forAll(a1 | te_tasks.notExists(a2 | a1 != a2 && a1.name.toUpperCase() ==
32  a2.name.toUpperCase() ));
33  context TypeElement ERROR
34      "TypeElement: Names of States must be unique" :
35      te_states.forAll(a1 | te_states.notExists(a2 | a1 != a2 && a1.name.toUpperCase() ==
36  a2.name.toUpperCase() ));
37  context TypeElement ERROR
38      "TypeElement: must have 1 state or more!" :
39      this.te_states.size != 0;
40  context TypeElement ERROR
41      "TypeElement: must have 1 task or more!" :
42      this.te_tasks.size != 0;
43  context WorkItem ERROR
44      "WorkItem: an Element must be selected" :
45      this.wi_element != null;
46  context WorkItem ERROR
47      "WorkItem: a Task must be selected" :
48      this.wi_task != null;
49  context Element ERROR
50      "Element: must have a Initial State!" :
51      this.initial_state != null;
52  context Element if this.initial_state != null ERROR
53      "Element: Initial state must be from same TypeElement!" :
54      this.eContainer == this.initial_state.eContainer;
55  context State if state_tasks.size!=0 ERROR
56      "State: Tasks must be from same TypeElement" :
57      state_tasks.forAll(a1 | a1.eContainer == this.eContainer);
58  context WorkItem if (wi_element.eContainer != null) && (wi_task.eContainer != null)  ERROR
59      "WorkItem: Element, Task and State must be from same TypeElement!" :
60      (wi_change_state == null &&  (wi_element.eContainer == wi_task.eContainer)) ||
61  (wi_change_state != null &&  (wi_element.eContainer == wi_task.eContainer) &&
62  (wi_task.eContainer == wi_change_state.eContainer));
```

Nombres únicos.

Un tipo de elemento debe tener al menos un estado y una tarea.

Un WorkItem debe tener un elemento y una tarea.

Un elemento debe tener un estado inicial.

No debemos mezclar elementos, tareas y estados de distintos tipos de elementos.

D- PLANTILLA DE TRANSFORMACIÓN

En este apéndice incluimos el fichero *template.xpt*, el cual tiene la plantilla de transformación xpand que será aplicada al modelo validado correctamente.

Las instrucciones del lenguaje van siempre entre los símbolos « y » , el resto de texto se genera tal cual está escrito. Las principales instrucciones son las siguientes:

Statement	Explicación
import	Hace conceptos del metamodelo visibles a la plantilla
Define	Define una nueva plantilla
File	Abre un fichero y dirige la salida hacia él
Expand	Llama a otra plantilla para uno o varios elementos
Foreach	Itera sobre cada elemento
if	Condición
Extensión	Importa un fichero de extensiones que incluye expresiones complejas
Error	Reporta un error
Let	Define una variable temporal
Rem	Comentarios
-	Eliminamos la línea en blanco

```
1  «IMPORT scenario»
2
3  «EXTENSION metamodel::Extensions»                    Importamos las funciones que extienden el metamodelo (AP. E)
4
5  «DEFINE Root FOR ScenarioModel»                      Empezamos por la raíz ScenarioModel
6      «FILE "index.html"-»                             Creamos el fichero index.html, e insertamos el bloque cod_inicial
7          «EXPAND cod_inicial FOR this-»
8  <ul id="home" title="Tasks" selected="true" hideBackButton="true" >
9        <li id="pending_group" class="group">Pending tasks</li>
10       <li id="ret-link"><a href="#pending"></a></li>
11       <div id="p-task"></div>
12         «FOREACH types AS ty-»
13         «REM»
14         CODIGO PARA CADA CADA TAREA PENDIENTE, pantalla inicial
15         «ENDREM»
16             «FOREACH ty.te_elements AS e-»
17                 «FOREACH e.initial_state.state_tasks AS t-»
18                     «IF t.pending==true-»
19     <li id="action-«t.name.replaceAll(' ','')»-«e.name.replaceAll(' ','')»"><a href="#decode"
20 >«t.name» «ty.name» «e.name»</a></li>
21                     «ENDIF-»
22                 «ENDFOREACH-»
23             «ENDFOREACH-»
24         «ENDFOREACH-»
25       <li id="actions_group" class="group">Actions</li>
26         «FOREACH types AS ty-»
27         «REM»
28         CODIGO PARA CADA CADA TAREA, pantalla inicial   Pantalla inicial del usuario
29         «ENDREM»
```

```
30      <li><div class="annotation" >«ty.name»</div></li>
31              «FOREACH ty.te_tasks AS t-»
32                  «IF t.pending!=true-»
33      <li><a href="#decode" onclick="setAction('«t.name.replaceAll(' ','')»');">«t.name»</a></li>
34                  «ENDIF-»
35              «ENDFOREACH-»
36          «ENDFOREACH-»
37  </ul>
38          «REM»
39          CODIGO PARA CADA ELEMENTO
40          «ENDREM»
41          «FOREACH types.te_elements AS e-»
42  <ul id="«e.name.replaceAll(' ','')»" title="Details" >
43      <li><h2>«e.name»</h2> <div class="annotation" id="«e.name.replaceAll(' ','')»-
44  status">«e.initial_state.name»</div></li>
45     <li id="actions-«e.name.replaceAll(' ','')»" class="group">Actions</li>
46          «FOREACH e.initial_state.state_tasks AS t-»
47              «IF t.pending!=true-»
48     <li id="action-«t.name.replaceAll(' ','')»-«e.name.replaceAll(' ','')»"><a
49  href="#«t.name.replaceAll(' ','')»-«e.name.replaceAll(' ','')»">«t.name»</a></li>
50              «ENDIF-»
51          «ENDFOREACH-»
52  </ul>
53          «ENDFOREACH-»
54          «REM»
55          CODIGO PARA CADA WORKITEM
56          «ENDREM»
57          «FOREACH workitems AS w-»
58              «IF w.wi_task.pending-»
59  <form id="«w.wi_task.name.replaceAll(' ','')»-«w.wi_element.name.replaceAll(' ','')»"
60  class="dialog" >
61          <fieldset>
62            <h1>Evento</h1>
63              «IF w.wi_change_state!=null-»
64        <a class="button blueButton"  type="cancel" onclick="remove('action-
65  «w.wi_task.name.replaceAll(' ','')»-«w.wi_element.name.replaceAll('
66  ','')»');«w.onclick()»">Ok</a>
67                  «ELSE-»
68        <a class="button blueButton" type="cancel" onclick="remove('action-
69  «w.wi_task.name.replaceAll(' ','')»-«w.wi_element.name.replaceAll(' ','')»');">Ok</a>
70              «ENDIF-»
71          </fieldset>
72          <div  class="panel">
73           <fieldset>
74             <div class="row">
75            <ul>
76            <li class="linklike"><h2> «w.wi_element.name» </h2></li>
77                  «REM»
78                  CODIGO PARA LA INFO ASOCIADA A LA TAREA
79                  «ENDREM»
80                  «FOREACH w.wi_task.task_info AS i-»
81                   «EXPAND informas FOR i-»
82                  «ENDFOREACH-»
83                  «REM»
84                  CODIGO PARA LA INFO ASOCIADA AL WORKITEM
85                  «ENDREM»
86                  «FOREACH w.wi_info AS i-»
87                   «EXPAND informas FOR i-»
88                  «ENDFOREACH-»
89              </ul>
```

Pantalla inicial del usuario (continuación).
Muestra las tareas pendientes (8-24)
Y las tareas posibles (25-37)

Código generado para cada elemento detectado.
Muestra las tareas disponibles según su estado.

Procesamos los workitems o tareas modeladas

Si la tarea que forma el workitem es una tarea pendiente estamos ante un evento. Generamos la pantalla asociada junto con su información.

```
90      </div>
91      </div>
92  </form>
93              «ELSE-»
94  <ul id="«w.wi_task.name.replaceAll(' ','')»-«w.wi_element.name.replaceAll(' ','')»"
95  title="«w.wi_task.name»" >
96          <li><h2>«w.wi_element.name»</h2></li>
97          <li class="group">«w.wi_task.name»</li>
98              «REM»
99                  CODIGO PARA LA INFO ASOCIADA A LA TAREA
100                 «ENDREM»
101                 «FOREACH w.wi_task.task_info AS i-»
102                  «EXPAND informas FOR i-»
103                 «ENDFOREACH-»
104                 «REM»
105                 CODIGO PARA LA INFO ASOCIADA AL WORKITEM
106                 «ENDREM»
107                 «FOREACH w.wi_info AS i-»
108                  «EXPAND informas FOR i-»
109                 «ENDFOREACH-»
110                 «IF w.wi_change_state!=null-»
111         <li><a href="#«w.wi_element.name.replaceAll(' ','')»" onclick="«w.onclick()»">Ok</a></li>
112                 «ELSE-»
113         <li><a href="#«w.wi_element.name.replaceAll(' ','')»">Ok</a></li>
114                 «ENDIF-»
115 </ul>
116             «ENDIF-»
117         «ENDFOREACH-»
118         «REM»
119         CODIGO FINAL DE INDEX.HTML
120         «ENDREM»
121         «EXPAND cod_final FOR this-»
122     «ENDFILE-»
123
124     «FILE "control.html"-»
125         «EXPAND control FOR this-»
126     «ENDFILE-»
127
128 «ENDDEFINE»
129
130 «DEFINE informas FOR scenario::InfoGroup-»
131                 «FOREACH this.group AS i-»
132             <ul><p style="font-size: 20px;font-weight: bold;"> «this.name» </p>
133                     «EXPAND informas FOR i-»
134             </ul>
135                 «ENDFOREACH-»
136 «ENDDEFINE»
137
138 «DEFINE informas FOR scenario::InfoLine-»
139             <li style="list-style: none"><p> «this.text» </p></li>
140 «ENDDEFINE»
141
142 «DEFINE informas FOR scenario::Info-»
143 «ENDDEFINE»
144
145 «DEFINE cod_inicial FOR scenario::ScenarioModel-»
146 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
147         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
148
149 <html xmlns="http://www.w3.org/1999/xhtml">
```

Si la tarea que forma el workitem no es pendiente, generamos la pantalla asociada, que se mostrará al usuario si este selecciona la tarea.

Insertamos el bloque cod_final
Cerramos el fichero index.html

Creamos el fichero control.html
Insertamos el bloque cod_final
Cerramos el fichero control.html

De aquí obtenemos la información asociada a una tarea y a un workitem.

```
<head>
<title>Smart Library</title>
<meta name="viewport" content="width=320; initial-scale=1.0; maximum-scale=1.0; user-
scalable=0;"/>
<script type="text/javascript" src="mootools/mootools-core.js"></script>
<script type="text/javascript" src="mootools/mootools-more.js"></script>
<style type="text/css" media="screen">@import url("iui/iui.css");</style>
<style type="text/css" media="screen">@import url("style/style.css");</style>
<script type="text/javascript" src="iui/iui.js"></script>

<script type="text/javascript">
var action=null;
var pending = false;

function setAction(act){
    action=act;
}

function setBack(bk){
$('backButton').href=bk;
$('backButton').set('html','Tasks');

}


function decode_response(responseText, responseXML){
try{
if(iui.getSelectedPage().id=="decode" || iui.getSelectedPage().id=="home"){
        if(responseText!="null" ){
        if(action!=null){
            iui.showPageById(action+'-'+responseText);
        }else{
            //evento asociado a tarea pendiente
            // si existe id =pending+'-'+responseText
            //var page=$('pending-'+responseText);
            //if (page!=null)
              //  {
                    iui.showPageById(responseText);
                //}
        }
        }
    }
}catch(err){}
}

var periodical=new Request({
    method: 'get',
     url: '/decode', onSuccess: decode_response,
    initialDelay: 1000,
    delay: 300,
    limit: 3000
});


function init(){

$('decode').addEvent('show', function(){periodical.startTimer();});
$('decode').addEvent('hide',function(){periodical.stopTimer();});

$('home').addEvent('show', function(){
```

```
210     setAction(null);
211     setBack("#");
212     periodical.startTimer();
213 });
214 $('home').addEvent('hide',function(){periodical.stopTimer();});
215
216     «FOREACH types.te_elements AS e-»
217 $('«e.name.replaceAll(' ','')»').addEvent('show', function(){
218     setBack("#home");
219 });
220     «ENDFOREACH-»
221
222 periodical.startTimer();
223 }
224
225
226 //function set_decode_title(t){
227 //$('decode_title').set('html', t);
228 //}
229
230
231 //function remove_book(obj){
232 // $(obj).destroy();
233 //}
234
235
236 function change_state(elemento, mensaje){
237 $(elemento+'-status').set('html',mensaje);
238 }
239
240 function remove(obj){
241 $(obj).destroy();
242 }
243
244 function add_element(idt,name,state) {
245 var s='<li><h2>'+name+'</h2> <div class="annotation" id="'+idt+'-status">'+state+'</div></li><li
246 id="actions-'+idt+'" class="group">Actions</li>';
247 var e=new Element('ul',{'id':idt,'title':'Details'});
248 e.injectBefore($('decode'));
249 e.set('html',s);
250 iui.showPageById(idt);
251 }
252
253 function add(idt,mensaje,obj,href)
254 {
255 //var s='<a href="#pending-'+idt+'" onclick="setAction("pending);">'+mensaje+'</a>';
256 var s='<a href="#'+ href + '" >'+mensaje+'</a>';
257 //"#decode"
258 var e=Element('li', {id:idt,  html:s});
259 e.inject($(obj),"after");
260 }
261
262 </script>
263
264 <!--
265 <script type="application/x-javascript" src="http://10.0.1.2:1840/ibug.js"></script>
266 -->
267 </head>
268
269 <body onload="init();" onclick="console.log('Hello', event.target);">
```

```
270
271     <!-- TOOLBAR:                        -->
272     <div class="toolbar">
273         <h1 id="pageTitle"></h1>
274         <a id="backButton" class="button" href="#"></a>
275         <div id="command">
276             <a class="button" href="#decode">sense</a>
277         </div>
278     </div>
279 «ENDDEFINE»
280
281 «DEFINE cod_final FOR scenario::ScenarioModel-»
282  <div id="decode" class="panel" title="Decode">
283         <h2><div id="decode_title">Decode</div></h2>
284         <p><div id="decode_subtitle"> </div></p>
285     <fieldset>
286       <div class="row" >
287       <div id="camera_container">
288
289       <p>
290       Sensing...
291       </p>
292     </div>
293     </div>
294     </fieldset>
295     </div>
296
297      <ul id="error" title="Error" >
298         <li><h2>Error</h2></li>
299         <li>You did not correctly followed the instructions.</li>
300     </ul>
301
302 </body>
303 </html>
304 «ENDDEFINE»
305
306 «DEFINE control FOR scenario::ScenarioModel-»
307 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
308         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
309
310 <html xmlns="http://www.w3.org/1999/xhtml">
311 <head>
312 <title>Lend Resource</title>
313 <meta name="viewport" content="width=320; initial-scale=1.0; maximum-scale=1.0; user-
314 scalable=0;"/>
315 <script type="text/javascript" src="/mootools/mootools-core.js"></script>
316 <script type="text/javascript" src="/mootools/mootools-more.js"></script>
317 <style type="text/css" media="screen">@import url("/iui/iui.css");</style>
318 <script type="application/x-javascript" src="/iui/iui.js"></script>
319
320 </head>
321
322 <body>
323
324 <!-- TOOLBAR:                        -->
325 <div class="toolbar">
326     <h1 id="pageTitle"></h1>
327         <a id="backButton" class="button" href="#"></a>
328         <a class="button" href="#searchForm">Search</a>
329 </div>
```

```
<ul id="home" title="Identifiers" selected="true">
    «FOREACH types.te_elements AS e-»
     <li><a href="/set/«e.name.replaceAll(' ','')»">«((TypeElement) e.eContainer).name»-
«e.name»</a></li>
    «ENDFOREACH»
    «FOREACH workitems AS w-»
        «IF w.wi_task.pending==true-»
     <li><a href="/set/«w.wi_task.name.replaceAll(' ','')»-«w.wi_element.name.replaceAll('
','')»">«((TypeElement) w.wi_element.eContainer).name»-«w.wi_task.name»
«w.wi_element.name»</a></li>
        «ENDIF-»
    «ENDFOREACH-»
    <li><a href="/set/null">Cancel</a></li>
</ul>

</body>
</html>
«ENDDEFINE»
```

Creamos los enlaces para introducir la detección de elementos físicos y de eventos

E- FUNCIONES DE APOYO

En este apéndice vemos el fichero de extensiones usado en nuestra propuesta, el cual está escrito mediante el lenguaje Xtend.

Xtend es un lenguaje que nos permite definer librerías de operaciones y extensiones para acceder al metamodelo basadas en métodos de Java o en expresiones OAW. Estas librerías pueden ser usadas desde los otros lenguajes de la plataforma OAW, xpand y check. La sintaxis básica de una extensión es:

ReturnType extensionName(ParamType1 paramName1, ParamType2...): expression-using-params;

```
1  import scenario;
2
3  String onclick(WorkItem w):"remove('"+w.wi_element.name.replaceAll(' ','')+"');add_element('"+w.wi_element.name.replaceAll('
4  ','')+"','"+w.wi_element.name+"','"+w.wi_change_state.name+"');"+actions(w.wi_change_state.state_tasks,w.wi_element);
5
6
7
8  String actions(List[Task] l,Element e) :
9   if l == null || l.isEmpty then ""
10     else if l.size == 1 then
11         ( l.last().pending ? "add('action-"+l.last().name+"-"+e.name.replaceAll(' ','')+"','"+l.last().name+" "+e.name+"','pending_group','decode');" : "add('action-
12 "+l.last().name+'-'+e.name.replaceAll(' ','')+"','"+l.last().name+"','actions-"+e.name.replaceAll(' ','')+"','"+l.last().name+'-'+e.name.replaceAll(' ','')+"');")
13     else
14         (l.last().pending ? "add('action-"+l.last().name+"-"+e.name.replaceAll(' ','')+"','"+l.last().name+" "+e.name+"','pending_group','decode');" + actions(l.reject(el |
15 l.last() == el),e) : "add('action-"+l.last().name+'-'+e.name.replaceAll(' ','')+"','"+l.last().name+"','actions-"+e.name.replaceAll(' ','')+"','"+l.last().name+'-
16 '+e.name.replaceAll(' ','')+"');"+actions(l.reject(el | l.last() == el),e));
```

Devuelve las llamadas a funciones javascript definidas en index.html: *remove(); add_element()*, que borran un elemento (su pantalla) y lo vuelven a crear, junto con el resultado de la función *actions*

Recibe como parámetro una lista de tareas y un elemento.
Devuelve llamadas a funciones javascript definidas en index.html: *add()*, que añaden dentro de la pantalla asociada al elemento, sus tareas asociadas a su estado actual. Si alguna es una tarea pendiente la función la añadirá en la pantalla inicial del usuario

F- Instalar y configurar el entorno Eclipse del presente desarrollo.

Los pasos a seguir para instalar Eclipse junto con todas las herramientas necesarias para realizar este trabajo son los siguientes:

1. Descargar e instalar el **JRE** (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
2. Descargar e instalar **Eclipse Galileo** (http://www.eclipse.org/downloads/packages/eclipse-modeling-tools-includes-incubating-components/galileosr2)
3. OAW
    a. Descargar primero http://archive.eclipse.org/tools/orbit/downloads/drops/R200706192011/bundles/org.apache.log4j_1.2.13.v200706111418.jar y copiarlo a la carpeta plugins. Reiniciar Elipse.
    b. En Eclipse->Help->Install new software->Add Site
    c. http://www.openarchitectureware.org/updatesite/milestone/site.xml
    d. Instalar OpenArchitectureware
4. Epsilon:
    a. En Eclipse->Help->Install new software->Add Site
    b. http://download.eclipse.org/modeling/gmt/epsilon/updates/
    c. Instalar Epsilon

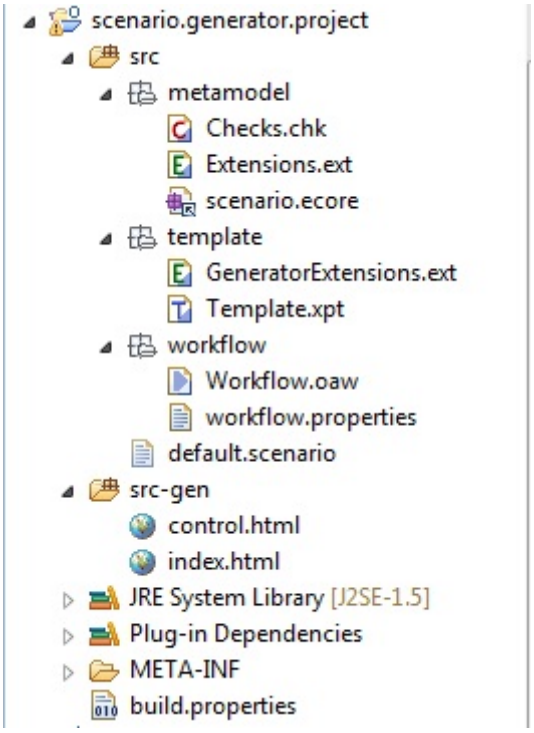
G. Diseño de la infraestructura para generar los prototipos.

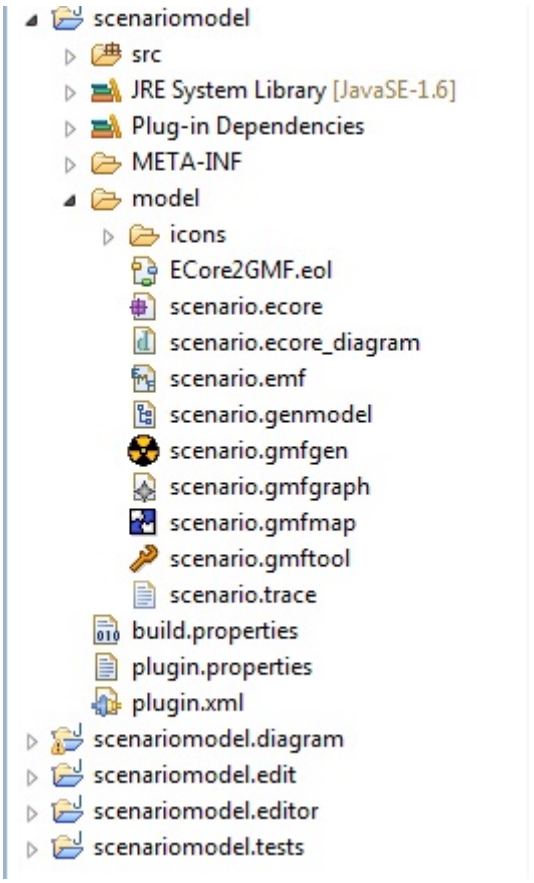
Por último vamos a hacer un pequeño resumen de los pasos que tenemos que ir siguiendo para obtener los prototipos. En primer lugar debemos tener el entorno de trabajo tal y como se explica en el apéndice F. Los pasos a seguir serían los siguientes:

1- Creamos un proyecto GMF. Lo llamamos *scenariomodel*.
   a. Marcamos la opción “Show dashboard for the created project”
2- Dentro de *scenariomodel*:
   a. Creamos un fichero de tipo “Ecore Diagram” y dibujamos el metamodelo del apéndice A. Le llamaremos scenario.ecore.
   b. A partir del fichero *scenario.ecore* generamos el fichero *scenario.emf*, este fichero contiene el metamodelo en formato Emfatic. Añadimos las anotaciones Eugenia necesarias para obtener el editor. Generamos ahora *scenario.ecore* a partir de *scenario.emf*.
   c. Creamos el fichero *ECore2GMF.eol* con los cambios que se aplicarán al editor.
   d. Desde el menú de Eugenia generamos los modelos *GMF tool (scenario.gmftool), graph (scenario.gmfgraph) and map (scenario.gmfmap)*, necesarios para generar el editor.
   e. Desde el Dashboard generamos el *genmodel*.
   f. A partir de *scenario.genmodel* generamos los proyectos *scenariomodel.edit, scenariomodel.editor* y *scenariomodel.tests*, así como el código asociado al metamodelo.
   g. A partir del modelo *scenario.gmfmap* del paso 2.d generamos el modelo *GMF generator (scenario.gmfgen).*
   h. Desde el menú de Eugenia sincronizamos *scenario.gmfgen* con *scenario.genmodel*.
   i. Modificamos las funciones *getText()* del proyecto scenariomodel.edit necesarias para que el editor nos muestre el nombre de la clase TypeElement.
   j. A partir de scenario.gmfgen generamos el plugin del editor y obtenemos el proyecto *scenario.diagram*.
3- Modificamos las clases java necesarias del proyecto *scenario.diagram*, para que en el editor aparezcan las etiquetas que nos interesan (figura 5.18).

4- Abrimos una nueva instancia de Eclipse.
    a. Dentro de un proyecto nuevo, creamos un fichero del tipo *Scenario Diagram*, que corresponde al plugin del editor que acabamos de crear.
    b. Dibujamos el escenario, obteniendo como resultado el fichero *default.scenario*. en este fichero está la instancia del metamodelo que acabamos de dibujar, con el escenario a partir del cual vamos a generar los prototipos.
5- Volvemos a la primera instancia de Eclipse y creamos un proyecto openArchitectureWare, y lo llamamos *scenario.generator.project*.
6- Dentro del proyecto *scenario.generator.project* :
    a. Añadimos el paquete *scenario* como dependencia.
    b. Copiamos el fichero *scenario.ecore*.
    c. Copiamos el fichero *default.scenario* obtenido con el editor.
    d. Creamos los ficheros:
        i. Checks.chk: con las reglas de validación.
        ii. Extensions.ext: con las funciones que extienden el metamodelo.
        iii. Template.xpt: con la plantilla de transformaciones.
        iv. Workflow.oaw y workflow.properties: para regular todo el proceso de validación y generación.
    e. Lanzamos el workflow.
    f. Si *default.scenario* se valida correctamente obtenemos en *src-gen* los ficheros con los prototipos.

En la siguiente figura podemos ver todos los proyectos que hemos generado:


scenario.generator.project
src
metamodel
Checks.chk
Extensions.ext
scenario.ecore
template
GeneratorExtensions.ext
Template.xpt
workflow
Workflow.oaw
workflow.properties
default.scenario
src-gen
control.html
index.html
JRE System Library [J2SE-1.5]
Plug-in Dependencies
META-INF
build.properties


scenariomodel
src
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
META-INF
model
icons
ECore2GMF.eol
scenario.ecore
scenario.ecore_diagram
scenario.emf
scenario.genmodel
scenario.gmfgen
scenario.gmfgraph
scenario.gmfmap
scenario.gmftool
scenario.trace
build.properties
plugin.properties
plugin.xml
scenariomodel.diagram
scenariomodel.edit
scenariomodel.editor
scenariomodel.tests