UNIVERSIDAD INTERNACIONAL DE LA RIOJA

TRABAJO FINAL
MASTER EN DIRECCIÓN E INGENIERÍA DE SITIOS WEB

Modelización de un formulario web colaborativo bajo un enfoque basado en procesos

Autor: **Raúl Castro Cerdeira**
Directora: **B. Cristina Pelayo García-Bustelo**

Madrid 2012

Resumen

Un formulario colaborativo puede ser representado mediante un grafo. Sus nodos establecerán los estados por los que pasan los registros creados. En la práctica, equivalen a una acción consistente en completar unos determinados campos del formulario. Esta tarea únicamente puede ser llevada a cabo por los usuarios que estén autorizados a ello. Por otra parte, los arcos señalan las transiciones posibles entre los diversos estados.

Según esta perspectiva, sería deseable poder disponer de una herramienta a través de la que se pueda definir la información propia de cada implementación de los formularios. Estos datos conformarían un modelo, es decir, una versión simplificada del sistema final. Tras la construcción de las partes comunes de la aplicación deseada, únicamente habría que generar las piezas dependientes de la especificación particular del formulario. Por último, se tendrían que ensamblar todos los componentes para lograr el resultado deseado.

La principal aportación de MDA consiste en la consecución secuencial de modelos, con distintos niveles de detalle. En última instancia, se llevaría a cabo una transformación final, que generará de manera automática el código ejecutable. Gracias a ello, no es necesario disponer de conocimiento sobre el proceso de implementación y codificación de la aplicación. Esta filosofía facilita la portabilidad, mantenimiento y documentación del software resultante.

Respecto al modelado de procesos, es importante ya que permite al diseñador de los mismos disponer a una visión global, para detectar aspectos mejorables. En algunos casos, pueden requerir una modificación del diseño. Con la incorporación de MDA, estos cambios se traducen en la alteración de los modelos iniciales. Finalmente, se producirá una versión mejorada del sistema.

Palabras Clave

Formulario colaborativo, modelo, proceso, flujo de trabajo, grafo, MDA, BPMN, XPDL, CIM, PIM, PSM, generación de código automática, gestión de procesos, modelado de procesos, abstracción.

Abstract

A collaborative form can be represented as a graph. Its nodes define the statuses through the created records pass. They also determine an action, which consists in the population of some particular fields in the form. This task can only be realized by the authorized users. On the other hand, the arcs in the graph show the possible transitions between the different statuses.

This perspective suggests that a tool to specify each form's implementation own information is needed. This data integrate a model. This is a simplified version of the final system. After the construction of common parts, it is only necessary to generate code sections then depend on the form's particular specification. The last step includes the assembly of the different elements.

MDA's main contribution is to provide different models that are sequentially provided. Each one gets a concrete detail level. The last transformation produces the final code, which is generated automatically. This way, knowledge about the process of the application's implementation and codification is not needed. This technique helps to make the generated software portable and easy to maintain and document.

Process modelling is important because it allows the designer to get a global view, so he or she can detect aspects that could work better. Even, these problems may require changes in the process design. MDA allows these modifications to be made directly in the models. Finally, a better version of the system will be created.

Keywords

Collaborative form, model, process, workflow, graph, MDA, BPMN, XPDL, CIM, PIM, PSM, automatic code generation, process management, process modelling, abstraction.

Agradecimientos

A Lore y a mis padres

Quiero transmitir mi agradecimiento a Cristina Pelayo, por su ayuda incondicional. Los ánimos nunca faltaron cuando me encontraba en horas bajas.

También deseo agradecer a mi familia sus ánimos en cuanto les comuniqué mi deseo de realizar este Máster, así como su apoyo en los buenos y los malos momentos.

Finalmente, a Lore, por estar siempre al pie del cañón.

Tabla de Contenidos

Tabla de Figuras

Tabla de Códigos

Índice de Tablas

PARTE I. Introducción a la Investigación

Capítulo 1. Introducción

Este primer capítulo tiene por finalidad principal familiarizar al lector con el problema cuyo estudio y resolución ocupa este trabajo. Se proporciona para ello una aproximación a la problemática, aportando ejemplos concretos que ayudarán a comprender las situaciones que se pretende solucionar. Esta contextualización se completa con la definición formal del tema tratado. Ésta se compone del establecimiento de la hipótesis que se desea demostrar, así como de los objetivos que se esperan alcanzar a través de la solución propuesta.

Posteriormente, se define el método científico seguido durante el desarrollo del trabajo, y que es el pilar fundamental para conferirle su carácter investigativo. Está caracterizado por una secuencia de fases, así como por unas actividades e hitos fruto de su realización.

El último apartado desgrana el índice que estructura este trabajo, señalando los bloques principales de la memoria y sintetizando los puntos más importantes de cada uno de ellos.

1.1 PLANTEAMIENTO Y JUSTIFICACIÓN DEL TRABAJO

En el mundo de las aplicaciones web, muchas de ellas disponen de formularios concebidos para la gestión colaborativa de la información. Su mantenimiento suele requerir la participación de distintas personas, que pueden actuar sobre los datos en paralelo o de manera secuencial. Desde esta perspectiva, surgen flujos de trabajo sobre dichos formularios. Por ejemplo, los casos siguientes hacen referencia a la aparición de este punto de vista en situaciones diversas.

- Herramientas de edición colaborativa de contenidos.

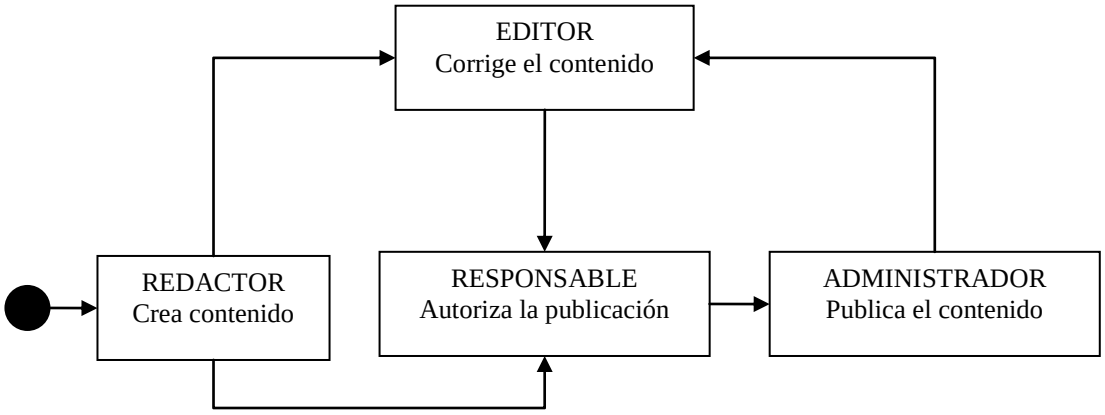


Figura 1.1. Flujo de trabajo de ejemplo en una herramienta colaborativa

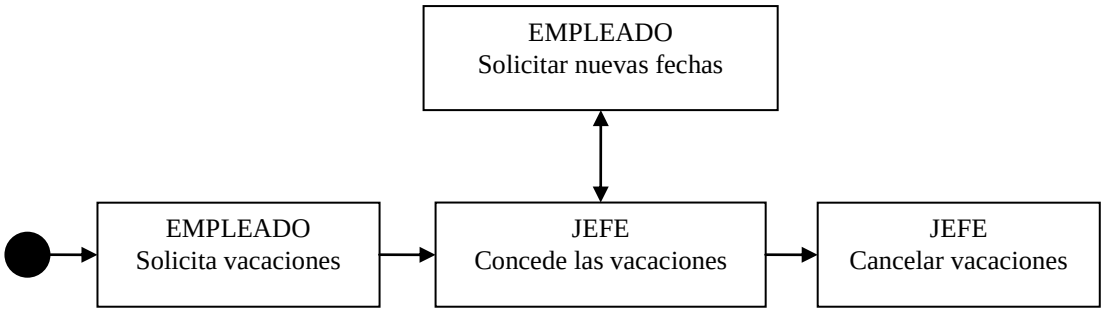


Figura 1.2. Flujo de trabajo de ejemplo en una herramienta de gestión de vacaciones

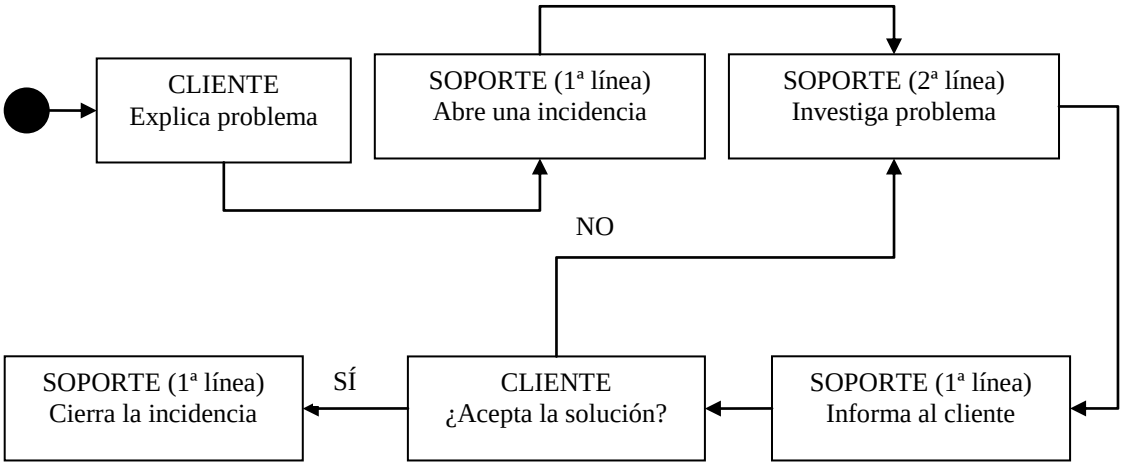


Figura 1.3. Flujo de trabajo de ejemplo en una herramienta de gestión de incidencias

A pesar de tratarse de contextos muy dispares, está claro que todos ellos están sustentados sobre una misma concepción: el mantenimiento colaborativo de la información de manera orquestada. Para su representación se puede emplear, como en los ejemplos anteriores, un grafo donde cada nodo refleja una acción (rellenar unos campos del formulario), a realizar por una persona (una en concreto, o una cualquiera perteneciente a un grupo definido). La condición que se requiere para la ejecución de esas actividades es que sólo puede ser llevada a cabo cuando las anteriores fases de la secuencia ya han sido acometidas. La manera en que las tareas están encadenadas es impuesta por la disposición de las transiciones. En conjunto, todo ello no deja de ser una modelización de un proceso.

En síntesis, el objetivo de este trabajo es la modelización de formularios web. La disponibilidad de los mismos para la edición de la información que albergan depende del propio estado de la información y del usuario que pretende acceder al mismo. Por su parte, cada fase concreta tendrá asociados unos campos particulares del formulario. El resto deben aparecer inactivos, esto es, se puede leer su contenido pero no alterarlo.

1.2 HIPÓTESIS Y OBJETIVOS

Se plantea la siguiente hipótesis de partida para este trabajo:

La especificación MDA unida al modelado de procesos permite crear de manera simple un formulario web gestionado mediante un workflow.

Esta investigación se centra por tanto en la manera de emplear las herramientas de modelización de procesos disponibles, para representar el ciclo de vida de un determinado conjunto de datos. Éstos serán administrados a través de un formulario que debe respetar ese workflow. Esto implica que debe restringir en cada fase los campos cuyo contenido se puede alterar, así como los participantes que pueden llevar a cabo esta acción.

El objetivo principal se puede desgranar en los siguientes logros parciales.

- *Estudio y valoración de las principales teorías relacionadas con el desarrollo dirigido por modelos y el modelado de procesos.*

  También se debe alcanzar una comprensión global de las tecnologías disponibles en estos campos, incluyendo patrones y frameworks que se podrían aplicar para alcanzar la meta.

- *Definición de una metodología para el desarrollo de formularios web mediante la modelización de su workflow subyacente.*

  Aparte de los flujos posibles, debe aceptar el modelado de los campos que componen cada formulario. Cada paso del proceso podrá ser llevado a cabo por un usuario, que estará en disposición de modificar aquellos campos que se especifique.

  Como requerimiento inicial, el resultado estará formado por una serie de ficheros escritos en el lenguaje de programación elegido. Cuando se ejecuten, aparte de llevar a cabo la lógica del proceso, producirán el código HTML de las páginas que componen el sitio. Este paquete debe ser completamente funcional. Respecto a la base de datos, se proveerán el conjunto de órdenes necesarias para su creación en cualquier servidor. De esta manera, se garantiza la portabilidad del producto final, que se podrá desplegar en toda ubicación que aporte los elementos necesarios para su puesta en servicio.

- *Se deben proveer dos sistemas anexos para la notificación de acciones en espera para la consecución de los procesos.*

  Concretamente, deben trasladarse estos avisos a través del correo electrónico y de un portal de usuarios. Sus destinatarios serán los usuarios encargados de ellas. La cuenta de envío empleada en el primer elemento debe ser configurable a través del modelo.

- *Crear un prototipo que implemente la metodología.*
- *Proporcionar una aplicación generada mediante la metodología establecida*.

  Para ello se hará uso del prototipo diseñado.

1.3 METODOLOGÍA SEGUIDA DURANTE LA INVESTIGACIÓN

Para la realización de este trabajo se ha seguido un método científico basado en las siguientes fases:

1. Planteamiento del problema.

En esta primera fase se realiza la definición del ámbito específico sobre el que se desea investigar. Concretamente, en este punto se debe establecer la hipótesis de partida y los objetivos primordiales que se espera lograr con el desarrollo de la investigación.

Esta decisión no suele ser inmediata, como ocurrió en este caso. Realmente, ha derivado de un primer enfoque difuso, dirigido hacia la búsqueda de una manera de combinar el desarrollo dirigido por modelos y la modelización de procesos.

2. Recopilación y estudio de información.

Establecido el objetivo final, se debe generar una biblioteca de documentación relacionada con la temática del que vertebra el trabajo. Una correcta organización de esta información es esencial para analizar adecuadamente su contenido y retener los conocimientos fundamentales, útiles para el desarrollo de la investigación. De esta manera, también se facilita la consulta futura de los contenidos compilados.

Obviamente, ésta no es una fase que ocurra durante un periodo concreto en los primeros tiempos de vida del proyecto. Esto es porque se debe prestar una atención continua a posibles nuevas aportaciones surgidas en el contexto investigado. Además, es muy posible que a medida que la investigación avance, se requiera el apoyo de nuevos textos que ayuden a resolver los conflictos que emerjan a lo largo de este camino.

Cabe señalar la relevancia que se debe conceder a la comprobación de las fuentes que se emplee en el transcurso de la investigación. Es importante también obtener literatura que incida sobre un tema ya analizado con anterioridad. Así, se pueden conocer puntos de vista alternativos que enriquezcan los conceptos aprehendidos. Los documentos empleados en este trabajo se han escogido respetando estas premisas.

3. Documentación de la base teórica.

La meta de este paso radica en la generación de una documentación que establezca una perspectiva amplia de aquellos aspectos relacionados con el tema tratado. En este caso, los ámbitos primordiales son el desarrollo dirigido por modelos y la modelización de procesos.

Es de vital importancia generar una base teórica de calidad. Con ella, la investigación dispondrá de unos buenos cimientos, básicos para alcanzar unas conclusiones irrefutables.

4. Análisis de trabajos relacionados con la investigación.

Habitualmente, un campo de estudio hacia el que un investigador se ha orientado no es un espacio inexplorado. Otras personas han tenido ya la misma idea o similar. Por tanto, este paso se debe aprovechar para comprobar los últimos descubrimientos en la materia. Especialmente importante es contrastar estos aportes con los objetivos establecidos, analizando las ventajas y los inconvenientes de los desarrollos existentes.

Es de esperar que las alternativas descubiertas implementen una propuesta muy similar a la idea concebida inicialmente. Sin embargo, ante este panorama, el investigador debe mostrar un carácter proactivo, adquiriendo nuevas ideas cuya aplicación aumentará la calidad de la investigación.

En conclusión, se establece una relación de simbiosis con las investigaciones que se encuentren. El investigador, mediante una postura analítica, descubrirá cómo la aportación del nuevo trabajo puede enriquecer los descubrimientos existentes, mientras aquél se nutre de las buenas ideas que ya se han aplicado con éxito.

5. Desarrollo de la investigación.

Todo el material acumulado en las fases anteriores (ideas primigenias, conocimientos, análisis proactivos, etc.) se empleará para el avance de la investigación. Con ello, se redactará una serie de capítulos donde se explique el proceso seguido.

Por tanto, la esencia de este paso incide sobre la explicación de la manera de relacionar los conocimientos recogidos, con el fin de lograr el objetivo.

6. Construcción de un prototipo.

Este prototipo es fundamental para demostrar la posibilidad de implementar soluciones al problema inicial, que respeten los requerimientos establecidos durante la investigación. Como es natural, no pretende proporcionar una solución global al problema de partida. Por ello, sus limitaciones deben quedar adecuadamente establecidas.

Se trata pues de un elemento tangible que demuestra la corrección de la investigación. Su uso permitirá a otros investigadores reproducir los experimentos realizados, además de servir como base, junto a esta memoria, para sus propias aportaciones sobre el tema.

7. Conclusiones.

Es la fase final del desarrollo de la investigación. Se trata de la exposición de los resultados extraídos de la misma, contrastados con la hipótesis y los objetivos iniciales.

Además, se debe aprovechar esta última fase para reflexionar sobre posibles ampliaciones u otros puntos de vista inexplorados. Nuevos investigadores podrán retomar la temática central recuperando uno de estos esbozos. Incluso, la

existencia de estos hilos conductores demostrará la inquietud y el interés que empujaron al autor a sopesar distintos escenarios en que ubicar su propio trabajo.

1.4 ORGANIZACIÓN DEL TRABAJO DE FIN DE MÁSTER

La memoria de este trabajo se organiza según una secuencia de capítulos que refleja el proceso de investigación descrito en la sección anterior.

1. Introducción a la investigación.

Esta primera sección se empleará para asentar el planteamiento del problema. Como introducción, se presentará una contextualización que servirá como primera aproximación al campo investigado. Seguidamente, se justificarán las circunstancias que han motivado este trabajo. Este razonamiento demostrará la necesidad de la investigación sobre la temática elegida, además de destacar las aportaciones derivadas del desarrollo de la misma.

La segunda parte está compuesta por una visión del planteamiento del problema desde un punto de vista formal. Para ello, se incluyen la hipótesis inicial y los objetivos que se espera lograr.

Finalmente, se hace referencia a la metodología científica seguida en la realización del trabajo, aparte de la organización de la memoria, plasmada en el presente capítulo.

2. Estado del arte.

El objetivo principal de esta parte es documentar toda la base teórica que sustentará la investigación.

Primeramente, se hace referencia al desarrollo dirigido por modelos. Después se presenta la especificación MDA, como una implementación real del concepto anterior. En este capítulo también se hace especial énfasis en las ventajas que aporta esta técnica. Para concluir, se aporta una relación de lenguajes disponibles.

El siguiente capítulo versa sobre la modelización de procesos. Como introducción, se incluye teoría básica acerca de los procesos empresariales y los flujos de trabajo (workflows). Finalmente, se añade una recopilación de los lenguajes destinados a la representación de procesos, en general, y de uno especialmente interesante por su orientación hacia los procesos web.

El episodio que continúa la serie consta de una reseña de aquellos patrones y frameworks que pudieran ser de utilidad en el transcurso de la investigación, concretamente en el diseño de la solución. La selección de los que cumplían esta característica se ha hecho en base a un conjunto de razonamientos. En primer lugar, muchos de los lenguajes referenciados en capítulos anteriores están basados en XML. Debido a ello, es prácticamente imprescindible el uso de una herramienta que ayude a interpretar la información contenida en este tipo de documentos. Por otra parte, el resultado del procesamiento del modelo, será un formulario ubicado en un sitio web. Por este motivo, es importante conocer buenas prácticas e implementaciones que apoyen la creación de sitios web

interactivos, así como el acceso a una base de datos desde las propias aplicaciones web.

En última instancia, se analizarán algunas soluciones ya existentes para el problema planteado. El objetivo de este capítulo es discernir si estas herramientas alcanzan los objetivos esperados en este trabajo, de qué manera y cómo se podrían mejorar contrastando sus inconvenientes. Las ventajas de su uso también serán objeto de estudio, pues pueden proveer ideas aplicables a la solución diseñada.

3. Desarrollo de la investigación.

Esta sección de la memoria está dedicada al proceso de investigación seguido para la obtención de la solución al problema planteado.

En primer lugar, debe analizarse las equivalencias entre los lenguajes de modelado de procesos. La idea es, a partir de un modelo construido en un lenguaje gráfico, generar un documento basado en XML. Éste se empleará como base para la construcción del formulario web, que se alcanzará como producto de una sucesión de transformaciones. De esta manera, se aunará el desarrollo dirigido por modelos con la modelización de formularios web mediante procesos.

Será muy importante discernir si el lenguaje gráfico elegido dispone de la notación suficiente para representar toda la información que se considere necesaria. Si no es así, podrán tomarse dos caminos. El primero consiste en añadir nuevos significados a los símbolos disponibles, que serán aceptados en el contexto de este trabajo. La otra opción es diseñar una herramienta auxiliar que acepte la definición de estos datos adicionales de una manera simple para el usuario.

Otro aspecto relevante es el sistema que se aplicará en el modelo para representar las notificaciones a los participantes en los flujos de trabajo. A priori, estos avisos deben tener lugar a través de dos medios, el correo electrónico y un portal del usuario. En ambos casos, además de contener el texto adecuado, se debe proveer una manera de acceder al formulario que corresponda.

Por último, es necesario realizar un diseño del prototipo capaz de solventar los objetivos requeridos. Este elemento se empleará para la elaboración de un formulario web basado en un proceso, a partir de su modelo.

4. Conclusiones.

Esta sección se destina en primer término a la verificación de los objetivos impuestos, contrastados frente a los resultados derivados de la investigación. La finalidad de esta comparación es dejar patentes los logros de las metas propuestas. Así, quedará demostrado el cumplimiento de la hipótesis de partida.

Seguidamente, se proporcionará un resumen del modelo propuesto y se incluirán los artículos derivados de este trabajo.

Para finalizar, se incorpora un episodio dedicado a sugerencias para posibles investigaciones que se puedan derivar de esta aportación. Éstas surgen de alternativas analizadas con menos detalle o directamente inexploradas, y posibles ampliaciones.

5. Bibliografía y referencias web.

En este apartado se detallan los documentos empleados durante la investigación, junto con la abreviatura empleada para las citas.

Estos se clasifican informaciones impresas (principalmente artículos, libros y especificaciones), y referencias de sitios web, junto a los que se especifica la fecha de la última consulta realizada.

6. Anexos.

La documentación que por sus características no tiene una cabida clara en otros capítulos, debe ser ubicada en esta sección.

Comprende, por una parte, informaciones no imprescindibles para la comprensión de la investigación y sus resultados. Sin embargo, su inclusión permite al lector disponer de un mayor nivel de detalle. Por este motivo, se incluye el código fuente del prototipo, así como de los componentes producto de su utilización.

Asimismo, dentro de los anexos se aprovechará para demostrar la calidad de las fuentes empleadas, además de adjuntar un artículo derivado de este trabajo. Aunque por sí solos no son suficientes, la existencia de estos dos documentos es determinante a la hora de establecer la validez de la investigación.

PARTE II. Estado del Arte

Capítulo 2. Modelos de software

La ingeniería dirigida por modelos (MDE, Model-Driven Engineering), es una práctica basada en la simplificación de sistemas informáticos a través de componentes capaces de abstraer al desarrollador de los aspectos más complejos de los mismos. Estos componentes son los modelos. Han sido comunes en el mundo de la Ingeniería del Software, desde el manejo de software indirecto a través de los servicios proporcionados por un sistema operativo, hasta los modelos capaces de generar código funcional con poco esfuerzo del programador. Gracias a esta técnica se reduce el tiempo de desarrollo y los errores insertados durante la codificación.

MDA (Model-Driven Architecture) es un ejemplo de aplicación de MDE. Destaca en primer lugar por la capacidad para representar un problema, independientemente de las tecnologías empleadas posteriormente en su resolución. Esta propiedad conlleva la aparición de herramientas que transforman, tras la ejecución de diversas fases, la información portada por el modelo en código funcional. Finalmente, el uso de estándares abiertos ha facilitado su difusión.

Este capítulo profundiza en estas concepciones, haciendo especial hincapié sobre las ventajas debidas a la aplicación de MDA, sus conceptos principales y los lenguajes que se pueden emplear en su implantación.

2.1 MODEL DRIVEN ENGINEERING (MDE)

Este apartado, resumen de [SCHM06], explica las motivaciones de las tecnologías MDE.

2.1.1 Introducción

En los principios de la computación, investigadores y desarrolladores de software hicieron un esfuerzo por crear abstracciones. Éstas representaban a los distintos elementos de un sistema informático. El objetivo de su existencia era ocultar sus complejidades tras una capa, que era la que verdaderamente realizaba las operaciones dificultosas. Por ejemplo, los primeros lenguajes de programación (ensamblador o Fortran), se concibieron para evitar al programador el uso de código máquina. Por su parte, los sistemas operativos primitivos (OS/360 o UNIX) proveían herramientas en que era posible delegar el manejo de los dispositivos hardware.

Estos avances supusieron la introducción del término abstracción. Este paradigma fue empleado inicialmente para representar objetos relacionados con la computación. Estaban generadas en particular para el espacio de la solución, es decir, propias de las herramientas tecnológicas empleadas en su consecución. Es por ello que no eran utilizadas para expresar diseños según los conceptos del dominio de la aplicación para el que se desarrollaba.

Esto continuó siendo así, hasta la llegada de los años 80. En los primeros años de esta década tuvo lugar la aparición de las herramientas CASE (computer-aided software engineering). Su aportación principal consistió en un sistema gráfico para la representación de los diseños de los desarrolladores. A partir de estos diagramas era posible generar código, reduciendo así los esfuerzos de programación manual (en oposición a la generada automáticamente por las herramientas), depuración y portabilidad. Sin embargo, a pesar de este potencial, se han utilizado en mayor medida para la generación de documentación. En la mayoría de casos, las representaciones realizadas servían como base para la implementación. Sin embargo, a medida que estas avanzaban, los diseñadores no se preocupan por la actualización de esta documentación, con lo que terminaba quedando obsoleta.

La razón principal fue la dificultad para acoplar el código producido con la tecnología sobre la que se sustentaría su ejecución. Habitualmente, los sistemas operativos existentes presentaban serias deficiencias en servicios básicos, como seguridad o tolerancia a fallos, que los programas debían suplir. Ello causaba que la complejidad del código proporcionado aumentara considerablemente. La consecuencia final es que las herramientas CASE y los códigos generados por ellas son difíciles de desarrollar, probar y evolucionar. Otro problema era su poca adaptabilidad a equipos de desarrollo con un buen número de componentes, al impedir el trabajo en paralelo sobre un mismo fichero de los que componen un proyecto. Además, son herramientas demasiado genéricas, con lo que no dan soporte a demasiados dominios de aplicación.

Gracias a los lenguajes de programación imperantes hoy día, mediante las librerías y frameworks desarrollados sobre ellos, no es necesario invertir tiempo en

servicios comunes como seguridad o transacciones. Esto demuestra que ha continuado la tendencia de aislar al programador de los elementos computacionales más complejos de aplicar. No obstante, el problema de la portabilidad aún sigue presente, más aún hoy con las grandes diferencias existentes entre las plataformas más empleadas, como pueden ser J2EE y .NET. Incluso, cada tecnología presenta una evolución continua, con lo que un desarrollo para ella debe seguir a la par. Este trabajo es manual, y requiere de un esfuerzo nada desdeñable, especialmente, si se desea conservar un buen rendimiento de la aplicación.

2.1.2 MDE: la solución a los problemas de portabilidad

Las tecnologías basadas en MDE combinan:

- *Lenguajes de modelado específicos del dominio (DSML).*

  Se utilizan para representar la estructura, el comportamiento y los requisitos en dominios particulares. Estos lenguajes son descritos mediante la definición de metamodelos, que establecen las relaciones entre conceptos de un dominio y especifican la semántica y las restricciones de dichos conceptos.

- *Motores de transformación y generadores*.

  Estos elementos analizan los modelos y crean a partir de ellos artefactos. Éstos pueden ser de varios tipos: código fuente, descriptores de despliegue en XML o representaciones alternativas del modelo inicial. Esta idea beneficia una correcta sintonía entre los modelos que recogen los requisitos y la implementación final. El gran mérito de este sistema es que se garantiza que los desarrollos son correctos por construcción, en contraposición al tediosos sistema tradicional de construir para corregir.

Las tecnologías MDE actuales no se basan en anotaciones genéricas, complicadas de adaptar al problema que requiere la implementación de un desarrollo. En su lugar, los DSML se crean a medida a través de metamodelos que relacionan semántica y sintaxis. Esto beneficia que los modelos producidos resulten más familiares a cualquier persona familiarizada con el dominio. Incluso, es posible imponer desde las primeras fases condiciones propias del dominio y comprobaciones que permitan la detección de errores. Por último, las herramientas MDE no necesitan ser particularmente complejas, pues pueden generar artefactos que deleguen sus tareas en APIs y frameworks estandarizados.

2.2 MODEL DRIVEN ARCHITECTURE (MDA)

2.2.1 Introducción

MDA [MILL03] es un ejemplo de aplicación de MDE. Fue propuesto por un grupo de expertos en la materia denominado OMG (Object Management Group). Parte de la tendencia a separar las operaciones que deben ser soportadas por un sistema, de la manera en que dicho sistema emplea las capacidades de la tecnología empleada.

MDA describe un enfoque orientado a:

- La especificación de un sistema, independientemente de la tecnología que lo sustenta.
- La especificación de plataformas.
- La selección de una plataforma particular para un sistema.
- La transformación de la definición de un sistema hacia uno en concreto para una tecnología particular.

Los objetivos principales de MDA son portabilidad, interoperabilidad y reusabilidad a través de la separación arquitectónica de aspectos.

Los principios sobre los que se sustenta este paradigma pueden extraerse del texto [BOOC04]. Fue redactado por varios autores de IBM, especialistas en distintas áreas como Ingeniería y Arquitectura de Software, estándares de intercambio de datos y, por supuesto, metodologías orientadas a modelos. Sus bases son establecidas por tres ideas fundamentales y complementarias, reflejadas en el diagrama siguiente:

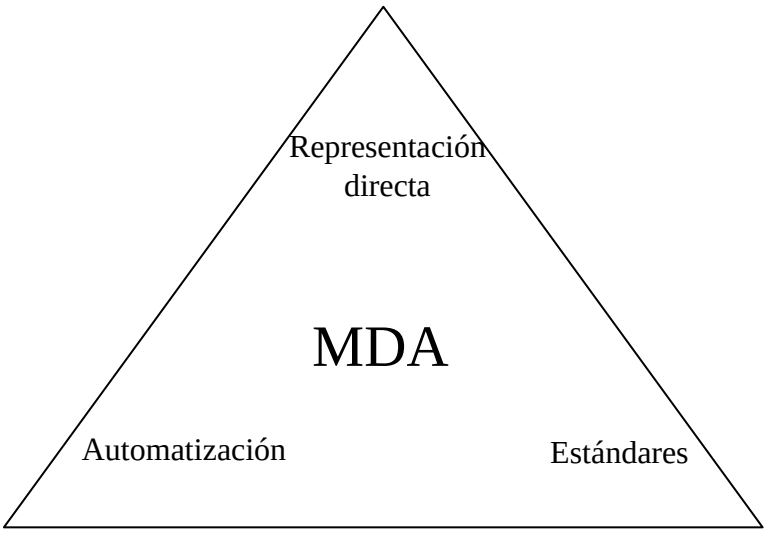


Figura 2.1. Conceptos fundamentales de MDA

- *La representación directa.*

El protagonismo se centra en el desarrollo del software hacia los conceptos del dominio del problema, pero aisladamente de la tecnología. El objetivo es reducir la distancia semántica entre el ámbito del problema, que consta de conceptos específicos del dominio, y su representación. Con ello, se consigue una mejor adaptación de las soluciones a los problemas, logrando diseños más adecuados y una mayor productividad.

Los dominios de aplicación son muy numerosos. A medida que el conocimiento y la experiencia sobre ellos aumentan, se hacen más sofisticados. Por eso, cada uno requerirá un lenguaje específico que represente adecuadamente los conceptos que el dominio encierra.

- *La automatización.*

Consiste en el uso de herramientas programadas que realizan trabajos mecánicos que no dependen del ingenio del ser humano. Una de estas tareas es la traducción desde el modelo al código. Si ésta se asigna a personas, le dedicarán un largo tiempo y causarán muchos errores. Es una tarea similar a la que se delega en los compiladores.

Por esta causa, una de las propuestas más importantes es esquivar el espacio semántico que existe entre los conceptos del dominio y la tecnología de implementación. Para ello, se modelan esas ideas mediante las tecnologías apropiadas, y después se emplea el conocimiento disponible sobre el resultado que producen, para la interpretación de la información obtenida. En este proceso, el desarrollador que emplea MDA no debe preocuparse por las transiciones. Para llevarlas a cabo, las herramientas de transformación realizan suposiciones sobre el dominio de la aplicación y el entorno de desarrollo. De esa forma, se restringe el modelo, y se logran conversiones eficientes entre él y el entorno de ejecución.

- *Los estándares abiertos.*

A lo largo de la Historia, los estándares han sido siempre buenos cimientos para los procesos. Su ventaja principal es la supresión de la diversidad superflua. Esto produce un conjunto de proveedores que crean herramientas especializadas o de propósito general que siguen los estándares. Con ello, se favorece el aumento de la popularidad de éstos entre los usuarios.

No obstante, su éxito sería imposible en un mercado muy fragmentado debido a una diversidad excesiva. Los fabricantes tendrían que enfrentarse a grandes dificultades para lograr productos que funcionaran en múltiples plataformas y hacerlos interesantes a la mayor proporción de usuarios posible. En consecuencia, los proveedores deben trabajar en el mismo sentido, por su bien común, cada uno cubriendo una necesidad concreta.

Además, los desarrollos de código abierto garantizan el cumplimiento de los estándares y fomentan la adopción de los mismos por parte de los fabricantes.

2.2.2 Conceptos clave de MDA

A continuación se presenta un listado de conceptos ampliamente utilizados en este ámbito.

- *Sistema.*

  Los conceptos de MDA son presentados en base a algún sistema planificado o existente. Éste puede contener cualquier elemento: un programa, un ordenador simple, una combinación de partes de distintos sistemas, la unión de varios sistemas (cada cual bajo un control propio), empresas, personas, etc.

- *Modelo.*

  Un modelo de un sistema es la descripción de ese sistema y su entorno para un propósito en concreto. Normalmente, se trata de una combinación de diagramas y textos. Éstos aparecen escritos en lenguaje de modelado o en lenguaje natural.

- *Orientación a modelo.*

  Es un enfoque en desarrollo de software que incrementa el protagonismo de los modelos. El uso del término "orientación" se debe a que se proveen formas de aplicación de los modelos para dirigir las fases de comprensión (del problema), diseño, construcción, despliegue, funcionamiento, mantenimiento y modificación.

- *Arquitectura.*

  La arquitectura de un sistema es la descripción de las distintas partes que lo componen, así como de la manera en que interaccionan entre sí. La arquitectura orientada a modelos hace referencia a determinados tipos de modelos que deben ser empleados y a las relaciones entre ellos.

- *Punto de vista.*

  Se trata de una técnica de abstracción para la visualización de un sistema. Consisten en emplear un conjunto concreto de conceptos arquitectónicos y reglas de estructuración, seleccionados para enfatizar determinados aspectos del sistema. El objetivo es, por tanto, abstraerse de detalles que carecen de interés para generar una versión simplificada.

  MDA establece tres puntos de vista de un sistema: independiente de la computación, independiente de la plataforma y específico de la plataforma. En la sección siguiente se describirá cada una detalladamente.

- *Vista.*

  Es la presentación de un sistema desde un punto de vista particular.

- *Plataforma.*

Es el conjunto de subsistemas y tecnologías que proporcionan una serie de funcionalidades a través de interfaces y patrones de uso. Cualquier aplicación soportada por esa plataforma puede hacer uso de ellas sin necesidad de que conozca sus detalles internos.

- *Independencia de la plataforma.*

Aplicada sobre un modelo, garantiza que éste no utiliza ninguna funcionalidad particular de cualquier plataforma existente.

Como otras cualidades, ésta admite graduación. Así, puede aceptar la existencia de cierta funcionalidad de una plataforma en concreto, mientras otro puede estar totalmente ligado con ella. Como ejemplo de este caso, se puede imagínese la existencia de dos modelos. El primero de ellos asume la existencia de la invocación remota, muy extendida actualmente. Por su parte, el segundo aplica funcionalidades propias de la tecnología CORBA.

2.2.3 Puntos de vista de MDA

Como ya se ha indicado anteriormente, son los que propone MDA:

- *Punto de vista independiente de la computación.*

En él, los protagonistas son el entorno del sistema y los requisitos que lo definen. Los detalles de su estructura o procesamiento están ocultos o ni siquiera se han determinado.

- *Punto de vista independiente de la plataforma.*

Destaca las operaciones propias del sistema, mientras esconde los detalles necesarios para una plataforma concreta. Dicho de otra manera, muestra aquellos aspectos que no varían entre distintas plataformas.

Puede emplearse un lenguaje de modelado de propósito general o un lenguaje específico del área en que se aplica.

- *Punto de vista específico de la plataforma.*

Es una especialización del anterior, de manera que se le añaden los detalles referidos al uso de una tecnología dada.

2.2.4 Modelos de MDA

Se describen seguidamente los modelos descritos por la especificación MDA.

2.2.4.1 Modelo independiente de la computación (CIM)

Se trata de una vista del sistema bajo la perspectiva independiente de la computación. No muestra la estructura del sistema. En ocasiones, se denomina modelo

de dominio, y emplea un vocabulario que resulta conocido para los profesionales de su ámbito.

Está claro que estos profesionales no tienen por qué comprender los modelos y artefactos empleados para implementar la funcionalidad de los requisitos reflejados en el CIM. Es por ello que este modelo juega un papel importante a la hora de vincular a los expertos en el dominio, con los conocedores del diseño y construcción de los productos que satisfarán los requisitos.

2.2.4.2 Modelo independiente de la plataforma (PIM)

Consiste en la visión de un sistema ofrecida a través del punto de vista independiente de la plataforma. Este grado de esta independencia debe ser tal que pueda ser admitido por diferentes plataformas de un mismo tipo.

Una técnica habitual para lograr la desvinculación total de la plataforma es crear un modelo para una máquina virtual tecnológicamente neutra. Esto quiere decir que dispone de una serie de servicios y componentes genéricos, que en cada plataforma tienen una realización exclusiva para ella. Una máquina virtual es una plataforma y, por tanto, existen modelos específicos para ella. No obstante, éstos son a su vez independientes de las distintas plataformas en que la máquina virtual ha sido implementada. En conclusión, estos modelos cumplen los criterios expuestos en la independencia de plataforma.

2.2.4.3 Modelo específico de la plataforma (PSM)

Este modelo combina las especificaciones expuestas en el PIM con los detalles que determinan la forma en que el sistema emplea un tipo de plataforma concreta.

2.2.4.4 Modelo de plataforma

Un modelo de plataforma consta de un grupo de conceptos técnicos. Éstos representan a los distintos elementos que conforman una plataforma y a los servicios que provee. También contiene información, aplicable en la construcción de PSMs, sobre los elementos de la plataforma que están a disposición de la aplicación resultante.

Por otra parte, también hace referencia a restricciones sobre las relaciones y el uso de distintas partes de la plataforma, así como a las conexiones entre la aplicación y la plataforma.

Por último, cabe señalar que un modelo de plataforma genérico puede llegar a establecer un estilo particular de arquitectura.

2.2.5 Transformación de modelos

La transformación de modelos es el proceso de paso de un modelo a otro del mismo sistema. En MDA se denomina mapeo al conjunto de especificaciones que permiten el paso de un PIM a un PSM para una plataforma concreta.

El modo de acometer el desarrollo de software a través de MDA se sintetiza en los siguientes pasos:

1. Construcción del PIM.

2. Utilizar una herramienta de transformación para convertir el PIM en uno o varios PSMs.

3. Utilizar una herramienta para obtener el código a partir del PSM.

Las herramientas empleadas en los pasos intermedios pueden ser distintas, o bien, tratarse de la misma. El siguiente diagrama representa la secuencia de transformaciones, aplicada a los distintos modelos.

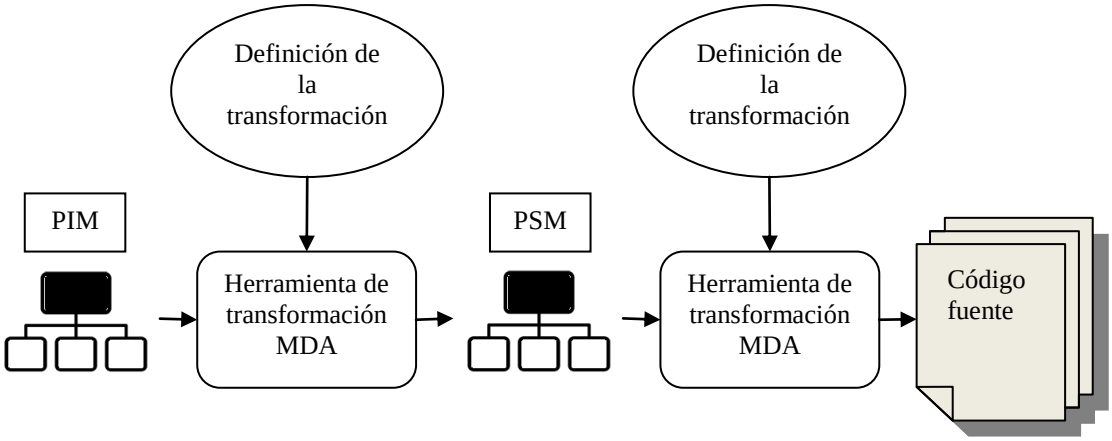


Figura 2.2. Secuencia de transformación de modelos

Es importante apreciar la separación existente entre cada herramienta de transformación y la definición de la transformación. La primera describe el proceso de conversión de un modelo en otro. La definición de la transformación podría entenderse como un conjunto de reglas, que constituyen una especificación inequívoca de la manera en que un modelo (o parte de él) puede transformarse en otro (o parte de otro). Estas reglas indican cómo convertir un elemento del lenguaje origen (el que se emplea para describir al primer modelo) en su equivalente del lenguaje destino (el que se emplea para describir al segundo modelo).

Las definiciones de transformaciones pueden ser de dos tipos:

- *Por tipo de modelo.*

  Especifica la conversión desde cualquier modelo construido mediante tipos especificados por el lenguaje usado en el PIM a modelos expresados a través de tipos del lenguaje del PSM. Por ejemplo, en J2EE, si un atributo *sharable* de una clase *entity* es cierto, entonces se transformará en una entidad EJB. En otro caso, dará lugar a una clase Java.

- *Por instancia de modelo.*

Se trata de identificar partes concretas del modelo PIM que deben ser transformadas de una forma determinada, permitiendo escoger la plataforma a que se orientará el PSM. Para agregar información adicional al modelo, se utilizan las marcas. Estas marcas también pueden indicar requisitos o restricciones que debe verificar el modelo destino, como por ejemplo, que un dato sea temporal o persistente.

En conclusión, es importante destacar que las herramientas de transformación, junto con las definiciones de las transformaciones, son componentes clave dentro de MDA.

2.2.6 Servicios difundidos

Son servicios disponibles en un amplio conjunto de tecnologías. MDA contiene modelos independientes de plataforma de servicios difundidos. Además, provee un sistema de conversión de los modelos que contienen estos PIMs de servicios difundidos a PSMs que emplean los servicios proporcionados por una plataforma determinada.

2.3 PROCESO DE DESARROLLO CON MDA

La evolución del desarrollo de software [KLEP04] no es tan cuantificable como la sucedida en torno al hardware, que se puede medir en términos de velocidad o rendimiento. No obstante, es evidente que ha ido mejorando, pues ahora es posible implementar sistemas más complejos con menor esfuerzo. Sin embargo, sigue tratándose de un tema complejo, debido a la diversidad en las tecnologías empleadas y a las interconexiones requeridas entre los distintos sistemas desarrollados o empleados. Esto se complica aún más si se añade el factor de los cambios continuos surgidos sobre los requisitos. En los siguientes apartados se explica cómo la aplicación de MDA puede ser de gran ayuda para estos casos.

2.3.1 Desarrollo de software tradicional

El sistema tradicional de desarrollo de software está organizado en las siguientes fases:

- *Recogida de requisitos.*
- *Análisis.*
- *Diseño.*
- *Codificación.*
- *Pruebas.*
- *Despliegue.*

La siguiente figura muestra el ciclo de vida del sistema tradicional de desarrollo de software.

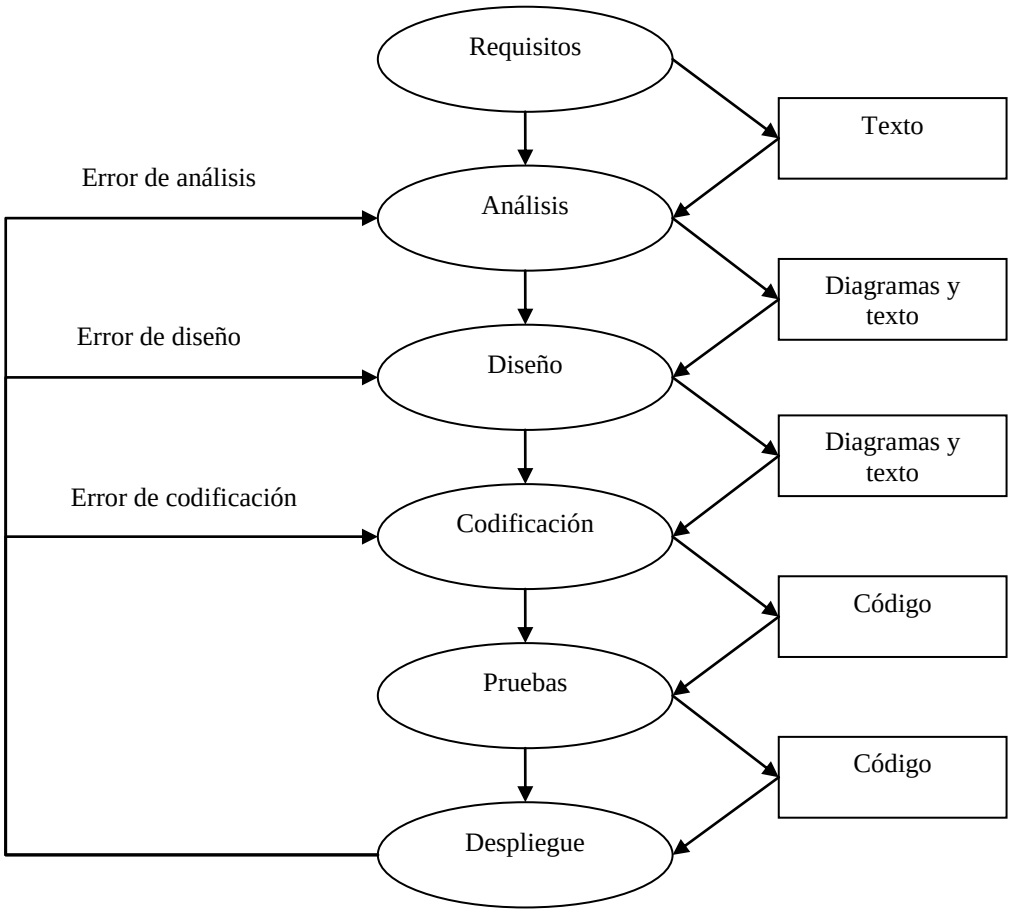


Figura 2.3. Ciclo de vida del sistema de desarrollo de software tradicional

A la vista de este sistema, se presenta un conjunto de problemas que se describen en las secciones siguientes.

2.3.1.1 Productividad

El resultado de las tres primeras fases [PELA07] consiste en una cantidad ingente de documentación que pierde valor a medida que avanza la codificación. Incluso, aunque en ésta se adopten modificaciones propias de análisis o diseño, estos cambios no se ven reflejados en los productos que se han generado anteriormente en ellas, por lo que resultan inútiles. Sería deseable que los cambios aplicados sobre cualquier nivel se propagaran de una manera simple al resto.

2.3.1.2 Portabilidad

Una característica única de la industria del software es la aparición continua de nuevas tecnologías que logran popularidad entre la comunidad de desarrolladores. El problema se complica con la aparición de nuevas versiones de una misma tecnología, que en muchos casos no son compatibles.

Algunas compañías se ven obligadas a mantenerse actualizadas, debido a que sus clientes las demandan, resuelven problemas conocidos o sus proveedores dejan de utilizar las tecnologías antiguas. A pesar de que esta transición supone mejoras, muchas empresas no pueden asumir la inversión que requiere. Además, se pierde el esfuerzo realizado para la adopción de las usadas anteriormente.

2.3.1.3 Interoperabilidad

Los sistemas software no suelen encontrarse de manera solitaria. Necesitan comunicarse con otros sistemas, habitualmente ya construidos. Incluso, aunque se trate de sistemas construidos desde cero, combinan múltiples tecnologías.

La tendencia ha sido evitar la construcción de sistemas monolíticos, remplazados por componentes que unidos participen en la realización de una funcionalidad global. Esta filosofía facilita la posibilidad de realizar cambios sobre el sistema. Los diferentes elementos utilizan la mejor tecnología para cada operación, pero necesitan interactuar con los demás.

2.3.1.4 Mantenimiento y documentación

La documentación es frecuentemente el punto débil de los desarrollos software. Los programadores opinan que su mantenimiento durante la fase de implementación reduce su productividad. Si lo hacen es simplemente por obligación.

Al final, la documentación no tiene la calidad deseada. Contiene una información desactualizada, ya que las personas que deberían revisarla no lo hacen. Creen, equivocadamente, que su única función es producir sistemas que puedan ser mantenidos o modificados posteriormente.

La solución a este problema a nivel de código es que la documentación se genere directamente a partir de él. Obviamente, la documentación (diagramas y textos) de alto nivel deben seguir siendo mantenidos manualmente. Éstos siguen siendo necesarios, dada la complejidad de los sistemas desarrollados.

2.3.2 Desarrollo de software con MDA

El ciclo de vida del desarrollo de software con MDA es similar al tradicional.

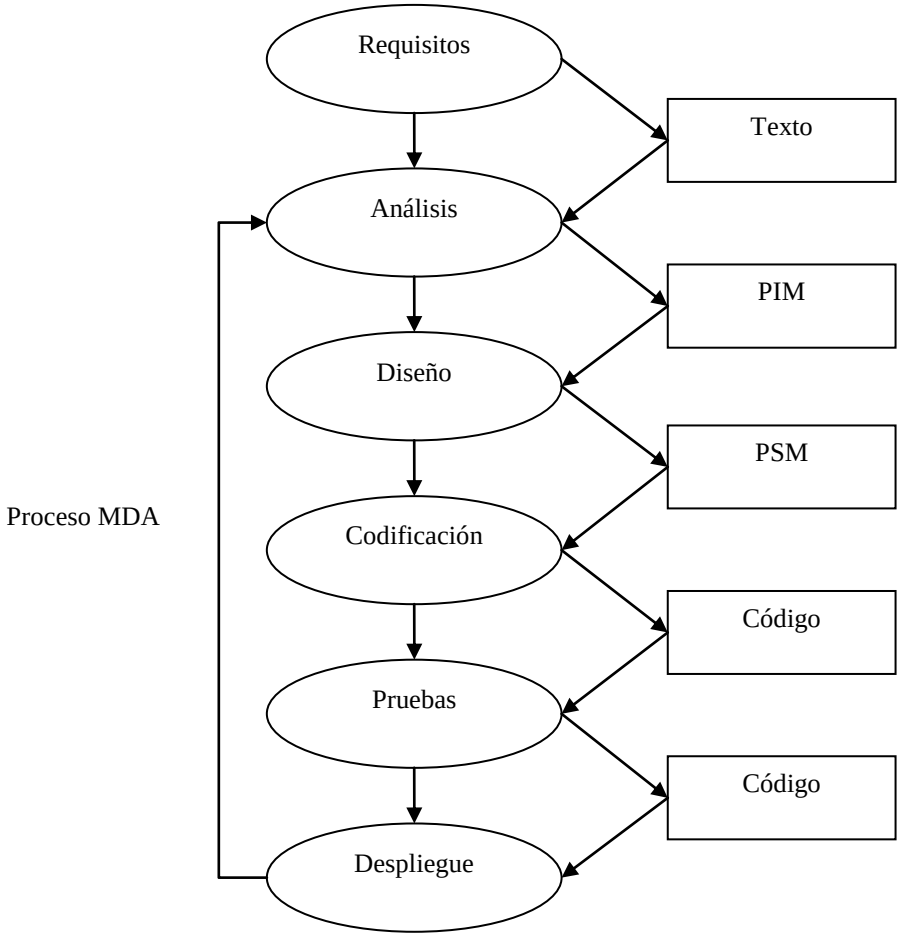


Figura 2.4. Ciclo de vida del sistema de desarrollo de software con MDA

Aunque a primera vista es muy similar al diagrama que caracteriza el sistema de desarrollo tradicional, existe una diferencia fundamental entre ambos. En MDA las transformaciones entre modelos o entre modelo y código se realizan de manera automática. Mientras, en el sistema tradicional, estas transiciones se producen manualmente o, como máximo, se generan plantillas con la estructura básica donde se debe completar el código.

En los apartados siguientes se hace referencia a las mejoras que MDA aporta al desarrollo de software tradicional.

2.3.2.1 Solución al problema de productividad

En MDA [PELA07] los elementos protagonistas son los PIMs. A partir de ellos, de manera automática, se obtienen los PSMs. El desarrollador puede centrar su atención en la implementación de estas transformaciones. Se trata de una tarea compleja que requiere especialización. No obstante, las transformaciones pueden emplearse en numerosos desarrollos distintos. Además, los PIMs aíslan los detalles concretos de cada tecnología, lo que logra que se puedan añadir nuevas funcionalidades con menor esfuerzo. La mayor parte de este trabajo corresponde a las transformaciones.

2.3.2.2 Solución al problema de portabilidad

En MDA, la portabilidad se logra centrando el protagonismo sobre los PIM. El mismo PIM puede ser transformado automáticamente en PSMs para distintas plataformas. Cualquier cosa especificada en el PIM es totalmente portable.

Sin embargo, la portabilidad lograda depende de las herramientas de transformación existentes. Éstas serán más comunes para tecnologías asentadas. En el caso de tecnologías de reciente creación, será el propio desarrollador el que deba componer sus propias transformaciones.

2.3.2.3 Solución al problema de interoperabilidad

Los PSMs generados a partir de un mismo PIM, no pueden comunicarse entre sí, al emplear plataformas diferentes. Por este mismo motivo, es necesario convertir los conceptos propios de una en los empleados por la otra. Para ello, MDA introduce el concepto de puentes entre PSMs, que son generados a la vez que éstos.

La definición de los puentes es simple. Se conoce para cada elemento del PIM qué elementos ha derivado en cada PSM. Obviamente, también es conocida la tecnología propia de cada plataforma, dado que en otro caso, la transformación de PIM a PSM sería imposible. En consecuencia, a través de esta información, se puede deducir qué componentes de los PSMs se relacionan entre sí. Los puentes también son aplicables a pares de códigos generados desde dos PSMs distintos.

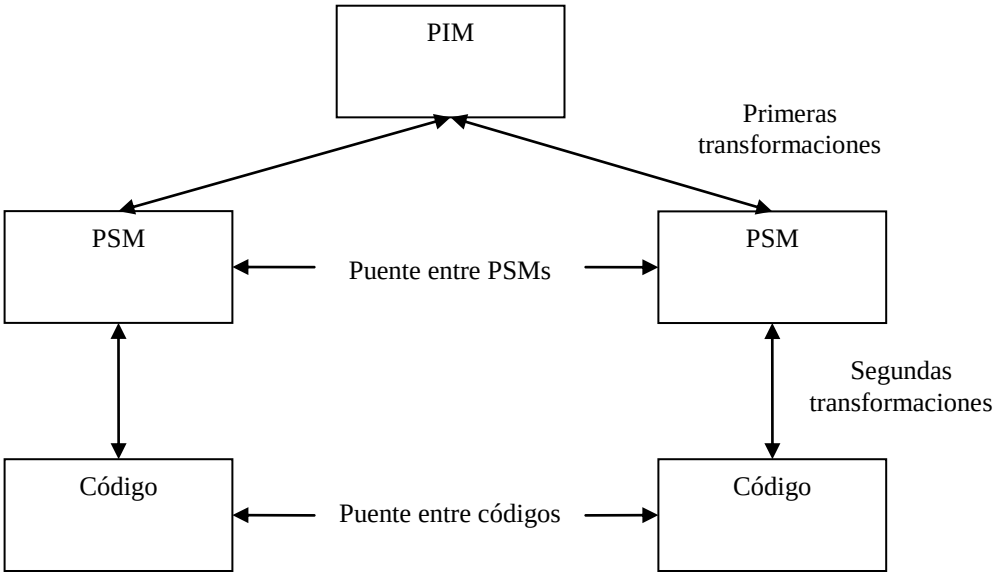


Figura 2.5. Relaciones entre los componentes de MDA

2.3.2.4 Solución al problema de mantenimiento y documentación

Mediante el ciclo de vida de MDA, los desarrolladores pueden centrarse en el PIM. Éste se emplea para la generación de los PSMs, que derivarán en el código. Por

tanto, existe una relación directa entre el modelo y el código. El PIM sirve como documentación de alto nivel.

Respecto al mantenimiento, puede ocurrir que se actualice el PIM para generar nuevos PSMs y, finalmente, versiones distintas del código. Otra opción es manipular directamente los PSMs. No obstante, existen herramientas que extienden las modificaciones al PIM, conservando la coherencia entre los modelos.

2.4 ESTÁNDARES EMPLEADOS EN MDA

Uno de los principios destacados en el texto [BOOC04] es el uso de los estándares. La difusión de su uso radica en ventajas que tanto usuarios como fabricantes pueden aprovechar. Los primeros, al emplear productos que verifiquen los estándares, tienen garantías de que serán compatibles en sus plataformas. Por su lado, los proveedores crearán desarrollos adaptados a las especificaciones de los estándares y, por tanto, tendrán un amplio público objetivo.

Concretamente, OMG propone en el sitio [OMG11a] un numeroso conjunto de especificaciones para el modelado.

2.4.1 Lenguajes de modelado

2.4.1.1 UML (Unified Modeling Language)

Se trata [FOWL04] de una familia de noticiones gráficas, respaldadas por un metamodelo simple. Su objetivo es la ayuda en la descripción y el diseño de sistemas software, especialmente si se desea implementarlos bajo el paradigma de orientación a objetos. Surgió en 1997 de la unificación de los distintos lenguajes gráficos de modelado orientados a objetos existentes desde finales de los años 80. Con ello, se ha suprimido la gran diversidad existente, lo que ha supuesto el primer logro de UML.

Existen diversas maneras en que se puede aplicar UML, heredadas de los antiguos lenguajes:

- *UML como croquis*.

  De esta manera, UML es empleado como una herramienta de comunicación, a través de la que los desarrolladores especifican diversas partes del sistema.

  Se puede aplicar tanto a través de ingeniería inversa (construcción de modelos sobre código ya construido) como de ingeniería directa (en contraposición a la inversa, la modelización gráfica precede a la codificación). En el primer caso, el objetivo principal es tratar alternativas posibles para un problema concreto. En el enfoque según ingeniería inversa, se trata de un primer acercamiento a la funcionalidad de una aplicación antes de indagar en el código.

  Estos dos usos también son viables en el siguiente punto de vista, con la diferencia de que se aplica un mayor nivel de detalle.

- *UML como plano (blueprint).*

  Los modelos constituyen una guía que el desarrollador puede seguir para implementar los modelos diseñados. Con ella, no tendrá que decidir ningún aspecto sobre el diseño, pues todas las determinaciones serán dirigidas.

- *UML como lenguaje de programación.*

El desarrollador emplea UML como un lenguaje de programación. Los diagramas son transformados directamente a código fuente. Para ello, se requieren herramientas de conversión muy complejas. En este caso, no cabe la distinción entre ingeniería inversa y directa, pues UML se concibe como un lenguaje de programación que es traducido a otro.

Una variante de UML es su versión ejecutable. Es similar a MDA, pero usa conceptos distintos. De la misma manera, se parte de un modelo independiente de plataforma representado mediante UML. Es similar al PIM. Sin embargo, a continuación se emplea un compilador de modelo, que convierte este UML directamente en un sistema desarrollado, listo para desplegar. Por tanto, no es necesario el uso del modelo PSM. Esta tecnología se basa en arquetipos, que describen el proceso de transformación para una plataforma determinada. La ventaja es que al mismo compilador de modelo se le pueden aplicar dos arquetipos (por ejemplo para .NET y J2EE), logrando así dos versiones del sistema.

2.4.1.2 OCL (Object Constraint Language)

En ocasiones, un diagrama UML no es suficientemente detallado como para mostrar todas las características relevantes de la especificación. Aspectos como restricciones adicionales sobre los modelos, terminan indicándose en lenguaje natural. Ello provoca la aparición de ambigüedades. OCL evita estas imprecisiones.

OCL [OMG06] permite definir expresiones cuya evaluación genera simplemente un valor. Esta operación no puede de ninguna forma influir en el modelo. No se trata de un lenguaje de programación. Debido a ello, no es posible escribir lógica de programa o control de flujo a través de él.

Sus funciones son básicamente las siguientes:

- Como un lenguaje de consulta.
- Especificar invariantes, precondiciones y postcondiciones de operaciones.
- Describir guardas.
- Hacer referencia a conjuntos de objetivos de mensajes u operaciones.
- Añadir restricciones a operaciones.
- Incluir reglas de derivación de atributos para una expresión del modelo UML.

2.4.1.3 MOF (Meta-Object Facility)

Define [OMG02] un lenguaje abstracto y un framework para especificar, construir y gestionar metamodelos independientes de tecnología. Se denomina metamodelo a un lenguaje para algún tipo de metadato.

Se trata de una arquitectura conceptual basada en capas. En ella, los elementos de una capa describen a los contenidos en la siguiente. Por ejemplo, el meta-metamodelo

MOF define el metamodelo UML, el cual define modelos UML, que, finalmente, describen un sistema computacional. También se aplica sobre el lenguaje IDL, diseñado para la especificación de interfaces y aplicado habitualmente a la descripción de interfaces entre distintas plataformas.

MOF define un framework para implementar repositorios que almacenan metadatos, por ejemplo modelos, descritos por los metamodelos. Usa relaciones estándar entre las tecnologías para transformar metamodelos MOF en APIs de metadatos. Con ello se logra un repositorio de APIs de metadatos consistente e interoperable, para diferentes proveedores e implementaciones.

El framework clásico para metamodelado está basado en una arquitectura de cuatro capas:

- La capa de información contiene los datos que se desea describir.
- La capa de modelo guarda los metadatos que describen los datos de la capa de información. Aplicando el concepto de forma no estricta, los metadatos son asimilados a los modelos.
- La capa de metamodelo incluye las descripciones que determinan la estructura y la semántica de los metadatos. Es un lenguaje abstracto que define diferentes tipos de datos. Por ello, no dispone de una sintaxis concreta.
- La capa de meta-metamodelo refleja la descripción de la estructura y la semántica de los meta-metadatos. En otras palabras, se trata de un lenguaje abstracto para definir diferentes tipos de metadatos.

MOF se basa en esta estructura, dando a cada capa una denominación concreta y un contenido definido:

- Capa M0: capa de datos.
- Capa M1: modelos UML e interfaces IDL.
- Capa M2: metamodelos UML e IDL.
- Capa M3: modelo MOF.

A pesar de esta similitud en la estructura, presenta una serie de características que diferencian esta arquitectura de la primitiva:

- Es orientada a objetos, con constructores de metamodelado alineados con los constructores de modelado de objetos UML.
- Los cuatro niveles no tienen por qué ser fijos. Pueden modificarse según las necesidades.
- Un modelo puede no estar restringido a un único meta-nivel. Por ejemplo, en el contexto de almacenamiento de datos puede ser útil pensar en el meta-

esquema tabla relacional y sus instancias (esquemas específicos) como un único modelo conceptual.

- Es autodescriptivo. De hecho, está definido de manera formal usando su propio constructor de metamodelado.

2.4.1.4 XMI (XML Metadata Interchange)

XMI [OMG11b] es un formato para el intercambio de modelos empleando XML. Este hecho aporta una serie de ventajas:

- Representación de objetos en base a los elementos y atributos XML.
- XMI dispone de mecanismos para relacionar objetos interconectados, situados en el mismo fichero o en varios distintos.
- Identidad unívoca de objetos a través de identificadores.
- Validación de los documentos XML empleando *schemas* XMI.

Los puntos obligatorios para el cumplimiento de la especificación XMI se clasifican en:

- *Cumplimiento del XMI schema.*

  El conjunto de XMI *schemas* generado debe ser equivalente a los generados mediante las reglas especificadas en [OMG11b]. Esto quiere decir que los documentos XMI válidos bajo un *XMI schema* generado a través de las reglas de producción para *XMI schemas* son válidos en el cumplimiento de *XMI schema*. A su vez, los documentos XMI no válidos bajo un *XMI schema* generado a través de las reglas de producción para *XMI schemas* no serían válidos en la verificación de *XMI schema*.

- *Cumplimiento del documento XMI.*

  El documento XMI debe ser válido y estar bien formado, según los requisitos impuestos por XML. Esto debe garantizarse, independientemente de que se haya contrastado mediante un *XMI schema*. El documento debe ser equivalente a los generados mediante las reglas especificadas en [OMG11b]. Esta equivalencia debe entenderse como la correspondencia uno a uno entre elementos de los documentos, incluyendo nombres de elementos, nombres y valores de atributos de elementos, y elementos contenidos.

2.4.1.5 CWM (Common Warehouse Metamodel)

El propósito principal de CWM [OMG03] es favorecer un intercambio sencillo de repositorios y metadatos de lógica de negocio entre herramientas y plataformas de repositorio en entornos distribuidos heterogéneos. Se basa en UML, MOF y XMI, combinados de la siguiente manera:

- Una arquitectura de metamodelado por capas. Se utiliza para la manipulación de metadatos en repositorios de objetos distribuidos, concretamente a través de UML y MOF.

- UML para representar metamodelos y modelos. En concreto, CWM incluye una versión reducida de UML llamada *Object Model*. A través de ella, el metamodelo define reglas formales para modelar instancias de repositorios de datos.

- El uso de modelos de información estándar para describir la semántica de los análisis de los objetos y modelos diseñados.

- La utilización de MOF para especificar y manipular metamodelos programáticamente.

- La aplicación de XMI al intercambio de metadatos.

2.4.1.6 SysML (Systems Modeling Language)

SysML [OMG10] es un subconjunto de UML 2, unido a extensiones necesarias para RFP (*request for proposal*, proceso de solicitud de una oferta a un proveedor por la concesión de un servicio) de ingeniería de sistemas.

Los principios fundamentales de SysML son:

- Orientación a los requisitos, concretamente para satisfacer aquéllos del modelo UML para RFP de ingeniería de sistemas.

- Reutilización de UML. Cuando ha sido preciso realizar modificaciones, se han llevado a cabo de una manera que minimizara los cambios sobre el lenguaje que lo sustenta. Por ello, SysML ha sido relativamente fácil de implementar para proveedores que soportan UML 2.

- Extensiones UML, para cumplir los requisitos para RFP.

- Particionado de la especificación en paquetes, que componen la unidad básica. Se trata de agrupaciones lógicas que suprimen las dependencias entre sus componentes.

- Clasificación en capas. SysML forma una capa que extiende el metamodelo UML.

- Interoperabilidad debida a la capacidad de intercambio de UML mediante XMI.

2.4.2 Lenguajes de definición de transformaciones

2.4.2.1 QVT (Query/View/Transformation)

Sus siglas [PELA07] hacen referencia a las tres abstracciones fundamentales que presenta:

- Consultas, que son expresiones que se evalúan sobre un modelo.
- Vistas, que se definen como modelos obtenidos a partir de otro modelo. Las consultas son un tipo restringido de vistas.
- Transformaciones, cuya función es la generación de un modelo a partir de otro.

La especificación QVT [OMG11c] presenta una naturaleza híbrida declarativa e imperativa. La parte declarativa está dividida en dos niveles:

- *Relations.*

  Se trata de una especificación declarativa de las relaciones existentes entre modelos MOF. Este lenguaje incluye patrones de relación complejos y registra los eventos sucedidos durante el proceso de transformación.

- *Core.*

  Es un lenguaje que soporta patrones de relación sobre un conjunto plano de variables. Estos se derivan de la evaluación de condiciones sobre las variables para un conjunto de modelos. Es tan potente como el anterior, aunque la semántica se puede describir de una manera más simple y la descripción de los procesos de transformación son más explicativos. Además, los modelos de traza pueden ser definidos explícitamente, en lugar de ser deducidos de la especificación de la transformación.

A partir de estos dos conceptos puede establecerse una analogía con la máquina virtual de Java. El lenguaje de relaciones equivale al lenguaje Java, mientras la semántica *core* sería la definición del comportamiento de la máquina.

Respecto a la parte imperativa de la especificación, existen dos mecanismos de implementación de las transformaciones entre los lenguajes *Relations* y *Core*.

- *Operational Mapping.*

  Es un estándar que provee implementaciones imperativas de las transformaciones. Estas implementaciones registran el mismo modelo de traza que el lenguaje *Relations*. Contiene extensiones OCL que le confieren un estilo procedural y sintaxis familiar para los programadores imperativos.

- *Black Box Implementations.*

Es posible derivar transformaciones *Relations* a operaciones MOF. De esta manera, permite codificar algoritmos complejos en cualquier lenguaje de programación usando MOF y que dicha codificación no se encuentre visible.

2.4.2.2 Mof2text (MOF Model to Text Transformation Language)

Se trata de un estándar [OMG08] que permite especificar la manera en que un modelo es transformado en diferentes artefactos basados en texto. Entre ellos, se encuentra el código, especificaciones de despliegue, documentos e informes. Una forma intuitiva de lograr este objetivo es aplicar un enfoque basado en plantillas. Éstas son parametrizadas mediante elementos del modelo.

Las plantillas reflejan un esqueleto de texto con lugares señalados donde ubicar datos extraídos del modelo. Básicamente son expresiones especificadas sobre elementos del metamodelo. En ellos, las protagonistas son las consultas necesarias para obtener los valores adecuados del modelo. Posteriormente, los valores se convierten a texto mediante las funciones adecuadas de transformación.

Las transformaciones pueden agruparse en módulos, formados por plantillas y consultas. Pueden tener parte pública y privada, que se diferencian por la posibilidad o no de ser invocadas desde otros módulos. Con este fin se proporciona la importación de módulos, de manera que los elementos públicos del importado quedan disponibles en el ámbito donde se incluye.

Capítulo 3. Modelado de procesos

A nivel empresarial, los procesos son cada vez más relevantes, puesto que sus cualidades redundan en la eficiencia de las cadenas productivas. Este hecho ha favorecido la aparición de técnicas orientadas a facilitar la implantación de los procesos desde su fase de diseño. Incluso, se permite la depuración de dichos procesos, cerrando así un ciclo de mejora continua.

Dado el protagonismo que los procesos tienen en este trabajo, se hace un repaso de los conceptos principales manejados en su teoría. Aunque se trata de ámbitos diferentes, el objetivo no deja de incluir un proceso empresarial a pequeña escala, donde el elemento producido es un registro de información. Obviamente, son menos las variables que entran en juego, pues no se requiere una gran inversión en maquinaria ni en recursos humanos. No obstante, comparten su esencia: un conjunto de recursos orquestados para la realización de un fin global.

A continuación, se introduce el concepto clásico de workflow, como una manera primitiva de modelización de procesos.

Posteriormente, se entra de lleno en la gestión de procesos (BPM, *Business Process Management*), como una alternativa a la gestión del workflow. Se trata de la disciplina que engloba todo lo que tiene que ver con los procesos industriales y sus representaciones. Su principal aportación es que aumenta la importancia concedida a la capacidad de análisis y mejora de los procesos implementados. Otra de sus partes afecta al modelado de procesos. Multitud de lenguajes han sido creados para esta finalidad, de los que se ha hecho una recopilación que incluye los más destacados.

3.1 GESTIÓN DE PROCESOS DE NEGOCIO

3.1.1 Introducción

Los procesos de negocio [KO09] de las compañías cada vez tienen una mayor relevancia. Esto viene impuesto por un cliente cada vez más exigente que requiere cambios sobre la marcha, además de una dura competencia. Para todo ello es fundamental una capacidad de decisión rápida. El enfoque [BROC10] de estos procesos de gran escala implica situarse en un alto nivel de las operaciones, para analizar sus impactos y resultados globales.

La definición [AALS03a] de un sistema de gestión de procesos de negocio es un sistema software genérico orientado a diseños de procesos explícitos para definir y gestionar procesos de negocio operacionales. El calificativo genérico hace referencia a que debe ser posible modificar los procesos que maneja.

Es necesario analizar una serie de tendencias para comprender el concepto central. Actualmente, los sistemas de información están formados por un conjunto de capas:

1. El sistema operativo, que, por ejemplo, maneja el hardware.

2. Aplicaciones genéricas, empleadas por un amplio espectro de empresas o incluso, varios departamentos pertenecientes a la misma. Se incluyen en este conjunto los sistemas de gestión de bases de datos o las aplicaciones para hojas de cálculo.

3. Aplicaciones específicas de un dominio, usadas sólo por unas empresas o departamentos concretos.

4. Aplicaciones a medida, desarrollados para organizaciones específicas.

Al principio, las dos capas intermedias no existían, pero han ido cobrando relevancia. Por ello, la tendencia actual consiste en orquestar fragmentos de software independientes, repartidos por las cuatro capas.

En segundo lugar, se han modificado los actores principales de estos sistemas. Inicialmente, se trataban enfoques orientados por los datos, punto inicial para su construcción. El problema era que los procesos tenían que adaptarse a la tecnología que sustentaba la información. No obstante, esta perspectiva ha evolucionado hacia un mayor protagonismo de los procesos, gracias a la aparición de nuevas tácticas, como la reingeniería de procesos.

Por último, se ha pasado de un diseño inicial cuidado, al rediseño y la evolución. Con la implantación de Internet y sus estándares, los sistemas de información cambian continuamente. Por ello, no suelen comenzar desde cero, sino que se aprovechan aplicaciones existentes. Con ello se consigue un desarrollo más dinámico del software.

La idea de aislar los procesos de negocio en un componente aúna las tres tendencias explicadas. Los sistemas de gestión de procesos sirven para eliminar la

codificación de los procesos en aplicaciones a medida. Además, soportan la orientación a proceso, el rediseño y la evolución. Por ejemplo, actualmente pueden ser integrados en aplicaciones existentes y modificar los procesos simplemente actualizando un diagrama de flujo.

Sin embargo, únicamente se aplica en ámbitos muy concretos, como la banca o las compañías de seguros. Uno de los problemas fundamentales es que ha sido difícil elaborar estándares de modelado de procesos de negocio.

Habitualmente se considera la gestión de procesos de negocio (BPM) como el siguiente paso a la moda del flujo de trabajo de la década de los 90. El flujo de trabajo es la automatización de un proceso de negocio, o parte, durante el que documentos, información o tareas pasan de un estado a otro, según un conjunto de reglas procedurales. Por su lado, un sistema de gestión de flujo de trabajo (WFMS, *Workflow Management System*) es un sistema capaz de gestionar la creación y la ejecución de flujos de trabajo en máquinas capaces de interpretar la definición de procesos.

Existen muchas definiciones para BPM, la mayoría de las cuales engloban la gestión de flujo de trabajo (WFM, *Workflow Management*). Se puede especificar como procesos de negocio que emplean métodos, técnicas y software para diseñar, difundir, controlar y analizar procesos operacionales que involucran a humanos, organizaciones, aplicaciones, documentos y otras fuentes de información. Esta definición es ilustrada por su ciclo de vida, dividida en las siguientes fases:

1. Fase de diseño, en la que los procesos son (re)diseñados.

2. Fase de implementación. En ella se implementan los procesos de negocio mediante la configuración de un sistema de información orientado a procesos.

3. Fase de difusión, que es en la que se ejecuta el proceso empleando el sistema configurado.

4. Fase de diagnóstico. Es el periodo durante el que el proceso se analiza para localizar problemas y posibles mejoras.

Esta planificación da lugar a un ciclo de vida reiterativo, como se puede apreciar en la imagen siguiente.

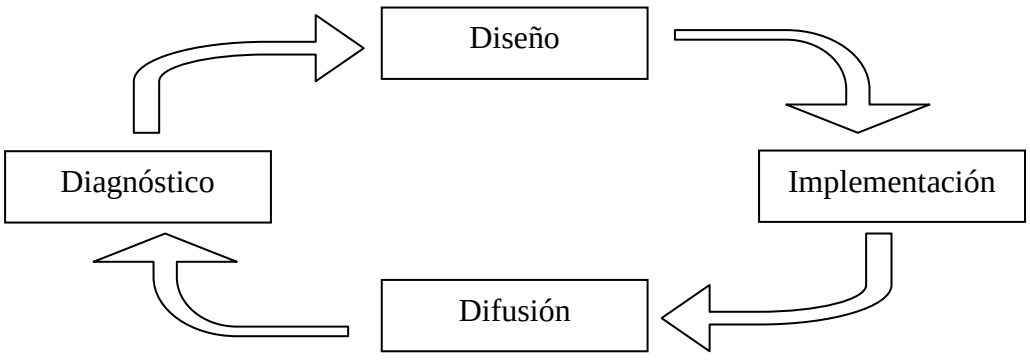


Figura 3.1. Ciclo de vida básico de BPM

Sin embargo, los WFMSs tradicionales se centran en las dos primeras fases. Para más inconveniente, no aportan en la fase de diseño un método de análisis y modelado como tal, sino únicamente un mero editor. Además, aunque algunos ofrecen herramientas de simulación, validación, recuperación de datos en tiempo real y registro de tareas ejecutadas, no aportan ninguna de diagnóstico. Otra diferencia entre ambos aparece en el rediseño de los procesos, donde BPM aventaja a WFM en que aporta dos nuevas tendencias:

- *Straight trough Processing (STP).*

  Tiene por objetivo una automatización compleja de los procesos. Obviamente, en muchos casos sólo es aplicable a un conjunto concreto de casos, pues algunos requerirán la intervención humana obligatoriamente. Al realizar esta separación, se reduce el periodo de duración de los procesos.

- *Case Handling (CH).*

  Esta técnica involucra a los procesos que por su complejidad son difícilmente representables en un diagrama de flujo. Se modela el caso básico o normal, lo que no quita que otros caminos estén también permitidos, salvo que se excluyan explícitamente. Una forma de conseguirlo es realizar flujos de trabajo orientados a los datos en lugar de a los procesos. Se trata de centrarse en el caso visto como un todo, en lugar de elementos individuales contenidos en una lista de tareas.

3.1.2 Los procesos de negocio

Existen muchos tipos de actividades. En todas ellas existe un elemento [AALS02], denominado caso, que es el componente que se modifica o se produce. Éste puede ser tangible, por ejemplo un coche o un edificio, o abstracto, como una reclamación. Cada caso implica un proceso o procedimiento que debe llevarse a cabo. Esto es un conjunto de tareas que necesitan ser completadas, cuya ordenación depende de un conjunto de condiciones. Por su parte, una tarea es una unidad lógica de trabajo que un recurso (persona o máquina, en grupo o individualmente) puede realizar.

Puede suceder que dos o más tareas deban ser ejecutadas en un orden invariable. En ese caso, se dice que forman una secuencia. Otra estructura es la selección, donde es posible elegir entre dos o más tareas. También aparece la ejecución en paralelo, de manera que las tareas que discurren de esta manera deben ser finalizadas antes de que comience la tarea dispuesta a continuación de ambas. Esto es la sincronización. Por último, también pueden aparecer iteraciones en los procesos, las cuales involucran a un conjunto de tareas cuya ejecución debe repetirse cíclicamente. Con estos mecanismos cualquier proceso puede ser representado mediante un modelo.

Respecto a su ejecución, algunas tareas no requieren intervención humana. Otras, en cambio, si necesitan una persona que interprete o decida. Se requiere para ello un conocimiento del proceso. Éste puede obtenerse mediante el aprendizaje (conocimiento explícito) o a través de la experiencia (conocimiento tácito). Su importancia es tal que existe una gestión del conocimiento, que determina la adquisición, enriquecimiento y distribución del mismo, cuando la persona participante lo necesita para ejecutar la tarea.

En el diseño de los procesos se debe tener en cuenta que siempre son más manejables los procesos simples. Para obtenerlos, un procedimiento complejo puede ser dividido en varios simples que deben ser ejecutados paralela o secuencialmente. Por otra parte, cuando un gran número de casos pertenecen a un mismo proceso es muy importante la elaboración de rutinas para logran una ejecución eficiente. Existe un símil muy gráfico en la industria textil: cada prenda de ropa no se confecciona desde 0, sino que existe un formato que es aplicado continuamente. Resulta más rápido y barato que una prenda a medida. Otra opción consiste en analizar el tipo de caso para escoger un proceso u otro, lo que requiere un gran esfuerzo de análisis.

Los procesos se organizan en tres categorías:

- *Procesos primarios.*

  Son aquéllos derivados de los productos o servicios generados por una compañía. También se conocen como procesos productivos. Involucran a casos destinados al cliente. Como regla general, son los procedimientos que generan beneficios y se orientan a cubrir una demanda.

- *Procesos secundarios.*

  Ofrecen un soporte para la ejecución de los primarios, por lo que son también llamados procesos de soporte. Ejemplos de ellos son aquéllos destinados al mantenimiento de los medios de producción (maquinaria, vehículos, etc.) o a la gestión de los recursos humanos.

- *Procesos terciarios.*

  Son procesos iniciados desde la gerencia, cuya finalidad es dirigir y coordinar los primarios y los secundarios. Además, se establecen sus precondiciones y objetivos.

En la siguiente imagen se visualizan las relaciones existentes entre ellos.

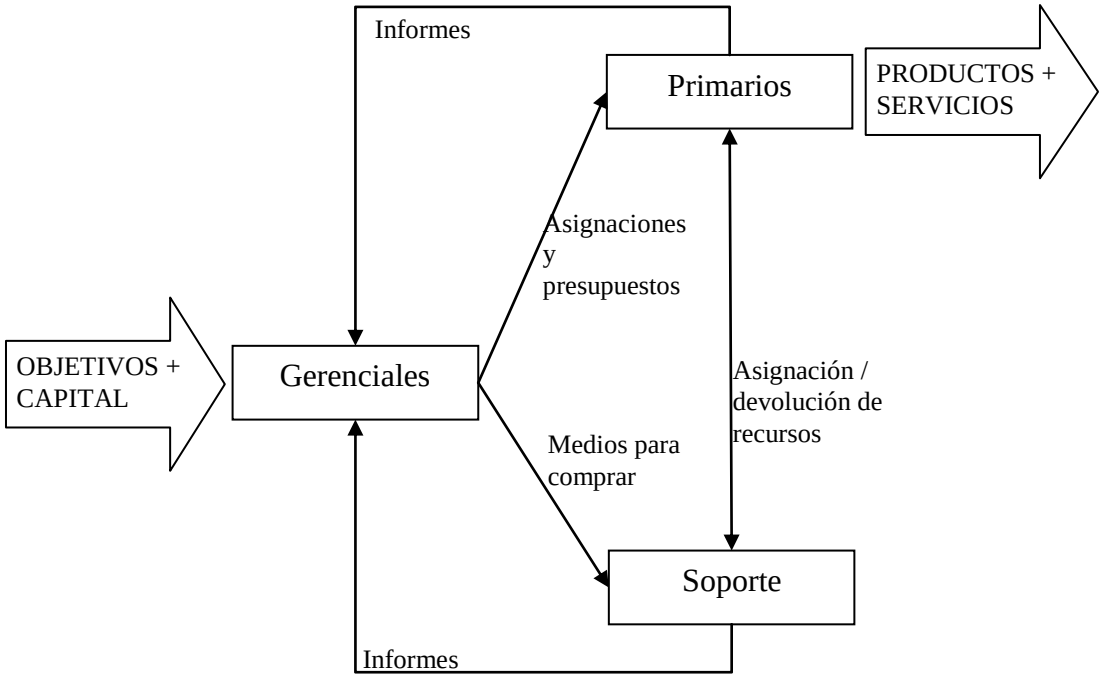


Figura 3.2. Relaciones entre tipos de procesos de negocio

Las personas que intervienen en los procesos, forman el conjunto de los actores. Se compone de los directores y los recursos. En el primer conjunto, se encuentran la gerencia y los clientes. Los primeros asignan las tareas necesarias para la elaboración del producto demandado por el cliente. Su relación es directa cuando el trabajo de los primeros redunda en el resultado demandado por los clientes. Si su vínculo es indirecto, es porque intervienen procesos secundarios o terciarios, que pueden ser delegados en los recursos. Esta cadena de delegaciones puede continuar hasta alcanzar a la persona adecuada para su realización. A su vez, las personas situadas en posiciones intermedias presentan un carácter dual, al ser a la vez directores y recursos. La decisión sobre el camino que siguen las delegaciones y quién se encarga de qué tarea se basa en la estructura jerárquica.

El organigrama organizativo relaciona, por tanto, directores y recursos. Por cada par de ellos relacionado, puede existir un contrato que especifique el producto deseado, el precio y la fecha de entrega. Cuando ambos son empresas grandes, es necesario formalizar su sistema de comunicación.

Un actor responsable de un proceso puede delegarlo como un todo, o bien descomponerlo en subprocesos y distribuirlos entre los recursos a su cargo. Este reparto genera lo que se denomina un árbol de contratación. Es posible aplicar a este árbol una cierta lógica, de manera que un tipo de caso concreto sea procesado por un recurso determinado. Para ello se requiere un diseño previo, donde se crea el árbol y se seleccionan los recursos. Este principio se aplica en el diagrama siguiente, donde los casos procesados se organizan en tipos clasificados de 1 a 8:

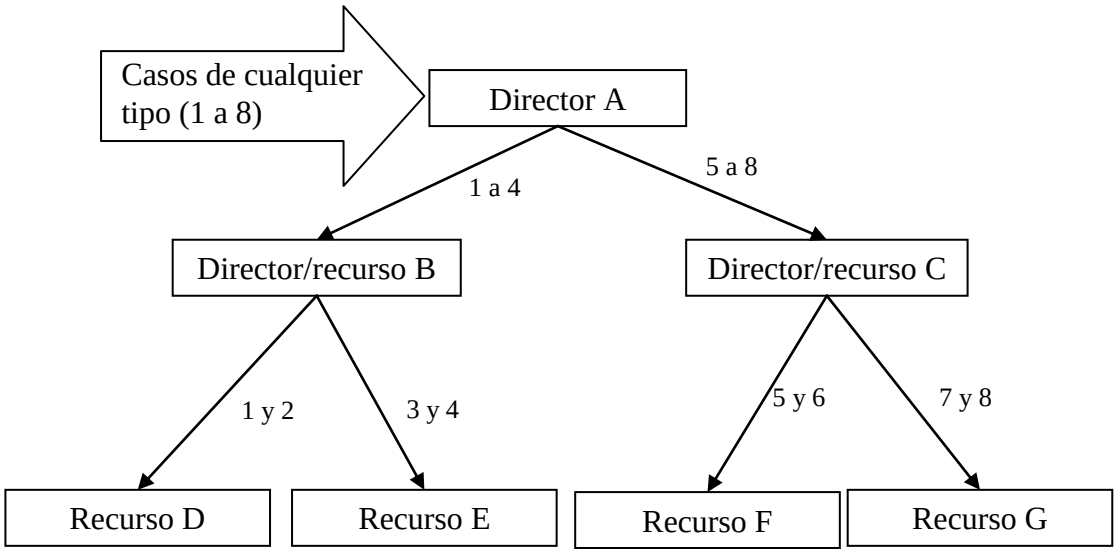


Figura 3.3. Ejemplo de delegación de procesos

Como se ha visto, la estructura organizativa juega un papel muy importante en la asignación de procesos a recursos agrupados según su cometido. Anteriormente, se ha mostrado una de tipo jerárquico, pero éste no es el único existente. A continuación, se analiza cada clase:

- *Organización jerárquica.*

  Es la más conocida y se distingue por su forma arbórea. Los nodos hoja indican un rol o una función individual. Por ello, se corresponden con conjuntos de empleados o departamentos. Los nexos indican que el actor de nivel superior tiene la capacidad de ordenar la relación de trabajos a las personas incluidas en el nodo inmediatamente inferior.

  También se da otro punto de vista en que la empresa se desgrana en función de la relación de pertenencia existente entre departamentos y recursos.

- *Grupos de capacitación.*

  Se trata de agrupar a las personas con una cualificación elevada, bajo un mismo departamento. Su encargado debe preocuparse de que no disminuya el nivel. Estos profesionales son cedidos a otras unidades de negocio. Un ejemplo es un grupo de ingenieros de mantenimiento.

- *Departamento funcional.*

  Define un conjunto interdependiente de tareas que requieren la misma cualificación. Dentro de este tipo se engloban secciones como marketing o departamento financiero.

- *Departamentos de proceso o producción.*

En este caso, el departamento es responsable de un proceso completo o de la elaboración de un producto.

Un director habitualmente no dispone de poder ilimitado, sino que las personas a su cargo suelen ser contadas. Bajo esta idea aparecieron las matrices organizativas. En ellas, los recursos son asignados en función de las tareas que es necesario realizar (columnas) y las capacidades de cada uno (filas). A continuación, se muestra un ejemplo, donde P_i, D_i y M_i representan a una persona con capacidad para programar, diseñar o maquetar, respectivamente.

		Tareas			
		Proyecto 1	Proyecto 2	Proyecto 3	Proyecto 4
Cualificación	Programación	P_1	P_2	P_3	P_4
	Diseño	D_1	D_2	D_3	D_4
	Maquetación	M_1	M_2	M_3	M_4

Tabla 3.1. Ejemplo de matriz organizativa

Puede suceder que algunos de estos profesionales sean capaces de diseñar y maquetar, por ejemplo. En ese caso se trataría de la misma persona desempeñando roles distintos.

La gestión de procesos distingue entre procesos de gestión y procesos gestionados. En primer lugar, es necesario definir el término sistema. Éste se refiere a personas, máquinas u ordenadores involucrados en la realización de un proceso. Un sistema gestionado puede ser dividido en un sistema de gestión de menor nivel y otro sistema gestionado. Esta separación puede aplicarse recursivamente. El sistema de menor nivel se denomina sistema de difusión. En el nivel más alto se encuentra siempre parte del sistema gestionado. Además un sistema de gestión puede manejar diversos sistemas gestionados, lo que garantiza que éstos puedan comunicarse con otro y con el mundo exterior a través del sistema de gestión de nivel superior. También existe comunicación entre el sistema de gestión y el gestionado. Esto permite al primero trasladar objetivos, precondiciones y decisiones. Como respuesta, el sistema gestionado remite informes. En base a esta retroalimentación, el sistema de gestión revisa los tres elementos, dando lugar al ciclo de planificación y control.

Existe una organización para la gestión de procesos, organizada en cuatro niveles. Se diferencian por la frecuencia y el ámbito de las decisiones tomadas. Por ámbito se entiende la unión del periodo de tiempo en que éstas tienen efecto y su impacto financiero potencial. La siguiente tabla resume los cuatro tipos existentes:

Tipo de gestión	**Frecuencia de decisión**	**Ámbito**	**Tipo de decisiones**
Tiempo real	De milisegundos a horas	Pequeñas consecuencias financieras	Control del equipo
Operacional	De horas a días	Consecuencias poco duraderas	Asignación de recursos

Táctica	De días a meses	Limitado	Planificación de la capacidad de los recursos y presupuesto
Estratégica	Una única vez o cada dos años a la sumo	Consecuencias duraderas (incluso años)	Diseño del proceso y de los tipos de recurso

Tabla 3.2. Clasificación de la gestión de procesos

Cada nivel tiene también la responsabilidad de controlar el correcto funcionamiento de las decisiones tomadas en las capas anteriores.

Finalmente, es importante destacar la importancia de una correcta toma de decisiones, basada en las fases de definición del problema, propuesta de soluciones, análisis de las mismas y la selección de una de ellas para su aplicación.

3.1.3 Sistemas de información para procesos de negocio

Como consecuencia de la complejidad de la organización del trabajo, se han introducido los sistemas de información computerizados para la gestión y coordinación de procesos. Atendiendo al rol desempeñado por el sistema dentro del proceso, surge la siguiente clasificación:

- *Sistemas de información de oficina.*

  Son aplicaciones que ayudan a los responsables de la plantilla en tareas básicas como escritura, realización de diagramas, archivo y comunicación. Este grupo contiene las hojas de cálculo, los procesadores de texto y el correo electrónico, por ejemplo. No disponen de ningún tipo de conocimiento acerca del proceso.

- *Sistemas de procesamiento transaccional.*

  Registran y comunican los cambios ocurridos en las circunstancias de los procesos. Algunos se especializan en la comunicación entre empresas diferentes, mediante el uso de estándares como XML. Poseen conocimientos básicos sobre el proceso, ya que son capaces de interpretar las transacciones entrantes y discernir dónde se almacenan los datos de entrada recibidos.

- *Sistemas de gestión del conocimiento.*

  Su cometido es la adquisición y distribución de conocimiento. Éste es manipulado para que pueda ser representado digitalmente. La realización más simple de este sistema es un maquina de búsqueda acoplada a un archivo de documentos. Habitualmente se usan repositorios de datos conectados a herramientas estadísticas. Un repositorio de datos almacena datos agregados, como el número de clientes procedentes de una determinada región que han comprado un producto durante un periodo en particular.

- *Sistemas con soporte para decisión.*

Las decisiones se toman a través de la interacción con las personas. Existen dos métodos para ello, utilizando modelos matemáticos (por ejemplo, para la planificación de los presupuestos y de la producción) o sistemas lógicos. Estos también son conocidos como sistemas expertos, como los empleados para dilucidar la causa de un defecto en una máquina.

- *Sistemas de control.*

Calculan e implementan decisiones de manera automática, apoyándose en los datos obtenidos del proceso. Ejemplos de este tipo son elementos de control de las condiciones ambientales o de realización de encargos de materias primas.

La evolución histórica del desarrollo de estos sistemas de información dio lugar a una serie de hitos. Su motivación fue la pretensión de extraer determinados componentes de la aplicación primitiva. El diagrama siguiente refleja el resultado final.

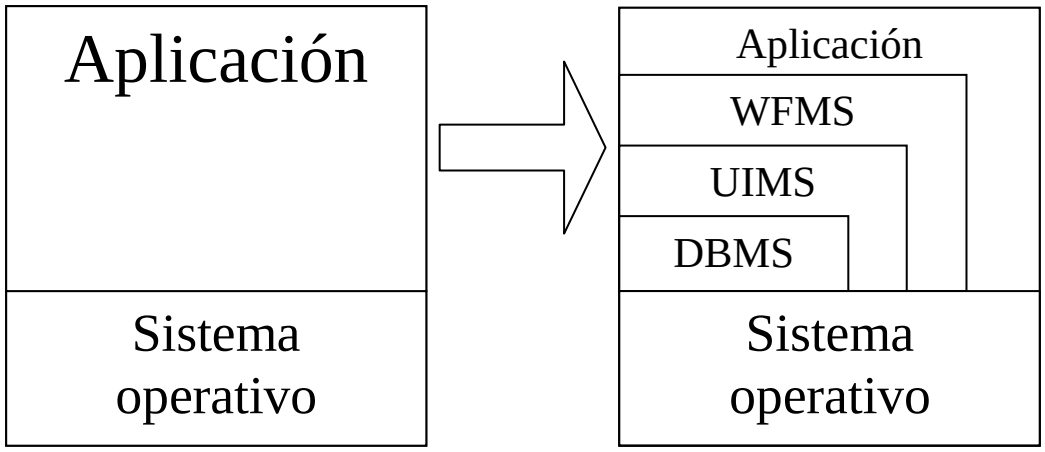


Figura 3.4. Evolución de los sistemas de información para procesos de negocio

En primer lugar (1965 a 1975), los sistemas de información se descompusieron en aplicaciones, cada uno con su base de datos y sin interfaz de usuario propia. El problema era que no era posible el intercambio de datos entre ellas.

Posteriormente (1975 a 1985), se introdujeron los sistemas de gestión de bases de datos (DBMS) con los que se logró extraer la información de las aplicaciones. De esta forma, los mismos datos eran accesibles para diversas aplicaciones.

La tercera fase (1985 a 1995) se caracterizó por la extracción de la interfaz fuera de las aplicaciones. Se dejó de dedicar tiempo en el desarrollo de las mismas, pasando a emplear los sistemas de gestión de interfaces de usuario (UIMS), caracterizados por una manera estándar para componerlas.

La última transición (1995-2005) consistió en el aislamiento de los procesos respecto a las aplicaciones. Era el último componente que quedaba por desprender de la aplicación, y los esfuerzos pudieron concentrarse en su desarrollo. Surgieron entonces los sistemas de workflow, capaces de trazar las rutas de los casos entre los recursos humanos mediante aplicaciones programadas. A su vez, un sistema de gestión de

workflow (WFMS) define y utiliza sistemas de workflow. Su gran ventaja estriba en la posibilidad de modificar procesos (por ejemplo, administrativos) sin requerir modificaciones posteriores sobre el software empleado. Con ello se favorece la reingeniería de procesos.

3.2 WORKFLOW CLÁSICO

3.2.1 Conceptos básicos

El objetivo primario de un sistema de workflow es el tratamiento de casos. Cada uno tiene una identidad única, que hace posible hacer referencia a él de manera unívoca. Además, tienen un tiempo de vida limitado, durante el cual permanecen en el sistema de workflow. A lo largo de su existencia dentro de él, exhiben un estado basado en tres componentes: sus atributos, las condiciones cumplidas y su contenido. Los atributos se emplean para gestionar el caso, por ejemplo para omitir ciertas tareas en determinadas circunstancias. Las condiciones (o fases) indican las tareas que han sido completadas y cuáles quedan por realizar. También se pueden considerar como requerimientos necesarios para iniciar una tarea. Por último, el sistema de workflow no suele mantener información sobre el contenido del caso, que puede registrarse en ficheros o bases de datos, que quedan fuera del ámbito del sistema de gestión de workflows.

Por su lado, las tareas son las unidades básicas de trabajo. En caso de que su ejecución resulte fallida, se debe retornar al punto inicial de la misma. Este comportamiento se denomina rollback. Si la tarea se relaciona con un caso en particular, entonces es necesario hablar de elemento de trabajo y actividad. El primero es una combinación de un caso y una tarea que está a punto de ser iniciada. En cambio, actividad hace referencia a la ejecución en curso de un elemento de trabajo. En síntesis, en el momento en que un elemento de trabajo comienza su ejecución, pasa a ser una actividad. El dibujo siguiente contiene esta especificación.

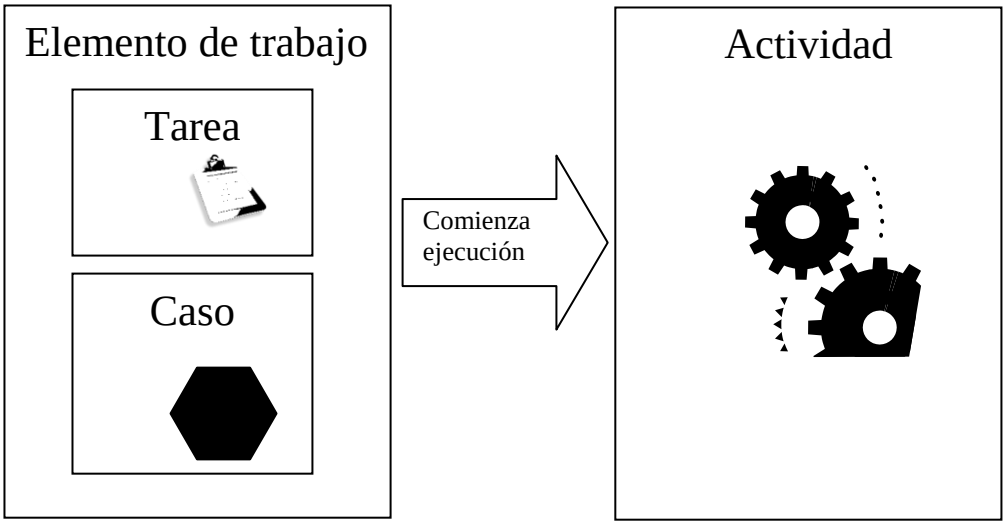


Figura 3.5. Elementos fundamentales del workflow

Un proceso indica el conjunto de tareas que deben ser realizadas. También se puede enfocar como un procedimiento para un tipo de caso determinado. En general, un mismo proceso es utilizado para manejar diversos casos. Por eso, es posible realizar

tratamientos concretos en función de los atributos de un caso particular. Las condiciones se emplean para determinar el orden de ejecución de las tareas.

El enrutamiento es el ciclo de vida seguido por un caso, definido por el proceso. Para su modelado, existen cuatro instrucciones básicas:

- *Ejecución secuencial*. Cada tarea es ejecutada cuando ha finalizado la anterior.
- *Ejecución paralela*. Se trata de dos tareas independientes que se necesita que estén completadas antes de proseguir.
- *Ejecución selectiva*. En este caso existe posibilidad de elección entre dos o más tareas. La opción se toma en base a los atributos del caso.
- *Ejecución iterativa*. Consiste en la repetición de una tarea mientras que una comprobación sea evaluada como válida.

En lo que afecta al desencadenamiento de la ejecución de los elementos de trabajo, éste puede requerir la aparición de un evento que lo propicie. Estos indicadores pueden deberse a la iniciativa de un recurso (por ejemplo, un empleado que toma sus tareas de una bandeja de entrada), por aparición de un evento (como la llegada de un mensaje de intercambio electrónico de datos) o por una señal temporal. Sin embargo, otros elementos de trabajo deben ser ejecutados inmediatamente, y, por tanto, no requieren de un evento desencadenante.

3.2.2 Modelado del workflow

Las Redes de Petri fueron introducidas por Carl Adam Petri. Él describió este sistema para la modelización y análisis de procesos. Proporciona una notación específica para representar cada componente interviniente en un proceso. A pesar de que existen otros sistemas de modelización, las Redes de Petri los aventajan en la eliminación de ambigüedades y contradicciones. Además, pueden ser usadas para realizar razonamientos sobre los procesos.

3.2.2.1 Las Redes de Petri clásicas

Los elementos básicos utilizados son las estaciones (círculos) y las transiciones (rectángulos). Son relacionados mediante arcos dirigidos, que unen siempre un lugar con una transición o a la inversa. Además, las estaciones contienen piezas (puntos negros), que representan objetos físicos o portadores de información. Una transición se habilita cuando todos sus lugares anteriores disponen de al menos una pieza. Cuando la transición se realiza, ésta consume piezas de entrada que son depositadas en la estación de salida. Por tanto, una transición representa un evento, una transformación, una operación o un trasvase. Las estaciones habitualmente representan un medio, un almacenamiento intermedio, una localización geográfica, un estado, una fase o una condición. A continuación se muestra una red de Petri típica que modela un sistema de reclamaciones:

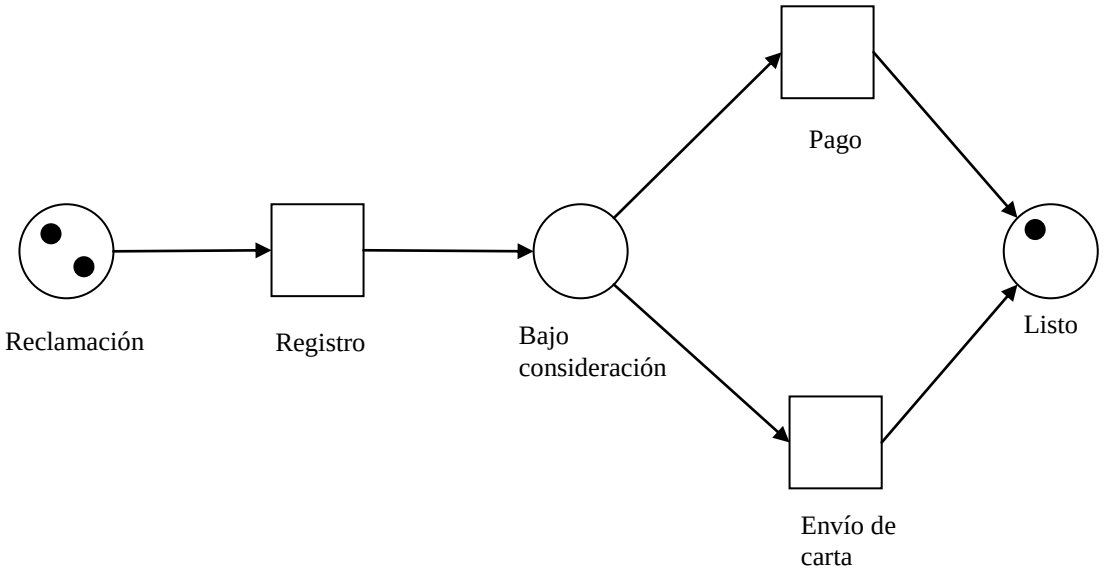


Figura 3.6. Red de Petri para un sistema de reclamaciones

3.2.2.2 Las Redes de Petri de alto nivel

A pesar de sus ventajas, las Redes de Petri también tienen inconvenientes. Principalmente se trata del tamaño inmanejable de los modelos y de la imposibilidad de representar actividades. Para solventar estos problemas, han aparecido las extensiones:

- *Extensión de color.*

Se utiliza para agregar características a las piezas, para así poder distinguirlas cuando se encuentren en la misma estación. En ello consiste "colorear" las piezas. También permiten establecer condiciones sobre los valores de las piezas en cada transición. De esta forma, para que se habilite una transición, todas las estaciones anteriores a ella deben contener al menos una pieza y las precondiciones deben haber sido satisfechas (por ejemplo, una pieza en una estación concreta debe tener un valor particular para poder ser consumido).

- *Extensión de tiempo.*

Esta extensión añade un control de la temporización de un proceso modelado. Esto se consigue estableciendo en cada pieza una marca temporal. Cada transición sólo se habilita cuando todas las piezas que aguardan su turno disponen de una marca temporal menor o igual al momento actual. Su orden de salida se determina según la forma FIFO (*first in, first out*), es decir, el primero que llega es el primero en salir. Si llegan a la vez, esto es, que tengan el mismo valor asignado a la marca temporal, el orden no se puede determinar a priori. En cada transición completada, se actualiza la marca de las piezas con un valor igual al momento de su ejecución más un retraso determinado.

Para ejemplificar este concepto puede pensarse en un semáforo cerrado. A cada vehículo se la asigna su instante de llegada más un margen adicional.

Cuando el instante actual supera la mayor de las marcas asignadas a los vehículos (correspondiente al último que llegó), es que ya se ha superado el periodo en rojo del semáforo, por lo que ya puede iluminar la luz verde y dejar paso.

- *Extensión jerárquica.*

Su finalidad consiste en agregar una dimensión jerárquica a los procesos modelados. El elemento empleado se denomina proceso y se representa mediante un cuadrado con doble borde. Representa una subred que emplea los mismos elementos que las de primer nivel: estaciones, transiciones, arcos y subprocesos.

Ante procesos complejos, se trata de aplicar la técnica de divide y vencerás, haciendo los procesos modelados más manejables. Incluso, presenta otra ventaja de gran importancia, que es la posibilidad de reutilizar procesos definidos con anterioridad. Así, un proceso establecido puede aplicarse como subproceso a otros de mayor orden. Por ejemplo, la colocación de las ruedas de un automóvil puede incluirse en el ensamblado del vehículo o en una operación de mantenimiento.

3.3 BUSINESS PROCESS MANAGEMENT (BPM)

3.3.1 Introducción

Al principio, [BROC10] se hicieron intentos por estabilizar el rendimiento del trabajo a través de la medición detallada de los resultados. A través de técnicas estadísticas, era posible aislar las causas básicas de los problemas de rendimiento. En los primeros estudios sobre el tema se establecieron una serie de conceptos fundamentales:

- Las operaciones tienen gran importancia y son merecedoras, por ello, de atención y gestión dedicadas.
- Se deben usar métricas para determinar si un trabajo desarrollado es satisfactorio.
- Las decisiones no se deben fundamentar en opiniones, sino en datos recogidos.
- En muchas ocasiones, malos resultados se deben a procesos ineficientes, no a los profesionales que intervienen en ellos.
- Las deficiencias en el rendimiento se pueden sintetizar en problemas concretos que se deben tratar.
- El proceso de mejora continua implica que la resolución de un problema no es la definitiva, sino una mejor base para afrontar el siguiente escollo.

Sin embargo, esta concepción presentaba dos limitaciones. La primera es la definición de proceso como una secuencia de actividades. Con ello, había que manejar una multitud de procesos, lo que dificultaba la adquisición de una visión estratégica de empresa y la gestión de los mismos de una manera uniforme. Por otra parte, el objetivo final es un rendimiento consistente, es decir, sin fallos de ejecución. Ello no implica que se logre el nivel de eficiencia requerido.

El siguiente paso fue la introducción de la reingeniería. Las ventajas que aportó fueron dos fundamentalmente. Primero, el cambio de punto de vista, pasando a percibir el proceso como una secuencia de extremo a extremo. Con ello, se permitía a la reingeniería mejorar los puntos débiles de la fragmentación: retrasos, pérdida de valor añadido, errores y complejidad. Todos estos males tienen su caldo de cultivo cuando el trabajo se reparte entre distintas organizaciones, con diferentes prioridades, fuentes de información y métricas. La segunda mejora fue el cambio de filosofía debido a la concesión de una mayor importancia al diseño del proceso frente a su ejecución. La fase de diseño es realmente donde se puede mejorar su rendimiento.

3.3.2 Características de BPM

Estos dos enfoques generales han sido unidos para generar la especificación moderna de BPM. Esto es un sistema integrado para la gestión de la eficiencia operacional a través de la administración de extremo a extremo de procesos de negocio.

Su ciclo de vida se basa en la necesidad de una gestión de los procesos de negocio vistos como un todo. En ellos se crea un producto demandado por el cliente.

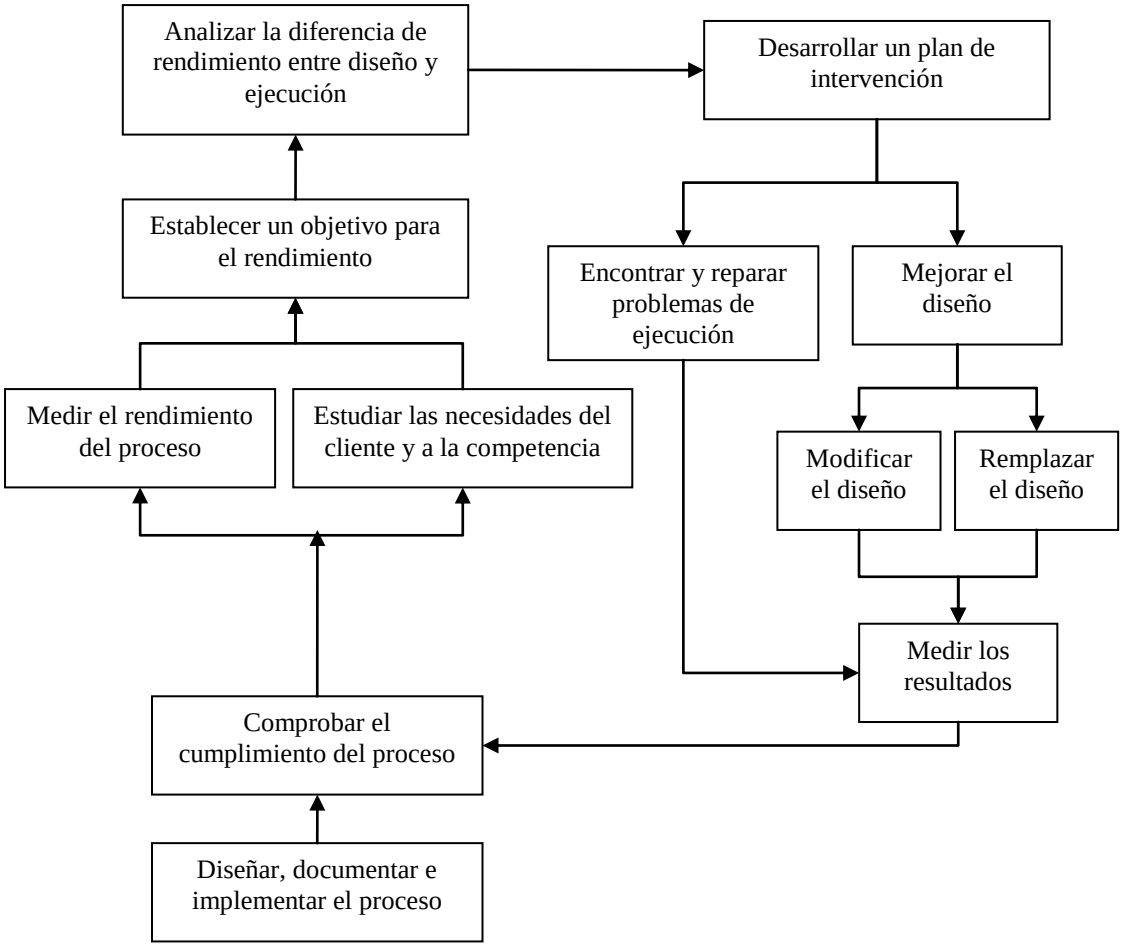


Figura 3.7. Ciclo de vida de BPM

En muchos casos, el diseño formal del proceso, paso inicial del ciclo de vida mostrado, es improvisado. Esto provoca que las empresas se vean en la obligación de realizar cambios sustanciales sobre las operaciones que realizan.

Una vez cumplida esta tarea, se deben establecer unos baremos que ayuden a medir la eficiencia del proceso, como pueden ser expectativas del cliente, análisis de la competencia o necesidades de la empresa, y, a partir de ellos, definir los objetivos.

Si este objetivo no se alcanza, se tienen que analizar las causas. Sus orígenes posibles son dos: el diseño y la ejecución. Los debidos a problemas en la ejecución son causadas por situaciones particulares (por ejemplo, falta de formación, equipos o recursos). Dada su gran diversidad son difíciles de determinar. Por otra parte, los

provocados por un diseño deficiente son repetitivos, con lo que son fácilmente detectables, mientras que son difíciles de corregir, ya que requieren una reorganización profunda de los procesos. Cuando se han aplicado las acciones necesarias, se debe medir su eficacia. Tras ello, comenzará una nueva iteración del ciclo.

La consecuencia principal de la aplicación de BPM es la creación de procesos empresariales de alto rendimiento. Ello redunda en la reducción de los costes y en el aumento de la velocidad y de la flexibilidad. A la vez, se aprovechan mejor los recursos disponibles. Todo ello, redunda en una mayor satisfacción del cliente.

3.4 MODELIZACIÓN DE PROCESOS DE NEGOCIO

3.4.1 Introducción

Un proceso [HAVE05] es definido primeramente en lenguaje natural. Un desarrollador de software extraerá de este texto un algoritmo, formado por una serie de pasos (actividades o tareas) con condiciones, bucles y ejecuciones paralelas. Finalmente, todo ello se traduce en un diagrama de flujo. Hipotéticamente, podría existir una máquina que ejecutara instancias definidas por un modelo expresado de una manera semántica y sintácticamente correcta.

La modelización de procesos de negocio aporta los siguientes beneficios:

- *Formalización de los procesos existentes y detección de mejoras necesarias.*

  Con ello, se consigue por una parte mejorar la comprensión de los procesos. Además, destaca posibles necesidades de mejora, como la eliminación y automatización de pasos, o de reingeniería del proceso.

- *Aplicación de un flujo de proceso automatizado y eficiente.*

  Los procesos se dividen en un conjunto de actividades. Por ello, es deseable perder el menor tiempo posible entre una y otra. La existencia de un software especializado permite vincular las tareas de la forma más eficiente posible, proponiendo incluso ejecuciones paralelas.

- *Aumento de la productividad.*

- *Concentración de las personas en los problemas reales.*

  Aunque esta filosofía tiende a reducir la participación humana en la administración de los procesos, mantiene flexibilidad a la hora de permitir su intervención para corregir problemas.

- *Simplifica los problemas con las regulaciones y su cumplimiento.*

  Mediante la modelización es posible definir procesos sobre los que se pueden realizar auditorías. Gracias a este aporte, las compañías pueden comparar el gasto debido a su complimiento frente a los beneficios derivados del control sobre los procesos.

Esta modelización sólo se adapta a aplicaciones con un sentido muy profundo de estado o proceso. Éstas se denominan aplicaciones orientadas a proceso, caracterizadas por una ejecución durante un tiempo indefinido, la persistencia del estado en una base de datos, la existencia de tiempos de espera hasta la aparición del siguiente evento desencadenante y la orquestación de comunicaciones entre sistemas o humanos.

3.4.2 Lenguajes de modelado

3.4.2.1 BPEL (Business Process Execution Language)

Este estándar se basa en XML para la representación de procesos. Fue propuesto por la organización OASIS, que está formada por las principales compañías de la computación, como Microsoft, Sun, IBM o DELL.

Consiste en una extensión para los estándares actuales de servicios web, con el fin de que puedan manejar procesos. Primeramente, los servicios web se limitaban a comunicaciones sin estado. No obstante, con la introducción de BPEL se ha conseguido construir procesos de negocio con estado y que interactúan entre sí, a partir de servicios web. Una definición de un proceso basada en BPEL se compone de dos elementos, que unidos consituyen un flujo de control de negocio. Éste puede actuar como una interfaz para elementos externos mediante los servicios web:

- *Ficheros WSDL (Web Service Definition Language).*

  Básicamente especifican la interfaz de los servicios web, así como de aquéllos implementados o invocados por los procesos.

- *Ficheros BPEL.*

  Se trata de la definición del proceso en formato XML. Especifican las actividades principales que lo forman, variables y manejadores para errores y eventos.

El código fuente contendrá un fichero BPEL y uno o varios WSDLs. Es desplegado en una máquina de ejecución BPEL, que supervisa la ejecución de la lógica del proceso.

3.4.2.2 BPMI (Business Process Modeling Initiative)

Se trata de una organización orientada a la creación de estándares y de una arquitectura común para BPM. Está compuesto por diversas empresas y, a su vez, forma parte de otras asociaciones, como OASIS, W3C u OMG. Sus aportaciones se centran en las siguientes propuestas:

- *BPMN (Business Process Modeling Notation).*

  Consiste en un lenguaje gráfico para la representación a través de diagramas de flujo de procesos de negocio. Su función principal es el diseño de los diagramas de procesos de negocio. Además, esta especificación provee un método para la conversión de BPMN en BPEL. De esta manera, el diagrama puede convertirse en una entidad ejecutable.

- *BPML (Business Process Modeling Language).*

  Se trata de un lenguaje XML que codifica el flujo de un proceso de negocio. El resultado puede ser interpretado por una máquina de ejecución de procesos. Describe la representación estructural del proceso y la semántica de su ejecución.

El código BPML contiene bucles, condiciones, ejecuciones paralelas, variables y manejo de excepciones, elementos todos ellos familiares para los desarrolladores.

- *BPSM (Business Process Semantic Model).*

  Se trata de la respuesta de BPMI a la solución creada por OMG para definición de modelos de procesos de negocio basada en MOF (*Meta-Object Facility*).

- *BPXL (Business Process Extension Layers).*

  Se trata de un conjunto de extensiones para BPEL, de manera que pueda incluir transacciones, reglas de negocio, gestión de tareas e interacción humana.

3.4.2.3 The Workflow Management Coalition (WfMC)

Su importancia se ha visto reducida en los últimos años por la extensión de BPEL. No obstante, introduce ideas interesantes sobre el modelado de procesos de negocio. Sus propuestas se sintetizan en los siguientes puntos:

- *XPDL (XML Process Definition Language).*

  Es el lenguaje de definición de procesos creado por WfMC. En su arquitectura, XPDL es la interfaz entre las herramientas de definición del proceso y la publicación del servicio. En concreto, los procesos diseñados son exportados en formato XPDL y cargados en un servicio de difusión para su ejecución.

  También se puede utilizar para trasladar definiciones entre distintas herramientas, a través de operaciones de importación o exportación con un XPDL como origen o destino, respectivamente.

- *WAPI (Workflow Management Application Programming Interface).*

  Se trata de una API ofrecida por el servicio de publicación. Puede ser empleada por aplicaciones de diversos tipos: cliente, invocadas o de administración y monitorización.

- *WfXML*.

  Es la denominación del protocolo de publicación de WfMC, construido sobre un modelo de servicios web. Ofrece un mecanismo para el control remoto y la monitorización de flujos con un periodo de ejecución largo.

3.4.2.4 World Wide Web Consortium (W3C)

Muchos autores diferencian entre los términos orquestación y coreografía, referidos a servicios web. Ambos implican coordinar la creación de servicios web individuales que colaboran en un proceso que los engloba. Por convención, la orquestación se refiere a la coordinación aplicada a un proceso con un único participante. Mientras, la coreografía hace referencia a una visión global, comprendiendo múltiples participantes.

El W3C se ha centrado en la coreografía y en sus lenguajes de modelado. Con ello, se pretende crear procesos con estado, conversacionales, de larga duración y con múltiples participantes, a partir de operaciones atómicas basadas en servicios web, básicas y sin estado.

El lenguaje BPEL aísla a cada participante y documenta su proceso interno y sus puntos de interacción. Sin embargo, desde el punto de vista global de la orquestación, únicamente importa el protocolo, es decir, las interacciones existentes entre los distintos participantes.

- *WSCDL (Web Services Choreography Description Language).*

  Se trata de un lenguaje basado en XML. Su finalidad es especificar contratos técnicos que capturan desde un punto de vista global el comportamiento común de los participantes autónomos que interactúan.

  Los participantes de los procesos pertenecen habitualmente a distintas compañías. Por ello, usan servicios web para comunicarse con los demás, y publican sus interfaces a través de WSDL.

  No existe un control central para gestionar las reglas de interacción. Cada participante tiene su propio control que ejecuta su parte del proceso. En cualquier caso, éste debe regirse por las reglas impuestas por la coreografía.

- *WSCI (Web Services Choreography Interface).*

  Su sintaxis y su semántica son similares a las descritas en BPML. Sin embargo, no se trata de una tecnología orientada a los procesos de negocio. Extiende WSDL con un lenguaje para la creación de procesos conversacionales con estado, a partir de servicios web atómicos sin estado. Por tanto, los servicios web son la parte sustancial de WSCI, mientras que para el resto de lenguajes son una tecnología sustituible por otra si no existiera.

  Los componentes integrantes se describen a continuación. En primer lugar, un documento WSDL define las interfaces del servicio web que va a ser coordinado. Una interfaz WSCI especifica el modo en que el WSDL es utilizado, habitualmente desde el punto de vista de uno de los participantes. Únicamente muestra el comportamiento público observable, dado que se orienta hacia la coreografía. Finalmente, el último elemento de un modelo global WSCI consiste en el intercambio de mensajes global. Está formato por un conjunto de conexiones que vinculan cada operación de un participante con la operación complementaria de otro participante.

- *WSCL (Web Services Conversation Languages).*

  Consiste en un estándar que nunca ha alcanzado popularidad entre los proveedores. Es una manera simple de implementar la coreografía, basándose en el concepto de las máquinas de estado.

3.4.3 Modelización de procesos web con WebML

Se trata [WEBM12] de un lenguaje de modelización gráfico de procesos propuesto por el departamento de Electrónica e Información del Politécnico de Milán. Sus objetivos principales son:

- Realizar una descripción a alto nivel de la estructura de una aplicación web, que facilita su mantenimiento y evolución.
- Generar múltiples vistas para un mismo contenido. Para ello, se provee una separación entre la información mostrada y las páginas, la navegación y la presentación.
- Recoger información surgida durante la fase de diseño del proceso, con la idea de aplicar posteriormente en la generación dinámica de páginas web.
- Definir usuarios y comunidades que permitan personalizar la aplicación según determinadas políticas.
- Posibilitar la especificación de operaciones que permitan la actualización del contenido del sitio y la comunicación con servicios externos.

En la siguiente imagen se puede apreciar el proceso sugerido para el uso de WebML.

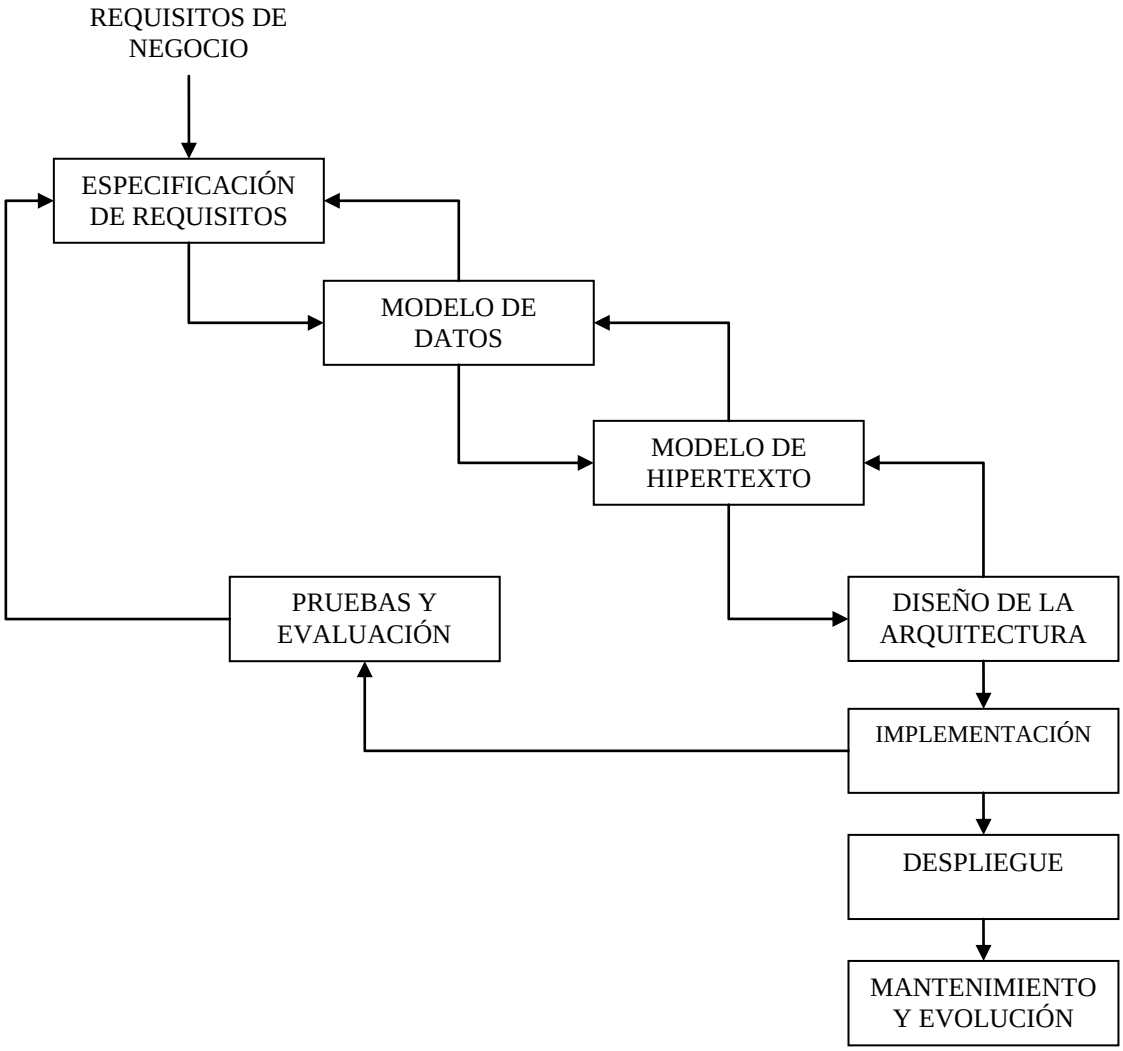


Figura 3.8. Ciclo de aplicación de WebML

Esta secuencia cíclica respeta el conjunto de pasos propuestos ya por los sistemas de gestión de procesos primitivos, añadiendo nuevas fases:

- *El modelo de datos.*

  Se asemeja al modelo entidad-relación de una base de datos o a un diagrama de clases UML. Se incluyen en él las entidades (contenedores de información) con sus propiedades. Es posible establecer entre ellas vínculos de generalización, dando lugar a una estructura jerárquica. Además, también se muestran las relaciones entre ellas, que pueden estar restringidas por una cardinalidad determinada.

  WebML incorpora una serie de reglas útiles para que el modelado de datos se adapte de la mejor manera posible a su explotación en una aplicación web.

- *El modelo de hipertexto.*

  Indica la composición y la navegación a través del sitio. Según su punto de vista, las páginas albergan unidades de contenido que se relacionan con algunas

de las entidades vistas en el modelo anterior. Por su parte, la navegación es provista por los hipervínculos. Estos pueden ser de dos tipos, contextuales o no, dependiendo de que porten algún tipo de datos sobre la navegación.

Debe establecerse en este paso la información circunstancial que permita determinar en qué casos son visibles unos datos concretos (por ejemplo, para determinados roles de usuarios) y cómo se relaciona esta información con entidades del modelo de datos.

- *El modelo de presentación.*

En esta fase se define el aspecto y el comportamiento de las páginas de un sitio web. Dado que WebML es representable mediante XML, el proceso de transformación se guía por un sistema de transformación de documentos. La meta final es la creación de páginas en un lenguaje concreto, como JSP o ASP .NET, usando para ello hojas de estilo XSL.

Capítulo 4. Patrones y frameworks

Estos dos elementos son fundamentales para el desarrollo de software. Por una parte, los patrones son soluciones prácticas para problemas típicos que pueden surgir en el diseño e implementación de programas. Respecto a los frameworks, encapsulan funcionalidades complejas, comunes en la creación de aplicaciones. De esta forma, se reduce el esfuerzo que se requiere para llevar a cabo una tarea recurrente y tediosa.

Debido a estas ventajas, conviene analizar aquellas soluciones que pudieran adoptarse en la implementación del sistema final. Esta clasificación vendrá dada por dos características fundamentales del sistema: la interactividad con el usuario y el mantenimiento de la información.

4.1 PATRONES EN INGENIERÍA DE SOFTWARE

4.1.1 Introducción

Cuando los desarrolladores [BUSC01] se enfrentan a un problema particular, tratan de recordar una solución ya aplicada en otra ocasión. Con ello, se pretende recuperar la esencia del camino seguido para, apoyándose en ella, generar una solución válida para el nuevo problema. A partir de los pares problema-solución que se van registrando, es posible observar factores comunes a los problemas y a las soluciones. Estas parejas pueden ser, por tanto, clasificadas en familias que presentan problemas y soluciones semejantes y que exhiben un patrón en ellos.

Los expertos en Ingeniería del Software conocen los patrones a través de la experiencia, esto es, de una manera no artificial. Los respetan para el desarrollo de aplicaciones con unas propiedades específicas. Con su utilización se alcanzan soluciones efectivas y elegantes para problemas de diseño recurrentes. Además, cada uno de ellos provee un vocabulario propio que facilita su comprensión. Gracias a él, se evita la necesidad de redactar una larga descripción de un problema con la solución aplicada. Simplemente, indicando la denominación del patrón e aclarando qué papel juega cada componente dentro de él, es posible alcanzar un alto grado de conocimiento del sistema desarrollado.

Los patrones facilitan la construcción de arquitecturas software heterogéneas. Cada uno de ellos es usado para implementar una parte concreta de la estructura, y puede ser empleado para formar un diseño más complejo. No obstante, en ningún caso se proporciona una solución completamente detallada. Se trata de un esquema que debe ser completado según los requerimientos. La ventaja primordial de esta táctica es que mejora el manejo de la complejidad del software. Con su aplicación, no es necesario invertir tiempo en pensar la manera de lograr la meta. El camino que se debe seguir es indicado por el patrón.

4.1.2 Componentes de un patrón

En la sección anterior ya se han mencionado alguno de los elementos que constituyen un patrón. Estos integrantes son: contexto, problema y solución. El primero de ellos define la situación que da pie a la aparición del problema.

Concretamente, este esquema visto como un conjunto establece la relación entre un contexto, el problema que surge en dicho contexto y la solución apropiada para su tratamiento.

- *El contexto.*

  Suele ser definido en términos generales. Por ejemplo, un contexto es el ámbito del desarrollo de software que presenta una interfaz hombre-máquina. A su vez, un contexto puede englobar distintos patrones.

Es difícil contemplar todas las situaciones en que un patrón puede ser aplicado. Un enfoque más pragmático consiste en listar situaciones conocidas donde el problema cubierto por el patrón pueda aparecer.

- *El problema.*

Esta parte de la especificación del patrón describe el problema que se detecta repetidamente en el contexto indicado. Comienza con una reseña general que explica su esencia, esto es, el aspecto concreto del diseño que se desea solventar.

Además también se señalan las fuerzas. Es un término proveniente de la arquitectura, que denota cualquier aspecto del problema que debe ser tenido en cuenta para su resolución, como requisitos, restricciones y propiedades deseables que la solución debería presentar. Un ejemplo de estas imposiciones es la necesidad de un sistema eficiente de comunicación entre procesos basado en un protocolo determinado. Las fuerzas requieren el análisis del problema desde distintos puntos de vista y ayudan a comprender sus detalles. En ocasiones, puede ocurrir que sean contradictorias, como el deseo de un código compacto unido a la necesidad de un sistema extensible basado en superclases. Por ello, es necesario calibrar la solución para dar respuesta a todas las fuerzas de la mejor forma posible. En situaciones extremas, el diseñador puede verse obligado a dejar determinadas fuerzas sin solución.

- *La solución.*

Se trata de la forma de resolver el problema, con un tratamiento ponderado de las fuerzas asociadas a él.

En primer lugar, describe la configuración espacial de los elementos que lo componen. De esta forma, una aplicación interactiva es dividida en las zonas de procesamiento, entrada y salida. Por tanto, consiste en la especificación del aspecto estático de la solución, que está formada por elementos y las relaciones entre ellos.

Por otra parte, cada patrón debe resumir el comportamiento que el sistema debe mostrar durante su ejecución. Comprende, por tanto, su parte dinámica, es decir, la manera en que los participantes colaboran y se comunican entre sí.

La gran ventaja de la solución propuesta es que es aplicable a un gran número de implementaciones que comparten su esencia.

4.1.3 Tipos de patrones

Un análisis detallado de los patrones existentes revela que cubren distintos niveles de abstracción. Estos incluyen desde la estructuración general del sistema hasta la implementación de aspectos de diseño en un lenguaje de programación concreto. También se organizan según la independencia del dominio que presenten, por ejemplo, desde componentes interactivos desacoplados hasta patrones para políticas de aplicaciones empresariales. De esta variedad se derivan tres categorías:

- *Patrones arquitectónicos.*

Expresan la organización estructural fundamental de sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluye reglas y guías para la gestión de las relaciones entre ellos.

Son plantillas para arquitecturas software concretas. Indican las propiedades globales de tipo estructural de una aplicación e influyen en la arquitectura de los subsistemas. Por tanto, la selección de un patrón arquitectónico es fundamental para el desarrollo de un sistema.

El patrón arquitectónico más conocido es el denominado Modelo-Vista-Controlador, al que se hará referencia en capítulos posteriores.

- *Patrones de diseño.*

Habitualmente, los subsistemas que conforman una arquitectura software y las relaciones entre ellos consisten en unidades arquitectónicas menores cuya especificación viene dada por los patrones de diseño. Es, por tanto, un esquema para el refinamiento de los componentes de un sistema y de las relaciones existentes entre ellos. Describe una estructuración de la comunicación entre componentes que soluciona un problema de diseño frecuente en un contexto concreto.

Muchos de ellos proporcionan estructuras capaces de descomponer servicios o elementos más complejos. Otros logran que la cooperación entre ellos se lleve a cabo de una manera efectiva.

A su vez, los patrones de diseño se clasifican en tres grupos [GAMM95], que, a su vez, se distinguen según se basen en la herencia de clases o en la composición de objetos para lograr sus fines:

  - *Creacionales*.

    Los patrones de este tipo realizan una abstracción del proceso de instanciación. Concretamente, hacen que el sistema sea independiente de la creación, composición y representación de los objetos.

    Con ello, se busca la definición de un conjunto de comportamientos fundamentales que pueden ser conjugados para la generación de uno más complejo. Estos patrones ocultan el conocimiento de las clases empleadas y su instanciación y unión de los objetos declarados.

  - *Estructurales.*

    Su acción se aplica a la composición de clases y objetos, unidos para formar estructuras más complejas.

    En lugar de emplear la composición o la herencia de clases e interfaces, estos patrones describen la capacidad de componer objetos para

obtener una nueva funcionalidad. Para ello, se basan en la posibilidad de alterar una composición en tiempo de ejecución.

- *De comportamiento.*

  Estos patrones están vinculados con los algoritmos y la asignación de responsabilidades entre objetos. Además de describir clases y objetos, también se especifican las relaciones entre ellos. Permiten abstraer el control de flujo en tiempo de ejecución para que el desarrollador se centre en la manera en que los objetos están interconectados.

- *Dialectos.*

Consiste en una práctica estándar vinculada a un lenguaje de programación. Describen cómo implementar aspectos particulares de los componentes o de las relaciones entre ellos usando las características de dicho lenguaje.

Muchos de ellos son específicos para un lenguaje de programación. Es más, puede ocurrir que respondan a características distintivas de los lenguajes. Por ejemplo, no tiene sentido aplicar en Java, con su recolector de basura, una gestión dinámica de referencias a memoria aplicado a C++ originalmente.

Los componentes de los patrones y las relaciones no son atómicos, como en un principio parecen ser. Por ello, la aplicación de un patrón se orienta a la resolución de un problema, pero da pie a la aparición de otros nuevos que serán, a su vez, tratados mediante otros patrones. En conclusión, los elementos y relaciones albergados por un patrón serán recursivamente descritos por nuevos patrones, todos ellos integrados en el de mayor nivel.

4.1.4 Patrones más relevantes

En esta sección, se detalla la definición de algunos patrones útiles para lograr la meta de este trabajo. Concretamente, se trata del patrón arquitectónico Modelo-Vista-Controlador (MVC), del patrón de diseño Cadena de Responsabilidad y del patrón DAO. El primero se ha escogido por su importancia dentro de las aplicaciones interactivas. Por ello, está presente en la mayoría de desarrollos web de la actualidad. Por su parte, el patrón Cadena de Responsabilidad está enmarcado en la organización del trabajo. En consecuencia, puede relacionarse con los procesos y su gestión. Por último, el patrón DAO especifica la manera de encapsular el acceso a distintas fuentes de información. Adicionalmente, se mencionan los patrones de workflow.

4.1.4.1 Patrón arquitectónico Modelo-Vista-Controlador (MVC)

Este patrón arquitectónico [BUSC01] se centra en las aplicaciones interactivas con una interfaz hombre máquina flexible. El problema a que hace frente surge por la proclividad de las interfaces a sufrir modificaciones, como añadir nuevas opciones a un menú o modificar el aspecto de una pantalla. Son cambios que requieren modificaciones en el código. Además, usuarios distintos pueden preferir métodos distintos para insertar (teclear o pulsar botones e iconos) y visualizar la información (en forma de texto plano, tabulada o enmarcada en un informe).

Por este motivo, se debe evitar que la interfaz de usuario conviva con el núcleo funcional de una manera acoplada. En caso contrario, todas las interfaces de usuario distintas requerirán un subsistema complejo propio.

Las fuerzas que intervienen en este problema son:

- Se desea que una misma información sea mostrada en distintos formatos.
- La apariencia y el comportamiento de la aplicación deben exhibir los cambios de manera inmediata.
- Se requiere que los cambios a la interfaz de usuario sean rápidos, preferiblemente en tiempo de ejecución.
- El cambio de aspecto o la sustitución de la interfaz no debería implicar la alteración del código que gestiona el funcionamiento interno del desarrollo.

La solución propuesta divide la aplicación interactiva en tres áreas: procesamiento, entrada y salida. Los componentes que participan en él son los siguientes:

- *El modelo*.

  Contiene la funcionalidad básica y los datos. Es independiente de representaciones particulares de la salida o comportamientos concretos en la entrada de datos.

- *La vista*.

  Muestra la información al usuario, extraída del modelo. Pueden existir diversas vistas para el mismo modelo.

- *El controlador*.

  Maneja las peticiones del usuario. Cada vista dispone de su propio controlador. Éste recibe una interacción del usuario (por ejemplo, un movimiento del ratón o la pulsación de una tecla). Esta petición es traslada al modelo o a la vista.

El siguiente diagrama muestra las relaciones existentes entre ellos. Si la flecha es punteada, indica que es posible que exista un elemento intermedio encargado de la comunicación (indicado por un color más tenue).

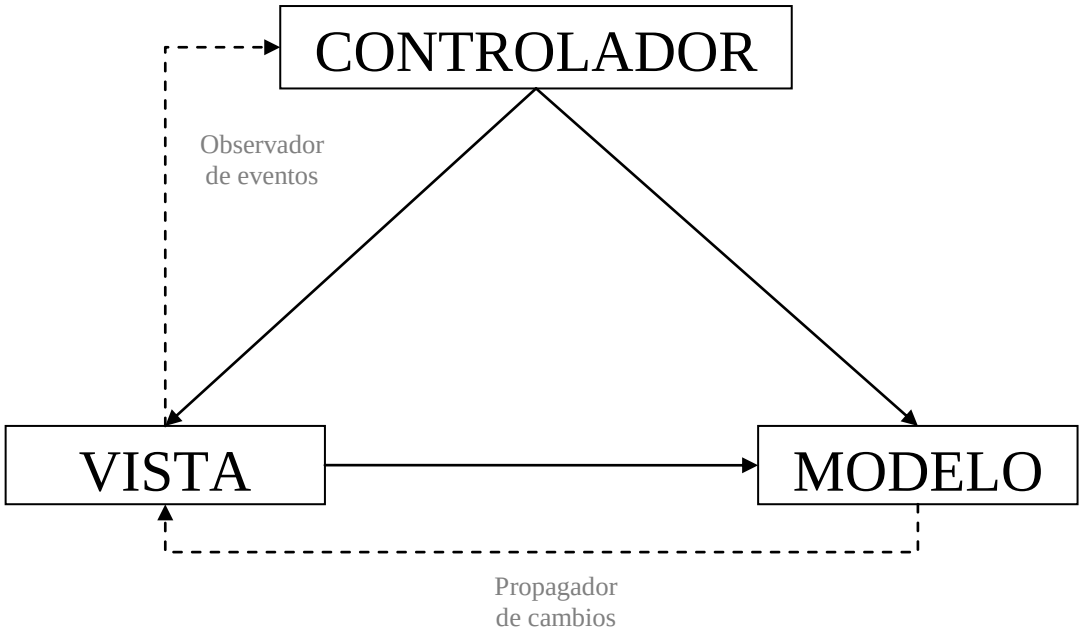


Figura 4.1. Representación del patrón Modelo-Vista-Controlador

El proceso que se genera entre ellos sigue la siguiente secuencia:

1. El controlador acepta peticiones a través de su manejador de eventos, identifica el evento acaecido e invoca el servicio adecuado del modelo.

2. El modelo realiza la operación requerida, que provoca una modificación en su estado.

3. Dependiendo de la implementación, existen dos opciones para esto paso. La primera consiste en que el modelo notifica a sus vistas relacionadas la modificación en su estado a través de un mecanismo de propagación de cambios. En otros casos, es el controlador quien se encarga de invocar a la vista adecuada, que recupera directamente del modelo la información que necesita.

4. Cada vista recupera los datos modificados del modelo y los muestra en pantalla.

5. El controlador vuelve a retomar el control de la situación.

4.1.4.2 Patrón de diseño Cadena de responsabilidad

La finalidad de este patrón [GAMM95] consiste en evitar el acoplamiento existente entre una petición y el destinatario de la misma. Posibilita que más de un objeto sea el responsable de su manipulación. Encadena dichos objetos, a través de los cuales va discurriendo la petición hasta que uno de ellos es capaz de gestionarla. El conjunto forma la denominada cadena de sucesión.

Un ejemplo es la gestión de la ayuda vinculada a los elementos de una pantalla. Si se solicita una explicación sobre un elemento (por ejemplo, un botón), existen dos opciones: que exista un elemento capaz de atender esa petición y mostrar la ayuda pedida, o bien que la petición sea trasladada a otro objeto de nivel superior, que gestiona las solicitudes sucedidas sobre el componente que contiene al anterior (por ejemplo, una ventana). Finalmente, tendrá que existir un participante de carácter generalista que sea capaz de atender todas las solicitudes. Entre éstas se incluyen tanto las destinadas específicamente a él, como las que no han podido ser atendidas anteriormente. Su respuesta a todas ellas consistirá, en el ejemplo, en una ayuda global de la aplicación.

Es por ello que este patrón puede aplicarse en las siguientes circunstancias:

- Más de un objeto puede manejar una petición. A priori se desconoce cuál será el encargado, que será elegido automáticamente.

- Se desea remitir una petición a varios objetos sin especificar su destinatario de forma explícita.

- El conjunto de elementos que pueden manejar una petición debe ser especificado dinámicamente.

La estructura se basa en un manejador base que únicamente recibe las peticiones y es capaz de trasladar una petición a su sucesor. Está relacionado, opcionalmente, con otro manejador. De este manejador fundamental heredan los manejadores específicos. Éstos ya cuentan con operaciones para el tratamiento de las solicitudes destinados a ellos.

4.1.4.3 Patrón Data Access Object (DAO)

Este patrón [DAO11] se aplica al contexto de acceso a datos, que varía según su localización. Este acceso depende del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objeto, ficheros de texto, etc.) y de la implementación del proveedor, especialmente cuando se trata de un sistema persistente. Debido a ello, existen variaciones entre las interfaces proporcionadas para emplear estos mecanismos de almacenamiento persistente. En consecuencia, si los componentes que realizan la comunicación con los sistemas de persistencia incluyen código de conexión y acceso a datos, surge un fuerte acoplamiento entre ellos. El resultado es un código confuso, que requiere cambios profundos si se desea que sea capaz de emplear otras fuentes de información.

La solución pasa por la aplicación de un objeto de acceso a datos (DAO) que encapsule cualquier acceso a la fuente de datos. Es por tanto dicho DAO el encargado de manejar la conexión con el origen de la información y de obtener y almacenar los datos.

Este patrón define los siguientes participantes:

- *El cliente de los datos.*

  Es el objeto que requiere el almacenamiento o la obtención de información.

- *El DAO.*

  Es el objeto principal del patrón. Oculta al cliente la implementación de acceso a datos, para así proporcionar un acceso transparente a la fuente de la información.

- *El origen de los datos.*

  Se trata de un objeto que abstrae a la fuente de la información. Ésta puede ser una base de datos relacional u orientado a objeto, un repositorio XML, un conjunto de ficheros de texto, otro sistema externo, un servicio (como una pasarela de pago) o un repositorio.

- *El objeto de transferencia.*

  Consiste en un objeto creado para portar la información recuperada desde el DAO al cliente. También es empleado por el cliente para indicar al DAO qué datos es necesario actualizar en la fuente de información.

Respecto a las estrategias que se pueden adoptar para su implementación, existe una especialmente interesante, basada en la generación automática de código. La base sobre la que se sustenta es la correspondencia existente entre un cliente y un DAO específico. El generador automático de código puede basarse en un fichero descriptor para crear el DAO, o, incluso, indagar en la base de datos y proporcionar el DAO adecuado para el acceso a ella. También se puede aprovechar la automatización para la implementación de propiedades como el cacheo de consultas y resultados, la integración con servidores de aplicaciones o la comunicación con otros componentes.

4.1.4.4 Patrones de workflow

Cabe reseñar también los patrones de workflow [WORK11]. Se trata de una iniciativa cuya finalidad es proveer una base conceptual para la tecnología de los procesos. Éstos se enfocan desde diversos aspectos que deben ser soportados por los lenguajes de workflow y de modelado de procesos de negocio. Estos puntos de vista [AALS03b] son:

- *Control de flujo.*

  Describe las actividades y su orden de ejecución. Éste se especifica a través de elementos que permiten el tránsito del control de la ejecución, por ejemplo, secuencial o paralelamente.

- *Datos.*

  La ejecución de las actividades viene determinada por su precondición y su postcondición. Están compuestas por los documentos que son trasladados entre las distintas actividades, así como por las variables locales al flujo.

- *Recursos.*

Se provee una estructura organizativa vinculada al proceso. En ella se especifican los roles responsables en la ejecución de las tareas, asignados a personas y dispositivos.

- *Operaciones.*

Básicamente definen la manera en que las acciones elementales ejecutadas por las actividades se comunican con la aplicación que sustenta el proceso. Especialmente, se trata del paso de información a través de interfaces específicas. Éstas también permiten la manipulación de los datos en la aplicación.

4.2 FRAMEWORKS

4.2.1 Introducción

Se denomina framework [FRAM11] a un conjunto de código o librerías que provee una funcionalidad común a un tipo de aplicación. Mientras una librería proporciona unas operaciones específicas, los frameworks ofrecen un rango más amplio. Éste cubre todos los requerimientos particulares de la clase de desarrollo que se trate.

Su ventaja principal es que evita la necesidad de recodificar aspectos comunes a todas las aplicaciones del tipo elegido. Además, se trata de código que ha sido usado y probado por una comunidad numerosa de programadores, lo que garantiza su fiabilidad. Todo ello redunda en la necesidad de un menor esfuerzo y en un código con una tasa reducida de errores. Por último, la implantación de los frameworks puede ayudar al mantenimiento de buenas prácticas, como la estructuración MVC.

Su uso también presenta una serie de inconvenientes, como la reducción del rendimiento. Puede ocurrir que el framework, diseñado para proveer soluciones globales, no esté optimizado para una situación concreta. También suelen ser complejos de utilizar y aplicar mejoras sobre ellos. Finalmente, los problemas derivados de errores y fallos de seguridad afectan a todas las aplicaciones construidas sobre ellos.

4.2.2 Frameworks más relevantes

En esta sección se repasan las características de los frameworks que pueden ser útiles para lograr los objetivos de este trabajo.

4.2.2.1 Struts

Se trata de un framework [STRU11] consistente en una solución de código abierto para la creación en Java de aplicaciones web. Éstas se caracterizan por la capacidad de generar una respuesta dinámica, construida mediante una lógica de negocio y consultas a una base de datos. Su finalidad es ayudar al desarrollador en la construcción de aplicaciones web que cumplan el patrón MVC. Para ello, emplea tres componentes fundamentales:

- Un manejador de peticiones (request) que es relacionado con una URL concreta.
- Un manejador de respuestas (response) capaz de transferir el control a otros recursos que implementen la respuesta.
- Una librería de etiquetas (tags) para la creación de aplicaciones interactivas basadas en formularios.

Antes de explicar sus características, conviene repasar los elementos que integran una aplicación web de tipo Java [ROUG07]:

- *Servlets.*

Proveen una manera de vincular una URL a una clase especial, cuyos métodos son invocados. En su momento, se reconoció el avance que supusieron en el desarrollo web. Pero pronto se comprobó que la generación de un documento HTML a través de código Java era muy problemático. Cada vez que se requería una modificación, era necesario recompilar y desplegar la aplicación en el servidor.

- *JSP y desarrollo de scriptlets.*

Dado el problema descrito, se llevó a cabo un cambio en la forma de programar. En lugar de enmarcar el código HTML en un *servlet*, el código Java fue trasladado al interior de un documento HTML, a través de *Java Server Pages* (JSP).

Sin embargo, esta solución presentaba un problema similar, ya que no permitía una estructura de métodos o clases Java.

- *Frameworks basados en acciones.*

Se trata de una combinación de *servlet* y JSP. De esta manera, se separa el procesamiento de las peticiones en dos partes: lógica de negocio y presentación.

Con ello, se logra la implementación del patrón MVC, de manera que el *servlet* es el controlador. A través de él, se relaciona una URL con una operación denominada acción. Ésta realiza una funcionalidad específica accediendo a distintos parámetros disponibles. A continuación, especifica el resultado a través del modelo, consistente en una clase Java estándar. Éste resultado es enviado por la acción a la página JSP, que es la vista.

- *Frameworks basados en componentes.*

A medida que las aplicaciones web se hicieron más complejas, se comprobó que la página había dejado de ser la unidad básica. Las páginas comenzaron a contener un gran número de elementos: formularios, enlaces y otros componentes, cada uno con su propia lógica para llevar a cabo sus tareas.

Por eso, su popularizaron estos frameworks. Proveen una relación entre los componentes de la interfaz de usuario y las clases que los representan y gestionan sus eventos.

En el diagrama siguiente se pueden observar los componentes participantes en este framework y las relaciones entre ellos.

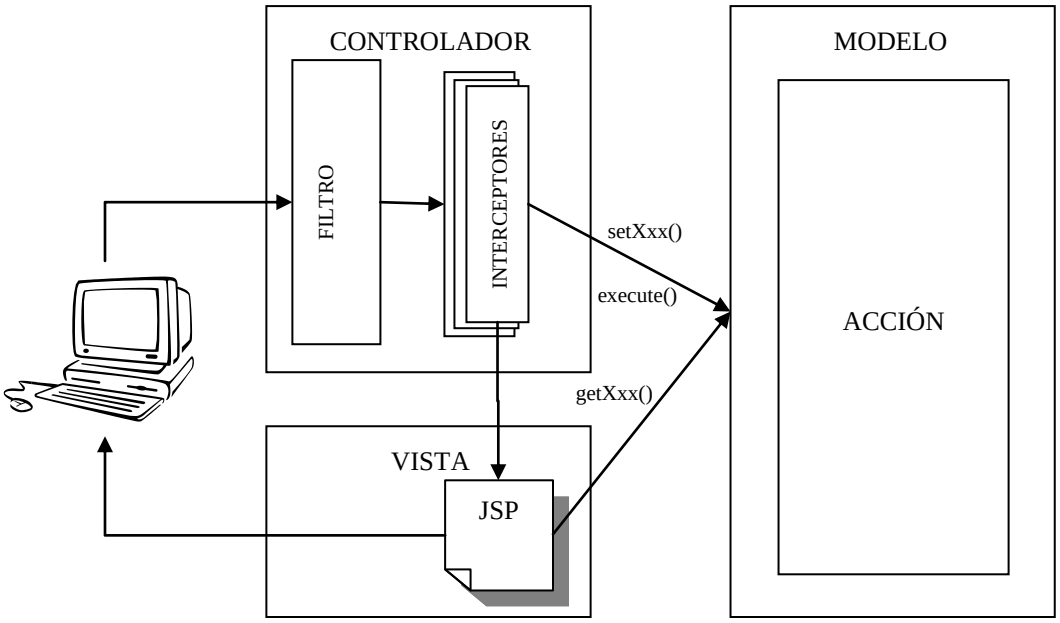


Figura 4.2. La arquitectura de Struts

Los interceptores, no mencionados hasta ahora, se encargan de diversas funcionalidades del framework. Entre ellas, destacan el manejo de excepciones, la carga (*upload*) de ficheros, funciones de devolución de llamada (*callbacks*) de ciclo de vida y validación.

Respecto a los métodos de traslado de datos, eran muy frecuente el uso del objeto *HttpServletRequest* (petición) y *HttpSession* (sesión). Struts introdujo una tercera vía a través de la llamada pila de valores. Este elemento es capaz de almacenar, en el orden en que aparecen: objetos temporales, el objeto modelo, el objeto acción y diversos objetos establecidos. Gracias a ella, es posible acceder directamente a datos del modelo mediante etiquetas incluidas en los ficheros JSP.

Entre las ventajas que aporta la aplicación de este framework, se encuentran las siguientes:

- *Separación de conceptos.*

  Estos conceptos son la lógica de la acción, el acceso y la obtención de los objetos de negocio, la conversión de tipos, la traducción de código HTML para ser trasladado a la acción y otros conceptos globales responsabilidad de los interceptores. Para el tratamiento de los objetos de negocio se basa en la inyección de dependencia, gracias a la cual, para tener objetos disponibles en la acción, sólo hay que proveer una función modificadora (*setter*).

- *Bajo acoplamiento.*

  Éste puede ser ejemplificado por la asignación de URLs a acciones, los diversos valores de retorno (de tipo cadena de texto) de una acción que pueden

vincularse a distintas páginas y la relación de las excepciones con una página de error.

- *Facilidad para la realización de pruebas unitarias.*

  Para verificar la corrección de la lógica de la acción, simplemente es necesario comprobar que el valor de retorno del método *execute()* y el estado final sean los adecuados tras su ejecución.

  La comprobación de los interceptores es más compleja, y para llevarla a cabo es necesario el empleo de *proxies*.

- *Modularización.*

  Ésta puede acometerse desde diversas vertientes, como son el reparto de la información de configuración entre distintos ficheros, la creación de aplicaciones modulares y el desarrollo de nuevas propiedades del framework, éstas dos últimas contenidas en *plug-ins*. Además, Struts acepta elementos de configuración adicionales requeridos por éstos.

- *Convención sobre configuración.*

  Heredada de Ruby on Rails, es una idea que se ha introducido para equilibrar la flexibilidad de este framework y la dificultad de su configuración. A través de las convenciones, se propicia una mayor productividad del desarrollador.

4.2.2.2 Hibernate

La finalidad de este framework [HIBE11] es lograr la persistencia. Con ella, los datos de las aplicaciones sobreviven a los propios procesos de dichas aplicaciones, empleando para ello una base de datos relacional.

Hibernate nació con el objetivo de facilitar el almacenamiento y obtención de objetos de dominio de tipo Java a través de un mapeo objeto/relacional. Actualmente, permite a los desarrolladores emplear modelos de dominio basados en POJO para aplicar el modelo objeto/relacional. Este acrónimo significa *Plain Old Java Object*, esto es, clases Java simples no asociadas a ningún framework particular.

En resumen, la meta última de Hibernate [MINT06] es el almacenamiento de POJOs en una base de datos. Para ello, emplea JDBC (*Java Database Connectivity*), que es un componente que facilita el acceso a una base de datos relacional. El siguiente diagrama muestra cómo se integra Hibernate en una aplicación Java.

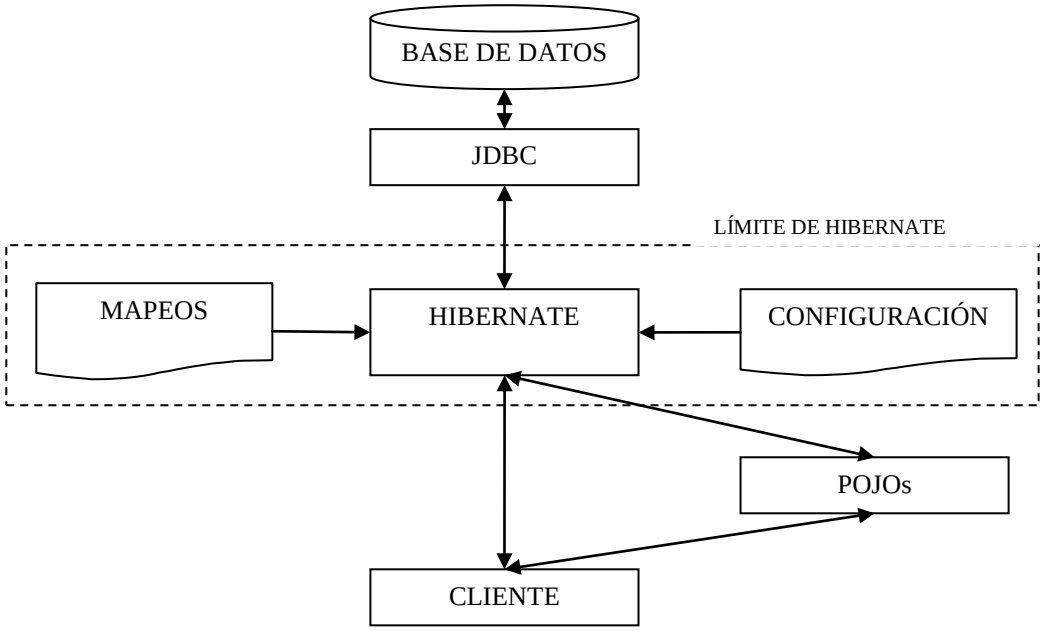


Figura 4.3. La arquitectura de Hibernate

La gran ventaja de Hibernate es que logra evitar el problema denominado *Object-Relational Impedence Mismatch*. Éste consiste en la dificultad de aunar objetos y bases de datos relacionales, caracterizada por cinco aspectos:

- *Granularidad.*

  En ocasiones, se tendrá un modelo de objetos que incluye más clases que el número de tablas disponibles (por ejemplo, una dirección).

- *Herencia.*

  La herencia es una práctica habitual en la programación orientada a objetos. Sin embargo, no se encuentra definida en el ámbito de las bases de datos relacionales de una manera totalmente estándar.

- *Identidad.*

  Las bases de datos relacionales poseen un concepto de igualdad único, basado en las claves primarias. Por su parte, Java define la identidad de objetos (dos variables que hacen referencia al mismo objeto en memoria) y la igualdad de objetos (dos objetos cuyo contenido es idéntico).

- *Asociaciones.*

  En los lenguajes orientados a objetos, las asociaciones se representan a través de referencias unidireccionales. En el ámbito de las bases de datos relacionales se habla de claves ajenas. Por ello, para implementar una relación bidireccional ésta debe ser especificada dos veces.

También existe el inconveniente de que no es posible determinar la multiplicidad (número de elementos vinculados entre sí) de una relación analizando el modelo de objetos de dominio.

- *Navegabilidad.*

En los lenguajes orientados a objeto el paso entre elementos se realiza a través de las relaciones entre ellos. Sin embargo, este proceso no es eficiente. Lo ideal sería especificar qué entidades se desea recuperar para luego formar la consulta adecuada, antes de recorrer el entramado de objetos.

Hibernate soporta la construcción de un POJO a partir de una selección de columnas de distintas tablas. De manera análoga, distintos POJOs pueden ser persistidos a través de una misma tabla.

Para la persistencia de los objetos, Hibernate implementa un ciclo de vida caracterizado por los estados *transient* (transitorio), *persistent* (persistente), *removed* (borrado) y *detached* (desprendido). Este ciclo se refleja en la imagen siguiente.

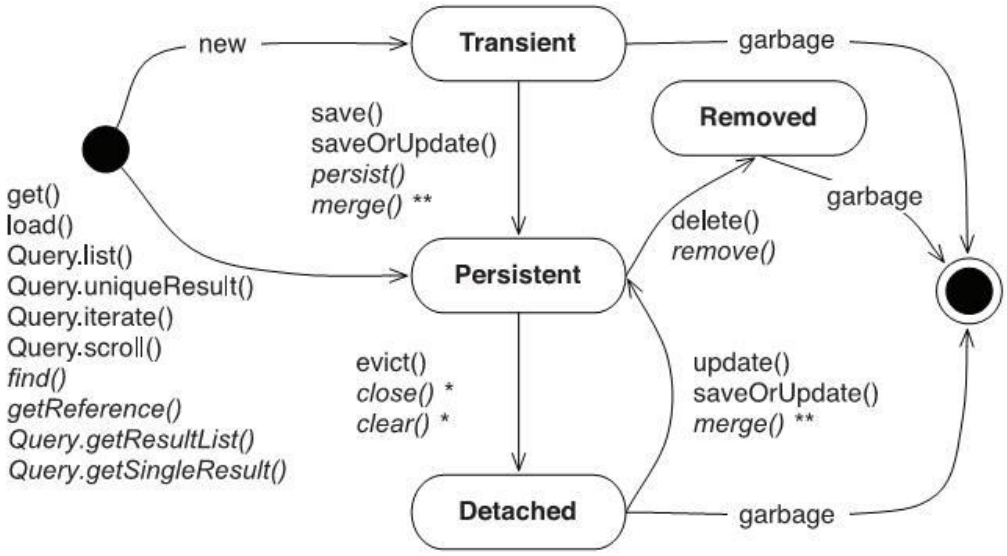


Figura 4.4. Ciclo de vida de los objetos en Hibernate

Por último, cabe señala que Hibernate dispone de un lenguaje de consultas orientado a objetos propio, denominado HQL. Permite extraer (operación *SELECT*) y modificar (operaciones *INSERT*, *UPDATE* y *DELETE*) la información contenida en la base de datos. Es similar a SQL, con la diferencia de que en lugar de tratar con tablas y columnas, emplea objetos Java y sus atributos. El propio framework se encarga de traducir las sentencias HQL, aunque también provee la posibilidad de especificar directamente consultas SQL.

4.2.2.3 Spring

Se trata de un framework [MINT08] que nació para solucionar el problema de la inyección de dependencias. Como normal general, este concepto no supone un inconveniente. Si se requiere que unas clases dependan de otras, simplemente se

especifica en el código. Sin embargo, existen determinados casos en que sí es problemático. Concretamente, cuando una clase depende de otra clase que puede presentar implementaciones alternativas o si esa clase externa contiene datos que pueden cambiar en el futuro. En estas circunstancias, sí se requiere de un componente que sea capaz de gestionar estas dependencias externas, permitiendo trasladar la especificación de sus detalles de configuración de la etapa de compilación a la de ejecución.

Sin embargo, Spring ha ido creciendo. Actualmente, proporciona un soporte potente para la programación de aplicaciones web integrales. De cara a este trabajo, son potencialmente interesantes los componentes [SPRI11] Spring MVC, Spring Web Flow y Spring Portlet MVC.

- *Spring MVC.*

  Su filosofía es similar a la aplicada al framework Struts.

- *Spring Web Flow.*

  Permite modelar el comportamiento de una aplicación web en términos del flujo seguido a través de distintos estados. Concretamente, se trata de los caminos que siguen los usuarios en un sitio web y que definen su comportamiento. Por ejemplo, el siguiente diagrama especifica el flujo especificado en una operación de creación de usuarios.

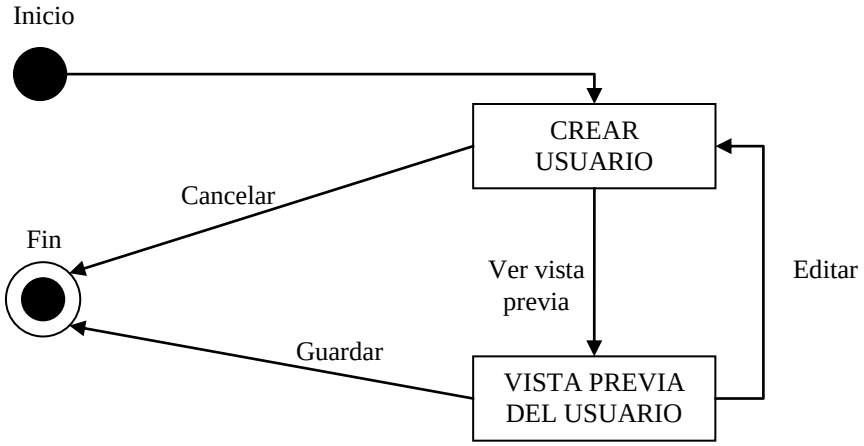


Figura 4.5. Flujo para el proceso de creación de usuarios

Spring Web Flow invoca la lógica de negocio apropiada según el usuario se va desplazando por la aplicación web. En cada transición, se traslada una petición a la acción adecuada. Éstas pueden ser llamadas cuando tienen lugar los siguientes eventos:

- Cuando empieza el flujo.
- Cuando el flujo alcanza un estado.

- Cuando el flujo va a avanzar hacia otro estado (o hacia el mismo en que se encontraba).
- Cuando el flujo está a punto de formar la vista asociada a un estado (útil para el rellenado de un formulario a partir de los datos referenciados).
- Cuando el flujo termina.

A su vez, existen otros eventos que ocurren cuando una operación no es exitosa, por ejemplo, durante la validación de un formulario.

- *Spring Portlet.*

Los portales permiten la creación de aplicaciones web extensas mediante un conjunto de subcomponentes que son ubicados en la misma página. Además, en ocasiones deben proveer capacidades estructurales como la autenticación y la autorización.

Spring Portlet consiste básicamente en una versión de Spring MVC enmarcada en un entorno orientado a la generación de portales.

Capítulo 5. Trabajos relacionados

La modelización del workflow de formularios web a través de BPMN es una idea que ya ha sido explorada e implementada en diversos productos disponibles en el mercado.

Este capítulo está dedicado a la revisión de las propuestas conocidas a través de artículos o mediante la experimentación directa. Con este análisis, se pretende contrastar el cumplimiento de los objetivos iniciales de este trabajo sirviéndose de estas herramientas.

5.1 ARTÍCULOS PUBLICADOS

El artículo [FREU07] se refiere también al modelado de workflows de formularios web. Se propone en él un lenguaje específico de dominio compuesto por tres componentes:

- *Modelo de dominio específico.*

  Especifica el esquema que deben respetar todos los desarrollos que empleen dicho lenguaje. Este elemento se ha sustentado sobre XPDL.

- *Modelo de interacción de dominio.*

  Su función es proporcionar una representación gráfica de los procesos. Su notación podría ser derivada de BPMN o de las Redes de Petri.

- *Bloque de construcción de la solución.*

  Es una máquina virtual capaz de ejecutar códigos escritos en este lenguaje. Aparte de los procesos especificados, que se pueden considerar como la configuración básica, acepta una personalización en detalle mediante el uso de ficheros XML.

La aplicación de este lenguaje fomenta la aparición de una serie de fases en el desarrollo de los formularios:

- *Modelado del proceso de negocio.*

  En esta fase se involucran a los principales usuarios del sistema.

- *Modelado del worklow.*

  El objetivo es indicar detalles específicos del sistema. Para ello, se apoya en un conjunto de elementos fundamentales, para la gestión de la comunicación, la obtención de los datos y el diálogo con servicios web.

- *Diseño físico y ejecución.*

  El producto de este paso es un prototipo ejecutable, con la especificación del proceso y sus detalles.

- *Evolución.*

  Este sistema permite realizar actualizaciones sobre el formulario resultante, tanto a nivel de diseño de workflow, como de configuración específica.

Las ideas proporcionadas por este artículo son muy interesantes de cara al desarrollo del presente trabajo. No obstante, tiene una orientación distinta, pues se fundamenta en la existencia de un elemento software encargado de la definición HTML final del formulario y los elementos circundantes. Con ello, se podrá encapsular en dicho elemento la adaptación a las peculiaridades de cada plataforma. Sin embargo,

surge un inconveniente, debido a que el resultado de la modelización es una serie de ficheros XML difíciles de manejar y modificar. Estos serán procesados por un programa que requerirá un gran trabajo de procesamiento por parte del servidor, y será una fuente de errores difíciles de depurar y corregir. Además, no se respeta completamente la estructura de la arquitectura MDA. Aunque no se indique explícitamente, se puede deducir la aparición de un CIM y un PIM, pero se hace innecesaria la derivación de un PSM y del código fuente.

Aunque no trata exactamente el tema de este trabajo, el interés del artículo [ESCO11] estriba en la mención del lenguaje específico de dominio denominado WebDSL [WEBD12]. Fue concebido principalmente para la modelización de la validación de la información y el control de acceso. Se han producido tentativas para ampliarlo [HEME08], de manera que cubriera el ámbito de los formularios basados en workflow. El problema es que es un lenguaje muy potente, lo que provoca que sea muy complejo. Por otra parte, se trata de un componente diseñado para Java, por lo que está fuertemente acoplado a una plataforma.

Por último, también se ha revisado el artículo [RYGG06]. En él se experimenta GPFlow. Se trata de una herramienta destinada a formar procesos en que intervienen un conjunto de programas. De esta manera, se forma una cadena, estableciendo la entrada de cada módulo como la salida del anterior. Se trata, por tanto, de un punto de vista generalista, que permite un abanico más amplio de actividades, más allá de la mera inserción de información en un formulario, como es el caso de este trabajo.

5.2 IMPLEMENTACIONES EXISTENTES

5.2.1 WebRatio

WebRatio [WEBR11] es un programa destinado a la modelización de procesos. Estos modelos pueden ser implementados en un prototipo y documentados mediante diversos formatos. Es distribuido en distintas versiones, unas orientadas hacia un uso básico y otras destinadas a su aplicación empresarial. En las secciones siguientes se desgranarán las características de las primeras de ellas. Las que tienen como objetivo el mundo empresarial queda fuera del ámbito de este trabajo, aparte de que no se dispone de documentación pública sobre su funcionamiento.

En primer lugar, existe una versión básica (WebRatio BPM Free). Emplea el estándar BPMN 1.2 para la representación de los procesos. A partir de esta información, el sistema permite generar una documentación en los formatos HTML, PDF o RTF. Otra opción es la posibilidad de transformar el modelo en un documento XPDL o bien, partiendo de uno en este formato, crear su diagrama BMPN equivalente. Por último, es capaz de generar un sitio web cuyos procesos son codificados partiendo de la especificación. La gran ventaja de esta propiedad estriba en la posibilidad inmediata de verificar la corrección de los procesos diseñados.

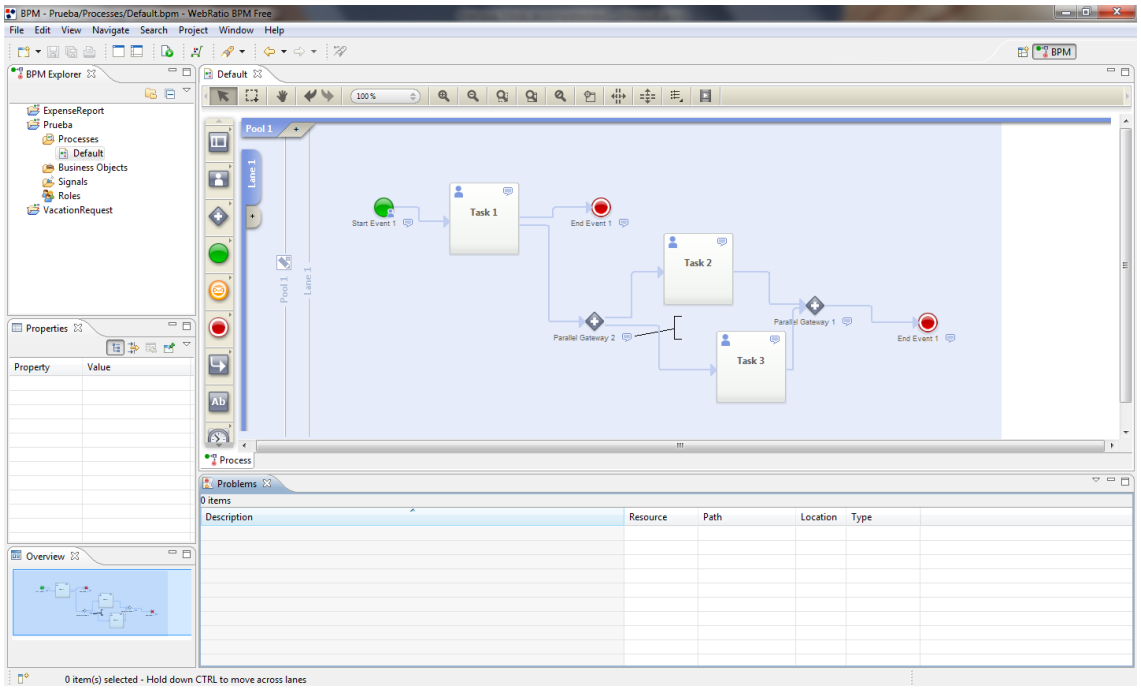


Figura 5.1. Aspecto de WebRatio BPM Free

En cada tarea que compone el proceso, es posible indicar qué campos aparecen en el formulario anexo a ella.

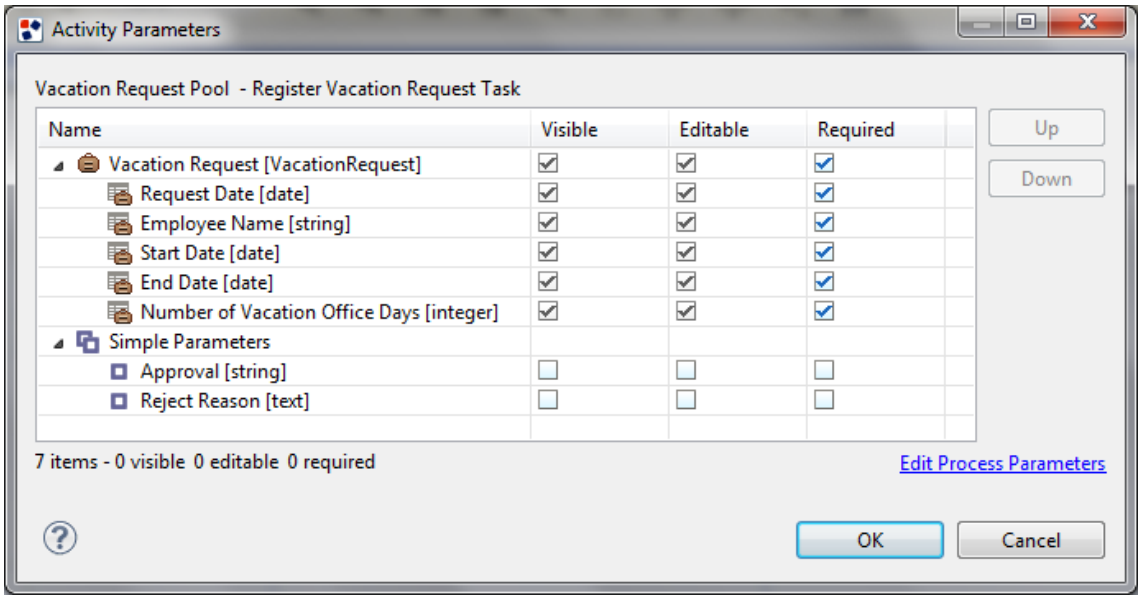


Figura 5.2. Especificación de los campos de un formulario en WebRatio BPM Free

Respecto a la versión de pago (WebRatio Personal), aparte de las propiedades aportadas por la edición gratuita, incorpora la posibilidad de emplear el estándar WebML. Además, dispone de un enfoque basado en desarrollo orientado a modelos para la generación de aplicaciones personalizadas.

En lo que atañe al empleo de esta herramienta como respuesta al objetivo planteado, cabe realizar las siguientes puntualizaciones:

- *No es código abierto.*

  Este producto no está disponible en una distribución en que sea posible la modificación de su código interno. Este extremo no facilita la investigación y experimentación sobre su funcionamiento original, con el fin de comprenderlo e intentar mejorarlo.

- *No es posible personalizar la aplicación resultante.*

  Aunque sea posible generar una aplicación a partir del modelo, la versión gratuita no permite la especificación de nuevas reglas de transformación. Por ello, una modificación de aspecto, por ejemplo, tendría que ser aplicada sistemáticamente a cualquier resultado generado. Sin embargo, esta salida se ve bloqueada por la propiedad especificada a continuación.

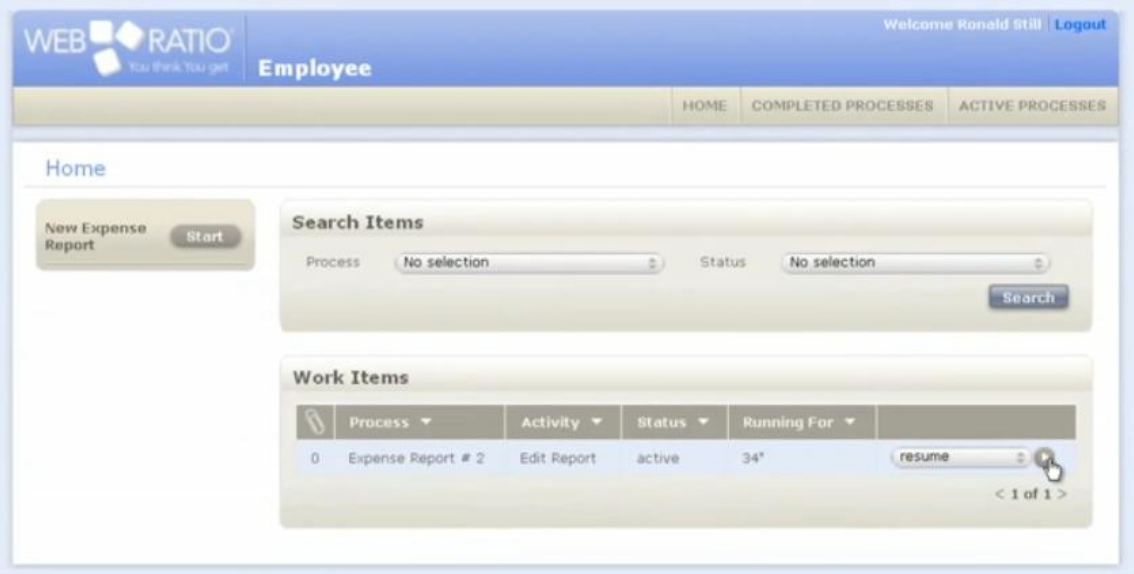


Figura 5.3. Ejemplo de prototipo generado por WebRatio

- *No es posible recuperar el código generado.*

  Sería deseable tener la posibilidad de disponer del código surgido a partir del modelo. Con ello, éste podría ser desplegado en un servidor que contase con las características necesarias. De esta forma, el resultado estaría visible a través de Internet. Esta propiedad sólo está disponible en las ediciones de pago.

- *Es un editor complejo.*

  La herramienta gráfica de modelado se presenta en un marco similar al empleado por Eclipse. Debido a ello, su uso no es intuitivo para las personas no habituadas a él. Muchas de sus funciones, aunque posiblemente interesantes, solamente complican la modelización de los procesos. Por añadidura, el código generado, basado en Java, debe pasar por los pasos de compilación y empaquetado. Éstos son susceptibles de provocar errores difíciles de resolver.

- *No dispone de una gran flexibilidad en la definición en los formularios.*

  Como se ha visto, esta herramienta no destaca por la libertad de configuración de los campos que componen los formularios. Especialmente, se echan en falta la posibilidad de añadir restricciones sobre los mismos.

En síntesis, esta herramienta puede aportar una gran ayuda en el modelado y verificación de procesos. Por otra parte, no ofrece un gran soporte para la generación y modificación de código portable. Éste sólo se proporciona cuando el usuario dispone de una licencia que sea de tipo anual o perpetua, con el consiguiente desembolso que ello implicaría. La posesión y manejo del código producido a partir del modelo BPMN es un objetivo fundamental en este trabajo.

5.2.2 ProcessMaker

Esta herramienta de código abierto [PROC12] ofrece otra propuesta para el modelado de workflows de formularios web a través de BPMN. Dispone para ello de una interfaz web basada en PHP.

En primer lugar, se compone de Dynaforms. Se trata de un módulo orientado hacia la definición de formularios. Gracias a él, es posible especificar no sólo los campos que los conforman, sino también restricciones sobre ellos. También es posible editar directamente el marcado HTML y vincular código Javascript a los controles.

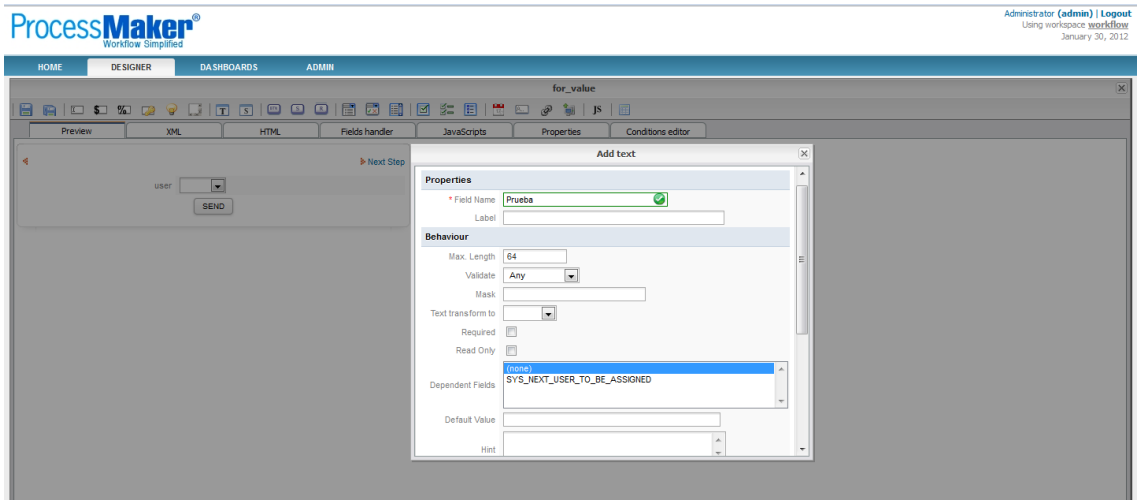


Figura 5.4. El componente Dynaforms de ProcessMaker

En lo que respecta al modelado de los procesos, dispone de un editor para tal fin. A su vez permite vincular directamente a cada tarea, los formularios que se desee del conjunto de ellos definidos a través de Dynaforms.

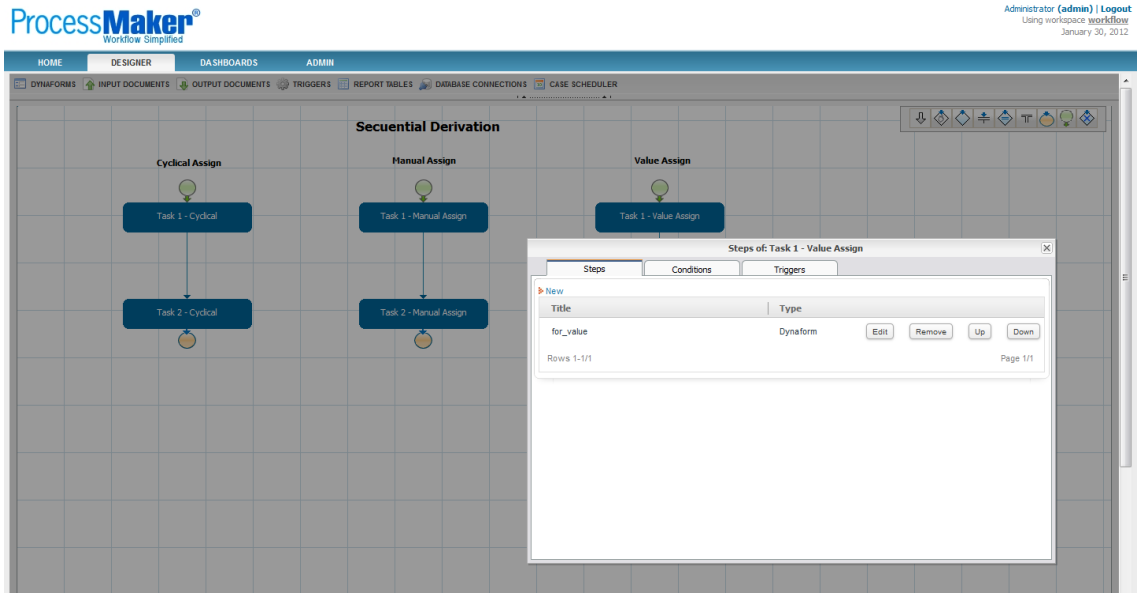


Figura 5.5. Editor BPMN de ProcessMaker

La ejecución de los procesos de realiza a través de casos. Básicamente, consisten en la asignación de tareas a usuarios.

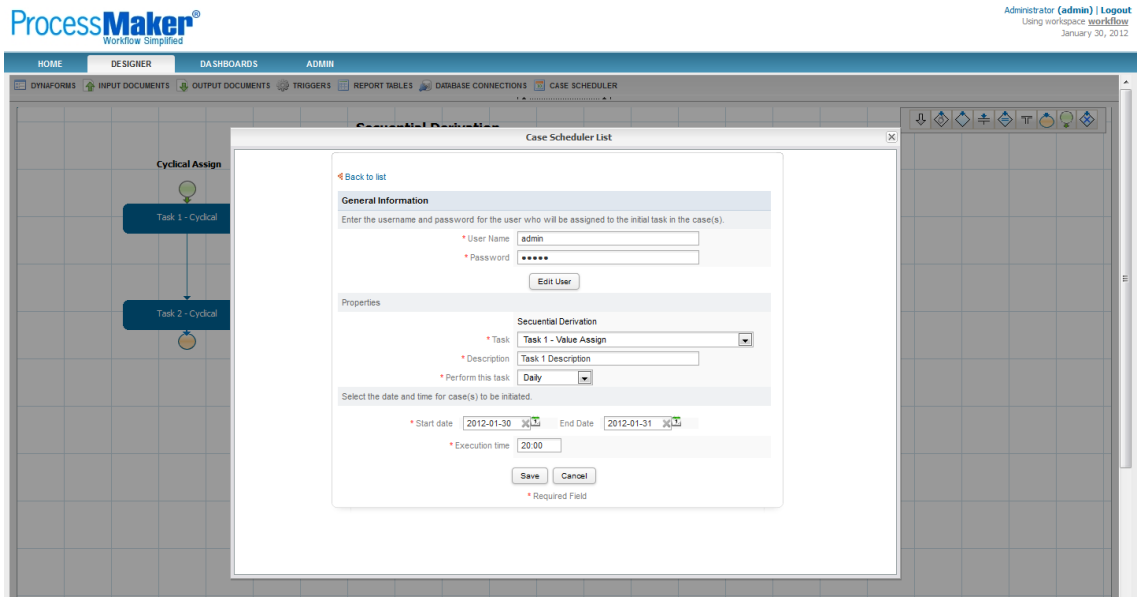


Figura 5.6. Definición de un caso en ProcessMaker

Para notificar al usuario que se requiere su participación para que un proceso dado siga su curso, se dispone de un panel de avisos. Además, también existe un sistema de envío de emails.

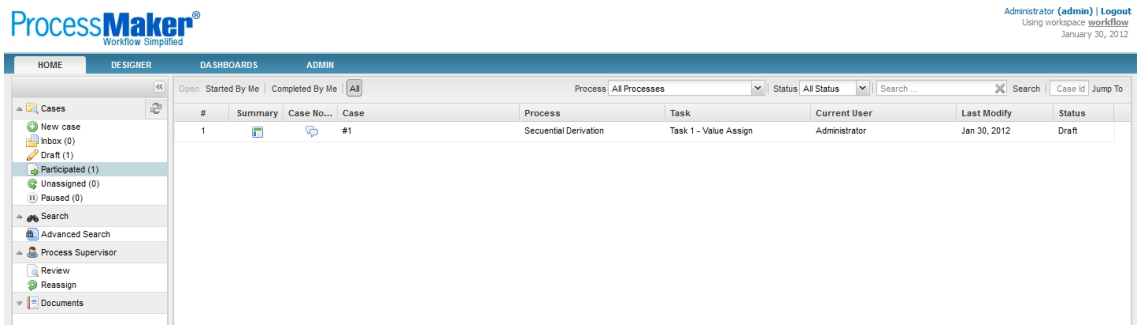


Figura 5.7. Panel de notificaciones en ProcessMaker

Desde cada tarea designada al usuario en sesión, éste puede acceder directamente al formulario que se ha dispuesto para llevarla a cabo.

Figura 5.8. Ejecución de una tarea (envío de un formulario) en ProcessMaker

Tras la realización de estas pequeñas pruebas, cabe destacar en primer lugar la gran capacidad de esta herramienta en todas las fases del modelado de procesos sobre formularios: diseño, ejecución y notificación de tareas. Todo ello se realiza en el marco de una aplicación web con un aspecto similar al de una herramienta de escritorio. En esta primera toma de contacto, ha quedado patente su validez de cara a lograr los objetivos marcados en este trabajo.

A pesar de ello, también presenta una serie de inconvenientes que hacen poco recomendable su aplicación.

- *Está basado en una aplicación web.*

  Al tratarse de una aplicación para la web, puede instalarse en un servidor para luego encontrarse accesible a través de la red. Ello posibilita la publicación en Internet de los formularios modelados. Esta característica también supone desventajas, especialmente en el rendimiento del sistema. A pesar de que el interfaz recuerda a la de un editor de escritorio, su eficiencia se ve reducida al tener que emitir constantemente peticiones al servidor que alberga la aplicación.

- *No es posible personalizar la aplicación resultante.*

  En los experimentos realizados ha quedado patente la flexibilidad de este sistema en lo que se refiere a la personalización de los formularios. Sin embargo, no es posible modificar el aspecto de la página web en que se enmarcan mediante la especificación de reglas de transformación distintas.

- *No es posible recuperar el código generado.*

  El prototipo se ejecuta sobre un servidor Apache. Con ello, es posible verificar el funcionamiento del proceso modelado a través de los formularios. El problema está en que no se puede recuperar el código que genera esta aplicación. Debido a ello, no se está en disposición obviamente de modificarlo o trasladarlo a otro servidor. No obstante, esta carencia se ve reducida gracias a que se trata de un desarrollo de código abierto. Aprovechando esta característica, se podría modificar su código interno, seguramente con un esfuerzo considerable, para que el resultado fuera el requerido en cada momento.

- *Se trata de un editor complejo.*

  La enorme flexibilidad ofrecida por este sistema redunda en una mayor complejidad en su funcionamiento. En ocasiones, la configuración de los formularios y los procesos es muy complicada. Además, existen apartados muy orientados a especialistas en desarrollo web (HTML y Javascript) y gestión de procesos. Esto dificulta la modelización de los procesos y los formularios, por simples que sean en origen. Por todo ello, para que pueda aprovechar su potencial, el usuario debe disponer de un alto grado de conocimiento de esta herramienta.

PARTE III. Desarrollo de la propuesta

Capítulo 6. Enfoque preliminar

Este primer capítulo de los que explican el desarrollo de la propuesta, pretende proporcionar un acercamiento al sistema ideado. Por ello, el objetivo principal es señalar cómo es posible aunar MDA y la modelización de procesos para lograr la consecución de la hipótesis.

En la introducción se desglosará el contenido de los elementos CIM, PIM y PSM, característicos de MDA, como fruto de diversas transformaciones. También se establece la funcionalidad que el código fuente generado en última instancia debe proporcionar al usuario.

A continuación, se detalla la estructura de estos cuatro elementos. De ellos, el CIM es especialmente crítico, y es el que más concienzudamente ha sido documentado. Esto se debe a que para su obtención se parte de un modelo gráfico expresado en BPMN, que será directamente convertido a un fichero basado en XPDL. Por este motivo, es crucial establecer las relaciones entre ambos sistemas, después de elegir los elementos BPMN necesarios para el cometido. Incluso, algunos de ellos se verán sometidos a nuevas interpretaciones. Tras una pequeña descripción del PIM y el PSM, el apartado dedicado al código fuente sienta las bases de la aplicación finalmente generada, desde los puntos de vista de presentación, estructuración interna y almacenamiento de la información requerida.

6.1 INTRODUCCIÓN

Como se ha indicado previamente, el objetivo final de este trabajo se logrará con la combinación de la modelización de procesos y MDA. Esta unión es adecuada por dos razones. Por una parte, los procesos deben ser modelados para obtener una visión global de los mismos. Ésta favorece la detección de los puntos débiles. Después, si se considera oportuno, se puede actuar sobre esas localizaciones señaladas, aplicando reingeniería. De esta manera, se retorna al punto inicial del ciclo, volviendo posteriormente a implementar el proceso con los cambios realizados en su modelo. El único aspecto que no es cubierto por MDA corresponde al análisis del proceso para detectar sus características mejorables.

Seguidamente, se establecen los elementos derivados de cada fase de la aplicación de la técnica de MDA, junto con una descripción somera que define la información que se espera obtener de su aplicación.

- *Modelo independiente de la computación (CIM).*

  Como sistema gráfico de representación de procesos se ha escogido BPMN. Esta elección se ha realizado basándose en que provee una forma intuitiva de reflejar los procesos. Por su parte, XPDL es un lenguaje basado en XML cuya notación está relacionada unívocamente con BPMN [WHIT03]. Esto facilita su interpretación con el objetivo de recuperar datos sobre el proceso modelado.

  En este primer paso, se estudiará el soporte que BPMN provee para la modelización de un flujo, basado en tareas asignadas a usuarios concretos y transiciones entre ellas. También se analizará si se proporciona algún método para especificar los campos del formulario disponibles para su modificación en cada paso del proceso. Puede suceder que sea necesaria la implantación de una herramienta auxiliar, que permita al usuario señalar determinada información.

  El contenido el CIM obtenido consistirá en una versión simplificada del documento XPDL, que aporte la información verdaderamente interesante. El formato elegido será XML, basado en un lenguaje específico de dominio propuesto.

- *Modelo independiente de la plataforma (PIM).*

  El objeto resultante será un documento basado en XML que amplíe los datos aportados por el anterior. El objetivo de esta fase consistirá en la imposición de reglas de validación sobre los valores de los campos contenidos por el formulario. También incluye la tarea de definir los valores incluidos en los campos seleccionables, como botones de tipo radio o menús desplegables. Se prevé la necesidad de implementar una herramienta para la captura de estos datos.

- *Modelo específico de plataforma (PSM).*

  En este caso, se procederá de nuevo a ampliar el XML resultante, adjuntando información referida principalmente al sistema de envío de emails. También se sopesará la posibilidad de indicar en este elemento la apariencia que

exhibirá el formulario y el portal de usuarios. Como para el resultado anterior, se requerirá el uso de una aplicación que acepte la recogida de estos datos.

- *Código fuente generado.*

  A partir de toda la información recopilada, se deben generar todos los archivos necesarios para proveer el formulario y sus sistemas anexos, es decir, el sistema de envío de correos electrónicos y el portal. En esta fase, también mediará una herramienta de conversión que genere los ficheros necesarios. Por último esta herramienta debe encargarse de crear la base de datos sobre la que se sustentará el nuevo sistema, así como de dejar instalado éste.

- *Reingeniería.*

  Tras la implantación y ejecución del código, se pueden establecer métricas sobre su eficiencia. En el caso concreto de este trabajo, se traducirán en cuellos de botella o ramas que nunca son utilizadas. Para realizar estas detecciones, habría que aplicar técnicas que no incumben a MDA. Estos resultados inducirán a introducir cambios sobre el proceso. Para llevarlos a cabo, se puede actuar sobre los modelos, evitando la manipulación directa del código fuente. Una nueva secuencia de transformaciones proporcionará la implementación del nuevo proceso. Esta fase no es abordada en esta investigación

En la siguiente sección se revisará en detalle el soporte de BPMN para representar la información relevante. Posteriormente, establecido el subconjunto útil de elementos de dicha notación, será necesario ver cómo son traducidos a XPDL. Esta comprobación es básica para diseñar el comportamiento de las herramientas de transformación.

6.2 OBTENCIÓN DEL CIM

6.2.1 Introducción

El objetivo de la modelización es la construcción de un formulario para la gestión colaborativa de la información según un flujo de trabajo específico. En base a ello, el modelo gráfico del sistema se asemejará a un grafo, donde aparte será necesario especificar cierta información adicional, referida al estado de los campos del formulario en cada paso.

En las siguientes secciones se analizan las notaciones gráficas existentes en BPMN para cada necesidad, su capacidad para indicar datos adicionales y su representación equivalente en el lenguaje XPDL. Con este fin se empleará el editor *Apia Facilis*, elegido por su simplicidad.

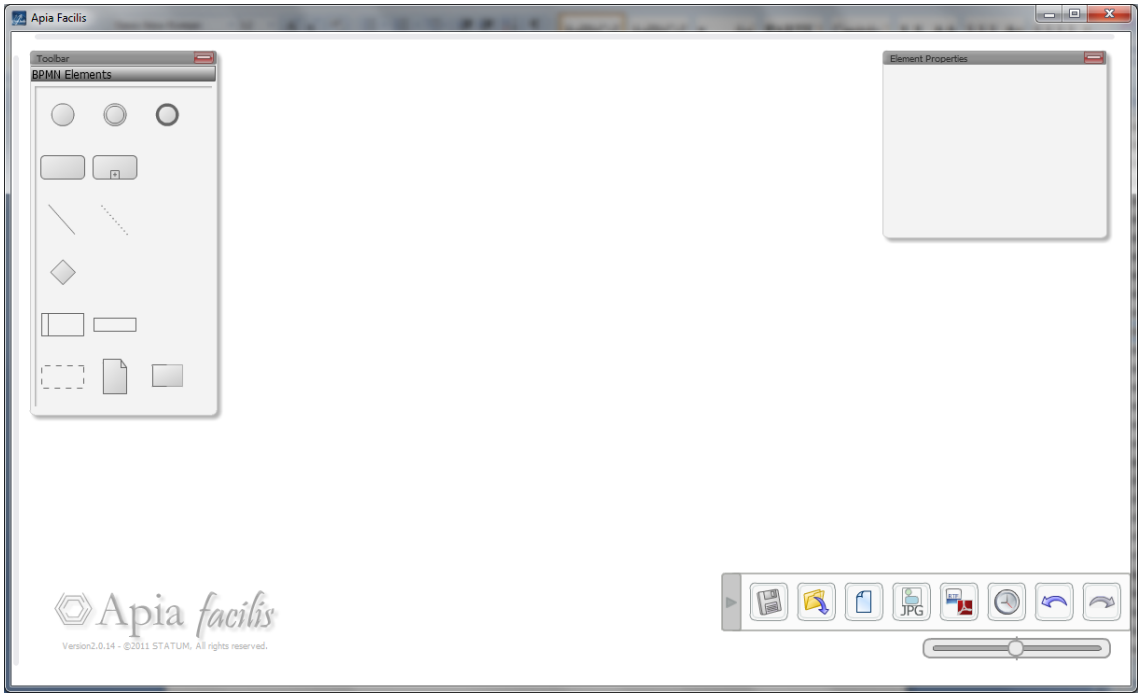


Figura 6.1. Ventana principal de Apia Facilis

6.2.2 Modelado de procesos orientado a la propuesta

6.2.2.1 El punto de comienzo del flujo de trabajo

En BPMN existe el llamado evento de inicio para representar este concepto. La única regla que se debe respetar es que no intervenga en el proceso como un elemento sucesor.

Figura 6.2. Símbolo del evento de inicio en BPMN

6.2.2.2 Las tareas

Son los elementos que reflejan las actividades cuya ordenación secuencial compone el proceso. Existen diversos tipos, pero las que interesan son las que son de tipo tarea de usuario.

Figura 6.3. Símbolo de la tarea de usuario en BPMN

A través de las propiedades de dicho elemento, se tiene acceso a dos características fundamentales para el cometido buscado.

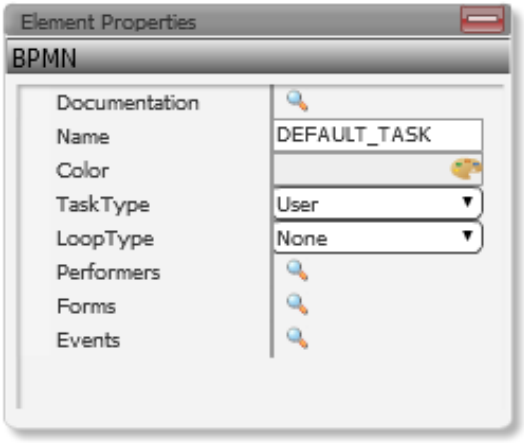


Figura 6.4. Propiedades de la tarea de usuario

La primera de ellas (*Performers*) permite asignar un conjunto de participantes a la tarea.

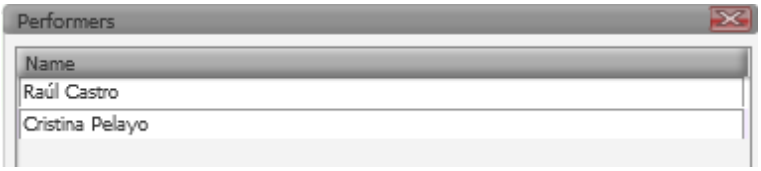


Figura 6.5. Asignación de participantes a la tarea de usuario

Por otra parte, es posible definir los formularios (*Forms*) que dichos usuarios tendrán disponibles para acometer la acción.

Figura 6.6. Declaración de un formulario

Posteriormente, también se soporta la definición de los campos y sus características, accesible a través de la opción *Attributes*. Entre las propiedades del campo se dispone de su nombre, su etiqueta en el formulario, un texto alternativo, el tipo de control empleado, su tipo de dato asociado y la posición en el formulario.

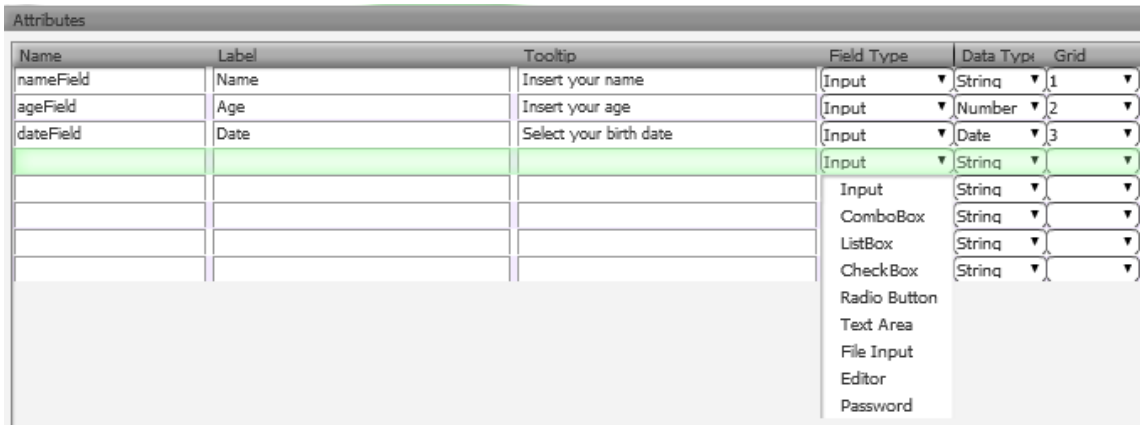


Figura 6.7. Declaración de los campos del formulario

6.2.2.3 Las transiciones

Ahora que ya se han definido los nodos que componen el grafo, es el turno de señalar la especificación de las transiciones entre ellos. En el diagrama BPMN, se trata de flechas unidireccionales dispuestas desde un elemento origen a otro destino, según el sentido de ejecución.

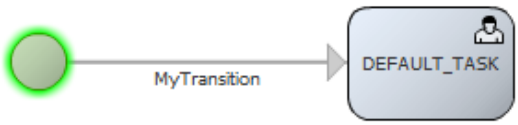


Figura 6.8. Símbolo de la transición en BPMN

6.2.2.4 Las bifurcaciones y las confluencias

Cuando un hilo es secuencial, el procesamiento sucede de manera consecutiva transitando entre las fases. Sin embargo, puede requerirse el uso de estructuras paralelas. Su función es indicar el desarrollo simultáneo de acciones diversas. Para ello, debe desmembrarse el proceso en ramas que se ejecutarán paralelamente. En otros casos, caminos separados deben unirse de nuevo. Para estas dos situaciones BPMN provee las pasarelas. Aunque existen más clases, son tres las más interesantes.

- *Pasarela exclusiva.*

Cuando interviene como elemento separador, se evalúan las condiciones de las salidas y escoge aquélla que se verifica, que debe ser única. Éste es el distintivo que les confiere la cualidad de la exclusividad. Por su parte, en su vertiente unificadora, aceptará los elementos que lleguen por cualquier camino y los traspasará a su salida en cuanto se detecte su llegada. Esta acción la realiza sin esperar a que el proceso alcance dicho elemento por todas sus entradas.

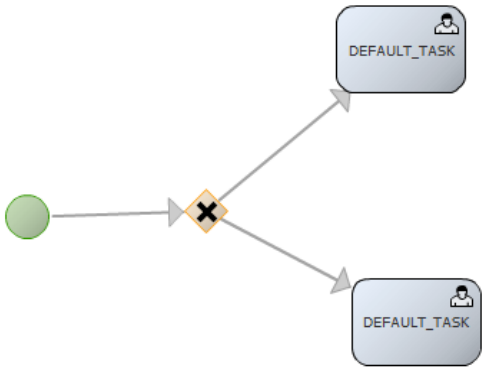


Figura 6.9. Símbolo de la pasarela exclusiva en BPMN

- *Pasarela paralela.*

Cuando se coloca en un modelo como separador, el proceso continuará en paralelo por todas las salidas de la pasarela. Finalmente, al reunir caminos separados, actúa síncronamente, es decir, manteniéndose a la espera de que el proceso llegue a través de todas las entradas antes de difundirlo a la siguiente fase.

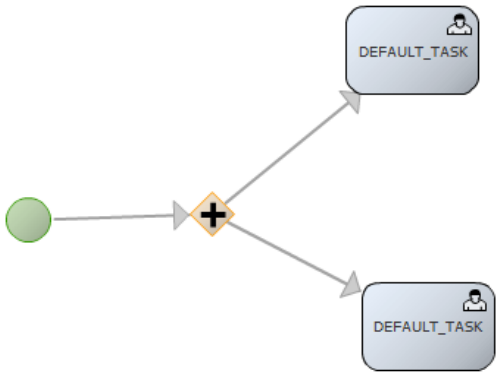


Figura 6.10. Símbolo de la pasarela paralela en BPMN

- *Pasarela inclusiva.*

Es similar a la exclusiva, con dos diferencias fundamentales. Al disgregar un camino, se establecen condiciones sobre las ramas salientes. Varias de estas condiciones pueden ser evaluadas simultáneamente como válidas, en cuyo caso el proceso continúa por todas las que lo sean. Además, cuando actúa para la unión de ramas, sí se espera a que el proceso haya llegado a ella a través de todas sus entradas, antes de transferirlo a las salidas.

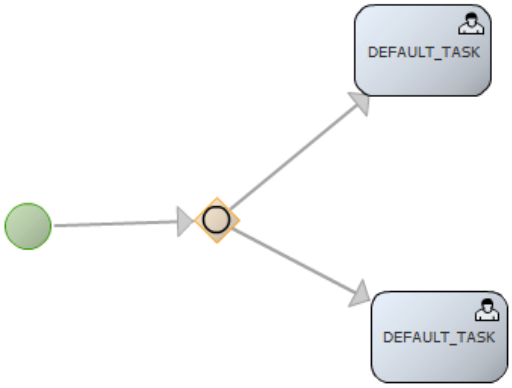


Figura 6.11. Símbolo de la pasarela inclusiva en BPMN

6.2.3 Interpretación particular de los elementos analizados

6.2.3.1 Evento de inicio.

La información gestionada por el formulario debe ser dada de alta en su primera fase. Éste es el punto de comienzo del proceso. Por ello, es necesario señalar esta entrada como punto de acceso en el diagrama. Gracias a esta indicación, se distinguirán las actividades relacionadas con la creación de datos, de las proporcionadas para su edición. Debido a esta norma, las tareas que suceden al evento de inicio no pueden suceder a ningún otro elemento. En esta circunstancia, una misma tarea se referiría a la creación y a la edición de un objeto, situación que resultaría paradójica.

La siguiente figura refleja el modo en que se aplicará. Según la convención indicada, las tareas *CREATION_1* y *CREATION_2* concluirán con la creación de un nuevo registro de información. Ello se debe a que son sucesoras directas del punto de inicio. El motivo de esta distinción entre distintos sistemas de creación se fundamenta en la posibilidad de que varias personas puedan crear registros, pero con diferentes campos del formulario disponibles.

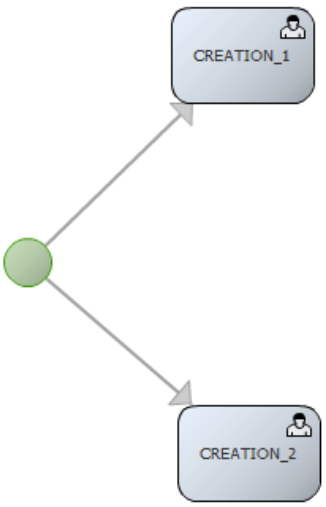


Figura 6.12. Evento de inicio y tareas de creación de registros

En esta bifurcación a partir del evento de arranque no se emplearán pasarelas. Ello es debido a que la generación de cualquier objeto es una operación que sucede una única vez para cada uno. Por esta razón, no tiene sentido que un mismo registro de información sea dado de alta a través de las dos operaciones. Por ello, se puede suponer la existencia implícita de una pasarela exclusiva entre el inicio y las tareas sucesoras.

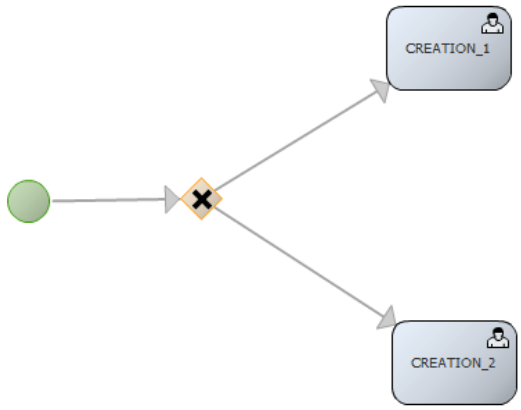


Figura 6.13. Evento de inicio, pasarela y tareas de creación de registros

6.2.3.2 Tareas.

Cuando se realice el modelado del workflow, el diseñador debe tener en cuenta que cada nodo refleja un estado de la información gestionada por el formulario modelado. Dicho estado, se alcanza cuando son rellenados por la persona adecuada los campos del formulario especificados. Según esta filosofía, el modelo debe percibirse como un grafo donde cada nodo indica un estado y las circunstancias que deben darse para alcanzarlo. En conclusión, la activación de un estado es posterior a la realización de las tareas asociadas en él. La norma de BPMN es que primero se alcanza una tarea, y a continuación, se realizarán los actos que requiere. Esta nueva convención se debe a que se desea aprovecha el potencial para contener la información necesaria de estos elementos de BPMN.

Para clarificar este punto de vista, se analizará el siguiente ejemplo. Partiendo de la figura anterior, se establece que *CREATION_1* puede ser ejecutada por los siguientes usuarios:

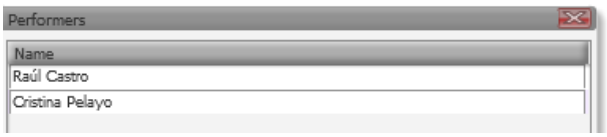


Figura 6.14. Usuarios autorizados a ejecutar CREATION_1

Por otra parte, este paso tiene asociados los siguientes campos.

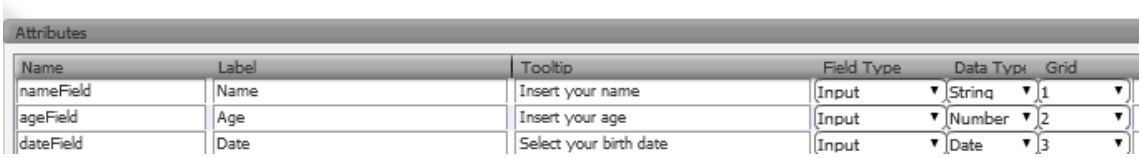


Figura 6.15. Campos relacionados con CREATION_1

La combinación de estas dos especificaciones implica que los usuarios incluidos estarán en disposición de trasladar un registro de información al estado *CREATION_1* cuando rellenen los campos especificados del formulario.

De los tipos de datos disponibles, no se aceptarán *File Input*, *Editor* y *Password*. Los dos primeros requieren de gestión y componentes adicionales que complicarán el sistema resultante. Por otra parte, dado el carácter de la información almacenada, no se espera que se requiera almacenar información privada.

Podría resultar más lógico emplear las condiciones de las transiciones, para mantener la concepción primitiva de la modelización de procesos. Esto implicaría una mayor complejidad en el desarrollo del modelo por el uso intensivo de pasarelas, requeridas para establecer condiciones sobre las transiciones. Además, estas comprobaciones deberían escribirse como una consulta lógica.

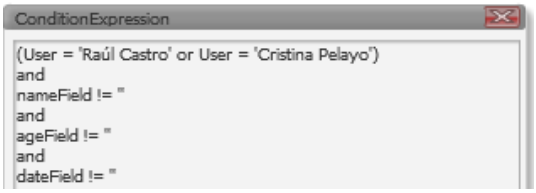


Figura 6.16. Condiciones impuestas sobre una transición

Desde esta perspectiva, una tarea debe verse más bien como el evento que regula el acceso a dicho nodo, presentando así un elemento dual capaz de señalar una acción necesaria (completar unos campos del formulario) regulada por un control de acceso.

Sin embargo esta distribución de participantes y campos entre las distintas tareas deja ver la aparición de un problema que complica el modelado. Normalmente, el conjunto de usuarios asignados al workflow del formulario será limitado. Por esta causa, algunos aparecerán asignados a diversas tareas, teniendo el usuario del editor que declararlos cada vez. Aparte de la redundancia en la información, es una fuente de inconsistencias. Puede suceder que se asocie a diversas fases los usuarios *C. Pelayo*, *Cris Pelayo* y *B. C. Pelayo*, sobreentendiendo que son distintos, cuando en realidad se trata de la misma persona. Análogamente sucede con los campos del formulario. Aquí, las incoherencias pueden ser más graves, pues entra en juego el tipo de dato. Por ejemplo, si a dos tareas se asignan dos campos que comparten nombre, pero uno es numérico y otro textual, no es posible determinar la opción correcta entre ambos.

Por este motivo, la solución lógica consiste en declarar una única vez ambos elementos. Posteriormente, se asignarían según conviniera, a través de referencias en cada tarea. El editor escogido no permite aplicar esta medida, y no se contempla la posibilidad de modificar su funcionamiento. Descartada esta opción, la solución parece ser que una tarea sea la encargada de portar esta información, y una herramienta adicional permita al usuario distribuir roles y campos entre los distintos estados del formulario, definidos por las tareas del modelo.

Tomada esta decisión, surge una nueva cuestión. Se debe determinar qué tarea es la adecuada para portar definiciones de usuarios y campos. A priori, ninguna dispone de

características distintivas que permitan localizarla de manera efectiva con el objetivo de recuperar las especificaciones. Dado que se trata de la definición del formulario y su workflow, se intuye que la opción natural sería emplear las fases iniciales del modelo, concretamente las destinadas a la creación de los registros de información. No obstante, como ya se ha indicado, la manera de darlos de alta no tiene por qué ser única.

En definitiva, se insertará una tarea única entre el evento de inicio y las referentes a la creación de registros. Desde el punto de vista del modelo, es una mera portadora de datos. Su existencia no redundará en la implementación del workflow del formulario. Sin embargo, su aparición es lógica dentro del modelo del proceso, como una estación dedicada a la creación de formularios, herramienta esencial para poder continuar el proceso, esto es, la gestión de la información.

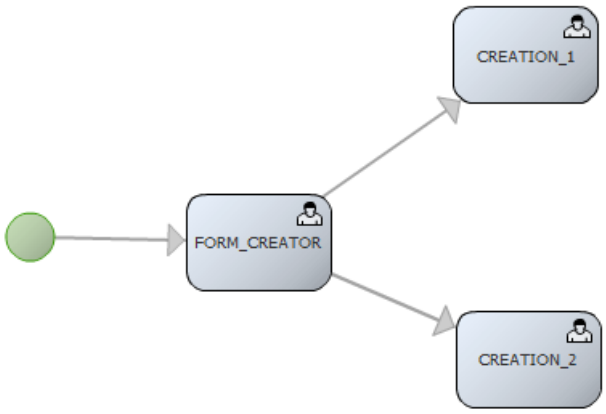


Figura 6.17. Tarea para la definición única de usuarios y campos

Bajo una perspectiva un tanto particular, este nuevo componente aporta una abstracción del proceso MDA que crea el formulario y su workflow. Por tanto, representa el procedimiento global explicado en este trabajo. Esta técnica, como así sucede en la realidad, es una parte primordial para poder llevar a cabo las tareas dispuestas a continuación. Si no existe un formulario, el flujo carece de sentido, al no disponer de los objetos básicos. El resultado es la relación bidireccional que se establece entre ambos sistemas, MDA y modelo. Por una parte, se emplea MDA para modelar workflows de formularios, mientras que el propio proceso modelado incluye la construcción del formulario y el workflow a través de MDA.

6.2.3.3 Transiciones.

La propuesta también afecta a la visión clásica de las transiciones. Normalmente, una transición se activa cuando ha finalizado la intervención de su elemento origen. En ese momento, el proceso avanza a su destino. En este caso concreto, una transición se abrirá cuando el usuario sea el señalado para la ejecución de la tarea destino, y además haya rellenado los campos especificados en el destino. Es entonces cuando la información procesada avanzará hacia el siguiente estado.

En origen, BPMN permite que un mismo elemento sea origen o destino de diversas transiciones. Ello implica que se deduce la existencia de una pasarela paralela, inherente al elemento. No obstante, se requerirá que ésta aparezca explícitamente, de manera que su tipo (exclusivo o paralelo) se haga explícito.

Únicamente se permite una excepción a esta norma. Se trata de las transiciones que parten de la tarea de definición del formulario. En este caso, ésta debe ser comunicada directamente con sus tareas sucesoras, correspondientes a la creación de registros. Semánticamente, implica que cuando el formulario es creado, es posible dar de alta información a través de una de las tareas establecidas para ello.

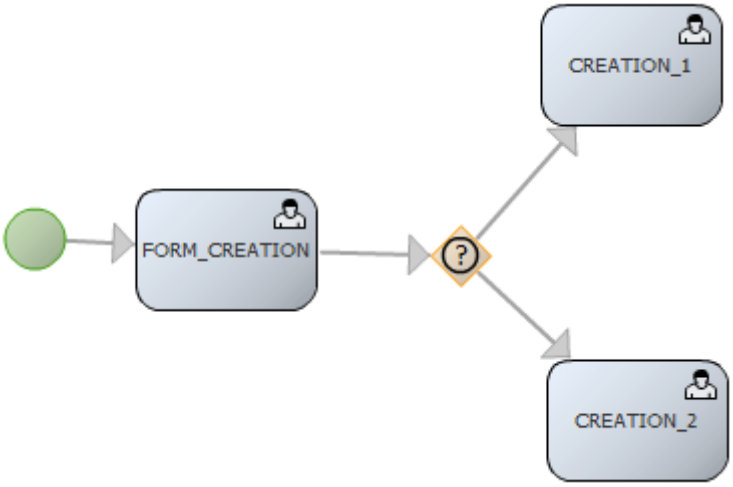


Figura 6.18. Puerta indeterminada tras la tarea de definición del formulario

Dado que esta selección es única, perece a priori que se trataría de una decisión alternativa exclusiva. No obstante, debido a los dos niveles de abstracción incluidos en el modelo, se podría pensar que estas transiciones establecen un vínculo entre la creación y sus distintos workflows (iniciados por la creación de la información). De esta forma, la decisión es coherente con el significado del modelo, dado que la vinculación entre ambos se establece simultáneamente a través de todos los enlaces definidos.

En cualquier caso, independientemente de esta consideración, no se admitirán pasarelas entre la definición del formulario y las tareas de creación de registros.

6.2.3.4 Pasarelas.

Respecto a su aplicación en la propuesta, la pasarela exclusiva presenta dos inconvenientes. En primer lugar, requiere condiciones en sus caminos de salida. Sin embargo, las condiciones empleadas en este sistema se basarán en los permisos de actuación de los usuarios y el estado de activación de las tareas según la inserción de datos en los campos especificados. Para ello se emplearán las notaciones introducidas en las secciones anteriores. Por otro lado, la no sincronización en las reunificaciones puede llevar a que un registro de información se encuentre simultáneamente en dos estados dispuestos secuencialmente. Debido a ello, podrían darse situaciones inconsistentes. Esta circunstancia se refleja en el siguiente ejemplo.

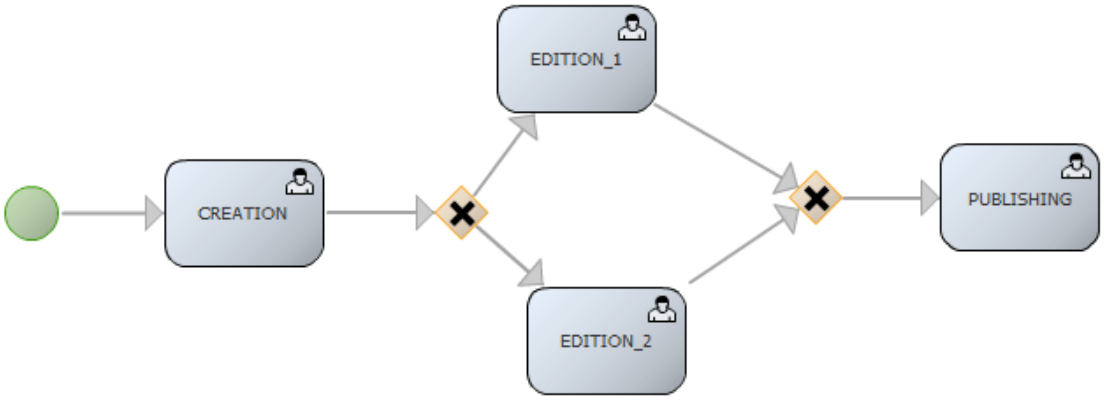


Figura 6.19. Caso problemático de uso de pasarelas exclusivas

Un contenido informativo que puede ser editado por dos personas indistintamente (acciones *EDITION_1* y *EDITION_2*) y en la práctica sólo ha sido alterado por una de ellas (*EDITION_1*). Posteriormente, el responsable lo publica (*PUBLISHING*), en respuesta a una solicitud de acción posterior a la fase de corrección. Mientras, es posible modificar el registro en aplicación de una fase previa (*EDITION_2*) a la publicación. Conceptualmente, no tiene sentido.

Según lo explicado anteriormente, se podrán usar estas pasarelas, pero de manera distinta a la estándar. Sirviendo como elementos disgregadores, no se les especificarán condiciones adicionales a las enumeradas en el párrafo anterior. Sin embargo, sí que serán útiles para indicar que sólo se tomará un camino, elegido entre sus salidas. En particular, el camino escogido dependerá de qué tarea accesible desde la pasarela es ejecutada en primer lugar. Ocurrido este suceso, las demás salidas resultarán inaccesibles para ese objeto desde la puerta. En el ejemplo anterior, cuando se dé de alta un registro los dos estados siguientes están accesibles a priori, cuando se verifiquen las restricciones impuestas por los estados. Si *EDITION_1* se ejecuta, entonces no se podrá realizar *EDITION_2*, y viceversa. Con ello, se marcará la existencia de alternativas opcionales y excluyentes, sin que provoquen que la información procesada aparezca posteriormente en varios estados incompatibles a la vez. Si se emplean en una confluencia, entonces debe tenerse siempre presente el problema indicado, a la hora de diseñar el proceso.

Las pasarelas paralelas resultarán útiles para señalar la división del proceso en caminos obligatorios y unificar de manera síncrona, es decir, cuando se registre el paso por los estados inmediatamente anteriores a la puerta. En el ejemplo, un registro debe haber atravesado *EDITION_1* y *EDITION_2* antes de poder ser publicado.

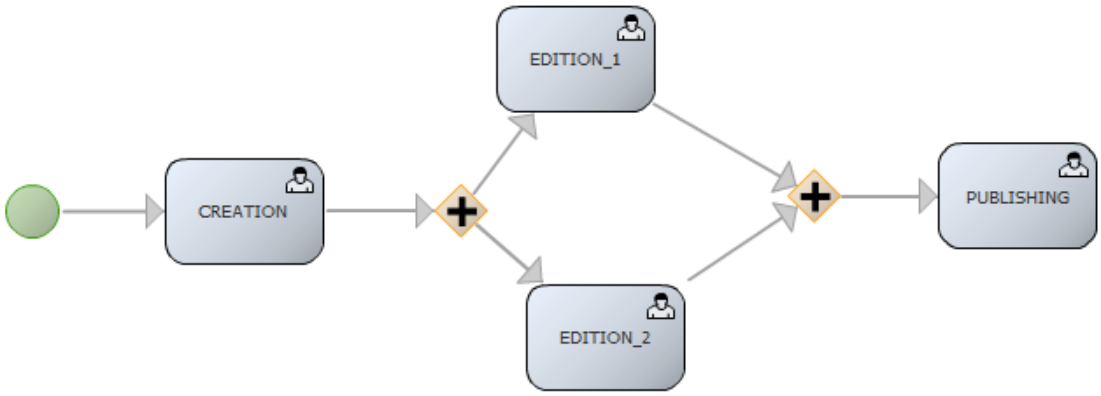


Figura 6.20. Ejemplo de uso de puertas paralelas

En tercer lugar, por causa de la introducción de las condiciones sobre las transiciones, las pasarelas inclusivas no aportarían ninguna ventaja adicional. Por ello, no serán aceptadas en la notación aceptada.

Por último, cabe señalar un caso particular en el empleo de las pasarelas, cuando se establecen de manera consecutiva. Debe ser tratado con cuidado. En la siguiente imagen se refleja este problema.

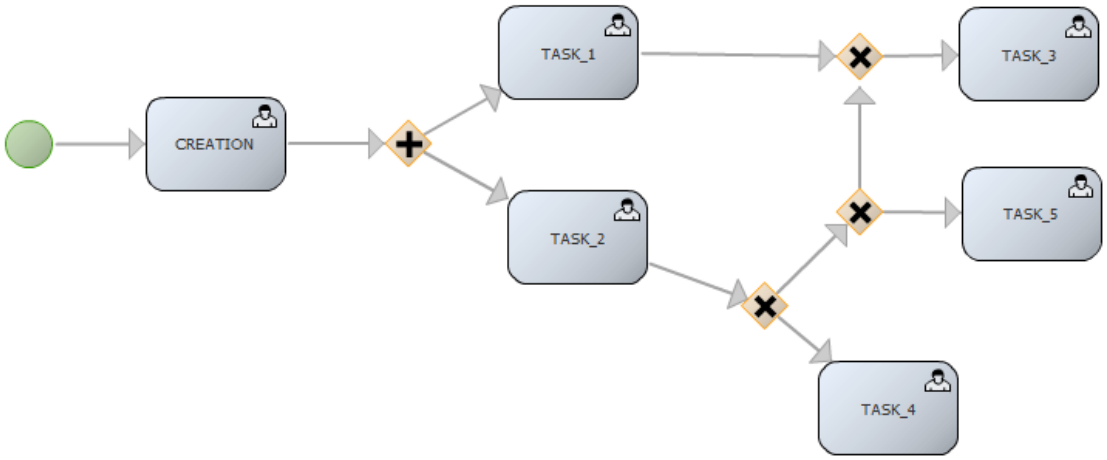


Figura 6.21. Uso consecutivo de pasarelas exclusivas

Se debe imaginar que un registro de información se encuentra a la vez en los estados *TASK_1* y *TASK_2*. Esta disposición es posible según el diagrama modelado. Adicionalmente, se establece que el control del proceso se fundamenta en el conocimiento sobre los estados accesibles según los que estén activos. En esta situación, los siguientes estados accesibles, cuando concurran las condiciones oportunas, son *TASK_3* (desde *TASK_1* y *TASK_2*), *TASK_4* y *TASK_5*.Entonces, se ejecuta *TASK_4*, con lo que como se ha escogido ya un camino, entonces *TASK_3* y *TASK_4* dejan de estar accesibles, según el funcionamiento especificado para las pasarelas exclusivas. El problema es que entonces *TASK_1* se convierte en un nodo sin salida.

En cambio, se puede optar por mantener un listado de transiciones cuyo elemento origen ha sido ejecutado y el destino está pendiente de ser sobrepasado. Con este punto

de vista, partiendo de la situación inicial, pertenecerán a este conjunto las transiciones que parten de *TASK_1* y de *TASK_2*. Desde ellas, se puede lanzar un elemento capaz de detectar los estados accesibles, atravesando las puertas que se encuentre. Esta alternativa tampoco es perfecta. Supongamos que la pasarela situada como antecesora de *TASK_5* es de tipo paralelo. En esa circunstancia, se acaba de ejecutar *TASK_2*. De esta manera, se activa la transición que sale de ella, con lo que son alcanzables los estados *TASK_3*, *TASK_5* y *TASK_4*. Si se llevara a cabo *TASK_5*, por la lógica del proceso, habría que dejar activar la transición que entra en *TASK_3*, después de comprobar las puertas existentes esta llegar a ésta.

A la vista de estos motivos, la concatenación de pasarelas es una fuente de problemas complejos. Por ello, se impondrá que estos elementos sólo puedan tener tareas como antecesores y sucesores.

6.2.4 Equivalencia XPDL de los elementos BPMN escogidos

6.2.4.1 Estructura básica del documento XPDL

Si se deja en blanco el diagrama BPMN, al convertirlo a XPDL se registra determinada información que puede resultar de interés.

```
<xpdl:Package Id="3" xmlns:xpdl="http://www.wfmc.org/2008/XPDL2.1">
   <xpdl:PackageHeader>
      <xpdl:XPDLVersion>2.1</xpdl:XPDLVersion>
      <xpdl:Vendor>STATUM</xpdl:Vendor>
      <xpdl:Created>2012/02/2512:45:56</xpdl:Created>
   </xpdl:PackageHeader>
   <xpdl:Pools>
      <xpdl:Pool Process="2" MainPool="true" BoundaryVisible="false"
            Id="1" Orientation="HORIZONTAL">
         <xpdl:NodeGraphicsInfos>
            <xpdl:NodeGraphicsInfo Width="0" Height="0">
               <xpdl:Coordinates YCoordinate="0" XCoordinate="0" />
            </xpdl:NodeGraphicsInfo>
         </xpdl:NodeGraphicsInfos>
      </xpdl:Pool>
   </xpdl:Pools>
   <xpdl:WorkflowProcesses>
      <xpdl:WorkflowProcess Id="2">
         <xpdl:ProcessHeader />
      </xpdl:WorkflowProcess>
   </xpdl:WorkflowProcesses>
</xpdl:Package>
```

Código 6.1. Documento XPDL básico

Tras la cabecera propia de un documento XML típico, se muestran datos de la versión de XPDL empleada, el fabricante de la herramienta, y la fecha de creación del fichero. Posteriormente se registra la información propia del proceso.

6.2.4.2 Evento de inicio

A continuación, se muestra su definición XPDL equivalente.

```
...
<xpdl:WorkflowProcess Id="2">
   <xpdl:ProcessHeader />
   <xpdl:Activities>
      <xpdl:Activity Id="3">
         <xpdl:Event>
            <xpdl:StartEvent Trigger="None" />
         </xpdl:Event>
         <xpdl:NodeGraphicsInfos>
            <xpdl:NodeGraphicsInfo Width="30" Height="30">
               <xpdl:Coordinates YCoordinate="181.5"
                     XCoordinate="271.5" />
            </xpdl:NodeGraphicsInfo>
         </xpdl:NodeGraphicsInfos>
      </xpdl:Activity>
   </xpdl:Activities>
</xpdl:WorkflowProcess>
...
```

Código 6.2. Representación XPDL del evento de inicio

Al observar el resultado, se aprecia que su definición se ha insertado en la sección *<xpdl:WorkflowProcesses>*, concretamente como un tipo *<xpdl:Activity>*. Adicionalmente, se aporta información sobre su representación gráfica.

6.2.4.3 Tareas

Una vez que se han salvado todas las modificaciones realizadas sobre el fichero BPMN, se proporciona a continuación el cambio experimentado en la conversión a XPDL.

En primer lugar, la sección *<xpdl:Performers>* contiene la relación de usuarios autorizados a realizar la acción.

```
...
<xpdl:Activity Name="DEFAULT_TASK" Id="4">
   <xpdl:Implementation>
      <xpdl:Task>
         <xpdl:TaskUser/>
      </xpdl:Task>
   </xpdl:Implementation>
   <xpdl:Performers>
      <xpdl:Performer>Raúl_Castro</xpdl:Performer>
      <xpdl:Performer>Cristina_Pelayo</xpdl:Performer>
   </xpdl:Performers>
   ...
```

Código 6.3. Representación XPDL de las tareas de usuario (usuarios autorizados)

Realmente, se trata de referencias a los participantes del proceso, cuya relación aparece en *<xpdl:Participants>*.

```
...
<xpdl:Participants>
   <xpdl:Participant Name="Raúl Castro" Id="Raúl_Castro">
      <xpdl:ParticipantType Type="ROLE" />
   </xpdl:Participant>
   <xpdl:Participant Name="Cristina Pelayo" Id="Cristina_Pelayo">
      <xpdl:ParticipantType Type="ROLE" />
   </xpdl:Participant>
</xpdl:Participants>
...
```

Código 6.4. Representación XPDL de los participantes del proceso

Por su parte, el apartado *<xpdl:Documentation>* contiene los formularios definidos, listados en *<forms>*. Para cada uno, se proporciona su información propia.

```
...
<xpdl:Activity Name="DEFAULT_TASK" Id="4">
   ...
   <xpdl:Object Id="7">
      <xpdl:Documentation>
         <forms>
            <form frmType="E" frmStepId="1" frmName="TestForm"
                  frmOrder="0" frmTitle="This is my test form">
               ...
            </form>
         </forms>
      </xpdl:Documentation>
   </xpdl:Object>
...
```

Código 6.5. Representación XML de las tareas de usuario (formulario)

A continuación, se muestra la información referente a los campos del formulario. Este fragmento aparece originalmente con los símbolos reservados de XML (por ejemplo, <, >, ") remplazados por sus valores entidad ("<", ">" y "&quote;"). Se han restablecido los elementos anteriores para facilitar su lectura.

```
...
   <form frmType="E" frmStepId="1" frmName="TestForm"
         frmOrder="0" frmTitle="This is my test form">
      <attribute attTooltip="Insert your name" regExp=""
            grid="1" datatype="String" fieldtype="Input"
            attName="nameField" attLabel="Name" />
         <attribute attTooltip="Insert your age" regExp=""
            grid="2" datatype="Number" fieldtype="Input"
            attName="ageField" attLabel="Age" />
         <attribute attTooltip="Insert your birth date" regExp=""
            grid="3" datatype="Date" fieldtype="Input"
            attName="birthDateField" attLabel="Birth date"/>
         <attribute attTooltip="Select your country" regExp=""
         grid="4" datatype="String" fieldtype="ComboBox"
         attName="countryField" attLabel="Country" />
         <attribute attTooltip="Select your interest fields"
            regExp=""  grid="5" datatype="String"
            fieldtype="Input"
            attName="inserestsField"
            attLabel="Interests" />
         <attribute
            attTooltip="Select if you want more information"
            regExp="" grid="6" datatype="String"
            fieldtype="CheckBox" attName="moreInformationField"
            attLabel="More information" />
         <attribute attTooltip="Select your sex" regExp=""
            grid="7" datatype="String" fieldtype="Input"
            attName="sexField" attLabel="Sex" />
         <attribute attTooltip="Insert a description about you"
            regExp="" grid="8" datatype="String"
            fieldtype="Input"
            attName="descriptionField"
            attLabel="Description" />
      </form>
...
```

Código 6.6. Representación XML de las tareas de usuario (campos del formulario)

El hecho de que esta sección no mantenga el formato XML se debe a que no forma parte de la especificación XPDL. De ahí que los elementos que aparecen en ella no tengan especificado el espacio de nombres habitual en el documento, es decir, "xpdl:". De esta manera, el conjunto es tratado como un elemento textual. Así, no aparecerán errores de validación cuando se coteje la estructura del fichero XPDL. Antes de recopilar el contenido de esta sección, será necesario transformarlo a XML.

Por tanto, la cualidad que presenta el editor empleado, de asignar formularios y campos a las tareas, es una ventaja adicional. Nunca se debe considerar parte del estándar ni, en consecuencia, presuponer que cualquier editor la incluirá.

Para proveer esta funcionalidad, *Apia Facilis* hace uso del campo *xpdl:Documentation* de las tareas.

Figura 6.22. Documentación de una tarea.

Por una parte, guarda en él la información relacionada con la propia tarea. Este contenido es el que originalmente guardaría *xpdl:Documentation*. Por otro lado, se especifica los datos referentes al formulario.

```
...
<xpdl:Documentation>
   &lt;documentation&gt;
      TASK&amp;apos;SDOCUMENTATION
   &lt;/documentation&gt;
   &lt;forms&gt;
      ...
   /&gt;&lt;forms&gt;
</xpdl:Documentation>
...
```

Código 6.7. Representación XPDL documentación y formularios de tarea

Esta anotación debe tenerse presente cuando se haga uso de otro editor distinto para la modelización BPMN. En ese caso, se debe proveer manualmente la información acerca del formulario, según el formato explicado. Se podría implementar una herramienta de transformación, responsable de solicitar y almacenar la información en el archivo XPDL.

Para finalizar la sección correspondiente a la tarea, se incorpora información necesaria para su representación gráfica.

```
...
<xpdl:Activity Name="DEFAULT_TASK" Id="4">
   ...
   <xpdl:NodeGraphicsInfos>
      <xpdl:NodeGraphicsInfo Width="90"
            Height="60" FillColor="">
         <xpdl:Coordinates YCoordinate="167" XCoordinate="417" />
      </xpdl:NodeGraphicsInfo>
   </xpdl:NodeGraphicsInfos>
</xpdl:Activity>
...
```

Código 6.8. Representación XPDL de las tareas de usuario (representación gráfica)

6.2.4.4 Transiciones

La incorporación de este componente ha supuesto la aparición de la sección *<xpdl:Transtions>* después de *<xpdl:Activities>*. Se distinguen perfectamente sus datos relevantes: nombre y elementos origen y destino.

```
<xpdl:Package Id="3" xmlns:xpdl="http://www.wfmc.org/2008/XPDL2.1">
...
         <xpdl:Transitions>
            <xpdl:Transition To="4" Id="5" From="3">
               <xpdl:ConnectorGraphicsInfos>
                  <xpdl:ConnectorGraphicsInfo>
                     <xpdl:Coordinates YCoordinate="196.5"
                           XCoordinate="286.5" />
                     <xpdl:Coordinates YCoordinate="197"
                           XCoordinate="463" />
                  </xpdl:ConnectorGraphicsInfo>
               </xpdl:ConnectorGraphicsInfos>
            </xpdl:Transition>
         </xpdl:Transitions>
      </xpdl:WorkflowProcess>
   </xpdl:WorkflowProcesses>
</xpdl:Package>
```

Código 6.9. Representación XPDL de la transición

6.2.4.5 Pasarelas exclusivas

Para ver cómo se comportan estos elementos, se tomará este modelo, que incluye una bifurcación y una unión.

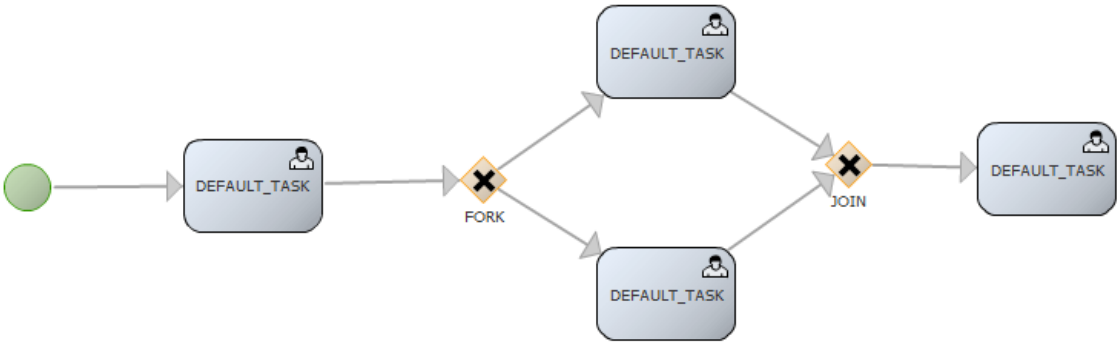


Figura 6.23. Modelo para el análisis de las pasarelas exclusivas

Al analizar el código XPDL generado, se comprueba que existen varias referencias a los componentes estudiados. Por una parte son declarados como actividades.

```
...
<xpdl:Activity Name="FORK" Id="5">
   <xpdl:Route GatewayType="Exclusive" ExclusiveType="Data" />
      <xpdl:TransitionRestrictions>
         <xpdl:TransitionRestriction>
            <xpdl:Split  Type="Exclusive">
               <xpdl:TransitionRefs>
                  <xpdl:TransitionRef Id="12" />
                  <xpdl:TransitionRef Id="13" />
               </xpdl:TransitionRefs>
            </xpdl:Split>
         </xpdl:TransitionRestriction>
      </xpdl:TransitionRestrictions>
   <xpdl:NodeGraphicsInfos>
      <xpdl:NodeGraphicsInfo Width="30" Height="30">
         <xpdl:Coordinates YCoordinate="245.5" XCoordinate="569.5" />
      </xpdl:NodeGraphicsInfo>
   </xpdl:NodeGraphicsInfos>
</xpdl:Activity>
...
<xpdl:Activity Name="JOIN" Id="8">
   <xpdl:Route GatewayType="Exclusive" ExclusiveType="Data" />
      <xpdl:NodeGraphicsInfos>
         <xpdl:NodeGraphicsInfo Width="30"       Height="30">
         <xpdl:Coordinates YCoordinate="235.5" XCoordinate="807.5" />
      </xpdl:NodeGraphicsInfo>
   </xpdl:NodeGraphicsInfos>
</xpdl:Activity>
...
```

Código 6.10. Representación XPDL de las pasarelas exclusivas

Analizado el resultado, cabe señalar que cuando actúa como elemento separador (*FORK*), se señalan los identificadores de las transiciones entrantes en la pasarela. En este caso, se trata de las que tienen como identificadores 12 y 13. Además se indica que es de tipo exclusivo.

```
...
<xpdl:Transition To="6" Id="12" From="5">
...
<xpdl:Transition To="7" Id="13" From="5">
...
```

Código 6.11. Representación XPDL transiciones salientes de la pasarela exclusiva

En cambio, cuando se coloca para unificar ramas (*JOIN*), las transiciones entrantes se conocerán buscando aquéllas que tienen por destino el identificador de la pasalela.

```
...
<xpdl:Transition To="8" Id="13" From="6">
   <xpdl:ConnectorGraphicsInfos>
      <xpdl:ConnectorGraphicsInfo>
         <xpdl:Coordinates YCoordinate="94" XCoordinate="745" />
         <xpdl:Coordinates YCoordinate="194.5" XCoordinate="884.5" />
         </xpdl:ConnectorGraphicsInfo>
      </xpdl:ConnectorGraphicsInfos>
   </xpdl:Transition>
<xpdl:Transition To="8" Id="16" From="7">
   <xpdl:ConnectorGraphicsInfos>
      <xpdl:ConnectorGraphicsInfo>
         <xpdl:Coordinates YCoordinate="314.5" XCoordinate="750.5" />
         <xpdl:Coordinates YCoordinate="194.5" XCoordinate="884.5" />
      </xpdl:ConnectorGraphicsInfo>
   </xpdl:ConnectorGraphicsInfos>
</xpdl:Transition>
...
```

Código 6.12. Representación XPDL transiciones entrantes de la pasarela exclusiva

También se ha verificado que este comportamiento se presenta cuando la pasarela actúa a la vez como componente reunificador y separador. En síntesis, la información proporcionada consiste en el tipo de pasarela, las transiciones que entran y salen de ella y, cuando une distintos hilos, los identificadores de las transiciones entrantes.

6.2.4.6 Pasarelas paralelas

Se partirá de un modelo de proceso similar al analizado en el caso anterior, sustituyendo las pasarelas.

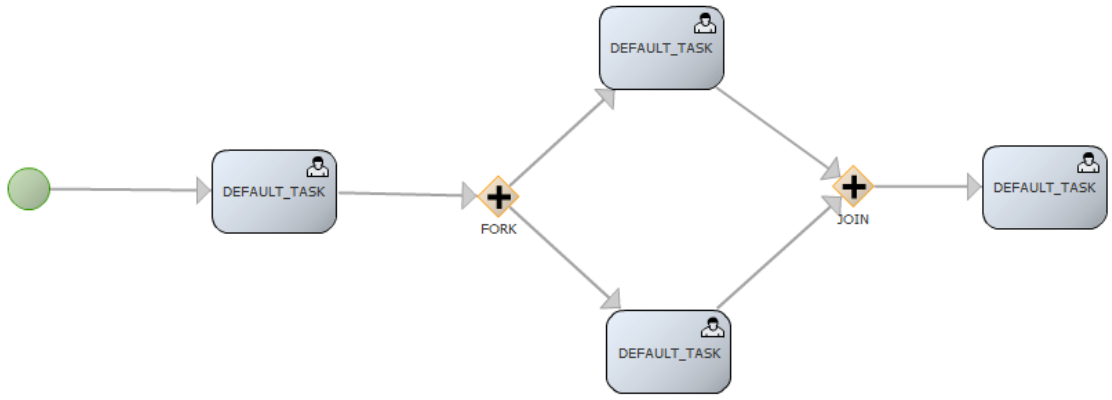


Figura 6.24. Modelo para el análisis de las pasarelas paralelas

El formato de la información almacenada es similar a la de las exclusivas, distinguiendo el tipo, que aquí es paralelo.

6.2.5 Descripción de la estructura del CIM

La estructura del CIM vendrá dada por un lenguaje específico de dominio propio basado en XML. Para determinar su sintaxis, y, en consecuencia, el formato del CIM, es

importante en primer lugar determinar cómo se procesará la información portada en este elemento. Otro de los objetivos que se pretende lograr es la limpieza del documento XPDL, obviando datos redundantes o irrelevantes para el cometido. Esta perspectiva permitirá disponer estos datos de una forma cómoda para su posterior recuperación. Por tanto, el primer paso consiste en determinar estos puntos teniendo en mente los criterios establecidos en la sección anterior.

- *El formulario.*

  El formulario manejado en la aplicación es único. Puede suceder que tareas distintas requieran la edición sobre campos comunes de este formulario. Por este motivo, es preferible mantener de forma independizada los datos que atañen a sus campos. Estos consisten en el nombre, la etiqueta, el tipo de dato y el tipo de control a través del que se gestionará. Posteriormente, se vincularán con las tareas.

- *Los usuarios.*

  Un mismo usuario puede ser requerido para la ejecución de tareas diferentes. Por tanto, lo mejor es mantener sus datos separados de las relaciones con las actividades.

- *Las tareas.*

  El paso de un registro a un estado determinado por una tarea está condicionado a que uno de los usuarios autorizados rellene los campos del formulario que se establezcan. Por ello, debe contener esa tarea referencias a usuarios y campos del formulario.

  Dado que la creación y la modificación de registros son operaciones conceptualmente distintas, se deberán diferenciar las tareas para dar de alta, sucesoras del evento de inicio, de las que causan una operación de edición.

- *Las pasarelas.*

  Este listado también será autónomo y es fundamental en él especificar su tipo. Así, se podrá determinar qué comprobaciones es necesario realizar. No contendrá información adicional referente a sus entradas y salidas. Para conocer la identidad de estos caminos se debe consultar la sección de transiciones.

- *Las transiciones.*

  Para cada transición se especificará su origen y su destino, que harán referencia a componentes de tipo tarea o pasarela.

  La propuesta es que el sistema generado sea capaz de mantener las transiciones que estén es disposición de ser franqueadas. Por este motivo, en cuando se lleve a cabo una tarea dada, será necesario realizar dos comprobaciones.

  En primer lugar, comprobar si el elemento sucesor es una pasarela. Si es de tipo exclusivo, entonces activar directamente la transición saliente de ésta. Cuando sea paralela, habrá que comprobar que no existan otras ramas entrantes o,

si existen, que estén activas. Si es así, se desactivarán todas las entrantes y se activarán las salientes. En otro caso, no se deberá realizar ninguna acción adicional.

Asimismo, es necesario verificar la clase de antecesor. Únicamente, cuando se trate de una puerta exclusiva, será necesario desactivar todas sus ramas salientes. Con ello, la exclusividad quedará garantizada.

La posibilidad de corroborar estas circunstancias resulta posible con la decisión tomada.

6.2.6 Resumen de reglas sobre el modelo BPMN

En las secciones anteriores se han perfilado distintas normas para la modelización del workflow a través de BPMN. A continuación se dispone un resumen de las mismas, para que el responsable de representar gráficamente el proceso las tenga a su disposición.

- El proceso debe comenzar en un evento de inicio único.
- Una tarea debe disponerse como único sucesor del evento de inicio. Ésta se aprovechará para definir los usuarios participantes y los campos del formulario.
- Como norma general, una tarea será destino y/u origen, como máximo, de una transición. Esto no se cumple para la tarea de definición, introducida en el punto anterior. Habitualmente, será origen de varias transiciones, cada una de las cuales trasladará a otra tarea cuyo cometido es dar de alta un registro de información.
- No puede haber dos tareas ni dos campos con el mismo nombre.
- Una pasarela nunca puede encontrarse a continuación de otra. Siempre serán de tipo paralelo o exclusivo.
- Las tareas deben tener asignados nombres distintos. De la misma manera sucede con los campos del formulario. El objetivo de este requisito es favorecer su distinción, para así evitar colisiones en la aplicación final.

6.3 OBTENCIÓN DEL PIM

En esta fase, se tomará el CIM construido y se le añadirá información adicional. Ésta está relacionada con las restricciones cuyo cumplimiento será necesario verificar sobre el formulario, antes de emprender el almacenamiento de los datos modificados. Las opciones de que se dispondrán dependerán del tipo de dato de que se trate, así como del control empleado.

En la siguiente tabla se sintetizan las restricciones aceptadas, según el dato y el control escogidos.

		Tipo de dato		
		Texto	Número	Fecha
Tipo de control	Input	Longitud máxima	Valores mínimo y máximo	Valores mínimo y máximo
	ListBox	Valores posibles (texto) y etiquetas	Valores posibles (numéricos) y etiquetas	Valores posibles (fechas) y etiquetas
	Check Box	No aplicable	No requiere información adicional	No aplicable
	Radio Button	Valores posibles (texto) y etiquetas	Valores posibles (numéricos) y etiquetas	Valores posibles (fechas) y etiquetas
	Text Area	Longitud máxima	No aplicable	No aplicable

Tabla 6.1. Restricciones aplicables según los tipos de dato y de control

- *Campo de inserción directa por teclado (Input).*

Si el dato es de tipo texto, se deberá establecer una longitud máxima permitida. En otros casos, se demarcará el rango de valores aceptados.

- *Campos de selección (ComboBox, ListBox y RadioButton).*

Se establecerá un conjunto de valores permitidos, cuyo tipo debe coincidir con el definido para el dato. Además, se señalará una etiqueta para cada uno.

- *Casilla de verificación (Check Box).*

El tipo de dato más apropiado es el booleano. Este tipo no está disponible en la construcción del CIM. Debido a ello, se empleará el tipo numérico, aunque posteriormente se interpretará debidamente. Este elemento no requiere más información complementaria.

- *Campo amplio de inserción directa por teclado (Text Area).*

Estará siempre vinculado con un tipo de dato textual. Esta decisión se fundamenta en que una cifra o una fecha no precisan un espacio extenso para su escritura.

Aunque este contenido podría ser considerado como propia del modelo CIM, se ha asignado a la segunda fase de manera justificada. Por una parte, el objetivo de aquél es la definición del flujo, a través de nodos y transiciones, y de un formulario. En consecuencia, la especificación de las restricciones sobre los campos de éste pertenece a un nivel de detalle superior. Parece lógico pensar que pertenece a otro paso distinto dentro de la secuencia de transformaciones. También avala esta decisión el hecho de que la información introducida se refiere a restricciones sobre la validación de los datos. Son, por tanto, operaciones para las que se requiere la intervención de un elemento computacional capaz de llevarlas a cabo.

Una vez que se ha recopilado toda esta información, se añadirá al XML inicial. Para cada campo del formulario, se agregarán los atributos definidos para señalar estas características de los mismos. El resultado conformará el PIM.

6.4 OBTENCIÓN DEL PSM

Este último modelo ampliará el anterior con las propiedades que lo vinculan a una plataforma concreta. Ésta radicará en tres aspectos concretos, que tendrán su representación en el XML resultante.

- *Configuración del portal del usuario.*

  Esta característica determinará qué campos del formulario se mostrarán en el portal del usuario. Además, diferenciará entre aquéllos que servirán como enlace al formulario desde este portal.

- *Configuración del sistema de envío de correos.*

  Servirá fundamentalmente para guardar la información que hace referencia a la cuenta de envío de tipo SMTP desde la que se enviarán los emails: servidor, número de puerto, uso de conexión segura, nombre de usuario y clave. También se solicitarán las direcciones de correo electrónico de los participantes. Estos datos no se han solicitado anteriormente porque están muy relacionados con la plataforma tecnológica empleada para la distribución de avisos. Gracias a ello, se admiten modelos PSM que incorporen métodos de notificación alternativos, por ejemplo, a través de mensajería telefónica.

  De manera análoga a como se definió para el portal, también se establecerán los campos de los registros que aparecerán en los emails de notificación.

- *Configuración del estilo.*

  Este dato guarda la ruta de la hoja de estilo CSS que se empleará para personalizar el aspecto del portal y del formulario.

Cuando la herramienta de conversión solicite esta información, el usuario puede que la desconozca y no sepa cómo obtenerla. Esto no debe ser un impedimento para la correcta generación de la aplicación final. Por ello, en la transformación se deben imponer valores por defecto que garanticen el adecuado funcionamiento del sistema resultante.

6.5 OBTENCIÓN DEL CÓDIGO FUENTE

El paso final de esta secuencia consiste ya en la creación de la codificación funcional del formulario, su workflow y los demás componentes. Tras este paso, el sistema debe quedar instalado y en funcionamiento.

6.5.1 Aspecto visual

Desde un punto de vista externo, el resultado estará compuesto por dos pantallas principales.

La primera es la correspondiente al portal del usuario, donde, previo inicio de sesión, un usuario será capaz de ver un resumen de las tareas que requieren su intervención. En la siguiente imagen se muestra el esquema de esta pantalla.

<table>
<tr><td colspan="2"><ÁREA PARA EL TÍTULO></td></tr>
<tr><td><ÁREA PARA EL LISTADO DE ESTADOS></td><td><ÁREA PRINCIPAL></td></tr>
</table>

Figura 6.25. Esquema de la pantalla del portal

Dispondrá para ello de un listado con todos los estados que forman el proceso que rige el formulario. La tarea que en el modelo primario partan del evento de inicio, correspondiente a la definición del formulario, no se incluye.

El término que se empleará para su designación, coincidirá con el nombre especificado para la tarea en el modelo BPMN. Cuando se trate de una tarea para la creación de registros, este texto contendrá un enlace que llevará al formulario. A través de él, se podrá añadir un nuevo registro. Sin embargo, cuando se trate de una acción de edición, se listará la relación de registros de información apropiados, es decir, los que están en disposición de avanzar hacia el estado elegido. Además, aparecerá con el total de las actuaciones del usuario en sesión pendientes, que provocarán el cambio hacia el estado concreto. A continuación, se muestra un ejemplo de esta pantalla con información hipotética.

<table>
<tr><td colspan="2">Portal del usuario</td></tr>
<tr><td>▪ CREATION
▪ EDITION_1 (0)
▪ EDITION_2 (2)
▪ PUBLISHING (0)</td><td>Bienvenido al portal del usuario</td></tr>
</table>

Figura 6.26. Ejemplo de pantalla principal del portal del usuario

Como se aprecia, un registro gestionado a través de la aplicación generada puede pasar por los estados *CREATION*, *EDITION_1*, *EDITION_2* y *PUBLISHING*. En este caso, el usuario que se ha registrado sólo tiene tareas pendientes que afectan al estado *EDITION_2*. El hecho de que aparezca subrayado denota que provee un hipervínculo, a diferencia de las otras dos líneas. Esto significa que debe rellenar los campos establecidos del formulario para que esos dos registros pasen a dicho estado, continuando así el workflow definido. En esta situación, sólo *EDITION_2* aparecerá marcado como un enlace. Al acceder a él, el usuario podrá ver qué registros de información concretos están a la espera de su actuación.

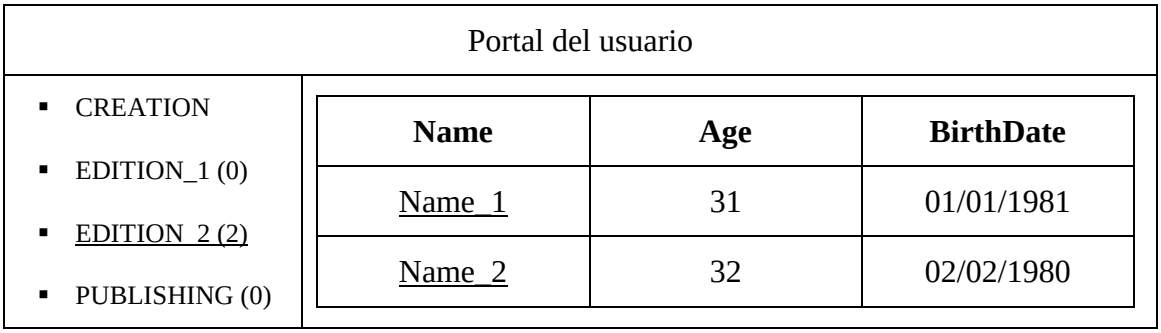


Figura 6.27. Ejemplo de pantalla del portal del usuario con un listado de registros

Finalmente, después de que el usuario haya pulsado el enlace correspondiente al registro que desea editar (situado sobre *Name_1*), visualizará sus datos sobre el formulario, todo ello enmarcado en el propio portal.

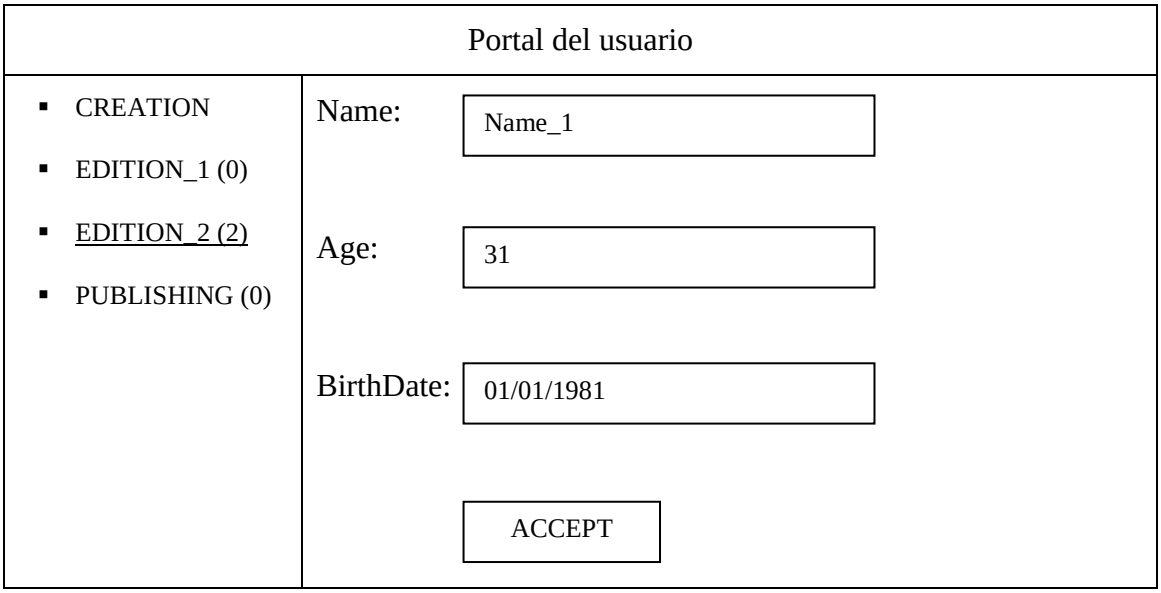
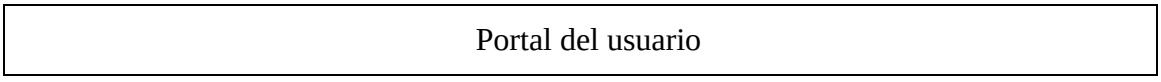


Figura 6.28. Ejemplo de pantalla del portal del usuario con el formulario

En caso de que el formulario disponga de más campos sobre los que el usuario no puede actuar, podrá leer su contenido, pero no modificarlo. Tras concluir la tarea, el portal debe retornar a su presentación inicial, en la que se han actualizado los conteos de los distintos estados.



▪ CREATION ▪ EDITION_1 (0) ▪ EDITION_2 (1) ▪ PUBLISHING (1)	Bienvenido al portal del usuario

Figura 6.29. Ejemplo de pantalla principal actualizada del portal del usuario

6.5.2 Sistema de almacenamiento persistente

El resultado empleará una serie de tablas en una base de datos, independientemente de la modelización del workflow:

- *Usuarios.*

  Esta tabla contendrá los datos de acceso de las personas autorizadas a ejecutar alguna tarea requerida para continuar el proceso del formulario. Básicamente, se tratará de su nombre personal, nombre de su usuario dentro del sistema y su contraseña.

- *Registros.*

  Los registros se información se almacenarán en una tabla única, cuyos campos dependerán de la modelización del proceso.

- *Transiciones activas.*

  Para cada registro creado del formulario, es necesario guardar qué transiciones son susceptibles de ser franqueadas, cuando se den las condiciones oportunas. Son fundamentales para controlar el acceso al formulario y enviar a la persona adecuada los correos de notificación. Cuando se traspase alguna de ellas, dejará de estar activa y será remplazada aquéllas que resulten de la aplicación de las reglas establecidas en las secciones anteriores.

El resto de información provista por la modelización del proceso influirá en la generación del código fuente. Por ello, debido a su invariabilidad a lo largo de la vida de la aplicación, estas imposiciones quedarán reflejadas en el código de una manera estática. Un ejemplo de ello es la decisión acerca de si un registro puede ser modificado para alcanzar un estado concreto, analizando para ello si el usuario está autorizado y si está activa la transición que llevará a ese estado.

6.5.3 Estructura de la aplicación

La aplicación generada tiene como principal distintivo su carácter interactivo. Por este motivo, su arquitectura ha de respetar el patrón Modelo-Vista-Controlador.

La vista contendrá las pantallas que se han señalado en el apartado anterior. El hecho de que la estructura general de estas pantallas esté bien definida desde el

principio, permitirá incluir ficheros que ya la especifiquen antes de cualquier modelización. De esta manera, se impone un diseño global fijo, donde se marcan las áreas dependientes de cada modelización: conjunto de estados y conteos, listado de registros y formulario. Gracias a esto, el proceso de creación de los ficheros que componen las pantallas será más liviano, pues existe una base sobre la que será necesario realizar los mínimos cambios posibles para su adaptación.

Por su parte, la responsabilidad principal del modelo será validar los datos remitidos por un usuario en un intento de acceso a la aplicación. También será el responsable de crear y modificar los registros de información alojados en la tabla correspondiente de la base de datos, después de verificar que no se violan las restricciones impuestas sobre los campos. También se ocupa de la función de envío de emails de notificación. Por último, se empleará este componente para la autenticación de los usuarios, así como para la obtención de listado de estados y conteos (teniendo en cuenta para éstos las acciones para las que el usuario en sesión está autorizado).

Para la escritura y la recuperación de la información gestionada por el formulario principal, se aplicará una simplificación del patrón DAO. Éste se basa en un elemento que provee las operaciones básicas de acceso a una base de datos: conexión, desconexión y ejecución de sentencias. Por otra parte, existirá una clase que emplea este componente, cuyos objetos almacenan la información contenida en una fila de una tabla en base de datos, empleando una correspondencia directa con sus atributos. Incluirá métodos que encapsularán la formación de las sentencias necesarias, así como la invocación adecuada para su ejecución. Por motivos de eficiencia y seguridad, en la práctica proveerá una función de este tipo por cada tarea o estado de los registros. La causa de esta distinción es garantizar que en cada operación sean actualizados exclusivamente aquellos campos que el usuario haya podido modificar. La ventaja de DAO radica en que se hace transparente al usuario la comunicación con la base de datos. Dado que está garantizado que cada estado posee un nombre que hace referencia a ella de manera unívoca, el título de estas funciones se puede componer a partir de este identificativo, con lo que se podrán establecer relaciones claras entre las tareas y sus funciones de guardado.

Finalmente, el controlador es el elemento mediador entre ellos. Por una parte, centraliza la recepción las peticiones del usuario debidas a su interactuación con las distintas pantallas. Éstas aparecerán como consecuencia de las pulsaciones sobre los enlaces, o de envíos de datos, como los de identificación del usuario o los contenidos en los campos del formulario.

Capítulo 7. Desarrollo del prototipo

Esta parte del trabajo está dedicada a la explicación del desarrollo del prototipo, haciendo hincapié en la justificación de las decisiones tomadas sobre las tecnologías usadas y las implementaciones realizadas.

Durante este proceso se hará referencia a los hitos que se han marcado como objetivos del trabajo, de manera que quede patente su consecución mediante la herramienta aportada.

7.1 LENGUAJE DE PROGRAMACIÓN ELEGIDO

Antes de comenzar el desarrollo del prototipo, es importante escoger bien los lenguajes de programación que servirán como base para su creación. Esta opción se debe tomar en base a una serie de criterios fundamentales.

- *Entorno de instalación.*

  En lo que afecta a la aplicación final, es de tipo web, por lo que el lenguaje seleccionado debe estar orientado a este ambiente. Respecto a las herramientas de conversión entre modelos, pueden estar destinadas a este contexto, pero también a un entorno de escritorio.

- *Conocimientos disponibles.*

  La persona encargada de desarrollar las herramientas de transformación entre modelos debe tener un dominio suficiente del lenguaje empleado para su creación. A su vez, también debe manejar con soltura el lenguaje sobre el que se creará la aplicación final, resultante del uso del prototipo.

- *Soporte para lograr los objetivos.*

  Existen frameworks y funcionalidades no disponibles en todos los lenguajes de programación. Por ello, se debe dedicar un tiempo a valorar las ventajas del uso de los potenciales escogidos. En lo que afecta al trabajo presente, existen tres características fundamentales: interactividad, persistencia de la información y manejo de XML.

- *Procesos adicionales.*

  Los lenguajes de programación están divididos en dos grandes grupos. Por una parte, los compilados, como J2EE o .NET, requieren de una conversión a un código que pueda ser interpretado por el sistema ejecutor correspondiente, además de un proceso de despliegue complejo. Por su parte, otros como PHP o RubyOnRails son directamente ejecutados por el servidor, y su despliegue consiste en copiar el directorio de la aplicación al servidor.

En primer lugar, el conocimiento de que dispone el desarrollador no debe forzar una decisión tomada únicamente según este criterio. Los lenguajes de programación tienen sus particularidades pero muestran una sintaxis similar. Por ello, emplear uno no usado anteriormente no debe ser traumático, cuando ya se cuenta con experiencia en la programación y se dispone de documentación sobre la que apoyarse. No obstante, un mínimo periodo de adaptación será necesario.

Como ya se indicó, el producto final consiste en un portal web. Las herramientas de conversión entre modelos admitirían perfectamente un desarrollo para escritorio. Sin embargo, para mantener la tendencia, y posibilitar que el prototipo y sus aplicaciones generadas convivan en el mismo entorno, se optará por implementarlas para la Web.

Finalmente, restan los factores verdaderamente determinantes. Como se ha visto durante el análisis del estado del arte, J2EE provee una serie de frameworks cuya aplicación sería muy útil en este prototipo. El inconveniente es que requiere unos procesos de compilación y despliegue que dificultarían la generación de los formularios. Aparte, se basa en diversos ficheros de configuración que es necesario proveer. Son todos estos puntos fuentes potenciales de un gran número de problemas adicionales. Sin embargo, la información recibida durante estas fases ayudará a depurar el funcionamiento de las herramientas de transformación. Se trata de una primera verificación de la validez del código generado, antes de probar el funcionamiento de la aplicación final.

Por su parte, PHP no aporta tanta funcionalidad, por lo que en muchos casos habrá que implementarla desde cero. Sin embargo, esta característica, junto con el hecho de que sea interpretado, hace que sea un lenguaje muy simple, que proporciona al programador una gran libertad de actuación. De esta manera, los desarrollos se pueden ajustar perfectamente a los requisitos descritos. Además, al no requerir compilaciones ni despliegues complejos, tanto el código de las aplicaciones de conversión como el del resultado estarán accesibles. De esta manera, cualquier persona que conozca su funcionamiento podrá alterarlo, con la ventaja de poder comprobar los resultados inmediatamente. En consecuencia, PHP es la opción escogida.

7.2 MATERIALES EMPLEADOS

Para poder ejecutar el prototipo, se requiere de una máquina de prestaciones medias. Ha sido desarrollado y probado sobre el sistema operativo Windows 7, por lo que se recomienda emplear el mismo. Debe tener instalado un servidor web capaz de interpretar PHP, así como un servidor de bases de datos compatible con MySQL. Es posible utilizar un servidor integrado, como XAMPP.

Los ficheros que componen el prototipo se encuentran agrupados en la carpeta */prototipo*. Se debe copiar la carpeta completa al directorio raíz del servidor.

Es importante señalar que para el empleo de las bases de datos, se establece una conexión con la máquina local (*localhost*), a través del usuario *root*, con la contraseña vacía. Son los parámetros por defecto de XAMPP. Si se desea modificarlos, es necesario actuar sobre los ficheros */model/ApplicationGenerator.php*, */model/DB_mysql.php* y */scaffold/model/DB_mysql.php*. En ningún caso se debe modificar el argumento que indica el nombre de la base de datos, pues puede provocar funcionamientos incorrectos.

A su vez, para que el envío de correos electrónicos funcione adecuadamente, se deben instalar los ficheros proporcionados para este fin. Por una parte, el contenido de la carpeta *librerias_openssl* (*libeay32.dll*, *php_openssl.dll* y *ssleay32.dll*), debe ser copiado al directorio *C:\Windows\System32*. Posteriormente, es necesario instalar *Win32OpenSSL-0_9_8j.exe*. El último paso consiste en insertar en el fichero *php.ini* la línea *extension=php_openssl.dll*, en la sección de extensiones activas.

7.3 PROC2FORM

La herramienta de transformación de modelos, permite en primer lugar seleccionar el fichero XPDL. Para el ejemplo, se partirá del siguiente diagrama BPMN.

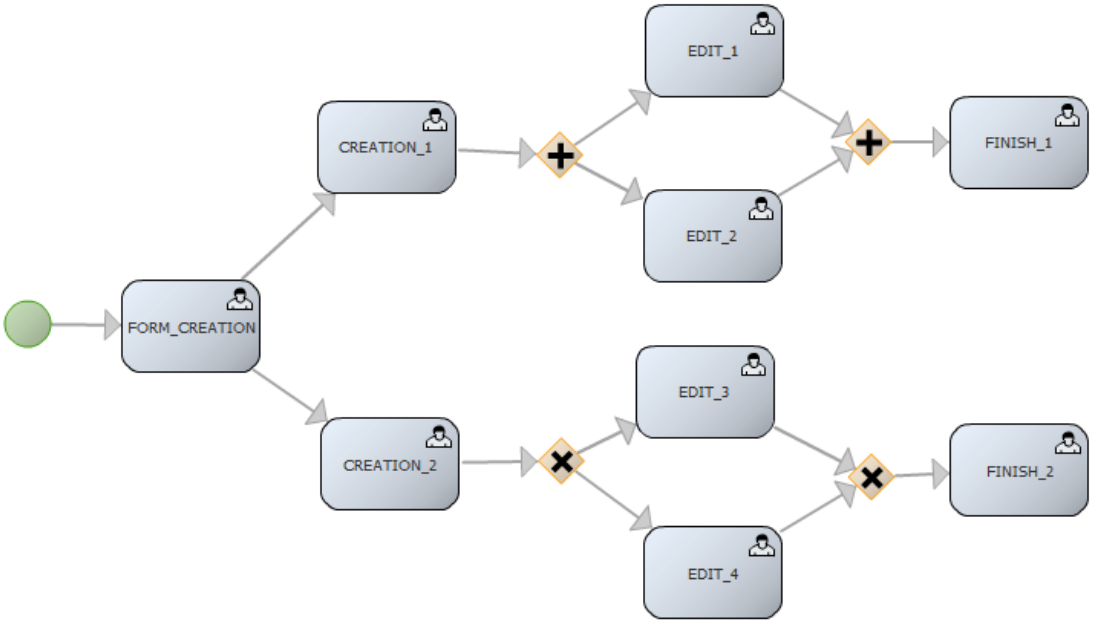


Figura 7.1 Diagrama BPMN de ejemplo

A partir de dicho documento XPDL, se recuperará la información básica del formulario y su flujo de trabajo. El diseñador del proceso debe ser muy cuidadoso a la hora de componer el diagrama, respetando las reglas relacionadas con su composición. Esto se debe a que el prototipo presupone que su formato se ajusta a dichas reglas. Por este motivo, no comprueba su cumplimiento. La no adopción de esas normas puede producir resultados impredecibles.

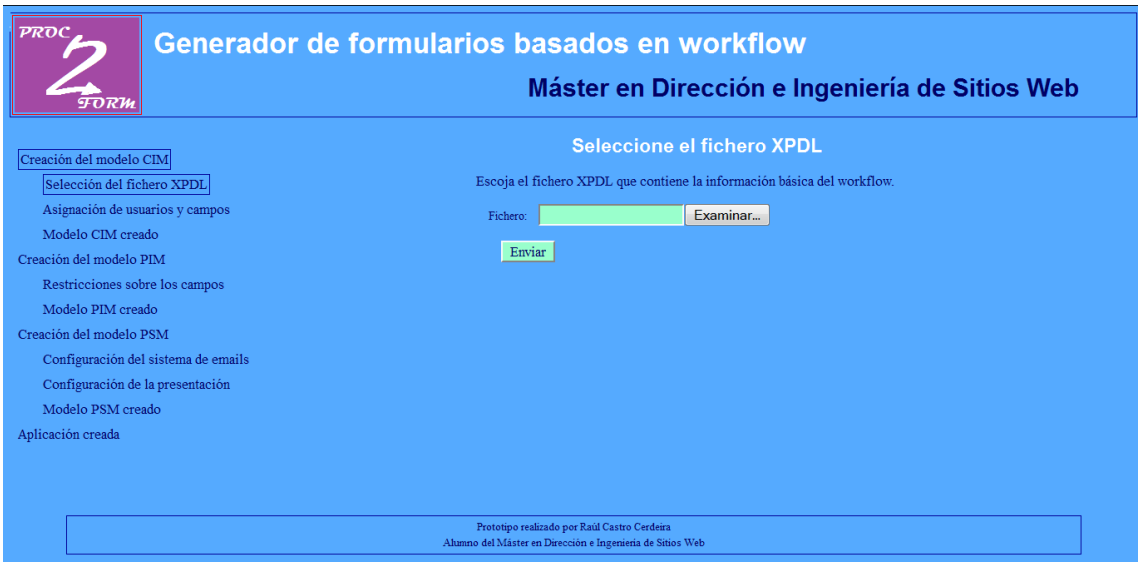


Figura 7.2. Pantalla de selección del fichero XPDL en Proc2Form

Tras detectar los estados especificados, así como los campos que conforman el formulario y los participantes, la aplicación permite realizar la asignación de estos elementos entre los distintos estados. A través de la pantalla resumen, es posible acceder al formulario de asignación de elementos, a través del hipervínculo dispuesto sobre los textos *X usuarios* ó *X campos*.

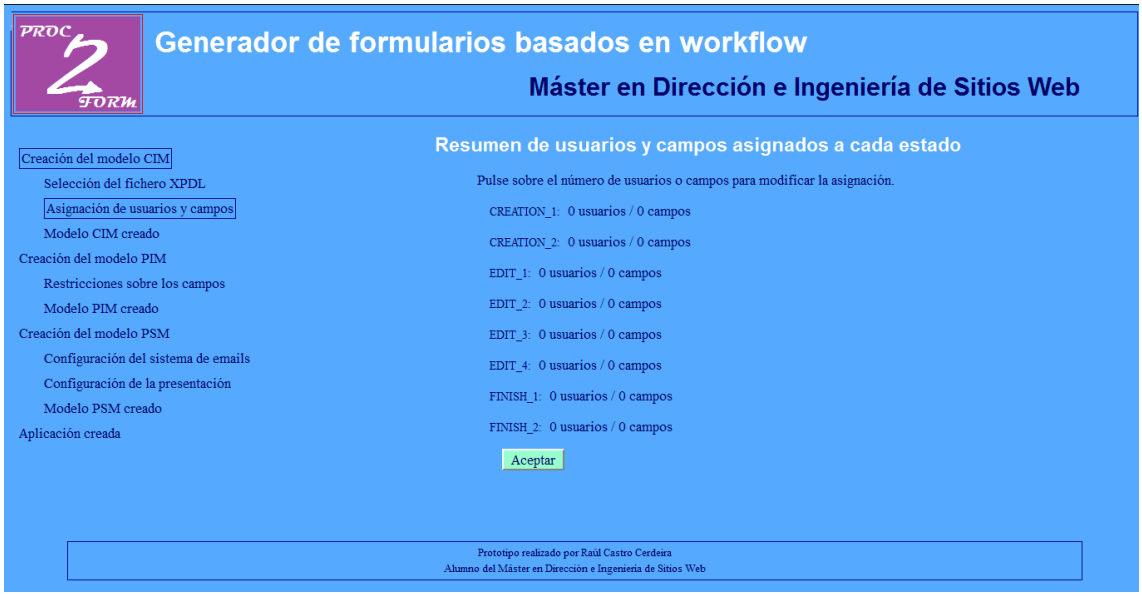


Figura 7.3. Pantalla de resumen de asignación de campos y usuarios en Proc2Form

Para el ejemplo, se asignará el primero de los dos usuarios (*Raúl Castro*) a las tareas de creación de registros.

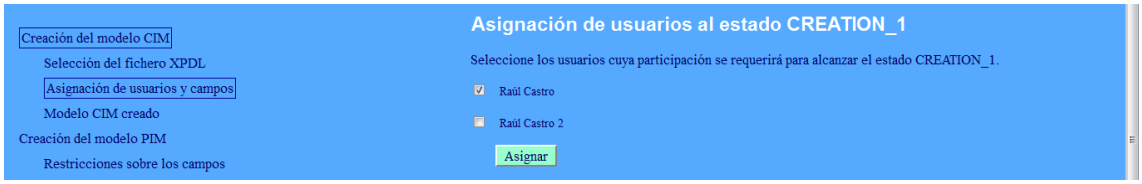


Figura 7.4. Pantalla de asignación de usuarios al estado CREATION_1

Por su parte, *Raúl Castro 2*, será vinculado con las tareas de edición.

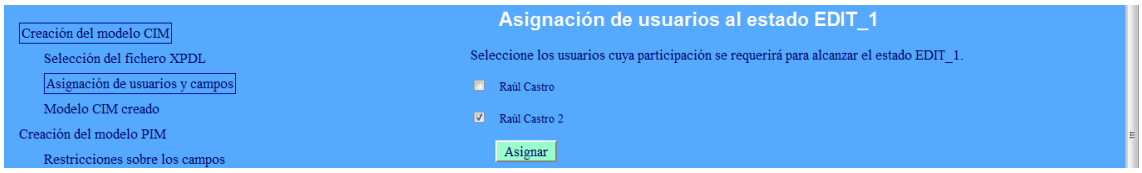


Figura 7.5. Pantalla de asignación de usuarios al estado EDIT_1

Respecto a los campos, los cinco primeros serán relacionados con las tareas de creación. Por su parte, las tareas restantes mostrarán activos los seis campos restantes. Tras ello, es posible generar el CIM.

Posteriormente, se recogerán los datos agregados al modelo, para la composición del PIM. Ésta afecta a restricciones sobre los valores aceptados en los campos de inserción directa y para los valores mostrados en los campos de selección (de tipo radio

y menús desplegables). Obviamente, los valores indicados tienen que poseer un formato que concuerde con el tipo de dato que se trate.

Figura 7.6. Restricciones sobre los campos del formulario en Proc2Form

En este punto, el sistema está en disposición de crear el modelo PIM.

El último modelo, PSM, dispondrá de información sobre el sistema de envío de emails. Ésta comprenderá la cuenta de envío de los mismos, así como las direcciones a que se remitirán las notificaciones a los usuarios. En la propuesta inicial, la personalización de la cuenta de envío admitiría una mayor flexibilidad (servidor, número de puerto, establecimiento de SSL, etc.). No obstante, estas características se han obviado, para simplificar el prototipo.

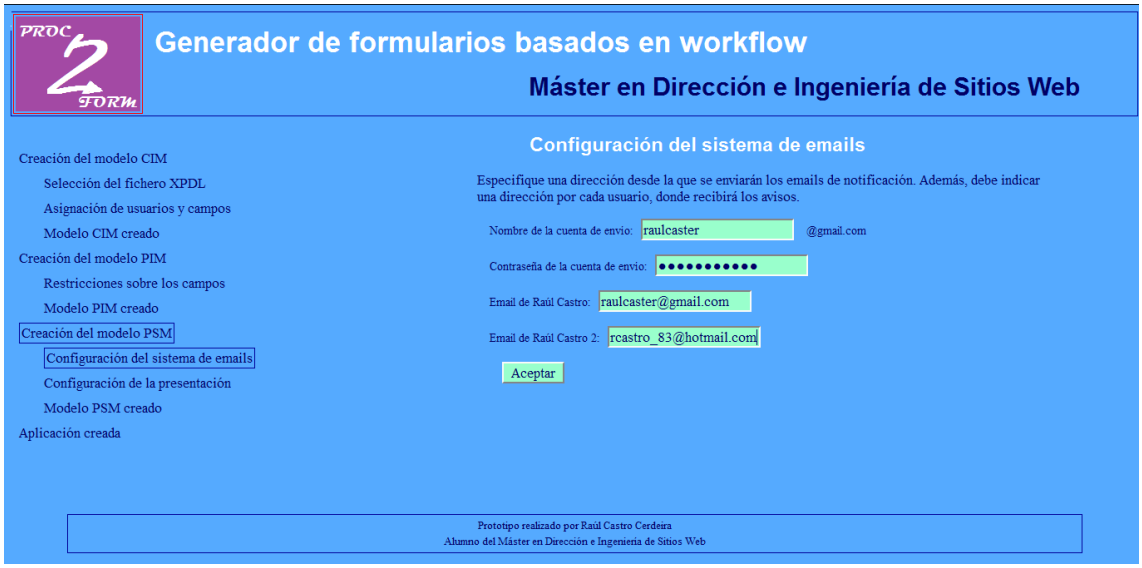


Figura 7.7. Definición del sistema de notificaciones por email en Proc2Form

El segundo paso afecta a aspectos visuales. En primer lugar, acerca del aspecto de la presentación de la aplicación generada. Si no se declara ninguna hoja de estilo, se tomará la empleada por el propia herramienta *Proc2Form*. También se especificarán los campos del formulario mostrados en los listados del portal, así como en los correos de notificación.

Figura 7.8. Definición de las características visuales en Proc2Form

La última transformación ya propicia la aparición de la nueva aplicación. Ésta incluye la generación del código fuente y de la base de datos requerida. Tras acceder a la nueva aplicación, se inicia la sesión con el usuario asignado a las tareas de creación.

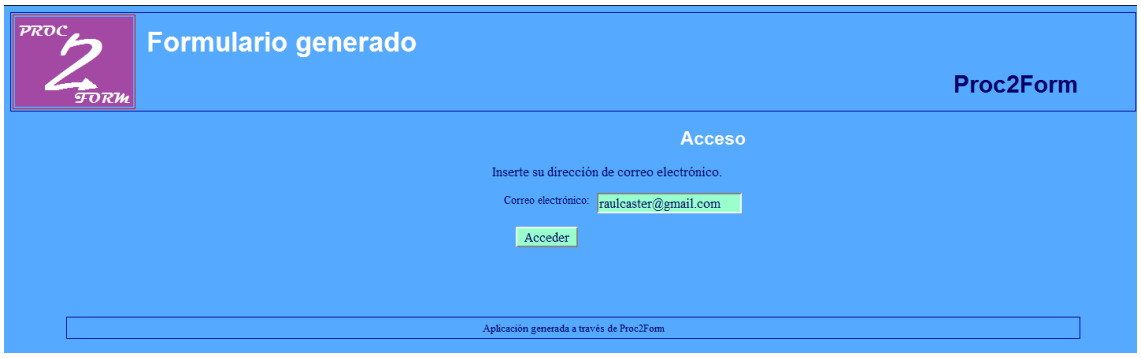


Figura 7.9. Inicio de sesión con el usuario asignado a las tareas de creación

A través, por ejemplo, de *CREATION_1*, se dará de alta un nuevo registro. En ese caso, el usuario sólo tendrá activos aquellos campos que esté autorizado a modificar. Cuando pulse sobre el botón *Guardar*, se verificará el cumplimiento de las validaciones establecidas.

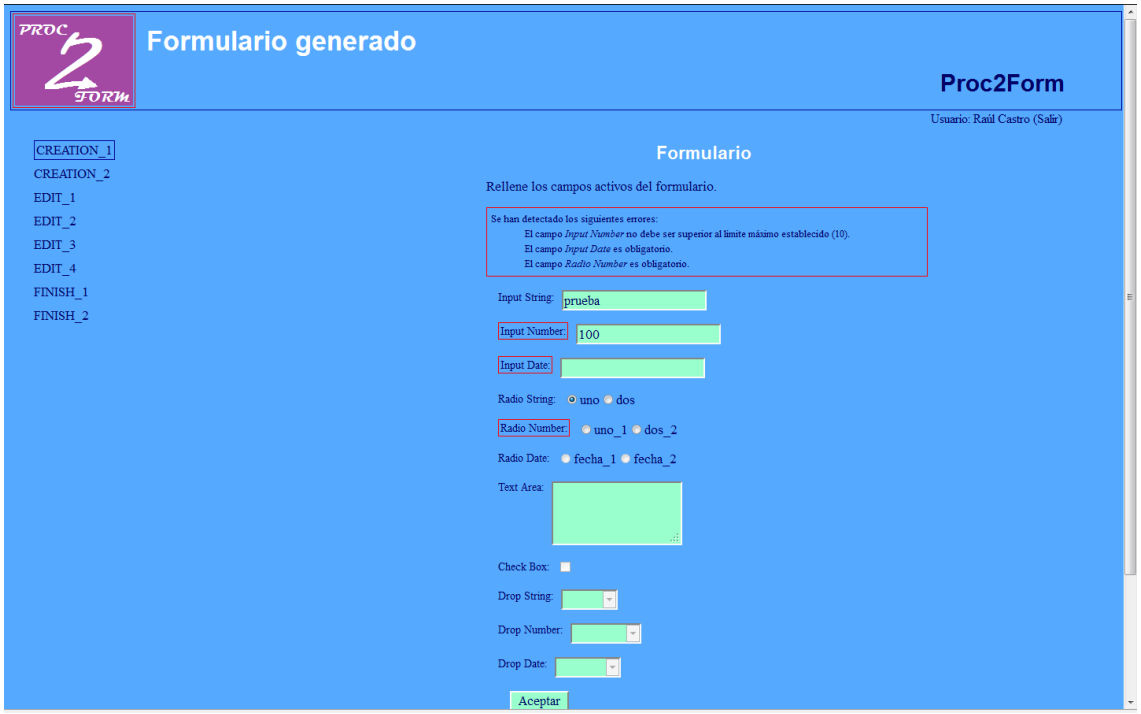


Figura 7.10. Intento fallido de creación de un registro

Cuando se superen todas las validaciones, se creará el registro en base de datos. En ese momento, el otro usuario recibirá los correspondientes emails de notificación.

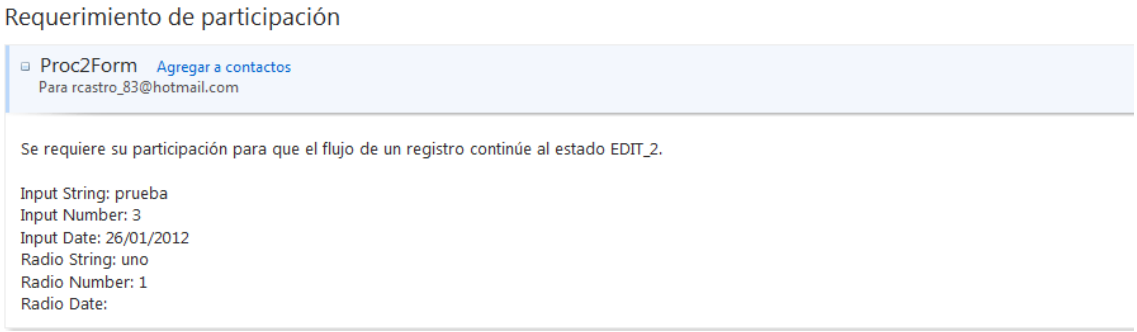


Figura 7.11. Email de notificación

A partir del aviso, este usuario puede proceder a modificar los datos que se hayan asignado a la tarea correspondiente.

Tanto el contenido de los modelos, como el código fuente de la aplicación generada pueden consultarse en la sección de anexos.

PARTE IV. Conclusiones

Capítulo 8. Conclusiones

La principal finalidad de este capítulo es demostrar la validez de la propuesta planteada. Éste quedará probada con la verificación de la hipótesis de partida. Concretamente, se debe revisar el cumplimiento de cada objetivo establecido.

A continuación, se resumen las propiedades del modelo propuesto. También se explican las ventajas que aporta, contrastadas con las características de las otras alternativas analizadas.

Finalmente, para poder dar continuación a esta investigación, se establecen unos nexos que permitan retomar este trabajo. Este listado corresponde, principalmente, a posibilidades no exploradas y ampliaciones posibles. Su aparición se corresponde con la amplitud del tema tratado, así como del interés por comparar diferentes opciones durante el desarrollo de la labor investigativa.

8.1 VERIFICACIÓN, CONTRASTE Y EVALUACIÓN DE LOS OBJETIVOS

Como se vio en la introducción los aspectos distintivos que caracterizan a los formularios colaborativos se pueden sintetizar en una serie de acciones que se disponen a lo largo de un proceso. Estas acciones determinan el hecho de completar los campos especificados del formulario definido. Son acometidas por unos usuarios autorizados, que forman un conjunto concreto establecido para cada tarea. También, cada una de estas tareas definirá un estado del ciclo de vida de los registros.

Salvando estos componentes diferenciadores, la infraestructura de la aplicación generada es común para todas las implementaciones de los formularios. Los elementos que en la práctica compondrán este sistema sustentador son:

- *El portal del usuario.*

  Aparte de proporcionar a la persona registrada información sobre los registros que requieren de su participación, ésta podrá operar sobre ellos a través del formulario enmarcado en el contexto que provee.

- *El sistema de almacenamiento.*

  El elemento encargado de la conexión con la base de datos, así como la ejecución de las diversas sentencias de escritura y consulta es común en todos los formularios derivados.

- *El sistema de notificaciones.*

  Su función es trasladar a los usuarios, vía correo electrónico, las notificaciones de requerimiento de participación. Éstas son debidas a la ocurrencia de registros que están a la espera de su intervención, para alcanzar el siguiente estado del flujo. Gracias a ella, podrán continuar su ciclo de vida.

En base a estas condiciones, se ha establecido la siguiente hipótesis de partida:

La especificación MDA unida al modelado de procesos
permite crear de manera simple un formulario web
gestionado mediante un workflow.

La demostración de la hipótesis se basó en la consecución de un objetivo global. Éste consiste en la posibilidad de emplear herramientas de modelización de procesos para representar el ciclo de vida de una información determinada. Para ello, se debe proporcionar un formulario que tenga activos los campos que corresponda en cada momento, de manera que sólo un usuario autorizado pueda completarlos. Ambos conjuntos, esto es, campos disponibles y usuarios autorizados, deben poder ser establecidos para cada tarea.

Éste objetivo principal se descompuso en múltiples metas. Su consecución es analizada a continuación.

- *Estudio y valoración de las principales teorías relacionadas con el desarrollo dirigido por modelos y el modelado de procesos.*

En primer lugar, en el capítulo *2 Modelos de software* se ha realizado un análisis de las principales teorías relacionadas con el desarrollo dirigido por modelos, haciendo especial énfasis en MDA. Se han aportado las ventajas que aporta al desarrollo de software. Este estudio también ha incluido especificaciones propuestas para la implementación de esta técnica, aunque posteriormente no han sido utilizadas.

Respecto al modelado de procesos, éste temática ha sido cubierta en el capítulo *3 Modelado de procesos*. Previamente a su desarrollo, se han establecido conceptos teóricos referidos a la gestión del flujo de trabajo y de los procesos empresariales. Por último, al igual que ocurrió para MDA, también se proporcionó una relación de lenguajes de modelado propios de este disciplina.

- *Definición de una metodología para el desarrollo de formularios web mediante la modelización de su workflow subyacente.*

El logro de este objetivo ha propiciado el capítulo *6 Enfoque preliminar*. En su primera parte, se describe la manera en que se aplicará MDA a la modelización del formulario. Ésta afecta a la creación de los sucesivos modelos (CIM, PIM y PSM), así como a los detalles propios de cada modelo.

Su contenido establece las notaciones BPMN útiles para el desarrollo de la propuesta. También se han impuesto reglas sintácticas sobre este lenguaje, para adecuar los diagramas a la modelización de los formularios. Se ha comprobado la equivalencia en XPDL de los elementos BPMN empleados. Esto fue fundamental para establecer los criterios que permitieran analizar este documento con el fin de extraer de él la información de interés.

Se detectaron e indicaron los casos en que BPMN no proporcionaba un soporte adecuado para la representación de la información necesaria. En esas ocasiones, se propuso la implementación de una herramienta que aceptara la definición de esos datos por parte del usuario.

- *Se deben proveer dos sistemas anexos para la notificación de acciones en espera para la consecución de los procesos.*

Dentro del diseño planteado en el capítulo *6 Enfoque preliminar*, se propuso un esbozo del portal del usuario. En éste, además de servir como contenedor para el formulario, se requirió la característica de informar a la persona en sesión sobre los registros que requerían su intervención para alcanzar un estado determinado. Dicho de otra manera, el portal debería informar sobre sus acciones pendientes sobre los registros, con el fin de que estos pueden continuar su ciclo de vida.

En lo que atañe al sistema de notificación a través del correo electrónico, la adquisición de la información requerida por él se vinculó con la formación del modelo PSM.

- *Crear un prototipo que implemente la metodología.*

El generador de código *Proc2Form* ha sido implementado según las características impuestas en el capítulo *6 Enfoque preliminar*. En el capítulo *7 Desarrollo del prototipo*, se puede visualizar la manera en que provee la funcionalidad requerida. Respecto a su código fuente, puede consultarse en el anexo *C Código fuente*.

- *Proporcionar una aplicación generada mediante la metodología establecida*.

Dentro del capítulo *7 Desarrollo del prototipo*, se desarrolla un ejemplo sobre el prototipo. En él se muestra el diagrama BPMN inicial, así como la secuencia de obtención de datos y las transformaciones necesarias para la creación de los diversos modelos. Finalmente, se presenta la aplicación producida, junto con su modo de operación. Tanto el contenido de los modelos como el código que implementa el formulario se encuentran disponibles en el anexo *C Código fuente*.

8.2 SÍNTESIS DEL MODELO PROPUESTO

El modelo propuesto persigue la creación de un formulario web colaborativo. Éste se sitúa en el contexto de un portal de usuarios, y con un sistema de aviso de requerimiento de participación a través de correo electrónico. Su ciclo de vida puede concebirse como un proceso formado por una secuencia de estados que serán atravesados por los registros. Cada uno de estos estados propiciará una acción, consistente en el cumplimiento de determinados campos del formulario, por el usuario adecuado. El objetivo de esta tarea es trasladar el registro al estado que representa.

En vista de que los aspectos distintivos de cada implementación de los formularios son conocidos (principalmente, ciclo de vida de los registros, campos asociados a cada tarea y usuarios autorizados a su realización), esto propicia la creación de una herramienta que acepte la definición de estos datos diferenciadores. Su unión formará un modelo, esto es, una simplificación del sistema resultante especificada a través de un lenguaje adecuado. A partir de ellos, mediante una aplicación de generación de código, será posible producir un nuevo desarrollo completamente funcional. El procedimiento consistirá en ensamblar sobre las áreas comunes a todos los formularios, aquellas particularidades debidas a la información señalada.

MDA es una técnica que permite la orientación a modelo de un desarrollo de software. Establece una serie de transformaciones entre modelos, para finalmente generar el código fuente. Estos modelos son:

- *Modelo independiente de la computación (CIM).*

  Únicamente contiene detalles sobre los requisitos de la aplicación. Nunca especifica aspectos de su estructura o su entorno.

- *Modelo independiente de la plataforma (PIM).*

  Oculta los aspectos del sistema relativos a particularidades de cualquier plataforma concreta. Introduce la abstracción de máquina virtual neutra. Ésta consiste en una máquina genérica, con funcionalidades globales incluidas en la práctica totalidad de las plataformas disponibles.

- *Modelo específico de la plataforma (PSM).*

  Consiste en el modelo PIM, especializado para una plataforma específica. Por ello, incluye los detalles relativos a la implementación sobre ella de la aplicación final.

Según esta secuencia, habrá que adaptar la modelización del formulario. Se trata, por tanto, de declarar qué información se recogerá en cada uno de los tres modelos.

- *Contenido del CIM.*

  Para su construcción, se partirá de un diagrama BPMN. Éste ya aporta la relación de campos que componen el formulario, el conjunto de estados disponibles y el ciclo de vida de los registros a través de ellos.

Tras la conversión del diagrama a su representación XPDL equivalente, se procederá al establecimiento de los campos activos para la realización de cada tarea. También se señalarán los usuarios que pueden acometer cada una de esas acciones.

Todos estos datos constituirán el modelo CIM.

- *Contenido del PIM.*

El modelo PIM está dedicado a la imposición de restricciones sobre los valores aceptados por los campos del formulario. Su espectro cubre validaciones de tipo longitud máxima (campos de entrada directa, de tipo textual), establecimiento de un rango de valores válidos (campos de entrada directa, de tipo numérico o fecha), y la definición de dos secuencias, una de valores y otra de etiquetas para campos seleccionables (botones de tipo radio y menús desplegables).

- *Contenido del PSM.*

Finalmente, la construcción del modelo PSM se divide en dos fases. La primera está orientada a la configuración del sistema de notificación a través del correo electrónico. Su objetivo es la asignación de una cuenta *Gmail* a través de la que se remitirán. Además, para cada usuario debe indicarse una dirección, a la que se le enviarán sus avisos.

La segunda parte se dedica a aspectos visuales de la aplicación final. Éstos comprenden el estilo mostrado y los campos del formulario que se incluyen en los listados de registros del portal y en los emails de notificación.

Tras la construcción del PSM, es posible proceder a la producción del código fuente. Éste se proporcionará en forma de un paquete que incluye todos los componentes necesarios.

8.3 APORTACIONES ORIGINALES

La propuesta planteada incorpora las siguientes aportaciones esenciales:

- *La modelización del flujo de trabajo permite adoptar una visión global del mismo.*

  De esta manera, se facilita al diseñador la detección de puntos débiles. En el tema concreto que ocupa este trabajo, estos consistirán en cuellos de botella o caminos con una baja circulación de registros.

  El primer caso se debe a una excesiva cantidad de registros a la espera de alcanzar un estado. La solución consistirá en realizar una bifurcación alternativa en el flujo y/o aumentar la asignación de participantes a la acción correspondiente. Respecto a las ramas que registran un tránsito de registros mínimo, puede ser conveniente desecharlas del modelo. Con ello también se conseguirá simplificar el modelo.

- *Esta modelización del formulario colaborativo se ha enfocado al cumplimiento de MDA.*

  Ello implica las mejoras en la portabilidad, documentación y mantenimiento. Todas estas ventajas se fundamentan en la relación directa que vincula los modelos con el código fuente que se produce gracias a su definición.

- *El diseñador no necesita conocer ningún detalle sobre la implementación del código final.*

  Únicamente se requiere que conozca el sistema de recopilación de datos. Éste se fundamenta, en primer lugar, en la definición BPMN del flujo y, posteriormente, en la agregación sucesiva de información mediante *Proc2Form*.

A continuación, se compararán las mejoras que proporciona el planteamiento, frente a otras investigaciones o productos relacionados.

- *El sistema propuesto garantiza la portabilidad.*

  El producto último es un paquete que incluye todos los componentes necesarios. En algunas de las soluciones analizadas, el código no está accesible como tal, en un formato manejable y fácilmente manipulable. Esto se debe, en unos casos, a que se imposibilita directamente llegar a él. En otras circunstancias, requiere el uso de una máquina virtual, es decir, un componente software que se encarga de manejar las diversas funcionalidades. Su comportamiento depende de ficheros de configuración XML. Éstos son los que realmente aportan las características distintivas del sistema del formulario.

- *La recolección de la información se hace de una manera simple.*

  Aunque se debe tener un conocimiento suficiente sobre los formatos empleados, éste no es particularmente extenso. Se trata de una consecuencia de

acotar su ámbito de aplicación al desarrollo de formularios web colaborativos con unas características concretadas de antemano. Las aplicaciones analizadas presentan éstas y otras capacidades de personalización. En muchos casos, su aparición complica innecesariamente la definición del sistema final. En los casos extremos, la herramienta de conversión entre modelos muestra una complejidad elevada en su manejo.

8.4 TRABAJOS DERIVADOS

Este Trabajo de Fin de Máster ha servido como base para la redacción de un artículo homónimo, aún inédito. En él se incluyen las investigaciones realizadas, además de presentar la técnica propuesta para el modelado de formularios colaborativos. Dicho artículo está disponible en el anexo *B Artículo presentado*.

8.5 LÍNEAS DE INVESTIGACIÓN FUTURAS

Se establecen las siguientes indicaciones para seguir desarrollando el tema tratado en el trabajo. Se deben a ampliaciones sugeridas sobre la propuesta e incursiones en ámbitos no explorados en esta investigación.

- *Representación gráfica completa del CIM.*

  Actualmente, parte de la inserción de los datos incluidos en el CIM, se lleva a cabo a través de una herramienta adicional. Sería muy interesante que esta información pudiera ser agregada directamente al diagrama BPMN que refleja el flujo de trabajo y la definición del formulario. Para ello, se pueden analizar las ventajas de otros editores distintos de *Apia Facilis* o, incluso, contemplar la posibilidad de proponer una nueva notación, con un editor (totalmente nuevo o modificado) capaz de manejar esa especificación.

- *Empleo de tecnologías MDA y XML.*

  MDA propone especificaciones para la definición de lenguajes. Éstas podrían ser empleadas para la definición formal del lenguaje diseñado para la composición de los modelos. Debido a que se ha empleado una estructura XML, se podría reforzar con los elementos de validación propios de esta tecnología, DTDs y esquemas. Gracias a ellos, se podrían introducir reglas sintácticas que ayudaran a comprobar la correcta estructura de los modelos. Por otra parte, XSLT aporta una buena opción para la conversión de modelos.

- *Aumento del conjunto de condiciones para la activación de las transiciones.*

  Para mejorar la definición de los flujos, se podría enriquecer el conjunto de condiciones necesarias. Ésta iría más allá de la mera estancia del registro que se desea modificar en un estado previo, unida a la autorización del usuario. Estas restricciones afectarían a los valores especificados en los campos de cada registro. De esta manera, el tránsito por los registros a través del flujo es dirigido a medida que se completan sus campos. Para ello, se puede imaginar un campo que presenta una pregunta. La respuesta, de tipo sí o no, determinará dos salidas diferentes, cuya apertura depende del valor escogido.

- *Inclusión de nuevas plataformas.*

  Una nueva implementación de *Proc2Form* podría generar versiones de los modelos PSM distintas a la proporcionada actualmente. Adicionalmente, el código generado podría ser implementado pensando en su ejecución sobre J2EE. Con ello, se podría aprovechar el estudio realizado sobre frameworks característicos de este entorno, como *Hibernate* o *Struts*. Finalmente, esta diversidad de versiones de PSMs y códigos fomentaría el estudio de los puentes definidos por MDA. Éstos se tenderían entre pares de elementos de ambos conjuntos, estableciendo una correspondencia entre componentes equivalentes.

- *Obtención de informes para la reingeniería del flujo de trabajo.*

La observación del funcionamiento de los procesos con el fin de cuantificar su eficiencia es una fase propia de su ciclo de vida. A partir de los resultados, es posible detectar puntos mejorables del diseño. Por ello, se podría añadir a la aplicación una nueva funcionalidad, consistente en la obtención de estas métricas.

En el caso de este trabajo, éstas estarían relacionadas con el tránsito de registros a través de los distintos estados del flujo. Mediante su análisis, se detectarían cuellos de botella o caminos con una circulación pequeña. El siguiente paso consistiría en un rediseño del proceso. En una versión avanzada, la herramienta podría disponer de cierta inteligencia para imponer de manera automática las modificaciones oportunas sobre los modelos y sobre las desarrollos generados.

Bibliografía

[AALS02] **W. M. P. van der Aalst, K. M. van Hee**; *Workflow Management: models, methods and systems*; The MIT Press, 2002.

[AALS03a] **W. M. P. van der Aalst, A. H. M. ter Hofstede, M. Weske**; *Business Process Management: A Survey*; junio 2003.

[AALS03b] **W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski y otros**; "Workflow Patterns"; *Workflow Patterns. Distributed and Parallel databases*; 2003.

[BOOC04] **G. Booch, A. Brown, S. Iyengar y otros**; "An MDA manifesto"; *The MDA Journal*, mayo 2004.

[BROC10] **J. Vom Brocke, M. Rosemann**; *Handbook on business process management 1*; Springer, 2010.

[BUSC01] **F. Buschmann, R. Meunier, H. Rohnert y otros**; *Pattern-oriented software architecture. A system of patterns. Volume 1*; Wiley, 2001.

[ESCO11] **E. Escott, P. Strooper, P. King y otros**; *Model-Driven Form Validation with UML and OCL*.

[FOWL04] **M. Fowler**; *UML distilled*; Addison-Wesley, 2004.

[FREU07] **P. Freudenstein, M. Nussbaumer y M. Gaedke**; "Model-driven Construction of Workflow-based web applications with domain-specific languages"*;* julio 2007.

[GAMM95] **E. Gamma, R. Helm, R. Johnson y otros**; *Design patterns. Elements of resusable object-oriented software*; Addison-Wesley, 1995.

[HAVE05] **M. Havey**; *Essential Business Process Modeling*; O'Really Media, 2005.

[HEME08] **Z. Hemel, R. Verhaaf, E. Visser**; "WebWorkFlow: An object-Oriented Workflow Modelling Language for Web Applications".

[KLEP04] **A. Kleppe, J. Warmer, W. Bast**; *MDA explained: The Model Driven Architecture: Practice and Promise*; Addison-Wesley, 2004.

[KO09] **R. K. L. Ko, Stephen S. G. Lee, E. W. Lee**; "Businees process management (BPM) standars: a survey"; *Business Process Management Journal, 2009*.

[MILL03] **J. Miller, J. Mukerji**; *MDA Guide version 1.0.1*; Object Management Group, 2003.

[MINT06] **D. Minter, J. Linwood**; *Beginning Hibertate. From Novice to Professional*; Apress, 2007.

[MINT08] **D. Minter**; *Beginning Spring 2: From Novice to Professional*; Apress, 2008.

[OMG02] **OMG**; *Meta Object Facility (MOF) specification*; Object Management Group, abril 2002.

[OMG03] **OMG**; *Common Warehouse Metamodel (CWM) Specification*; Object Management Group, marzo 2003.

[OMG06] **OMG**; *Object Constraint Language*; Object Management Group, mayo 2006.

[OMG008] **OMG**; *MOF Model to Text Transformation Language, v1.0*; Object Management Group, enero 2008.

[OMG10] **OMG**; *OMG Systems Modeling Language (OMG SysML)*; Object Management Group, junio 2010.

[OMG11b] **OMG**; *OMG MOF 2 XMI Mapping Specification*; Object Management Group; agosto 2011.

[OMG11c] **OMG**; *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*; enero 2011.

[PELA07] **B. C. Pelayo García-Bustelo**; "TALISMAN: desarrollo Ágil de Software con Arquitecturas Dirigidas por Modelos"; Universidad de Oviedo, 2007.

[ROUG07] **I. Roughley**; *Starting Struts 2*; C4Media 2007.

[RYGG06] **A. Rygg, P. Roe, O. Wong**; "GPFlow: An Intuitive Environment for Web Based Scientific Workflow". *IEEE*, 2006.

[SCHM06] **D. C. Schmith**; "Model-Driven Ingeneering"; *Computer, febrero 2006*.

[WHIT03] **S. A. White**; *Workflow handbook*; Future Strategies Inc., 2003.

Referencias web

[DAO11] *Data Access Object*.
Consultado en diciembre de 2011.
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[FRAM11] *Frameworks*.
Consultado en diciembre de 2011.
http://docforge.com/wiki/Framework

[HIBE11] *Hibernate*.
Consultado en diciembre de 2011.
http://www.hibernate.org/

[OMG11a] *Catalog of OMG Modeling And Metadata Specifications*.
Consultado en noviembre de 2011.
http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[PROC12] *ProcessMaker*.
Consultado en enero de 2012.
http://www.processmaker.com/

[SPRI11] *Spring*.
Consultado en diciembre de 2011.
http://www.springsource.org/

[STRU11] *Struts*.
Consultado en diciembre de 2011.
http://struts.apache.org/

[WEBD12] *WebDSL*.
Consultado en febrero de 2012.
http://webdsl.org

[WEBM12] *WebML*.
Consultado en enero de 2012.
http://www.webml.org/

[WEBR12] *WebRatio*.
Consultado en enero de 2012.
http://www.webratio.com/

[WORK11] *Workflow Patterns*.
Consultado en diciembre de 2011.
http://www.workflowpatterns.com

PARTE V. Anexos

Anexo A. Calidad de las fuentes

Durante el proceso de revisión de documentación, el investigador debe disponer de un criterio que permita determinar si la información aportada es válida. Esta validación es fundamental, pues sin ella, puede cimentar su trabajo sobre información errónea. Esta circunstancia concluirá con la nula fiabilidad del resultado y, por tanto, un esfuerzo del científico desperdiciado.

En este informe, se clasifican las fuentes empleadas en función de su temática. Se ha hecho un esfuerzo considerable en la búsqueda de información, siempre de diversas fuentes. Esta variedad permite contrastar los datos extraídos de su lectura, al revisar las referencias existentes entre documentos distintos que tratan una misma temática y la similitud de los conocimientos disponibles en ellos.

A.1 MODELOS DE SOFTWARE

La documentación leída para lograr un primer acercamiento a este tema fue *Model-Driven Engineering* de D. C. Schmith. Este artículo ha sido también empleado como referencia en la obra *TALISMAN: desarrollo Ágil de Software con Arquitecturas Dirigidas por Modelo*, de B. C. Pelayo García-Bustelo. En él, se establece un paradigma que es descrito en todas las obras relativas a este mismo tema, o a la estructura arquitectónica dirigida por modelos (MDA). Adicionalmente, se han encontrado artículos que inciden sobre esta temática, como es el caso de *MDE between Promises and Challenges*, de T. Gherbi, D. Meslati y I. Borne. Todos estos trabajos señalan unas características similares de este paradigma, además de describir las mismas ventajas en su aplicación.

MDA manifesto es una obra de referencia en este ámbito. Aparece frecuentemente en las referencias bibliográficas de cualquier artículo o trabajo que trate este tema. Esto es debido a que consiste en el asentamiento de las bases que caracterizan la arquitectura dirigida por modelos.

Otra gran parte del índice bibliográfico, incluye fuentes empleadas para obtener documentación sobre los lenguajes y estándares típicos de MDA. En todos los casos, se trata de literatura extraída directamente del sitio web del conjunto de investigadores Object Management Group (OMG), exceptuando alguno para el que se ha consultado su propia página informativa acerca de su desarrollo. Para cada caso, se ha encontrado una especificación definida. Éste es el trabajo de un equipo de investigación, por lo que se trata de información totalmente fidedigna. Como punto de partida para conocer el conjunto de ellos disponibles, se ha usado el índice disponible en la página web correspondiente. Ésta provee enlaces a las especificaciones mencionadas anteriormente.

Realizado también por integrantes de este grupo de investigación, se ha empleado el documento *MDA Guide Version 1.0.1*, de J. Miller y J. Mukerji, donde se detallan sus propiedades. Adicionalmente, se han encontrado menciones a estos documentos en las tesis doctorales *TALISMAN: desarrollo Ágil de Software con Arquitecturas Dirigidas por Modelo*, de B. C. Pelayo García-Bustelo, *Modelado específico de dominio para la construcción de learning objects independientes de la plataforma*, de C. E. Montenegro Marín, y *MDCI: Model-Driven Continous Integration*, de Vicente García Díaz. El artículo antes señalado, *MDE between Promises and Challenges*, también contiene una selección de los lenguajes característicos.

Finalmente, cabe señalar un conjunto de artículos que explican la técnica MDA: *An introduction to Model Driven Architecture* (A. Brown) y *Model Driven Architecture* (R. Soley). Muestran una información similar a la especificada en los documentos señalados anteriormente. También ha sido de utilidad el libro titulado *MDA explained: The Model Driven Architecture: Practice and Promise* (A. Kleppe, J. Warmer y W. Bast). En este escrito se revisan las mismas características, ventajas y especificaciones que se han destacado en otros elementos consultados.

De manera adicional, cabe destacar el empleo de un libro que trata una temática, el lenguaje de modelado UML, no directamente relacionada con el hilo conductor de este trabajo. Se trata de *UML distilled*, escrito por M. Fowler. Sin embargo, ha sido útil para conocer una especificación de modelización ampliamente extendida. Esta obra

también ha sido empleada en las tesis doctorales mencionadas, así como en la mayoría de artículos revisados.

A.2 GESTIÓN DE PROCESOS

Para la extracción de información acerca de la gestión de procesos empresariales y flujos de trabajo, se han seguido una serie de artículos que tratan el tema desde diversas perspectivas. Este conjunto está formado por: *Workflow Management: models, methods and systems* (W. M. P. van der Aalst, K. M. van Hee), *Business Process Management: A Survey* (W. M. P. van der Aalst, A. H. M. ter Hofstede, M. Weske), *Workflow Patterns* (W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski y otros) y *Businees process management (BPM) standars: a survey* (R. K. L. Ko, Stephen S. G. Lee y E. W. Lee). El primero de ellos ha sido publicado por *The MIT Press*, es decir, la editorial del Instituto Tecnológico de Massachussetts, centro de referencia mundial de la investigación tecnológica. Como se aprecia, algunos autores han participado en la redacción de varios de estos artículos. Esta anotación no justifica la calidad da las ideas reflejadas, pero sí es un indicio de que se trata de entendidos en la materia. Para esto es necesario revisar documentación de autores diversos.

Asimismo, se ha consultado un libro que constituye un compendio de los conocimientos aportados por los artículos leídos. Éste es *Handbook on business process management 1*, de J. Vom Brocke y M. Rosemann.

A.3 MODELIZACIÓN DE PROCESOS DE NEGOCIO

Para un primer acercamiento al modelado de procesos de negocio, se ha empleado el libro titulado *Essential Business Process Modeling* (M. Havey). Su principal interés estriba en la descripción de las principales técnicas disponibles.

Como en el caso de MDA, para la modelización de procesos de negocio, la mayoría de los lenguajes y especificaciones han partido de OMG. Es por ello que su sitio web es una fuente muy fiable para ampliar y refutar los datos obtenidos tras la lectura del primer documento.

También se hace referencia a lenguajes no desarrollados por OMG. Para ampliar la información disponible sobre ellos, se ha consultado en los sitios web de sus impulsadores, como es el caso de OASIS (*Organization for the Advancement of Structured Information Standards*).

A.4 PATRONES Y FRAMEWORKS

Para documentar la clasificación de patrones y conocer aquéllos que mejor se podían adaptar a la temática de este trabajo, se ha empleado el documento *Pattern-oriented software architecture. A system of patterns. Volume 1*, escrito por F. Buschmann, R. Meunier, H. Rohnert y otros. El investigador de la Universidad de Oviedo, J. M. Cueva Lovelle, lo señala en su página web como una referencia importante.

A continuación se ha profundizado en los patrones de diseño. Para ello ha sido fundamental la lectura del libro *Design patterns. Elements of resusable object-oriented software* (E. Gamma, R. Helm, R. Johnson y otros). Este documento también se ha aprovechado para contrastar la información relacionada con los patrones de diseño provista por el primer documento. También ha servido como base para otros artículos, que lo mencionan entre las referencias bibliográficas y reafirman sus ideas, como *Messaging Patterns in Service-Oriented Architecture* (S. Chatterjee) o *Patterns of Aspect-Oriented design* (J. Noble, A. Schmidmeier, D. J. Pearce y otros).

Frecuentemente, se encuentran juntos estos dos documentos, dentro de las referencias bibliográficas de artículos referidos a patrones en el mundo de la Ingeniería del Software. Por ejemplo, sus ideas básicas se han visto reflejadas en los documentos *Patterns and Software: Essential Concepts and Terminology*, de B. Appleton, y *Software Patterns*, de D. C. Schmidt, R. E. Johnson y M. Fayad. Por su parte, la obra de M. Fowler titulada *Patterns of Enterprise Application Architecture*, concede una visión práctica de los patrones.

De manera separada, se ha estudiado el patrón DAO. Se trata de uno especialmente definido para el núcleo de J2EE. Debido a ello, la especificación que se encuentra en el sitio web de dicho entorno es especialmente fiable. Su éxito se percibe en la existencia de trabajos que amplían el tema, como *Advanced DAO programming*, de S. Sullivan.

A su vez, se han descubierto la existencia de patrones orientados a la gestión de procesos. Se encuentran disponibles en el sitio web titulado *Workflow Patterns*. En esta iniciativa están participando autores como W. M. P. van der Aalst y A. H. M. ter Hofstede. Estos nombres han aparecido en diversos artículos revisados en la recopilación de información sobre la gestión de procesos. Incluso, también han publicado sus resultados en artículos, como estos tres mencionados a continuación: *Workflow Patterns*, *Advanced Workflow Patterns* y *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages*. Además, sus aportaciones son empleadas para el desarrollo de otros artículos. Entre éstos se encuentran *Workflow patterns in process modeling* (A. Fortis) y *Workflow Patterns for Business Process Modeling* (L. H. Thom, C. Iochpe1 y M. Reichert).

Para documentar los frameworks de interés, se ha empleado la información oficial disponible en sus sitios web. Incluso, se han empleado libros que presentaban el mismo conocimiento, desde una perspectiva práctica.

A.5 SOLUCIONES EXISTENTES

En principio, se ha analizado alguna investigación orientada a la búsqueda de una solución similar a la formulada en este trabajo. Se han encontrado dos especialmente interesantes: *GPFlow: An Intuitive Environment for Web Based Scientific Workflow* (A. Rygg, P. Roe, O. Wong) y *Model-driven Construction of Workflow-based web applications with domain-specific languages* (P. Freudenstein, M. Nussbaumer y M. Gaedke). Sobre GPFlow se ha hallado numerosa información adicional que corrobora la valía de este desarrollo: *Comparative Studies Made Simple in GPFlow*, de L. Buckingham, J. M. Hogan, P. Roe y otros, y *GPFlow: A Pilot Workflow System for Interactive Bioinformatics*, de J. Hogan.

Desde un punto de vista práctico, se han encontrado diversos productos que proveen soluciones para el problema planteado. Sus sitios web son una buena fuente para obtener información técnica sobre ellos. Sin embargo, se debe tener presente su objetivo comercial, por lo que en determinados casos pueden estar sobrevaloradas sus bondades. Por ello, la verdadera comprobación de su valía se ha realizado mediante la experimentación práctica.

Anexo B. Artículo presentado

El tema de este TFM ha sido objeto de la presentación de un artículo. En este capítulo se incluye su contenido.

Modelización de formularios web colaborativos bajo un enfoque basado en procesos

Raúl Castro Cerdeira

Universidad Internacional de La Rioja

Resumen. La Web presenta actualmente multitud de formularios. El objetivo de muchos de ellos es el mantenimiento colaborativo de la información. En ellos se requiere la participación ordenada de distintos usuarios. Cada intervención debe tener lugar en un punto concreto del ciclo de vida de los registros. El responsable llevará a cabo la acción, consistente en completar los campos del formulario que tenga disponibles. Este enfoque aplicable a este tipo de formularios permite modelar flujo de trabajo, participantes y campos disponibles en cada momento. Con ello, se pretende automatizar la generación del código. Dicho código implementa la funcionalidad requerida para la aplicación final. Ésta incluye la composición del formulario y los sistemas de almacenamiento y notificación de requerimiento de participación.

Palabras clave: modelo, proceso, flujo de trabajo, formulario, colaborativo, MDA, CIM, PIM, PSM, BPMN, XPDL, patrón, MVC, DAO.

1 Introducción

Actualmente, Internet dispone de un enorme número de aplicaciones fundamentadas en el mantenimiento colaborativo de la información. Su extensión abarca desde herramientas Web 2.0 a intranets propias de ambientes corporativos. En ellas, la gestión de la información requiere la participación de distintas personas. Su actuación sobre los datos puede ocurrir de manera secuencial o en paralelo. Esta organización deriva en la aparición de flujos de trabajo sobre los formularios.

Desde esta perspectiva, un sistema de este tipo se puede representar a través de un grafo dirigido. Cada nodo representa un estado. Para alcanzarlo, tiene que llevarse a cabo una acción consistente en completar unos campos determinados del formulario. Su realización es responsabilidad de una persona designada previamente. Para que una acción pueda iniciarse, todas las fases anteriores en la secuencia tuvieron que ser acometidas. En conjunto, se dispone de una representación o modelo de un proceso.

El objetivo primordial de este trabajo es proveer la generación automática de un formulario y su entorno (portal y sistema de notificaciones por correo electrónico). Este conjunto debe presentarse en un paquete integrado portable. Para ello se parte de su representación como proceso. Debe lograrse de manera que no se requieran

conocimientos especiales, más allá de los necesarios para la especificación de la información básica requerida.

Este tema ya ha sido objeto de investigación. Existen soluciones a su problemática. En ocasiones son excesivamente complejas debido a que pretenden abarcar un ámbito genérico. Por ello, el uso de esas herramientas añade una nueva dificultad. En otros casos, existen dificultades para disponer del resultado. Esto se debe a que, o bien no se encuentra accesible, o bien está formado por ficheros de configuración que requieren de un componente adicional para su ejecución.

2 Modelos de software

Desde los inicios de la computación, la Ingeniería del Software [1] ha dedicado un gran esfuerzo a crear abstracciones. Éstas representan a los distintos componentes informáticos. Su función es ocultar sus complejidades tras una capa. Pasa a ser la responsable de la realización de las tareas tediosas y proclives a errores. Estas actividades son típicas en la interacción con los sistemas encapsulados. Ejemplos de esta tendencia se encuentran en los lenguajes de programación de alto nivel y en los sistemas operativos. Los primeros están concebidos para evitar el manejo directo de código máquina. En los sistemas operativos se ha delegado la comunicación con el hardware.

Esta práctica también se aplicó posteriormente al software, con la aparición de las herramientas CASE. Éstas permiten representar mediante diagramas los diseños de los sistemas. Según esta especificación, se genera código cuya corrección está garantizada ya en el proceso de construcción. Debido a esta característica, se reducen los tiempos de codificación manual, pruebas y depuración. Sin embargo, su uso quedó limitado a la documentación de los proyectos. Ésta tendía a quedar obsoleta a medida que los desarrollos avanzaban.

La razón principal de este desfase era debida a la dificultad para acoplar el código producido con la tecnología que sustentaría su ejecución. Las herramientas de producción de código debían proveer servicios básicos, como tolerancia a fallos o seguridad. Tanto estas aplicaciones como las implementaciones creadas por ellas eran difíciles de probar y evolucionar.

Ante esta problemática, MDE (Model-Driven Engineering) proporciona una serie de tecnologías que combinan lenguajes de modelado específicos de dominio (DSML), junto con motores de transformación y generadores. Los DSMLs permiten establecer la estructura y los requisitos en ámbitos particulares. Estas características se sintetizan en modelos. Los motores de transformación toman éstos para generar nuevos artefactos. Pueden ser código fuente, ficheros de descripción en formato XML o representaciones alternativas del modelo inicial.

La gran aportación de MDE consiste en el acercamiento entre semántica y sintaxis. De esta manera, los modelos iniciales resultan más familiares a los expertos en el dominio. Con ello, también se facilita la imposición de condiciones y requisitos propios del ámbito específico, ya en las primeras fases del diseño.

3 MDA.

Es una implementación práctica de MDE. Se trata de una especificación propuesta por un grupo de expertos en la materia denominado OMG (Object Management Group). Su motivación [2] es el aislamiento entre las operaciones que deben ser soportadas por un sistema y la manera en que éste emplea las capacidades de la tecnología sobre la que se sustenta. Por ello, está enfocado a definir una funcionalidad, en inicio, independiente de una plataforma. Será transformada para su implementación sobre el entorno seleccionado.

Este paradigma se constituye sobre tres ideas fundamentales [3]. La primera es la representación directa. Consiste en la separación entre conceptos del dominio y tecnología, a que anteriormente se ha hecho referencia. El segundo aspecto es la automatización. Ésta aparece en el momento de desarrollar de manera eficiente trabajos mecánicos que no requieren del ingenio humano. Estas tareas comprenden la conversión del modelo en un artefacto adaptado al entorno de ejecución. La característica final se basa en la selección de estándares abiertos. Esta opción reduce la diversidad superflua. También fomenta la aparición de herramientas muy especializadas. Además, facilita que los proveedores desarrollen sus productos para un mercado poco fragmentado.

Según MDA, un modelo de un sistema es la descripción de ese sistema y su entorno para un propósito definido. Normalmente, se trata de una combinación de diagramas y textos (en lenguaje natural o de modelado). El enfoque orientado a modelos provee formas de aplicación de estos elementos para dirigir las fases de comprensión del problema, así como de construcción, despliegue, funcionamiento y modificación.

Uno de los términos clave de MDA es el punto de vista. Hace referencia a una técnica de abstracción para la visualización de un sistema. Emplea un conjunto concreto de conceptos arquitectónicos y reglas de estructuración. Unos aspectos determinados del sistema son destacados sobre los demás. Con ello, se genera una versión simplificada que obvia los detalles que carecen de interés desde la perspectiva adoptada. A continuación se explican los puntos de vista típicos de MDA.

En el punto de vista independiente de la computación, los protagonistas son el entorno del sistema y los requisitos que lo definen. Los pormenores de su estructura o de su forma de procesamiento no están visibles. Incluso, puede suceder que éstos aún no hayan sido definidos. El sistema observado bajo esta perspectiva recibe la denominación de modelo independiente de la computación (CIM). Para su definición, se emplea un lenguaje conocido por los profesionales del ámbito. Por este motivo, establece un vínculo entre estos expertos y los conocedores del diseño y construcción de los productos que satisfarán los requisitos impuestos.

El punto de vista independiente de la plataforma recoge aquellos aspectos del sistema que no varían entre distintas plataformas. La percepción del sistema desde esta situación proporciona el denominado modelo independiente de la plataforma (PIM). Habitualmente, se suele abstraer su ejecución en una máquina virtual tecnológicamente neutra. Ésta contendría una serie de componentes y servicios genéricos, comunes a la mayoría de plataformas.

Por último, el punto de vista específico de la plataforma proporciona una especialización del anterior. Concretamente, le añade los detalles relacionados con el uso de una tecnología dada. La composición del sistema obtenido mediante este punto de vista se denomina modelo específico de la plataforma (PSM). Combina las especificaciones expuestas en el PIM con los detalles de cómo interactuará con una plataforma particular.

MDA propone un estilo de desarrollo de software [4] basado en transformaciones secuenciales entre estos modelos. Este método aporta una serie de ventajas sobre el método tradicional. La primera es una mayor productividad. El desarrollador puede centrar su atención en las conversiones entre modelos. Éstas pueden ser reutilizadas posteriormente. También se mejora la portabilidad. Esta característica se debe a que un PIM puede ser transformado en diversos PSM. En tercer lugar, se fomenta la interoperabilidad entre PSM generados a partir de un mismo PIM. Para ello, MDA introduce el concepto de puentes. Estos componentes se encargan de la traducción entre conceptos particulares de distintas plataformas. Se pueden aplicar sobre las implementaciones debidas a los diversos PSMs. Finalmente, también aporta soluciones para los problemas del mantenimiento y de la documentación. Debido a que existe una relación directa entre el PIM y el código, el primero sirve como documentación de alto nivel. Se asegura que siempre está actualizada. En lo que atañe al mantenimiento, la mejora consiste en la posibilidad de introducir modificaciones en el PIM que causarán alteraciones sobre el código final.

4 Modelado de procesos.

Los procesos [5] son secuencias de actividades de diversos tipos. En todas ellas, existe un elemento denominado caso. Es la unidad que se modifica o se produce. Ésta puede ser tangible, como un registro de información, o abstracto, como una solicitud de un periodo vacacional. Cada caso determina un procedimiento que debe llevarse a cabo. La ordenación de las tareas que constituyen el proceso depende de un conjunto de condiciones.

Puede ocurrir que dos o más tareas deban ser ejecutadas en un orden invariable. En ese caso, se dice que forman una secuencia. Otra disposición es la selección, donde es posible escoger entre diversas actividades. También es frecuente la ejecución en paralelo. En ella, las tareas colocadas según este sistema deben haber sido todas ellas acometidas antes de pasar a la acción ubicada a continuación de ellas. Por último, también pueden aparecer iteraciones en los procesos. Involucran un conjunto de tareas cuya realización debe repetirse cíclicamente. Cualquier sistema de modelado que incluya estos conceptos es válido para la representación de procesos.

En un principio, la implantación de los procesos empresariales se llevaba a cabo aplicando una gestión del workflow [6]. Esta disciplina cubría las fases de diseño e implementación de dichos flujos. Respecto a la medición de su eficiencia, únicamente permitía realizar deducciones sobre la misma en base a técnicas estadísticas. Esta deficiencia se palió mediante la introducción de un nuevo enfoque del proceso, a

través de una gestión global. Esta nueva perspectiva facilita la detección de puntos débiles en el proceso. Posteriormente, se puede actual sobre ellos mediante la reingeniería del flujo. La importancia que anteriormente se le concedía a la ejecución del proceso pasa a ubicarse en la fase de diseño.

En este ambiente de mejora continua es muy importante la modelización de los procesos. El objetivo principal es su definición formal. Ésta tiene que proporcionar la información necesaria para su comprensión. A la vista de esta representación, es posible analizar los puntos donde es necesario aplicar cambios para mejorar su productividad. Además un software especializado podría analizar el modelo. A partir de este estudio, sería capaz de diseñar una disposición más eficiente, proponiendo incluso ejecuciones paralelas.

Un proceso [7] es definido primeramente en lenguaje natural. Un desarrollador de software extraerá de este texto un algoritmo, formado por una serie de pasos (actividades o tareas) con condiciones, bucles y ejecuciones paralelas. Finalmente, todo ello se traduce en un diagrama de flujo. Hipotéticamente, podría existir una máquina que ejecutara instancias definidas por un modelo expresado de una manera semántica y sintácticamente correcta.

El primer sistema de modelado de procesos que se utilizó fueron las Redes de Petri. Éstas permitían representar los procesos mediante una serie de nodos, unidos mediante arcos dirigidos. Estos nodos se clasifican en estaciones y transiciones. Las primeras representan normalmente un estado del proceso o una localización geográfica. Por su parte, las estaciones equivalen a eventos u operaciones realizados en el marco del proceso. Los objetos físicos o elementos de información que circulan por el proceso son modelados mediante las llamadas piezas. La activación de una transición depende de que su estación predecesora contenga alguna pieza.

El problema de las Redes de Petri es que los modelos resultan de un tamaño inmanejable. Además, no permite diferenciar los distintos objetos procesados. Para ello, se agregaron una serie de ampliaciones a la especificación original. La primera de ellas es la extensión de color. Ésta permite diferenciar las piezas mediante datos distintivos. Usando estas características, se pueden establecer condiciones adicionales sobre la activación de las transiciones. La extensión de tiempo añade a las piezas una marca temporal. Con ello, se permite establecer periodos de espera y una selección de los elementos a procesar mediante una cola FIFO (*first in, first out*). En este sistema, el primero que llega es el primero en salir. Por último, la extensión jerárquica posibilita la capacidad de representar subredes con los mismos elementos que las de nivel superior. Ello posibilita la división del proceso en fragmentos manejables.

Aparte de las Redes de Petri, existen una serie de lenguajes de modelado. El primero es BPEL. Está basado en XML. Se apoya en los servicios web para la ejecución de las operaciones. Proporciona una estructura que les permite interactuar entre ellos y mantener su estado. Por su parte, BPMN consiste en un lenguaje gráfico basado en diagramas de flujo. XPDL es un formato XML empleado para trasladar la información de la modelización de un proceso desde las herramientas de definición a las aplicaciones encargadas de su implantación.

5 Patrones de Ingeniería del Software relevantes

Cuando los desarrolladores [8] se enfrentan a un problema particular, tratan de recordar una solución ya aplicada en otra ocasión. Con ello, se pretende recuperar la esencia del camino seguido para, apoyándose en ella, generar una solución válida para el nuevo problema. A partir de los pares problema-solución que se van registrando, es posible observar factores comunes a los problemas y a las soluciones. Estas parejas pueden ser, por tanto, clasificadas en familias que presentan problemas y soluciones semejantes y que exhiben un patrón en ellas.

Debido a la interactividad característica de la aplicación, el patrón MVC (Modelo-Vista-Controlador) resulta adecuado. Separa los sistemas interactivos en tres partes. El modelo contiene la funcionalidad básica y los datos. La vista muestra al usuario la información extraída del modelo. Finalmente, el controlador maneja las peticiones del usuario debidas a sus propias interacciones y las remite al modelo o a la vista.

La segunda característica del sistema es la comunicación con una base de datos. El patrón DAO (Data Access Object) [9], propio de J2EE, es aplicable a cualquier sistema de almacenamiento persistente. Este patrón se fundamenta en dos participantes fundamentales. El DAO oculta la implementación del acceso a datos. El objeto de transferencia se encarga de recuperar los datos del DAO. Posteriormente, se los traslada al solicitante.

También existen patrones para el workflow. Están enfocados a los diversos aspectos que deben ser soportados por los lenguajes de modelado de procesos de negocio: control de flujo, ejecución condicional, asignación de recursos y comunicación entre el proceso central y sus actividades.

6 Resultados.

Partiendo de la representación gráfica de un proceso, se plantea la posibilidad de generar una aplicación totalmente funcional. Para lograr este objetivo, se construyó un prototipo empleando para ello un ordenador personal de prestaciones medias, un servidor web (XAMPP), un editor de texto (Notepad++) y un editor BPMN (Apia Facilis).

El resultado proporcionado por dicho prototipo debe estar formado principalmente por el formulario, ubicado en el marco de un portal de usuarios. También debe incluir un sistema de notificación por correo electrónico. Este conjunto total estará integrado en un paquete independiente. Mediante este procedimiento, se automatizará la implementación del código que constituye el resultado final. La ventaja más importante estriba en la eliminación de tareas tediosas y repetitivas. Además, se garantiza la corrección del código producido.

La gestión de procesos fue tratada mediante un enfoque basado en MDA. La primera fase, el diseño, se asimiló con la construcción de los sucesivos modelos del sistema: CIM, PIM, PSM y aplicación generada. La implantación del proceso sería

análogo a la instalación y ejecución del código fuente producido. A continuación se po-dría obtener información acerca del rendimiento del proceso. Esta fase queda fuera del control de MDA, pero es propia del ámbito de la gestión de procesos. Esta información puede ser empleada para modificar su diseño. MDA facilita la implementación de estos cambios. Esta técnica permite la especificación de las modificaciones en el modelo adecuado, que derivarán en una nueva versión del código final.

Para acometer el desarrollo del formulario web colaborativo y su workflow, es necesario previamente señalar qué información del mismo será aportada por cada modelo MDA.

6.1 Construcción del CIM.

Como sistema gráfico de representación de procesos se ha escogido BPMN. Esta elección se ha realizado basándose en que provee una forma intuitiva de reflejar los procesos. Por su parte, XPDL es un lenguaje basado en XML cuya notación está relacionada unívocamente con BPMN. Esto facilitará su interpretación con el objetivo de recuperar los datos del proceso modelado. Por otra parte, la mayoría de editores BPMN exportan los procesos modelados en formato XPDL. Otros lenguajes de modelado, como BPEL, incluyen soporte para elementos que no interesan en este trabajo, como los servicios web.

El CIM consistió en un documento XML establecido sobre un lenguaje específico de dominio. Se creó de manera que contuviera la información verdaderamente relevante de la definición XPDL.

- El formulario. Dado que es único en la aplicación generada, también lo es en el XPDL. Puede ocurrir que tareas distintas requieran la edición de campos comunes de este formulario. Por este motivo, los datos referentes a los campos se mantuvieron en una sección independiente: nombre, etiqueta, tipo de dato y tipo de control. En el diagrama BPMN la definición del formulario sucedió en una tarea declarada para este fin. Se ubicó entre el inicio del proceso y las tareas de creación de registros. Sin embargo, ésta no se vio reflejada en la aplicación generada. Se trata de una mera convención auxiliar. Posteriormente, los usuarios fueron vinculados con las tareas. Es necesario aclarar que la definición del formulario no pertenece al estándar declarado por XPDL. Se trata de una mera funcionalidad adicional provista por el editor BPMN empleado (Apia Facilis). Por esta causa, si no se emplea esta herramienta, habrá que proveer otro sistema para la especificación del formulario. Otra opción es imitar el modo en que este editor almacena estos datos en el XPDL.
- Los usuarios. Un mismo usuario podría ser requerido para la ejecución de tareas diferentes. Por tanto, se tomó la decisión de mantener sus datos separados de las relaciones con las tareas.
- Las tareas. El paso de un registro a un estado determinado por una tarea se condicionó a que uno de los usuarios autorizados rellene los campos correspondientes del formulario. Por ello, debe contener esa tarea referencias a

usuarios y campos. Además, la creación y edición de registros son operaciones conceptualmente distintas. Por este motivo, se diferenciaron las tareas para dar de alta, sucesoras del evento de inicio, de las que causan una operación de modificación de una información existente.

- Las pasarelas. Este listado se consideró autónomo. Para cada una fue necesario especificar su tipo. Éste valor influenció en la manera de controlar el flujo. La identidad de los elementos sucesores y antecesores se pudo conocer consultando la sección de las transiciones.
- Las transiciones. Para cada una, se especificó su origen y su destino. Para ello, se indicó la referencia que identifica de manera unívoca una tarea o una pasarela.

6.2 Construcción del PIM.

En esta fase, se actuó sobre las especificaciones relacionadas con los valores de los campos del formulario. Las opciones disponibles dependieron del tipo de campo tratado en cada caso.

Cuando el tipo era de entrada directa (input), se pudieron establecer límites sobre los valores permitidos. Si es dato es textual, el límite estuvo relacionado con la longitud máxima aceptada. Para los otros dos tipos de datos, número y fecha, se señalaron los límites superior e inferior del rango de valores admitidos. En este caso, el formato empleado debió estar acorde el tipo de dato de que se tratara.

Para los controles de selección, es decir, el botón de radio y el menú desplegable, se pudo especificar los valores disponibles así como la etiqueta que para cada uno se visualizaría. Para ello, se emplearon sendas sucesiones de valores separados por un carácter concreto. Se verificó que los valores respetaran el formato impuesto por el tipo de dato.

Respecto al campo de área de texto, fue posible imponer una longitud máxima. Siempre se interpretó como un dato de tipo textual, a pesar de que se especificara otro distinto.

Finalmente, las casillas de verificación no aceptaron ningún tipo de información adicional. Independientemente de la elección del tipo de dato, fue tratado como un valor binario.

6.3 Construcción del PSM.

Este modelo amplió al anterior con las propiedades que lo vinculaban a una plataforma concreta. Se incidió en tres aspectos concretos.

- Configuración del portal del usuario. Principalmente, se permitió decidir qué campos del formulario se mostrarían en los listados de registros.
- Configuración del sistema de envío de correos. Por una parte, se estableció la configuración de la cuenta desde la que se remiten. Además, se asignó a cada usuario una dirección en la que recibiría sus notificaciones. Por último, se pudieron

seleccionar los campos de los registros que se mostrarían en el cuerpo de los emails.
- Configuración del estilo de la aplicación. Para ello, se habilitó la posibilidad de emplear una hoja de estilos propia.

6.4 Obtención del código fuente.

A partir de toda la información recopilada, la aplicación fue capaz de generar el formulario y sus sistemas anexos, esto es, portal del usuario y sistema de envío de emails. También fue responsabilidad de la herramienta crear la base de datos necesaria. Con todo ello, el resultado estuvo en disposición de ser ejecutado sin problemas. Además, se le concedió al usuario la opción de recuperar el paquete completo e instalarlo en un equipo ajeno al que ejecuta el generador de código.

7 Análisis del resultado.

Mediante el desarrollo de este trabajo se ha proporcionado una herramienta capaz de generar un formulario con un flujo de trabajo subyacente. Se ha diseñado de manera que la especificación de los requisitos por parte del usuario sea simple. En ningún caso requeriría de conocimientos acerca del desarrollo de la aplicación final. La disposición del código resultante se ha orientado a facilitar su accesibilidad y portabilidad.

La información solicitada al usuario para conformar la aplicación final se ha relacionado meramente con los datos variables entre diversas implementaciones del formulario. Éstos son básicamente el flujo de trabajo, los campos del formulario y la asignación de campos y participantes a las diversas tareas. Sin embargo, la generación del código común (áreas comunes de las pantallas, sistemas de validación y almacenamiento y envíos de emails) se ha creado para que sea autosuficiente. Por ello, no requiere la intervención del usuario. El resultado final es una aplicación que integra ambas partes. En definitiva, se ha aumentado el acoplamiento entre la definición del formulario final y su propia implementación. Se ha mejorado la adaptabilidad del sistema ante cambios requeridos en el flujo o en los usuarios asignados a cada tarea.

8 Conclusiones y trabajo futuro.

A la vista del resultado, se puede afirmar que es posible combinar MDA con la modelización de procesos para la obtención de un formulario web colaborativo.

Para futuras líneas de investigación, se pueden explorar los siguientes caminos:

- Capacitar a la herramienta para que informe sobre el tránsito a través del flujo establecido. De esta manera se pueden detectar cuellos de botella o caminos no empleados. El análisis de estos datos puede implicar el rediseño del proceso. Una

evolución posterior podría conceder a la aplicación generada la capacidad de implementar cambios sobre su propio flujo de trabajo. Esta aportación vincularía aún más los resultados con la gestión de procesos de negocio.

- Uso de los estándares MDA para la especificación formal de lenguajes para los modelos. Análogamente, podría ser interesante formalizar mediante lenguajes estándar las transformaciones realizadas entre los distintos modelos.
- Realizar nuevas implementaciones de las herramientas de obtención del modelo PSM y del código fuente. Esta diversidad propiciaría el estudio del establecimiento de puentes entre pares de PSMs y de códigos fuente producidos. A través de estos puentes, se recopilarían reglas de conversión entre ellos.

9 Referencias.

1. D. C. Schmith; "Model-Driven Ingeneering"; *Computer, febrero 2006*.
2. J. Miller, J. Mukerji; *MDA Guide version 1.0.1*; Object Management Group, 2003.
3. G. Booch, A. Brown, S. Iyengar et al.; "An MDA manifesto"; *The MDA Journal*, mayo 2004.
4. A. Kleppe, J. Warmer, W. Bast; *MDA explained: The Model Driven Architecture: Practice and Promise*; Addison-Wesley, 2004.
5. W. M. P. van der Aalst, K. M. van Hee; *Workflow Management: models, methods and systems*; The MIT Press, 2002.
6. J. Vom Brocke, M. Rosemann; *Handbook on business process management 1*; Springer, 2010.
7. M. Havey; *Essential Business Process Modeling*; O'Really Media, 2005.
8. F. Buschmann, R. Meunier, H. Rohnert et al.; *Pattern-oriented software architecture. A system of patterns. Volume 1*; Wiley, 2001.
9. Data Access Object. Consultado en diciembre de 2011. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
10. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski et al.; "Workflow Patterns"; *Workflow Patterns. Distributed and Parallel databases*; 2003.

Anexo C. Código fuente

Este anexo contiene el código fuente que constituye el prototipo, así como el que produce éste en la generación de un formulario de ejemplo.

C.1 CÓDIGO FUENTE DEL PROTOTIPO

C.1.1 Directorio /control

C.1.1.1 attachUsersAndFields.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();
$workflow = unserialize($_SESSION["workflow"]);

$users = $workflow->getUsersArray();
$fields = $workflow->getFieldsArray();
$tasks = $workflow->getTasksArray();
$errorList = array();
$errorFields = array();

if (isset($_POST["assign"]))
   {
   $items = array();
   if (isset($_POST["items"]))
      $items = $_POST["items"];
   $taskId = $_GET["taskId"];
   $type = $_GET["type"];
   $workflow->updateTask($taskId, $type, $items);
   $_SESSION["workflow"] = serialize($workflow);
   include "../view/usersAndFieldsSummary.php";
   }
else if (isset($_GET["type"]) && isset($_GET["taskId"]))
   {
   $taskId = $_GET["taskId"];
   $type = $_GET["type"];
   $selectedTask = $workflow->getTaskById($taskId);
   $formTarget = "../control/attachUsersAndFields.php?" .
         "taskId=" . $selectedTask->getId() . "&amp;type=". $type;
   $itemName = "";
   $itemDescription = "";
   $items = array();
   $selectedItems = array();
   if ($type == "users")
      {
      $selectedItems = $selectedTask->getUsers();
      $itemName = "usuarios";
      $itemDescription =
            "usuarios cuya participación se requerirá";
      $items = $users;
      }
   else if ($type == "fields")
      {
      $selectedItems = $selectedTask->getFields();
      $itemName = "campos";
      $itemDescription =
            "campos que estarán disponibles";
      $items = $fields;
      }
   include ("../view/attachUsersAndFields.php");
   }
else if (isset($_POST["accept"]))
   {
   $validationResult =
         $workflow->validateUsersAndFieldsOfTasks($errorList);
   if (count($validationResult->getErrors()) != 0)
      {
```

```
      $errorList = $validationResult->getErrors();
      $errorFields = $validationResult->getErrorFieldNames();
      include "../view/usersAndFieldsSummary.php";
      }
   else
      {
      header("Location: cimCreated.php");
      exit;
      }
   }
else
   {
   include "../view/usersAndFieldsSummary.php";
   }

?>
```

C.1.1.2 cimCreated.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

include "../view/cimCreated.php";

?>
```

C.1.1.3 displayProperties.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();
$workflow = unserialize($_SESSION["workflow"]);

$fieldsInPortal = array();
$fieldsInEmails = array();

if (isset($_POST["accept"]))
   {
   $cssFile = $_FILES["cssFile"];
   if (isset($_POST["showInPortal"]))
      $fieldsInPortal = $_POST["showInPortal"];
   if (isset($_POST["showInEmails"]))
      $fieldsInEmails = $_POST["showInEmails"];
   $workflow->setDisplayProperties($cssFile,
        $fieldsInPortal, $fieldsInEmails);
   $_SESSION["workflow"] = serialize($workflow);
   header("Location: psmCreated.php");
   exit;
   }
else
   {
   $fields = $workflow->getFieldsArray();
   $fieldsInPortal = $workflow->getFieldsShowInPortal();
   $fieldsInEmails = $workflow->getFieldsShowInEmail();
   include "../view/displayProperties.php";
   }

?>
```

C.1.1.4 extendFields.php

```
<?php

include "../include/classInclude.php";

session_start();
```

```
$workflow = unserialize($_SESSION["workflow"]);

$workflow->extendFields();

$_SESSION["workflow"] = serialize($workflow);

header("Location: fieldsConstraints.php");
exit;

?>
```

C.1.1.5 fieldsConstraints.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();

$workflow = unserialize($_SESSION["workflow"]);
$fields = $workflow->getFieldsArray();

$errorList = array();
$errorFields = array();

if (isset($_POST["accept"]))
   {
   foreach ($fields as $field)
      {
      switch (get_class($field))
         {
         case "FieldDirectInputString":
            {
            $paramName = "checkMaxLength_" . $field->getId();
            $checkMaxLength = $_POST[$paramName];
            $field->setCheckMaxLength($checkMaxLength);
            break;
            }
         case "FieldDirectInputNonString":
            {
            $paramName = "checkMinValue_" . $field->getId();
            $checkMinValue = $_POST[$paramName];
            $field->setCheckMinValue($checkMinValue);
            $paramName = "checkMaxValue_" . $field->getId();
            $checkMaxValue = $_POST[$paramName];
            $field->setCheckMaxValue($checkMaxValue);
            break;
            }
         case "FieldSelectionInput":
            {
            $paramName = "values_" . $field->getId();
            $values = $_POST[$paramName];
            $field->setValues($values);
            $paramName = "labels_" . $field->getId();
            $labels = $_POST[$paramName];
            $field->setLabels($labels);
            break;
            }
         }
      }
   $validationResult = new ValidationResult();
   $workflow->validateFieldsAdditionalInformation($validationResult);
   $errorFields = $validationResult->getErrorFieldNames();
   $errorList = $validationResult->getErrors();
   if (count($errorList) != 0)
      include ("../view/fieldsConstraints.php");
   else
      {
      $_SESSION["workflow"] = serialize($workflow);
      include ("../view/pimCreated.php");
```

```
      }
   }
else
   include ("../view/fieldsConstraints.php");

?>
```

C.1.1.6 getApplication.php

```
<?

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();

$workflow = unserialize($_SESSION["workflow"]);
$applicationGenerator = new ApplicationGenerator($workflow);

$rootDirectory = $applicationGenerator->create();

include ("../view/applicationCreated.php");

?>
```

C.1.1.7 getCim.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

header("Content-type: text/xml");

session_start();

$workflow = unserialize($_SESSION["workflow"]);

echo "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
echo $workflow->getCim();

?>
```

C.1.1.8 getPim.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

header("Content-type: text/xml");

session_start();

$workflow = unserialize($_SESSION["workflow"]);

echo "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
echo $workflow->getPim();

?>
```

C.1.1.9 getPsm.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

header("Content-type: text/xml");

session_start();
```

```
$workflow = unserialize($_SESSION["workflow"]);

echo "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
echo $workflow->getPsm();

?>
```

C.1.1.10 pimCreated.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

include "../view/pimCreated.php";

?>
```

C.1.1.11 psmCreated.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

include "../view/psmCreated.php";

?>
```

C.1.1.12 selectFile.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

$errorList = array();
$errorFields = array();

if (isset($_POST["send"]))
   {
   $xpdlFile = new XpdlFile($_FILES["xpdlFile"]["name"]);
   $validationResult = $xpdlFile->validate();
   if (count($validationResult->getErrors()) != 0)
      {
      $errorList = $validationResult->getErrors();
      $errorFields = $validationResult->getErrorFieldNames();
      include ("../view/selectFile.php");
      }
   else
      {
      $xpdlFile->save($_FILES["xpdlFile"]["tmp_name"]);
      $xpdlFile->cleanXpdlFile();
      $users = $xpdlFile->getWorkflow()->getUsersArray();
      $fields = $xpdlFile->getWorkflow()->getFieldsArray();
      $tasks = $xpdlFile->getWorkflow()->getTasksArray();
      session_start();
      $_SESSION["workflow"] = serialize($xpdlFile->getWorkflow());
      header("Location: attachUsersAndFields.php");
      exit;
      }
   }
else
   include ("../view/selectFile.php");

?>
```

C.1.1.13 usersEmails.php

```
<?php
```

```
include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();
$workflow = unserialize($_SESSION["workflow"]);

$users = $workflow->getUsersArray();
$errorList = array();
$errorFields = array();
$emailAccountName = $workflow->getEmailAccountName();
$emailAccountPassword = $workflow->getEmailAccountPassword();

if (isset($_POST["accept"]))
   {
   foreach ($users as $user)
      {
      $paramName = "email_" . $user->getId();
      $user->setEmail($_POST[$paramName]);
      }
   $emailAccountName = $_POST["emailAccountName"];
   $emailAccountPassword = $_POST["emailAccountPassword"];
   $workflow->setMailingAccount($emailAccountName, $emailAccountPassword);
   $validationResult = new ValidationResult();
   $workflow->validateMailingAccount($validationResult);
   $workflow->validateUsersEmails($validationResult);
   $errorFields = $validationResult->getErrorFieldNames();
   $errorList = $validationResult->getErrors();
   if (count($errorList) != 0)
      include ("../view/usersEmails.php");
   else
      {
      $_SESSION["workflow"] = serialize($workflow);
      header("Location: displayProperties.php");
      exit;
      }
   }
else
   include "../view/usersEmails.php";

?>
```

C.1.2 Directorio /images

Este directorio contiene el logotipo mostrado en el prototipo, bajo el nombre *logo.png*.

C.1.3 Directorio /inbox

Esta ubicación está destinada al almacenamiento de los ficheros XPDL subidos al servidor.

C.1.4 Directorio /include

C.1.4.1 classInclude.php

```
<?php

require_once ("../model/ApplicationGenerator.php");
require_once ("../model/DB_mysql.php");
require_once ("../model/Error.php");
```

```
require_once ("../model/Field.php");
require_once ("../model/FieldDirectInputNonString.php");
require_once ("../model/FieldDirectInputString.php");
require_once ("../model/FieldSelectionInput.php");
require_once ("../model/Gateway.php");
require_once ("../model/Task.php");
require_once ("../model/Transition.php");
require_once ("../model/User.php");
require_once ("../model/ValidationResult.php");
require_once ("../model/Workflow.php");
require_once ("../model/XpdlFile.php");


?>
```

C.1.4.2 footer.php

```
<p>
   Prototipo realizado por Raúl Castro Cerdeira
</p>
<p>
   Alumno del Máster en Dirección e Ingeniería de Sitios Web
</p>
```

C.1.4.3 header.php

```
<a href="../index.php">
   <img src="../images/logo.png" alt="Inicio" />
</a>
<h1>Generador de formularios basados en workflow</h1>
<h2>Máster en Dirección e Ingeniería de Sitios Web</h2>
```

C.1.5 Directorio /lib

C.1.5.1 error.php

```
<?php

/*************************************************************************/

function getErrorTextualDescription($errorCode, $errorField)
{
$errorDescription = "";
switch($errorCode)
   {
   case 0:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "es obligatorio.";
      break;
      }
   case 1:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "no especifica un archivo de tipo XPDL.";
      break;
      }
   case 2:
      {
      $errorDescription = "El estado <i>$errorField</i> ".
            "no tiene usuarios asignados.";
      break;
      }
   case 3:
      {
      $errorDescription = "El estado <i>$errorField</i> " .
            "no tiene campos asignados.";
      break;
      }
```

```
   case 4:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "es obligatorio.";
      break;
      }
   case 5:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "debe ser un número entero mayor que 0.";
      break;
      }
   case 6:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "debe ser menor que 5000.";
      break;
      }
   case 7:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "debe ser numérico.";
      break;
      }
   case 8:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "tiene un formato incorrecto.";
      break;
      }
   case 9:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "contiene elementos vacíos.";
      break;
      }
   case 10:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "contiene elementos no numéricos.";
      break;
      }
   case 11:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "contiene elementos con un formato incorrecto.";
      break;
      }
   case 12:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "contiene un número de elementos distinto al de los valores.";
      break;
      }
   case 13:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "debe ser mayor que el mínimo.";
      break;
      }
   default:
      {
      $errorDescription = "Error sobre <i>$errorField</i> no identificado.";
      break;
      }
   }
return $errorDescription;
}

/*****************************************************************************/

?>
```

C.1.5.2 helpers.php

```
<?php

include "../lib/error.php";

/*****************************************************************************/

function renderItemsSelectors($items, $selectedItems)
{
foreach($items as $item)
   {
   $itemId = $item->getId();
   $itemName = $item->getName();
   echo "<p><input type=\"checkbox\" name=\"items[]\" value=\"$itemId\" ";
   if (in_array($itemId, $selectedItems))
      echo "checked=\"checked\"";
   echo "/><span class=\"label\">$itemName</span></p>";
   }
}

/*****************************************************************************/

function renderTasksSummary($tasks, $errorFields)
{
$link = "../control/attachUsersAndFields.php?";
foreach ($tasks as $task)
   {
   echo "<p>";
   getFormFieldLabel($errorFields, $task->getName(), $task->getName() . ":");
   echo "<a href=\"$link" . "taskId=" . $task->getId() . "&amp;type=users\">" .
         $task->getNumberAssignedUsers() . " usuarios</a>" . " / ";
   echo "<a href=\"$link" . "taskId=" . $task->getId() . "&amp;type=fields\">" .
         $task->getNumberAssignedFields() . " campos</a>" . " ";
   echo "</p>";
   }
}

/*****************************************************************************/

function renderMenu($phase, $subphase)
{
echo "<ul>";
echo "<li>";
renderMenuOption($phase, $subphase, 1, 0, "Creación del modelo CIM");
echo "<ul>";
echo "<li>";
renderMenuOption($phase, $subphase, 1, 1, "Selección del fichero XPDL");
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 1, 2, "Asignación de usuarios y campos");
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 1, 3, "Modelo CIM creado");
echo "</li>";
echo "</ul>";
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 2, 0, "Creación del modelo PIM");
echo "<ul>";
echo "<li>";
renderMenuOption($phase, $subphase, 2, 1, "Restricciones sobre los campos");
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 2, 2, "Modelo PIM creado");
echo "</li>";
echo "</ul>";
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 3, 0, "Creación del modelo PSM");
echo "<ul>";
echo "<li>";
```

```
renderMenuOption($phase, $subphase, 3, 1, "Configuración del sistema de emails");
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 3, 2, "Configuración de la presentación");
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 3, 3, "Modelo PSM creado");
echo "</li>";
echo "</ul>";
echo "</li>";
echo "<li>";
renderMenuOption($phase, $subphase, 4, 0, "Aplicación creada");
echo "</li>";
echo "</ul>";
}

/*****************************************************************************/

function renderMenuOption($currentPhase, $currentSubphase,
      $optionPhase, $optionSubphase, $optionLabel)
{
$spanStart = "<span class=\"wrap\">";
$spanEnd = "</span>";
$activeOption = false;
if ($currentPhase == $optionPhase)
   {
   if ($optionSubphase == 0)
      $activeOption = true;
   else if ($optionSubphase == $currentSubphase)
      $activeOption = true;
   }
if ($activeOption == true)
   echo $spanStart;
else if ($optionSubphase != 0)
   {
   if (($optionPhase < $currentPhase) ||
         ($optionPhase == $currentPhase && $optionSubphase < $currentSubphase))
      echo getLink($optionPhase, $optionSubphase);
   }

echo $optionLabel;
if ($activeOption == true)
   echo $spanEnd;
else if ($optionSubphase != 0)
   {
   if (($optionPhase < $currentPhase) ||
         ($optionPhase == $currentPhase && $optionSubphase < $currentSubphase))
      echo "</a>";
   }
}

/*****************************************************************************/

function getLink($phase, $subphase)
{
$link = "<a href=\"../control/";
switch ($phase)
   {
   case 1:
      {
      switch ($subphase)
         {
         case 1:
            {
            $link .= "selectFile.php";
            break;
            }
         case 2:
            {
            $link .= "attachUsersAndFields.php";
            break;
            }
```

```
         case 3:
            {
            $link .= "cimCreated.php";
            break;
            }
         }
      break;
      }
   case 2:
      {
      switch ($subphase)
         {
         case 1:
            {
            $link .= "fieldsConstraints.php";
            break;
            }
         case 2:
            {
            $link .= "pimCreated.php";
            break;
            }
         }
      break;
      }
   case 3:
      {
      switch ($subphase)
         {
         case 1:
            {
            $link .= "usersEmails.php";
            break;
            }
         case 2:
            {
            $link .= "displayProperties.php";
            break;
            }
         case 3:
            {
            $link .= "psmCreated.php";
            break;
            }
         }
      break;
      }
   }
$link .= "\">";
return $link;
}

/*****************************************************************************/

function renderErrorList($errorList)
{
echo "<div id=\"errors\">\n";
echo "<p>Se han detectado los siguientes errores:</p>";
echo "<ul>\n";
foreach ($errorList as $error)
   echo "<li>" . getErrorDescription($error) . "</li>";
echo "</ul>\n";
echo "</div>";
}

/*****************************************************************************/

function getErrorDescription($error)
{
$errorCode = $error->getErrorCode();
$errorField = $error->getErrorFieldLabel();
$errorDescription = getErrorTextualDescription($errorCode, $errorField);
```

```
return $errorDescription;
}

/*************************************************************************/

function getFormFieldLabel($errorFields, $fieldName, $fieldLabel)
{
echo "<span class=\"";
if (in_array($fieldName, $errorFields))
   echo "errorLabel";
else
   echo "label";
echo "\">$fieldLabel</span>";
}

/*************************************************************************/

function renderFieldsControls($fields, $errorFields)
{
foreach ($fields as $field)
   {
   $fieldClass = get_class($field);
   if ($fieldClass == "FieldDirectInputString")
      renderDirectInputStringControl($field, $errorFields);
   else if ($fieldClass == "FieldDirectInputNonString")
      renderDirectInputNonStringControl($field, $errorFields);
   else if ($fieldClass == "FieldSelectionInput")
      renderSelectionInputControl($field, $errorFields);
   }
}

/*************************************************************************/

function renderDirectInputStringControl($field, $errorFields)
{
$label = "Longitud máxima de " . $field->getName() . ":";
$paramName = "checkMaxLength_" . $field->getId();
$paramValue = $field->getCheckMaxLength();
echo "<p>";
getFormFieldLabel($errorFields, $paramName, $label);
echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
      "/></p>";
}

/*************************************************************************/

function renderDirectInputNonStringControl($field, $errorFields)
{
$type = "";
if ($field->getDataType() == "Date")
   $type = "fecha formato dd/mm/aaaa";
else if($field->getDataType() == "Number")
   $type = "número";
$label = "Valor ($type) mínimo de " . $field->getName() . ":";
$paramName = "checkMinValue_" . $field->getId();
$paramValue = $field->getCheckMinValue();
echo "<p>";
getFormFieldLabel($errorFields, $paramName, $label);
echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
      "/></p>";
$label = "Valor ($type) máximo de " . $field->getName() . ":";
$paramName = "checkMaxValue_" . $field->getId();
$paramValue = $field->getCheckMaxValue();
echo "<p>";
getFormFieldLabel($errorFields, $paramName, $label);
echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
      "/></p>";
}

/*************************************************************************/

function renderSelectionInputControl($field, $errorFields)
```

```
{
$type = "";
if ($field->getDataType() == "Date")
   $type = "fechas formato dd/mm/aaaa";
else if($field->getDataType() == "Number")
   $type = "números";
else if($field->getDataType() == "String")
   $type = "textos";
$label = "Valores ($type) de " . $field->getName() . " separados por ;:";
$paramName = "values_" . $field->getId();
$paramValue = $field->getValues();
echo "<p>";
getFormFieldLabel($errorFields, $paramName, $label);
echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
      "/></p>";
$label = "Etiquetas de " . $field->getName() . " separadas por ;:";
$paramName = "labels_" . $field->getId();
$paramValue = $field->getLabels();
echo "<p>";
getFormFieldLabel($errorFields, $paramName, $label);
echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
      "/></p>";
}

/*****************************************************************************/

function renderEmailAccountFields($emailAccountName, $emailAccountPassword,
      $errorFields)
{
$paramName = "emailAccountName";
echo "<p>";
getFormFieldLabel($errorFields, $paramName, "Nombre de la cuenta de envío:");
echo "<input type=\"text\" name=\"$paramName\" value=\"$emailAccountName\" " .
      "/><span class=\"label\">@gmail.com</span></p>";
$paramName = "emailAccountPassword";
echo "<p>";
getFormFieldLabel($errorFields, $paramName, "Contraseña de la cuenta de envío:");
echo "<input type=\"password\" name=\"$paramName\" " .
      "value=\"$emailAccountPassword\" /></p>";
}

/*****************************************************************************/

function renderEmailFields($users, $errorFields)
{
foreach ($users as $user)
   {
   $paramName = "email_" . $user->getId();
   $paramValue = $user->getEmail();
   $label = "Email de " . $user->getName() . ":";
   echo "<p>";
   getFormFieldLabel($errorFields, $paramName, $label);
   echo "<input type=\"text\" name=\"$paramName\" value=\"$paramValue\" " .
         "/></p>";
   }
}

/*****************************************************************************/

function renderDisplayFieldsSelector($fields, $fieldsInPortal, $fieldsInEmail)
{
$errorFields = array();
echo "<p>";
getFormFieldLabel($errorFields, "cssFile", "Fichero CSS:");
echo "<input type=\"file\" id =\"cssFile\" name=\"cssFile\" /></p>";
foreach ($fields as $field)
   {
   $fieldId = $field->getId();
   echo "<p>";
   getFormFieldLabel(array(), $field->getName(), $field->getName() . ":");
   echo "<input type=\"checkbox\" name=\"showInPortal[]\" " .
         "value=\"$fieldId\" style=\"margin-left: 20px\" ";
```

```
   if (in_array($fieldId, $fieldsInPortal))
      echo "checked=\"checked\" ";
   echo "/>";
   echo "<span class=\"label\">Mostrar en el portal</span>";
   echo "<input type=\"checkbox\" name=\"showInEmails[]\" value=\"$fieldId\" " .
        "style=\"margin-left: 20px\" ";
   if (in_array($fieldId, $fieldsInEmail))
      echo "checked=\"checked\" ";
   echo "/>";
   echo "<span class=\"label\">Mostrar en los emails</span>";
   echo "</p>";
   }
}

/*****************************************************************************/

?>
```

C.1.5.3 utils.php

```
<?php

/*****************************************************************************/

function is_valid_date($date)
{
if ($date == null)
   return false;
$dateArray = explode("/", $date);
if (count($dateArray) != 3)
   return false;
if (strlen($dateArray[1]) != 2 || strlen($dateArray[0]) != 2 ||
      strlen($dateArray[2]) != 4)
   return false;
return checkdate($dateArray[1], $dateArray[0], $dateArray[2]);
}

/*****************************************************************************/

function compareDates($date1, $date2)
{
if (is_valid_date($date1) && is_valid_date($date2))
   {
   $date1Array = array_reverse(explode("/", $date1));
   $convertedDate1 = implode($date1Array);
   $date2Array = array_reverse(explode("/", $date2));
   $convertedDate2 = implode($date2Array);
   return $convertedDate1 < $convertedDate2;
   }
}

/*****************************************************************************/

function dateToString($date)
{
$dateArray = array_reverse(explode("/", $date));
return implode($dateArray);
}

/*****************************************************************************/

function saveUploadedFile($fileName, $fileTempName)
{
$targetDirectory = "./../inbox/";
$targetFileName = $targetDirectory . $fileName;
if (file_exists($targetFileName))
   $targetFileName = $targetDirectory . time() . $fileName;
move_uploaded_file($fileTempName, $targetFileName);
return $targetFileName;
}

/*****************************************************************************/
```

```
?>
```

C.1.6 Directorio /model

C.1.6.1 ApplicationGenerator.php

```
<?php

include "../include/classInclude.php";

class ApplicationGenerator
{

private $workflow;
private $rootDirectory;
private $dataBaseName;
private $myBD;

/*************************************************************************/

public function ApplicationGenerator($workflow)
{
$this->workflow = $workflow;
}

/*************************************************************************/

public function create()
{
$this->createBase();
$tableFieldsQuery = $this->createModel();
$this->getFormRenderingFunction();
$this->getRecordsTableRenderingFunction();
$this->createTables($tableFieldsQuery);
return $this->rootDirectory;
}

/*************************************************************************/

private function createBase()
{
$applicationName = $this->workflow->getFormName();
if ($applicationName == "")
   $applicationName = "Formulario";
$this->rootDirectory = "../../" . $applicationName;
if (file_exists($this->rootDirectory))
   {
   $applicationName .= time();
   $this->rootDirectory .= "../../" . $applicationName;
   }

$this->copyDirectory("../scaffold", $this->rootDirectory);

// Copy uploaded CSS file content to used css file
$sourceCssFileName = $this->workflow->getCssFile();
if ($sourceCssFileName != "")
   {
   $sourceCssFile = fopen($sourceCssFileName, "r");
   $style = fread($sourceCssFile, filesize($sourceCssFileName));
   fclose($sourceCssFile);
   $targetCssFileName = $this->rootDirectory . "/style.css";
   $targetCssFile = fopen($targetCssFileName, "w+");
   fwrite($targetCssFile, $style);
   fclose($targetCssFile);
   }

$this->createDatabase($applicationName);
}
```

```
/*****************************************************************************/

private function createModel()
{
$userClassConstructor = "\$this->myDB = new DB_mysql(\"" .
      $this->dataBaseName . "\", \"localhost\", \"root\", \"\");\n}\n";
$userClassConstructor .= $this->addSeparator() . "}\n?>";

$file = fopen($this->rootDirectory. "/model/User.php", "a");
fwrite($file, $userClassConstructor);
fclose($file);

$file = fopen($this->rootDirectory . "/model/Form.php", "w");
$formFileContent = "<?php\n\n";
$formFileContent .= "include \"../include/classInclude.php\";\n";
$formFileContent .= "include \"../lib/utils.php\";\n\n";
$formFileContent .= "class Form\n{\n\n";

$tableFieldsQuery = "";
$fieldsDeclaration = "private \$myDB;\n";
$fieldsDeclaration .= "private \$phpmailer;\n";
$fieldsDeclaration .= "private \$id;\n";
$constructor = "public function Form()\n{\n}\n" . $this->addSeparator();
$constructor .= "public function FormFromArray(\$fieldsArray)\n{\n";
$constructor .= "\$this->myDB = new DB_mysql(\"" . $this->dataBaseName .
      "\", \"localhost\", \"root\", \"\");\n";
$constructor .= "\$this->phpmailer = new PHPMailer(true);\n";
$constructor .= "if (isset(\$fieldsArray[\"id\"]))\n";
$constructor .= "\t\$this->id = \$fieldsArray[\"id\"];\n";

foreach ($this->workflow->getFieldsArray() as $field)
   {
   $fieldsDeclaration .= "private $" . $field->getName() . ";\n";
   $tableFieldsQuery .= $field->getName() . " " .
         $this->getSqlType($field) . " NULL, ";
   $constructor .= "if (isset(\$fieldsArray[\"" . $field->getName() . "\"]))\n";
   $constructor .= "\t\$this->" . $field->getName() . " = " .
         "\$fieldsArray[\"" . $field->getName() . "\"];\n";
   }
$constructor .= "}\n";

$fieldsFunctions = "";
$fieldsGetters = "";
$statusFunctions = "";

$statusRecordsFunction = $this->getCalculateRecordsFunction();

$statusRecordsFunction .= $this->addSeparator();

$statusRecordsFunction .= "public function getStatusInformation()\n{\n" .
      "switch(\$this->id)\n\t{\n";

$allStatusFunction = "public static function getAllStatus()\n{\n";
$allStatusFunction .= "\$result = array();\n";

$this->getStatusFunctions($allStatusFunction, $statusRecordsFunction,
      $statusFunctions);

$this->getFieldsFunctions($fieldsGetters, $fieldsFunctions);

$formFileContent .= $fieldsDeclaration;
$formFileContent .= $this->addSeparator();
$formFileContent .= $constructor;
$formFileContent .= $this->addSeparator();
$formFileContent .= $this->emailSendingFunction();
$formFileContent .= $this->addSeparator();
$formFileContent .= $fieldsGetters;
$formFileContent .= $this->getTransitionsControlFunctions();
$formFileContent .= $this->addSeparator();

$formFileContent .= $fieldsFunctions;
```

```
$formFileContent .= $statusFunctions;
$formFileContent .= "}\n\n?>";
fwrite($file, $formFileContent);
fclose($file);

$statusRecordsFunction .= "\t}\n}\n";
$statusRecordsFunction .= $this->addSeparator();
$statusRecordsFunction .= $allStatusFunction . "return \$result;\n}\n";
$statusRecordsFunction .= $this->addSeparator();
$statusRecordsFunction .= "}\n\n?>";
$file = fopen($this->rootDirectory . "/model/Status.php", "a");
fwrite($file, $statusRecordsFunction);
fclose($file);

return $tableFieldsQuery;
}

/*****************************************************************************/

public function emailSendingFunction()
{
$emailSettings = "private function sendEmail(\$statusName, \$addresses)\n{\n";
$emailSettings .= "\$body = \"Se requiere su participación para que \" .\n";
$emailSettings .= "\t\t\"el flujo de un registro \" .\n";
$emailSettings .= "\t\t\"continúe al estado \$statusName.<br />\";\n";

$emailSettings .= "\$this->phpmailer->IsSMTP();\n";
$emailSettings .= "try\n\t{\n";

$emailSettings .= "\t\$this->phpmailer->SMTPAuth = true;\n";
$emailSettings .= "\t\$this->phpmailer->SMTPSecure = \"ssl\";\n";
$emailSettings .= "\t\$this->phpmailer->Host = \"smtp.gmail.com\";\n";
$emailSettings .= "\t\$this->phpmailer->Port = 465;\n";
$emailSettings .= "\t\$this->phpmailer->CharSet = \"UTF-8\";\n";

$emailSettings .= "\tforeach (\$addresses as \$address)\n";
$emailSettings .=
      "\t\t\$this->phpmailer->AddAddress(\$address, \$address);\n";

$emailAccountName = $this->workflow->getEmailAccountName() .
      "@gmail.com";
$emailAccountPassword = $this->workflow->getEmailAccountPassword();

$emailSettings .=
      "\t\$this->phpmailer->Username = \"$emailAccountName\";\n";
$emailSettings .= "\t\$this->phpmailer->Password " .
      "= \"$emailAccountPassword\";\n";
$emailSettings .=
      "\t\$this->phpmailer->SetFrom(\"$emailAccountName\", \"Proc2Form\");\n";
$emailSettings .=
      "\t\$this->phpmailer->Subject = \"Requerimiento de participación\";\n";

foreach ($this->workflow->getFieldsShowInEmail() as $fieldId)
   {
   $field = $this->workflow->getFieldById($fieldId);
   $emailSettings .= "\t\$fieldValue = \$this->get_" .
         $field->getName() . "();\n";
   $emailSettings .= "\t\$body .= \"<br />" . $field->getLabel() .
         ": \$fieldValue\";\n";
   }

$emailSettings .= "\t\$this->phpmailer->MsgHTML(\$body);\n";

$emailSettings .= "\t\$this->phpmailer->Send();\n\t}\n";
$emailSettings .= "catch(Exception \$e)\n\t{\n" .
      "\techo \$e->getMessage();\n\t}\n}";

return $emailSettings;
}

/*****************************************************************************/
```

```
public function getStatusFunctions(&$allStatusFunction,
      &$statusRecordsFunction, &$statusFunctions)
{
foreach ($this->workflow->getTasksArray() as $task)
   {
   $taskId = $task->getId();
   $taskName = $task->getName();
   $taskType = $task->getType();
   $taskUsers  = $this->arrayToString($task->getUsers(), false);
   $taskFields = $this->arrayToString($task->getFields(), false);
   $taskIngoings = $this->workflow->getTransitionsToTarget($taskId);
   $taskIngoing = "null";
   if (count($taskIngoings) > 0)
      $taskIngoing = $taskIngoings[0]->getId();

   $allStatusFunction .= "\$newStatus = new Status($taskId);\n";
   $allStatusFunction .= "array_push(\$result, \$newStatus);\n";

   $statusRecordsFunction .= "\tcase $taskId:\n\t\t{\n";
   $statusRecordsFunction .= "\t\t\$this->name = \"$taskName\";\n";
   $statusRecordsFunction .= "\t\t\$this->type = \"$taskType\";\n";
   $statusRecordsFunction .=
         "\t\t\$this->authorizedUsers = array($taskUsers);\n";
   $statusRecordsFunction .=
         "\t\t\$this->activeFields = array($taskFields);\n";
   $statusRecordsFunction .=
         "\t\t\$this->ingoing = $taskIngoing;\n";
   $statusRecordsFunction .=
         "\t\t\$this->records = \$this->calculateRecords();\n";
   $statusRecordsFunction .= "\t\tbreak;\n\t\t}\n";

   $statusFunctions .= "public function validateTask_" . $task->getName() .
         "(&\$validationResult)\n" . "{\n";

   foreach ($task->getFields() as $taskField)
      {
      $field = $this->workflow->getFieldById($taskField);
      $statusFunctions .= "\$this->validateField_" . $field->getName() .
            "(\$validationResult);\n";
      }

   $statusFunctions .= "}\n";
   $statusFunctions .= $this->addSeparator();

   $statusFunctions .= "public function save_" . $task->getName() .
         "()\n" . "{\n";
   $statusFunctions .= "\$this->myDB->connect();\n";
   $statusFunctions .= "\$query = \"" .
         $this->createQuery($task) . "\";\n";
   $statusFunctions .= "\$this->myDB->query(\$query);\n";
   if ($taskType == "create")
      $statusFunctions .= "\$this->id = \$this->myDB->getId();\n";
   $statusFunctions .= "\$this->myDB->disconnect();\n";
   $statusFunctions .= $this->checkInGoingTransitions($taskId);
   $statusFunctions .= $this->checkOutGoingTransitions($taskId);

   $statusFunctions .= "}\n";
   $statusFunctions .= $this->addSeparator();
   }
}

/*****************************************************************************/

public function getFieldsFunctions(&$fieldsGetters, &$fieldsFunctions)
{
$fieldsGetters .= "public function getId()\n{\n";
$fieldsGetters .= "return \$this->id;\n}\n";
$fieldsGetters .= $this->addSeparator();

foreach ($this->workflow->getFieldsArray() as $field)
   {
   $fieldsGetters .= "public function get_" . $field->getName() . "()\n{\n";
```

```
   if ($field->getDataType() == "Date")
      $fieldsGetters .= "return stringToDate(\$this->" . $field->getName() .
         ");\n}\n";
   else
      $fieldsGetters .= "return \$this->" . $field->getName() . ";\n}\n";
   $fieldsGetters .= $this->addSeparator();

   $fieldsFunctions .= "public function validateField_" .
         $field->getName() . "(&\$validationResult)\n{\n";
   if (get_class($field) == "FieldDirectInputString")
      {
      $fieldsFunctions .=
            $this->createTextInputFieldValidationFunction($field);
      }
   else if (get_class($field) == "FieldDirectInputNonString")
      {
      $fieldsFunctions .=
            $this->createNonTextInputFieldValidationFunction($field);
      }
   else if (get_class($field) == "FieldSelectionInput")
      {
      $fieldsFunctions .=
            $this->createSelectionInputFieldValidationFunction($field);
      }
   else
      $fieldsFunctions .= "return;\n";
   $fieldsFunctions .= "}\n";
   $fieldsFunctions .= $this->addSeparator();
   }
}

/*****************************************************************************/

public function getCalculateRecordsFunction()
{
$statusRecordsFunction = "private function calculateRecords()\n{\n";
$statusRecordsFunction .= "\$result = array();\n";
$statusRecordsFunction .= "if (\$this->ingoing == null)\n\treturn \$result;\n";
$statusRecordsFunction .= "\$myDB = new DB_mysql(\"$this->dataBaseName\",
      \"localhost\", \"root\", \"\");\n";
$statusRecordsFunction .= "\$myDB->connect();\n";
$statusRecordsFunction .=
      "\$query = \"SELECT * FROM Records " .
      "WHERE id IN\n" .
      "(SELECT recordId FROM ActiveTransitions\n" .
      "WHERE transitionId=\$this->ingoing)\";\n";
$statusRecordsFunction .= "\$myDB->query(\$query);\n";
$statusRecordsFunction .= "while (\$myDB->numRecords() > 0 && " .
      "\$row = \$myDB->resultArray())\n\t{\n";
$statusRecordsFunction .= "\t\$newForm = new Form();\n";
$statusRecordsFunction .= "\t\$newForm->formFromArray(\$row);\n";
$statusRecordsFunction .= "\tarray_push(\$result, \$newForm);\n\t}\n";

$statusRecordsFunction .= "\$myDB->disconnect();\nreturn \$result;\n}\n";
return $statusRecordsFunction;
}

/*****************************************************************************/

public function createTables($tableFieldsQuery)
{
$myDB = new DB_mysql();
$myDB->connect();
$myDB->useDB($this->dataBaseName);
$query = "";
$query = "CREATE TABLE Records (" .
  "id int(11) NOT NULL AUTO_INCREMENT, " .
  $tableFieldsQuery . " " .
  "PRIMARY KEY (id)) COLLATE 'utf8_unicode_ci'";
if (!$myDB->query($query))
   echo $myDB->getError();
```

```
$query = "CREATE TABLE ActiveTransitions (" .
   "id int(11) NOT NULL AUTO_INCREMENT, " .
   "recordId int(11) NOT NULL, " .
   "transitionId int(11) NOT NULL, " .
   "PRIMARY KEY (id)) COLLATE 'utf8_unicode_ci'";
if (!$myDB->query($query))
   echo $myDB->getError();

$myDB->disconnect();
}

/*****************************************************************************/

private function createDatabase($dataBaseName)
{
$this->dataBaseName = $dataBaseName;
$myDB = new DB_mysql();
if ($myDB->connect() == 0)
   echo $myDB->getError();
$query = "CREATE DATABASE $dataBaseName COLLATE 'utf8_unicode_ci'";
if (!$myDB->query($query))
   echo $myDB->getError();
$myDB->useDB($dataBaseName);
$this->createAndPopulateUsersTable($dataBaseName, $myDB);
$myDB->disconnect();
}

/*****************************************************************************/

private function createAndPopulateUsersTable($dataBaseName, $myDB)
{
$query = "";
$query = "CREATE TABLE Users (" .
  "id int(11) NOT NULL AUTO_INCREMENT, " .
  "name varchar(100) NOT NULL, " .
  "email varchar(100) NOT NULL, " .
  "idInModel int(11) NOT NULL, ".
  "PRIMARY KEY (id)) COLLATE 'utf8_unicode_ci'";
if (!$myDB->query($query))
   echo $myDB->getError();
foreach ($this->workflow->getUsersArray() as $user)
   {
   $query = "INSERT INTO Users (name, email, idInModel) VALUES ";
   $query .= "(";
   $query .= "'" . $user->getName() . "', ";
   $query .= "'" . $user->getEmail() . "', ";
   $query .= $user->getId(). ")";
   if (!$myDB->query($query))
      echo $myDB->getError();
   }
}

/*****************************************************************************/

private function checkIngoingTransitions($taskId)
{
$function = "";
$ingoing = $this->workflow->getTransitionsToTarget($taskId);
if (count($ingoing) > 0)
   {
   $ingoing = $ingoing[0];
   $transitionId = $ingoing->getId();
   $function .= "\$this->disableTransition($transitionId);\n";
   $element = $ingoing->getSourceId();
   // The ancestor is an exclusive gateway
   if ($this->workflow->getGatewayById($element) != null)
      {
      $gateway = $this->workflow->getGatewayById($element);
      // Exclusive: diable all gateway outgoing transitions
      if ($gateway->getType() == "Exclusive")
         {
         $outgoingTransitions =
```

```
                $this->workflow->getTransitionsFromSource($gateway->getId());
          foreach ($outgoingTransitions as $outgoingTransition)
             {
             $outgoingTransitionId = $outgoingTransition->getId();
             $function .= "\$this->disableTransition($outgoingTransitionId);\n";

             }
          }
       }
    }
return $function;
}

/*****************************************************************************/

private function checkOutgoingTransitions($taskId)
{
$function = "";
$outgoing = $this->workflow->getTransitionsFromSource($taskId);
if (count($outgoing) > 0)
   {
   $outgoing = $outgoing[0];
   $transitionId = $outgoing->getId();
   $function .= "\$this->enableTransition($transitionId);\n";
   $element = $outgoing->getTargetId();
   // The sucessor is another status
   if ($this->workflow->getTaskById($element) != null)
      $function .= $this->createNotification($element);
   // The sucessor is a gateway
   if ($this->workflow->getGatewayById($element) != null)
      {
      $gateway = $this->workflow->getGatewayById($element);
      // Exclusive: enable gateway outgoing transitions.
      if ($gateway->getType() == "Exclusive")
         $function .=
               $this->checkExclusiveGatewayOutTransitions($gateway, $transitionId);
      // Parallel: enable gateway outgoing transitions
      // when all ingoing are enabled
      if ($gateway->getType() == "Parallel")
         $function .=
               $this->checkParallelGatewayOutTransitions($gateway, $transitionId);
      }
   }
return $function;
}

/*****************************************************************************/

private function checkExclusiveGatewayOutTransitions($gateway, $transitionId)
{
$function = "";
$transitions =
      $this->workflow->getTransitionsFromSource($gateway->getId());
// Enable all outgoing transitions
$targetTasks = array();
foreach ($transitions as $transition)
   {
   $outTransitionId = $transition->getId();
   $function .= "\$this->enableTransition($outTransitionId);\n";

   $outTransitionTaskTarget = $transition->getTargetId();
   $function .= $this->createNotification($outTransitionTaskTarget);
   }
// Disable current transition
$function .= "\$this->disableTransition($transitionId);\n";
return $function;
}

/*****************************************************************************/

private function checkParallelGatewayOutTransitions($gateway, $transitionId)
{
```

```
$function = "";
$ingoingTransitions =
      $this->workflow->getTransitionsToTarget($gateway->getId());
// Pass gateway when all ingoing transitions are enabled
$function .= "if (1==1";
foreach ($ingoingTransitions as $ingoingTransition)
   {
   $ingoingTransitionId = $ingoingTransition->getId();
   $function .= " && \$this->isEnabled($ingoingTransitionId)";
   }
$function .= ")\n\t{\n";
// Then enable all outgoing transitions
$outgoingTransitions =
      $this->workflow->getTransitionsFromSource($gateway->getId());
foreach ($outgoingTransitions as $outgoingTransition)
   {
   $outgoingTransitionId = $outgoingTransition->getId();
   $function .=
         "\t\$this->enableTransition($outgoingTransitionId);\n";

   $outTransitionTaskTarget = $outgoingTransition->getTargetId();
   $function .= "\t" . $this->createNotification($outTransitionTaskTarget);
   }
// And disable all ingoing transitions
foreach ($ingoingTransitions as $ingoingTransition)
   {
   $ingoingTransitionId = $ingoingTransition->getId();
   $function .=
         "\t\$this->disableTransition($ingoingTransitionId);\n";
   }
$function .= "\t}\n";
return $function;
}

/*****************************************************************************/

private function createNotification($taskId)
{
$task = $this->workflow->getTaskById($taskId);
$authorizedUsers = $task->getUsers();
$addresses = array();
foreach ($authorizedUsers as $authorizedUser)
   {
   $user = $this->workflow->getUserById($authorizedUser);
   array_push($addresses, $user->getEmail());
   }
$notification = "\$this->sendEmail(\"" . $task->getName() .
      "\", array(" . $this->arrayToString($addresses, true) . "));\n";
return $notification;
}

/*****************************************************************************/

private function getTransitionsControlFunctions()
{
$functions = "private function isEnabled(\$transitionId)\n{\n";
$functions .= "\$this->myDB = new DB_mysql(\"$this->dataBaseName\", " .
      "\"localhost\", \"root\", \"\");\n";
$functions .= "\$this->myDB->connect();\n";
$functions .= "\$query = \"SELECT * FROM ActiveTransitions\n".
      "\t\tWHERE recordId = \$this->id AND transitionId = \$transitionId\";\n";
$functions .= "\$this->myDB->query(\$query);\n";
$functions .= "\$numRows = \$this->myDB->numRecords();\n";
$functions .= "\$this->myDB->disconnect();\nreturn \$numRows > 0;\n}\n";
$functions .=  $this->addSeparator();

$functions .= "private function enableTransition(\$transitionId)\n{\n";
$functions .= "\$this->myDB = new DB_mysql(\"$this->dataBaseName\", " .
      "\"localhost\", \"root\", \"\");\n";
$functions .= "\$this->myDB->connect();\n";
$functions .= "\$query = \"INSERT INTO ActiveTransitions".
      "(recordId, transitionId)\n\t\tVALUES (\$this->id, \$transitionId)\";\n";
```

```
$functions .= "\$this->myDB->query(\$query);\n";
$functions .= "\$this->myDB->disconnect();\n}\n";
$functions .=  $this->addSeparator();

$functions .= "private function disableTransition(\$transitionId)\n{\n";
$functions .= "\$this->myDB = new DB_mysql(\"$this->dataBaseName\", " .
      "\"localhost\", \"root\", \"\");\n";
$functions .= "\$this->myDB->connect();\n";
$functions .= "\$query = \"DELETE FROM ActiveTransitions\n".
      "\t\tWHERE recordId = \$this->id AND transitionId = \$transitionId\";\n";
$functions .= "\$this->myDB->query(\$query);\n";
$functions .= "\$this->myDB->disconnect();\n}\n";

return $functions;
}

/*****************************************************************************/

private function getFormRenderingFunction()
{
$function =
      "function renderFormFields(\$form, \$activeFields, \$errorFields)\n{\n";

foreach ($this->workflow->getFieldsArray() as $field)
   {
   $fieldClass = get_class($field);
   $fieldId = $field->getId();
   $fieldName = $field->getName();
   $fieldLabel = $field->getLabel();
   $fieldType = $field->getFieldType();

   $function .= "\$fieldValue = \$form->get_" . $fieldName . "();\n";
   $function .= "\$fieldIsActive = in_array($fieldId, \$activeFields);\n";

   if ($fieldClass == "FieldDirectInputString" ||
         $fieldClass == "FieldDirectInputNonString")
      {
      if ($fieldType == "Text Area")
         $function .= "renderTextAreaControl";
      else
         $function .= "renderDirectInputControl";
      $function .= "(\"$fieldName\", \"$fieldLabel:\", \$fieldValue, " .
            "\$fieldIsActive, \$errorFields);\n";
      }
   else if ($fieldClass == "FieldSelectionInput")
      {
      $values = $field->getValues();
      $labels = $field->getLabels();
      if ($fieldType == "Radio Button")
         $function .= "renderRadioButtonControl";
      else if ($fieldType == "ListBox")
         $function .= "renderDropDownControl";
      $function .= "(\"$fieldName\", \"$fieldLabel:\", \$fieldValue, " .
            "\$fieldIsActive, \$errorFields, \"$values\", ".
            "\"$labels\");\n";
      }
   else
      {
      $function .= "renderCheckBoxControl";
      $function .= "(\"$fieldName\", \"$fieldLabel:\", \$fieldValue, " .
            "\$fieldIsActive, \$errorFields);\n";
      }
   }
$function .= "}\n" . $this->addSeparator();
$file = fopen($this->rootDirectory . "/lib/helpers.php", "a");
fwrite($file, $function);
fclose($file);
}

/*****************************************************************************/

private function getRecordsTableRenderingFunction()
```

```
{
$function =
      "function renderRecords(\$statusId, \$records, \$userIsAuthorized)\n{\n";

$showFields = $this->workflow->getFieldsShowInPortal();

$fieldNames = array();
$fieldLabels = array();

foreach ($showFields as $showField)
{
$field = $this->workflow->getFieldById($showField);
array_push($fieldNames, $field->getName());
array_push($fieldLabels, $field->getLabel());
}
$function .= "renderRecordsTable(array(" .
      $this->arrayToString($fieldNames, true) .
      "),\n\t\tarray(" . $this->arrayToString($fieldLabels, true) .
      "),\n\t\t\$statusId " .
      ",\n\t\t\$records " .
      ", \n\t\t\$userIsAuthorized);\n";
$function .= "}\n" . $this->addSeparator() . "?>";
$file = fopen($this->rootDirectory . "/lib/helpers.php", "a");
fwrite($file, $function);
fclose($file);
}

/*********************************************************************************/

private function arrayToString($array, $hasStrings)
{
$arrayStr = "";
foreach ($array as $element)
   {
   if ($hasStrings)
      $arrayStr .= "\"";
   $arrayStr .= $element;
   if ($hasStrings)
      $arrayStr .= "\"";
   $arrayStr .= ", ";
   }
return substr($arrayStr, 0, strlen($arrayStr) - 2);
}

/*********************************************************************************/

private function getSqlType($field)
{
$sqlType = "";
switch (get_class($field))
   {
   case "Field":
      {
      $sqlType = "bit(1)";
      break;
      }
   case "FieldDirectInputString":
      {
      $maxValue = 1000;
      if ($field->getCheckMaxLength() != null)
         $maxValue = $field->getCheckMaxLength();
      $sqlType = "nvarchar($maxValue)";
      break;
      }
   case "FieldDirectInputNonString":
      {
      switch ($field->getDataType())
         {
         case "Number":
            {
            $sqlType = "float";
            break;
```

```
            }
         case "Date":
            {
            $sqlType = "date";
            break;
            }
         }
      break;
      }
   case "FieldSelectionInput":
      {
      switch ($field->getDataType())
         {
         case "String":
            {
            $sqlType = "nvarchar(1000)";
            break;
            }
         case "Number":
            {
            $sqlType = "float";
            break;
            }
         case "Date":
            {
            $sqlType = "date";
            break;
            }
         }
      break;
      }
   }
return $sqlType;
}

/*****************************************************************************/

private function addSeparator()
{
return "\n/*************************************************" .
      "*************************/\n\n";
}

/*****************************************************************************/

private function createTextInputFieldValidationFunction($field)
{
$fieldName = $field->getName();
$fieldCheckMaxLength = $field->getCheckMaxLength();

// Generate code to check field exists
$functionBody = "";
$functionBody .= "if (\$this->$fieldName == null)\n\t{\n";
$functionBody .= "\t\$validationResult->addErrorNameAndLabel(0, \"" .
      $field->getName() . "\", \"" . $field->getLabel() .
      "\", \"\");\n\treturn;\n\t}\n";

// Generate code to check field length
if ($fieldCheckMaxLength != null)
   {
   $functionBody .= "if (strlen(\$this->$fieldName) " .
         "> $fieldCheckMaxLength)\n\t{\n";
   $functionBody .= "\t\$validationResult->addErrorNameAndLabel(2, \"" .
         $field->getName() . "\", \"" . $field->getLabel() .
         "\", \"$fieldCheckMaxLength\");\n\treturn;\n\t}\n";
   }
return $functionBody;
}

/*****************************************************************************/

private function createNonTextInputFieldValidationFunction($field)
```

```
{
$fieldName = $field->getName();
$fieldLabel = $field->getLabel();
$checkMinValue = $field->getCheckMinValue();
$checkMaxValue = $field->getCheckMaxValue();
$type = $field->getDataType();

$checkMinValueDisplay = dateToString($checkMinValue);
$checkMaxValueDisplay = dateToString($checkMaxValue);
if ($type == "Date")
   {
   $checkMinValueDisplay = "'" . $checkMinValueDisplay . "'";
   $checkMaxValueDisplay = "'" . $checkMaxValueDisplay . "'";
   }

// Generate code to check field exists
$functionBody = "";
$functionBody .= "if (\$this->$fieldName == null)\n\t{\n";
$functionBody .= "\t\$validationResult->addErrorNameAndLabel(0, \"" .
      $fieldName . "\", \"" . $fieldLabel . "\", \"\");\n\treturn;\n\t}\n";

// Generate code to check the field is a valid date
if ($type == "Date")
   {
   $functionBody .= "if (!is_valid_date(\$this->$fieldName))\n\t{\n";
   $functionBody .= "\t\$validationResult->addErrorNameAndLabel(3, \"" .
         $fieldName . "\", \"" . $fieldLabel . "\", \"\");\n\treturn;\n\t}\n";

   }
// Generate code to check the field is a number
else
   {
   $functionBody .= "if (!is_numeric(\$this->$fieldName))\n\t{\n";
   $functionBody .= "\t\$validationResult->addErrorNameAndLabel(4, \"" .
         $fieldName . "\", \"" . $fieldLabel . "\", \"\");\n\treturn;\n\t}\n";
   }

// Generate code to check lower limit
if ($checkMinValue != null)
   {
   if ($type == "Date")
      $functionBody .= "if (dateToString(\$this->$fieldName) ";
   else
      $functionBody .= "if (\$this->$fieldName ";
   $functionBody .= "< $checkMinValueDisplay)\n\t{\n";
   $functionBody .= "\t\$validationResult->addErrorNameAndLabel(5, \"" .
   $fieldName . "\", \"" . $fieldLabel .
         "\", \"$checkMinValue\");\n\treturn;\n\t}\n";
   }
// Generate code to check upper limit
if ($checkMaxValue != null)
   {
   if ($type == "Date")
      $functionBody .= "if (dateToString(\$this->$fieldName) ";
   else
      $functionBody .= "if (\$this->$fieldName ";
   $functionBody .= "> $checkMaxValueDisplay)\n\t{\n";
   $functionBody .= "\t\$validationResult->addErrorNameAndLabel(6, \"" .
   $fieldName . "\", \"" . $fieldLabel .
         "\", \"$checkMaxValue\");\n\treturn;\n\t}\n";
   }
return $functionBody;
}

/*************************************************************************/

private function createSelectionInputFieldValidationFunction($field)
{
$fieldName = $field->getName();
$fieldLabel = $field->getLabel();
$functionBody = "";
$functionBody .= "if (\$this->$fieldName == null)\n\t{\n";
```

```
$functionBody .= "\t\$validationResult->addErrorNameAndLabel(0, \"" .
      $fieldName . "\", \"" . $fieldLabel . "\", \"\");\n\treturn;\n\t}\n";
return $functionBody;
}

/*****************************************************************************/

private function createQuery($task)
{
$query = "";
if ($task->getType() == "create")
   $query .= "INSERT INTO Records \n" . $this->createInsertQuery($task);
else
   $query .= "UPDATE Records SET \n" . $this->createUpdateQuery($task);
return $query;
}

/*****************************************************************************/

private function createInsertQuery($task)
{
$fieldsList = "";
$valuesList = "";
foreach ($task->getFields() as $taskField)
   {
   $field = $this->workflow->getFieldById($taskField);
   $fieldName = $field->getName();
   $fieldsList .=  "$fieldName,\n";
   $valuesList .= $this->getSqlFormat($field) . ",\n";
   }
$fieldsList = substr($fieldsList, 0, strlen($fieldsList)-2);
$valuesList = substr($valuesList, 0, strlen($valuesList)-2);
return "($fieldsList)\nVALUES\n($valuesList)";
}

/*****************************************************************************/

private function createUpdateQuery($task)
{
$query = "";
foreach ($task->getFields() as $taskField)
   {
   $field = $this->workflow->getFieldById($taskField);
   $fieldName = $field->getName();
   $query .= "$fieldName = " . $this->getSqlFormat($field) . ",\n";
   }
$query = substr($query, 0, strLen($query)-2);
$query .= "\nWHERE id = \$this->id";
return $query;
}

/*****************************************************************************/

private function getSqlFormat($field)
{
$base = "\$this->" . $field->getName();
if ($field->getDataType() == "String")
   return "'$base'";
if ($field->getDataType() == "Date")
   return "'\" . dateToString($base) . \"'";
return $base;
}

/*****************************************************************************/

private function copyDirectory($source, $destination)
{
if (is_dir($source))
   {
   mkdir($destination, 777);
   $directory = dir($source);
   while (($readDirectory = $directory->read()) != false)
```

```
      {
      if ($readDirectory == '.' || $readDirectory == '..' )
         continue;
      $PathDir = $source . '/' . $readDirectory;
      if (is_dir($PathDir))
         {
         $this->copyDirectory($PathDir, $destination . '/' . $readDirectory);
         continue;
         }
      copy($PathDir, $destination . '/' . $readDirectory);
      }

   $directory->close();
   }
else
   copy($source, $destination);
}

/**************************************************************************/

}

?>
```

C.1.6.2 DB_mysql.php

```
<?php

class DB_mysql
{

private $database;
private $server;
private $user;
private $password;
private $connectionId = 0;
private $queryId = 0;
private $errno = 0;
private $error = "";

/**************************************************************************/

public function DB_mysql($database = "", $server = "localhost", $user = "root",
      $password = "")
{
$this->database = $database;
$this->server = $server;
$this->user = $user;
$this->password = $password;
}

/**************************************************************************/

public function connect()
{
$this->connectionId =
      mysql_connect($this->server, $this->user, $this->password);
if (!$this->connectionId)
   {
   $this->error = "Intento de conexión fallido";
   return 0;
   }
}

/**************************************************************************/

public function useDB($database)
{
$this->database = $database;
if (!@mysql_select_db($this->database, $this->connectionId))
   {
   $this->error = "No se ha podido abrir " . $this->database;
```

```
   return 0;
   }
return $this->connectionId;
}

/*************************************************************************/

public function disconnect()
{
if (!mysql_close($this->connectionId))
   {
   $this->error = "Intento de desconexión fallido";
   return 0;
   }
return 1;
}

/*************************************************************************/

public function query($query)
{
mysql_query("SET names UTF8");
if ($query == "")
   {
   $this->error = "No se ha especificado una consulta SQL";
   return 0;
   }
$this->queryId = mysql_query($query, $this->connectionId);
if (!($this->queryId))
   {
   $this->errno = mysql_errno();
   $this->error = mysql_error();
   }
return $this->queryId;
}

/*************************************************************************/

public function numFields()
{
return mysql_num_fields($this->queryId);
}

/*************************************************************************/

public function numRecords()
{
return mysql_num_rows($this->queryId);
}

/*************************************************************************/

public function nombreCampo($numField)
{
return mysql_field_name($this->queryId, $numField);
}

/*************************************************************************/

public function getError()
{
return $this->error;
}

/*************************************************************************/

}

?>
```

C.1.6.3 Error.php

```
<?php

include "../include/classInclude.php";

class Error
{

private $errorCode;
private $errorFieldName;
private $errorFieldLabel;

/*****************************************************************************/

function Error($errorCode, $errorFieldName, $errorFieldLabel)
{
$this->errorCode = $errorCode;
$this->errorFieldName  = $errorFieldName;
$this->errorFieldLabel = $errorFieldLabel;
}

/*****************************************************************************/

function getErrorCode()
{
return $this->errorCode;
}

/*****************************************************************************/

function getErrorFieldName()
{
return $this->errorFieldName;
}

/*****************************************************************************/

function getErrorFieldLabel()
{
return $this->errorFieldLabel;
}

/*****************************************************************************/

}

?>
```

C.1.6.4 Field.php

```
<?php

include "../include/classInclude.php";

class Field
{

protected $id;
protected $name;
protected $label;
protected $dataType;
protected $fieldType;
protected $tooltip;
protected $grid;

/*****************************************************************************/

public function Field($id, $name, $label, $dataType, $fieldType, $tooltip, $grid)
{
$this->id = $id;
$this->name = $name;
```

```
if ($label == "")
   $this->label = $name;
else
   $this->label = $label;
$this->dataType = $dataType;
$this->fieldType = $fieldType;
$this->tooltip = $tooltip;
$this->grid = $grid;
}

/*****************************************************************************/

public function getId()
{
return $this->id;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getLabel()
{
return $this->label;
}

/*****************************************************************************/

public function getFieldType()
{
return $this->fieldType;
}

/*****************************************************************************/

public function getDataType()
{
return $this->dataType;
}

/*****************************************************************************/

public function getTooltip()
{
return $this->tooltip;
}

/*****************************************************************************/

public function getGrid()
{
return $this->grid;
}

/*****************************************************************************/

public function getFieldObject()
{
return new Field($this->id, $this->name, $this->label, $this->dataType,
      $this->fieldType, $this->tooltip, $this->grid);
}

/*****************************************************************************/

public function validateAdditionalInformation(&$validationResult)
{
return;
```

```
}

/*************************************************************************/

public function toXml()
{
$result = "";
$result .= "<field id=\"$this->id\" name=\"$this->name\" " .
      "dataType=\"$this->dataType\" fieldType=\"$this->fieldType\" " .
      "tooltip=\"$this->tooltip\" grid=\"$this->grid\" />";
return $result;
}

/*************************************************************************/

}

?>
```

C.1.6.5 FieldDirectInputNonString.php

```
<?php

include "../include/classInclude.php";
include "../lib/utils.php";

class FieldDirectInputNonString extends Field
{

private $checkMinValue;
private $checkMaxValue;

/*************************************************************************/

public function FieldDirectInputNonString($field)
{
parent::Field($field->getId(), $field->getName(), $field->getLabel(),
      $field->getDataType(), $field->getFieldType(), $field->getTooltip(),
      $field->getGrid());
$this->checkMinValue = null;
$this->checkMaxValue = null;
}

/*************************************************************************/

public function getCheckMinValue()
{
return $this->checkMinValue;
}

/*************************************************************************/

public function getCheckMaxValue()
{
return $this->checkMaxValue;
}

/*************************************************************************/

public function setCheckMinValue($checkMinValue)
{
$this->checkMinValue = $checkMinValue;
}

/*************************************************************************/

public function setCheckMaxValue($checkMaxValue)
{
$this->checkMaxValue = $checkMaxValue;
}

/*************************************************************************/
```

```
public function validateAdditionalInformation(&$validationResult)
{
$checkMinValueFieldName = "checkMinValue_" . $this->getId();
$checkMaxValueFieldName = "checkMaxValue_" . $this->getId();

if ($this->dataType == "Number")
   {
   $checkMinValueFieldLabel = "Valor (número) mínimo de " . $this->getName();
   $checkMaxValueFieldLabel = "Valor (número) máximo de " . $this->getName();
   // checkMinValue is numeric
   if ($this->checkMinValue != null && !is_numeric($this->checkMinValue))
      $validationResult->addErrorNameAndLabel(7,
            $checkMinValueFieldName, $checkMinValueFieldLabel);
   // checkMaxValue is numeric
   if ($this->checkMaxValue != null && !is_numeric($this->checkMaxValue))
      $validationResult->addErrorNameAndLabel(7,
            $checkMaxValueFieldName, $checkMaxValueFieldLabel);
   // checkMaxValue is greater than checkMinValue
   if (is_numeric($this->checkMaxValue) && is_numeric($this->checkMinValue) &&
         $this->checkMaxValue <= $this->checkMinValue)
      $validationResult->addErrorNameAndLabel(13,
            $checkMaxValueFieldName, $checkMaxValueFieldLabel);
   }

if ($this->dataType == "Date")
   {
   $checkMinValueFieldLabel = "Valor (fecha formato dd/mm/aaaa) " .
         "mínimo de " . $this->getName();
   $checkMaxValueFieldLabel = "Valor (fecha formato dd/mm/aaaa) " .
         "máximo de " . $this->getName();
   // checkMinValue is valid date (format dd/mm/yyyy)
   if ($this->checkMinValue != null & !is_valid_date($this->checkMinValue))
      $validationResult->addErrorNameAndLabel(8,
            $checkMinValueFieldName, $checkMinValueFieldLabel);
   // checkMaxValue is valid date (format dd/mm/yyyy)
   if ($this->checkMaxValue != null & !is_valid_date($this->checkMaxValue))
      $validationResult->addErrorNameAndLabel(8,
            $checkMaxValueFieldName, $checkMaxValueFieldLabel);
   // checkMaxValue is greater than checkMinValue
   if (is_valid_date($this->checkMinValue) &&
         is_valid_date($this->checkMaxValue) &&
         compareDates($this->checkMinValue, $this->checkMaxValue) == false)
      $validationResult->addErrorNameAndLabel(13,
            $checkMaxValueFieldName, $checkMaxValueFieldLabel);
   }
}

/*****************************************************************************/

public function toXml()
{
$xml = new SimpleXMLElement(parent::toXml());
$xml->addAttribute("checkMinValue", $this->checkMinValue);
$xml->addAttribute("checkMaxValue", $this->checkMaxValue);
$fieldXml = $xml->asXML();
$fieldXmlArray = explode('?>', $fieldXml);
if (count($fieldXmlArray) == 2)
   return $fieldXmlArray[1];
return $fieldXmlArray[0];
}

/*****************************************************************************/

}

?>
```

C.1.6.6 FieldDirectInputString.php

```
<?php
```

```
include "../include/classInclude.php";

class FieldDirectInputString extends Field
{

private $checkMaxLength;

/*************************************************************************/

public function FieldDirectInputString($field)
{
parent::Field($field->getId(), $field->getName(), $field->getLabel(),
      $field->getDataType(), $field->getFieldType(), $field->getTooltip(),
      $field->getGrid());
$this->checkMaxLength = null;
}

/*************************************************************************/

public function getCheckMaxLength()
{
return $this->checkMaxLength;
}

/*************************************************************************/

public function setCheckMaxLength($checkMaxLength)
{
return $this->checkMaxLength = $checkMaxLength;
}

/*************************************************************************/

public function validateAdditionalInformation(&$validationResult)
{
if ($this->checkMaxLength != null)
   {
   $fieldName = "checkMaxLength_" . $this->getId();
   $fieldLabel = "Longitud máxima de " . $this->getName();
   // checkMaxLength is positive integer
   if (!is_numeric($this->checkMaxLength) ||
         ($this->checkMaxLength <= 0) ||
         strrpos($this->checkMaxLength, ".") ||
         strrpos($this->checkMaxLength, ","))
      $validationResult->addErrorNameAndLabel(5, $fieldName, $fieldLabel);
   // checkMaxLength is not greater than 5000
   else if($this->checkMaxLength > 5000)
      $validationResult->addErrorNameAndLabel(6, $fieldName, $fieldLabel);
   }
}

/*************************************************************************/

public function toXml()
{
$xml = new SimpleXMLElement(parent::toXml());
$xml->addAttribute("checkMaxLength", $this->checkMaxLength);
$fieldXml = $xml->asXML();
$fieldXmlArray = explode('?>', $fieldXml);
if (count($fieldXmlArray) == 2)
   return $fieldXmlArray[1];
return $fieldXmlArray[0];
}

/*************************************************************************/

}

?>
```

C.1.6.7 FieldSelectionInput.php

```
<?php

include "../include/classInclude.php";

class FieldSelectionInput extends Field
{

private $values;
private $labels;

/*****************************************************************************/

public function FieldSelectionInput($field)
{
parent::Field($field->getId(), $field->getName(), $field->getLabel(),
      $field->getDataType(), $field->getFieldType(), $field->getTooltip(),
      $field->getGrid());
$this->values = null;
$this->labels = null;
}

/*****************************************************************************/

public function getValues()
{
return $this->values;
}

/*****************************************************************************/

public function getLabels()
{
return $this->labels;
}

/*****************************************************************************/

public function setValues($values)
{
$this->values = $values;
}

/*****************************************************************************/

public function setLabels($labels)
{
$this->labels = $labels;
}

/*****************************************************************************/

public function validateAdditionalInformation(&$validationResult)
{
$fieldValueName = "textos";
if ($this->dataType == "Date")
   $fieldValueName = "fechas formato dd/mm/aaaa";
else if ($this->dataType == "Number")
   $fieldValueName = "números";

$valuesFieldName = "values_" . $this->getId();
$valuesFieldLabel = "Valores ($fieldValueName) de " .
      $this->getName() . " separados por ;";
$labelsFieldName = "labels_" . $this->getId();
$labelsFieldLabel = "Etiquetas de " . $this->getName() . " separadas por ;";

// It makes validation over value elements, then over
// their labels.
if ($this->values == null)
   $validationResult->addErrorNameAndLabel(0, $valuesFieldName,
         $valuesFieldLabel);
```

```
// Check there are values
else
   {
   $valuesArray = explode(";", $this->values);
   foreach ($valuesArray as $value)
      {
      // Check value is not empty
      if ($value == "")
         {
         $validationResult->addErrorNameAndLabel(9, $valuesFieldName,
               $valuesFieldLabel);
         return;
         }
      // Check value is numeric when it's expected to be
      if ($this->dataType == "Number" && !is_numeric($value))
         {
         $validationResult->addErrorNameAndLabel(10, $valuesFieldName,
               $valuesFieldLabel);
         return;
         }
      // Check value is date when it's expected to be
      if ($this->dataType == "Date" && !is_valid_date($value))
         {
         $validationResult->addErrorNameAndLabel(11, $valuesFieldName,
               $valuesFieldLabel);
         return;
         }
      }
   // If it passes checks over values, then check labels
   // Check there are labels
   if ($this->labels == null)
      $validationResult->addErrorNameAndLabel(0, $labelsFieldName,
            $labelsFieldLabel);
   else
      {
      $labelsArray = explode(";", $this->labels);
      // Check there's a label for each value
      if (count($labelsArray) != count($valuesArray))
         $validationResult->addErrorNameAndLabel(12, $labelsFieldName,
               $labelsFieldLabel);
      }
   }
}

/*****************************************************************************/

public function toXml()
{
$xml = new SimpleXMLElement(parent::toXml());
$xml->addAttribute("values", $this->values);
$xml->addAttribute("labels", $this->labels);
$fieldXml = $xml->asXML();
$fieldXmlArray = explode('?>', $fieldXml);
if (count($fieldXmlArray) == 2)
   return $fieldXmlArray[1];
return $fieldXmlArray[0];
}

/*****************************************************************************/

}

?>
```

C.1.6.8 Gateway.php

```
<?php

include "../include/classInclude.php";

class Gateway
{
```

```
private $id;
private $type;

/*************************************************************************/

public function Gateway($id, $type)
{
$this->id = $id;
$this->type = $type;
}

/*************************************************************************/

public function getId()
{
return $this->id;
}

/*************************************************************************/

public function getType()
{
return $this->type;
}

/*************************************************************************/

public function toXml()
{
$result = "";
$result .= "<gateway id=\"$this->id\" type=\"$this->type\" />";
return $result;
}

/*************************************************************************/

}

?>
```

C.1.6.9 Task.php

```
<?php

include "../include/classInclude.php";

class Task
{

private $id;
private $type;
private $name;
private $users;
private $fields;

/*************************************************************************/

public function Task($id, $type, $name)
{
$this->id = $id;
$this->type = $type;
$this->name = $name;
$this->users = array();
$this->fields = array();
}

/*************************************************************************/

public function getId()
{
return $this->id;
```

```
}

/*************************************************************************/

public function getName()
{
return $this->name;
}

/*************************************************************************/

public function getNumberAssignedUsers()
{
return count($this->users);
}

/*************************************************************************/

public function getNumberAssignedFields()
{
return count($this->fields);
}

/*************************************************************************/

public function getUsers()
{
return $this->users;
}

/*************************************************************************/

public function getFields()
{
return $this->fields;
}

/*************************************************************************/

public function getType()
{
return $this->type;
}

/*************************************************************************/

public function setUsers($users)
{
$this->users = $users;
}

/*************************************************************************/

public function setFields($fields)
{
$this->fields = $fields;
}

/*************************************************************************/

public function setItems($items, $type)
{
if ($type == "users")
   $this->users = $items;
else if ($type == "fields")
   $this->fields = $items;
}

/*************************************************************************/

public function toXml()
{
```

```
$result = "";
$result .= "<task id=\"$this->id\" name=\"$this->name\">";
$result .= "<fields>";
foreach ($this->fields as $field)
   $result .= "<field ref=\"$field\" />";
$result .= "</fields>";
$result .= "<users>";
foreach ($this->users as $user)
   $result .= "<user ref=\"$user\" />";
$result .= "</users>";
$result .= "</task>";
return $result;
}

/*****************************************************************************/

}

?>
```

C.1.6.10 Transition.php

```
<?php

include "../include/classInclude.php";

class Transition
{

private $id;
private $sourceId;
private $targetId;

/*****************************************************************************/

public function Transition ($id, $sourceId, $targetId)
{
$this->id = $id;
$this->sourceId = $sourceId;
$this->targetId = $targetId;
}

/*****************************************************************************/

public function getId()
{
return $this->id;
}

/*****************************************************************************/

public function getSourceId()
{
return $this->sourceId;
}

/*****************************************************************************/

public function getTargetId()
{
return $this->targetId;
}

/*****************************************************************************/

public function toXml()
{
$result = "";
$result .= "<transition id=\"$this->id\" sourceId=\"$this->sourceId\" " .
      "targetId=\"$this->targetId\" />";
return $result;
}
```

```
/*****************************************************************************/

}

?>
```

C.1.6.11 User.php

```
<?php

include "../include/classInclude.php";

class User
{

private $name;
private $id;
private $email;

/*****************************************************************************/

public function User($name, $id)
{
$this->name = $name;
$this->id = $id;
$this->email = null;
}

/*****************************************************************************/

public function getId()
{
return $this->id;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getEmail()
{
return $this->email;
}

/*****************************************************************************/

public function setEmail($email)
{
$this->email = $email;
}

/*****************************************************************************/

public function validateEmail(&$validationResult)
{
$fieldName = "email_" . $this->getId();
$fieldLabel = "Email de " . $this->getName();
if ($this->email == null)
   $validationResult->addErrorNameAndLabel(0, $fieldName, $fieldLabel);
else
   {
   $regExp = "/^([a-zA-Z0-9])+([a-zA-Z0-9\._-])" .
         "*@([a-zA-Z0-9_-])+([a-zA-Z0-9\._-]+)+$/";
   $checkMail = preg_match($regExp, $this->email);
   if ($checkMail == false)
      $validationResult->addErrorNameAndLabel(8, $fieldName, $fieldLabel);
```

```
   }

}

/*************************************************************************/

public function toXml($phase)
{
$result = "";
$result .= "<user id=\"$this->id\" name=\"$this->name\" ";
if ($phase == 2)
   $result .= "email=\"$this->email\" ";
$result .= "/>";
return $result;
}

/*************************************************************************/

}

?>
```

C.1.6.12 ValidationResult.php

```
<?php

include "../include/classInclude.php";

class ValidationResult
{

private $errors;

/*************************************************************************/

function ValidationResult()
{
$this->errors = array();
}

/*************************************************************************/

function addError($errorCode, $errorField)
{
$this->addErrorNameAndLabel($errorCode, $errorField, $errorField);
}

/*************************************************************************/

function addErrorNameAndLabel($errorCode, $errorFieldName, $errorFieldLabel)
{
array_push($this->errors,
      new Error($errorCode, $errorFieldName, $errorFieldLabel));
}

/*************************************************************************/

function getErrorFieldNames()
{
$errorFields = array();
foreach ($this->errors as $error)
   array_push($errorFields, $error->getErrorFieldName());
return $errorFields;
}

/*************************************************************************/

function getErrors()
{
return $this->errors;
}
```

```
/*******************************************************************************/

}

?>
```

C.1.6.13 Workflow.php

```
<?php

include "../include/classInclude.php";

class Workflow
{

private $fileName;
private $transitions;
private $gateways;
private $tasks;
private $formName;
private $formTitle;
private $fields;
private $users;
private $emailAccountName;
private $emailAccountPassword;
private $cssFile;
private $fieldsShowInPortal;
private $fieldsShowInEmail;

/*******************************************************************************/

public function Workflow($fileName)
{
$this->fileName = $fileName;
$this->transitions = array();
$this->gateways = array();
$this->tasks = array();
$this->formName = "";
$this->formTitle = "";
$this->fields = array();
$this->users = array();
$this->cssFile = "";
$this->fieldsShowInPortal = array();
$this->fieldsShowInEmail = array();
}

/*******************************************************************************/

private function getXpdlSource()
{
$xpdl = simplexml_load_file($this->fileName);
$xpdl->registerXPathNamespace("xpdl", "http://www.wfmc.org/2008/XPDL2.1");
return $xpdl;
}

/*******************************************************************************/

private function getXpdlData()
{
$xpdl = $this->getXpdlSource();
return $xpdl->xpath("xpdl:WorkflowProcesses/xpdl:WorkflowProcess");
}

/*******************************************************************************/

public function cleanWorkflowData()
{
// Get file content and decode html entities
// (changes "&amp;" by "&" and so on...).
// This is necesary to make form information readable.

$file = fopen($this->fileName, "r+");
```

```
$fileSize = filesize($this->fileName);
$fileContent = fread($file, $fileSize);

$fileContent = html_entity_decode($fileContent);

ftruncate($file, 0);

rewind($file);

fwrite($file, $fileContent);

fclose($file);

$this->getTransitions();
$this->getTasks();
$this->getUsers();
}

/*****************************************************************************/

private function getTransitions()
{
// Get important information
$workflow = $this->getXpdlData();

// Get transitions
$transitions = $workflow[0]->xpath("xpdl:Transitions/xpdl:Transition");

foreach ($transitions AS $transition)
   {
   $myTransition = new Transition((string)$transition["Id"],
         (string)$transition["From"], (string)$transition["To"]);
   array_push($this->transitions, $myTransition);
   }
}

/*****************************************************************************/

private function getTasks()
{
// Get important information
$workflow = $this->getXpdlData();

// Get tasks: start event, user tasks and gateways
$tasks = $workflow[0]->xpath("xpdl:Activities/xpdl:Activity");

$startEventId = null;

foreach ($tasks AS $task)
   {
   $startEvent = $task->xpath("xpdl:Event/xpdl:StartEvent");
   $userTask = $task->xpath("xpdl:Implementation/xpdl:Task");
   $gateway = $task->xpath("xpdl:Route");

   if ($startEvent != null)
      $startEventId = (string)$task["Id"];
   else if ($userTask != null)
      array_push($this->tasks, $task);
   else if ($gateway != null)
      {
      $gatewayId = $task["Id"];
      $gatewayType = $gateway[0]["GatewayType"];
      $myGateway = new Gateway((string)$gatewayId, (string)$gatewayType);
      array_push($this->gateways, $myGateway);
      }
   }

$transitionsFromFormDefinitionTask =
      $this->getFormDefinitionTaskInfo((string)$startEventId);

// Convert tasks to Task type.
```

```
$this->convertTasks($transitionsFromFormDefinitionTask);
}

/*****************************************************************************/

private function getFormDefinitionTaskInfo($startEventId)
{
// Find id of transition from start event
$transitionFromStartEvent = $this->getTransitionsFromSource($startEventId);

// Find this transition target id
$transitionFromStartEventTarget = $transitionFromStartEvent[0]->getTargetId();
$formDefinitionTask = null;
$found = false;
for ($i = 0; $i < count($this->tasks) && !$found; $i++)
   {
   $task = $this->tasks[$i];
   if ((string)$task["Id"] == $transitionFromStartEventTarget)
      {
      $found = true;
      $formDefinitionTask = $task;
      }
   }

// Get form fields information
$form =
      $formDefinitionTask->xpath("xpdl:Object/xpdl:Documentation/forms/form");
$form = $form[0];

$this->formName = (string)$form["frmName"];
$this->formTitle = (string)$form["frmTitle"];

$fieldsArray = array();

$fields = $form->xpath("attribute");

// Counter for field variable
$fieldId = 0;
foreach($fields as $field)
   {
   $myField = new Field((string)$fieldId, (string)$field["attName"],
         (string)$field["attLabel"],(string)$field["datatype"],
         (string)$field["fieldtype"], (string)$field["attTooltip"],
         (string)$field["grid"]);
   array_push($this->fields, $myField);
   $fieldId++;
   }

$transitionsFromFormDefinitionTask =
      $this->getTransitionsFromSource($formDefinitionTask["Id"]);

$creationTasks = array();

// Get creation tasks
foreach ($transitionsFromFormDefinitionTask as $transition)
   array_push($creationTasks, $transition->getTargetId());

// Get transitions to delete: from start event and from form definition task
$transitionsToDelete = array_merge($transitionFromStartEvent,
      $transitionsFromFormDefinitionTask);

// Delete transitions from start event and from form definition task
$this->deleteTransitions($transitionsToDelete);
// Delete form definition task.
$this->deleteTask($transitionFromStartEventTarget);

return $transitionsFromFormDefinitionTask;
}

/*****************************************************************************/

private function getUsers()
```

```
{
$xpdl = $this->getXpdlSource();
$users = $xpdl->xpath("xpdl:Participants/xpdl:Participant");
$i = 0;
foreach($users as $user)
   {
   $myUser = new User((string)$user["Name"], (string)$i);
   array_push($this->users, $myUser);
   $i++;
   }
}

/*****************************************************************************/

public function getTransitionsFromSource($sourceId)
{
$transitionsFromSource = array();
foreach ($this->transitions as $transition)
   {
   if ($transition->getSourceId() == $sourceId)
      array_push($transitionsFromSource, $transition);
   }
return $transitionsFromSource;
}

/*****************************************************************************/

public function getTransitionsToTarget($targetId)
{
$transitionsToTarget = array();
foreach ($this->transitions as $transition)
   {
   if ($transition->getTargetId() == $targetId)
      array_push($transitionsToTarget, $transition);
   }
return $transitionsToTarget;
}

/*****************************************************************************/

private function deleteTransitions($transitionsToDelete)
{
$transitionsToDeleteIds = array();
$cleanTransitions = array();

foreach ($transitionsToDelete as $transitionToDelete)
   array_push($transitionsToDeleteIds, $transitionToDelete->getId());

foreach ($this->transitions as $transition)
   {
   if (!in_array($transition->getId(), $transitionsToDeleteIds))
      array_push($cleanTransitions, $transition);
   }
$this->transitions = $cleanTransitions;

}

/*****************************************************************************/

private function deleteTask($taskToDeleteId)
{
$cleanTasks = array();
foreach ($this->tasks as $task)
   {
   if ((string)$task["Id"] != $taskToDeleteId)
      array_push($cleanTasks, $task);
   }
$this->tasks = $cleanTasks;
}

/*****************************************************************************/
```

```
private function convertTasks($transitionsFromFormDefinitionTask)
{
$creationTasks = array();
$convertedTasks = array();
foreach($transitionsFromFormDefinitionTask as $transition)
   array_push($creationTasks, $transition->getTargetId());
foreach ($this->tasks as $task)
   {
   $type = "edit";
   $taskId = (string)$task["Id"];
   $taskName = (string)$task["Name"];
   if (in_array($taskId, $creationTasks))
      $type = "create";
   array_push($convertedTasks, new Task($taskId, $type, $taskName));
   }
$this->tasks = $convertedTasks;
}

/*****************************************************************************/

public function getUsersArray()
{
return $this->users;
}

/*****************************************************************************/

public function getFieldsArray()
{
return $this->fields;
}

/*****************************************************************************/

public function getTasksArray()
{
return $this->tasks;
}

/*****************************************************************************/

public function getTaskById($taskId)
{
foreach ($this->tasks as $task)
   {
   if ($taskId == $task->getId())
      return $task;
   }
}

/*****************************************************************************/

public function getGatewayById($gatewayId)
{
foreach ($this->gateways as $gateway)
   {
   if ($gatewayId == $gateway->getId())
      return $gateway;
   }
}

/*****************************************************************************/

public function updateTask($taskId, $type, $items)
{
$task = $this->getTaskById($taskId);
$task->setItems($items, $type);
}

/*****************************************************************************/

public function validateUsersAndFieldsOfTasks()
```

```
{
$errors = new ValidationResult();
foreach ($this->tasks as $task)
   {
   if ($task->getNumberAssignedUsers() == 0)
      $errors->addError(2, $task->getName());
   if ($task->getNumberAssignedFields() == 0)
      $errors->addError(3, $task->getName());
   }
return $errors;
}

/*****************************************************************************/

public function extendFields()
{
// Extend field objects to store constraints
// informations. Checkbox fields do not need
// additional data.
foreach ($this->fields as &$field)
   {
   if (get_class($field) == "Field")
      {
      $fieldType = $field->getFieldType();
      $dataType = $field->getDataType();
      if ($fieldType == "Input")
         {
         if ($dataType == "String")
            $field = new FieldDirectInputString($field);
         else
            $field = new FieldDirectInputNonString($field);
         }
      else if ($fieldType == "Text Area")
         $field = new FieldDirectInputString($field);
      else if ($fieldType != "CheckBox")
         $field = new FieldSelectionInput($field);
      }
   }
}

/*****************************************************************************/

public function validateFieldsAdditionalInformation(&$validationResult)
{
foreach ($this->fields as $field)
   $field->validateAdditionalInformation($validationResult);
}

/*****************************************************************************/

public function validateUsersEmails(&$validationResult)
{
foreach ($this->users as $user)
   $user->validateEmail($validationResult);
}

/*****************************************************************************/

public function validateMailingAccount(&$validationResult)
{
$fieldName = "emailAccountName";
$fieldLabel = "Nombre de la cuenta de envío";
if ($this->emailAccountName == null)
   $validationResult->addErrorNameAndLabel(0, $fieldName, $fieldLabel);
$fieldName = "emailAccountPassword";
$fieldLabel = "Contraseña de la cuenta de envío";
if ($this->emailAccountPassword == null)
   $validationResult->addErrorNameAndLabel(0, $fieldName, $fieldLabel);
}

/*****************************************************************************/
```

```
public function setMailingAccount($emailAccountName, $emailAccountPassword)
{
$this->emailAccountName = $emailAccountName;
$this->emailAccountPassword = $emailAccountPassword;
}

/*****************************************************************************/

public function getEmailAccountName()
{
return $this->emailAccountName;
}

/*****************************************************************************/

public function getEmailAccountPassword()
{
return $this->emailAccountPassword;
}

/*****************************************************************************/

public function setDisplayProperties($cssFile, $fieldsShowInPortal,
      $fieldsShowInEmail)
{
if ($cssFile["name"] != "")
   $this->cssFile = saveUploadedFile($cssFile["name"], $cssFile["tmp_name"]);
$this->fieldsShowInPortal = $fieldsShowInPortal;
$this->fieldsShowInEmail  = $fieldsShowInEmail;
}

/*****************************************************************************/

public function getCssFile()
{
return $this->cssFile;
}

/*****************************************************************************/

public function getFieldsShowInPortal()
{
return $this->fieldsShowInPortal;
}

/*****************************************************************************/

public function getFieldsShowInEmail()
{
return $this->fieldsShowInEmail;
}

/*****************************************************************************/

public function toXml($phase)
{
$result = "";
$result .= "<workflow";
if ($phase == 2)
   $result .= " cssFile=\"$this->cssFile\" " .
         "emailAccountName=\"$this->emailAccountName\" " .
         "emailAccountPassword=\"$this->emailAccountPassword\"";
$result .= ">";
$result .= "<fields>";
foreach ($this->fields as $field)
   {
   if ($phase == 0)
      {
      $fieldObject = $field->getFieldObject();
      $result .= $fieldObject->toXml();
      }
   else
```

```
      $result .= $field->toXml();
   }
$result .=  "</fields>";
$result .= "<users>";
foreach ($this->users as $user)
   $result .= $user->toXml($phase);
$result .=  "</users>";
$result .= "<tasks>";
foreach ($this->tasks as $task)
   $result .= $task->toXml();
$result .=  "</tasks>";
$result .= "<gateways>";
foreach ($this->gateways as $gateway)
   $result .= $gateway->toXml();
$result .= "</gateways>";
$result .= "<transitions>";
foreach ($this->transitions as $transition)
   $result .= $transition->toXml();
$result .= "</transitions>";
if ($phase == 2)
   {
   $result .= "<showInPortal>";
   foreach ($this->fieldsShowInPortal as $field)
      $result .= "<field ref=\"$field\" />";
   $result .= "</showInPortal>";
   $result .= "<showInMail>";
   foreach ($this->fieldsShowInEmail as $field)
      $result .= "<field ref=\"$field\" />";
   $result .= "</showInMail>";
   }
$result .=  "</workflow>";
return $result;
}

/*****************************************************************************/

public function getCim()
{
return $this->toXml(0);
}

/*****************************************************************************/

public function getPim()
{
return $this->toXml(1);
}

/*****************************************************************************/

public function getPsm()
{
return $this->toXml(2);
}

/*****************************************************************************/

public function getFormName()
{
return $this->formName;
}

/*****************************************************************************/

public function getFieldById($fieldId)
{
foreach ($this->fields as $field)
   {
   if ($field->getId() == $fieldId)
      return $field;
   }
}
```

```
/*************************************************************************/

public function getUserById($userId)
{
foreach ($this->users as $user)
   {
   if ($user->getId() == $userId)
      return $user;
   }
}

/*************************************************************************/

}

?>
```

C.1.6.14 XpdlFile.php

```
<?php

include "../include/classInclude.php";

class XpdlFile
{

private $fileName;
private $workflow;

/*************************************************************************/

public function XpdlFile($xpdlFileName)
{
$this->fileName = $xpdlFileName;
$this->workflow = null;
}

/*************************************************************************/

public function validate()
{
$errors = new ValidationResult();
if ($this->fileName == "")
   $errors->addErrorNameAndLabel(0, "xpdlFile", "Fichero");
else
   {
   $xpdlFileNameArray = explode(".", $this->fileName);
   $extension = $xpdlFileNameArray[count($xpdlFileNameArray) - 1];
   if ($extension != "xpdl")
      $errors->addErrorNameAndLabel(1, "xpdlFile", "Fichero");
   }
return $errors;
}

/*************************************************************************/

public function save($sourceXpdlFileName)
{
$targetDirectory = "./../inbox/";
$targetXpdlFileName = $targetDirectory . $this->fileName;
if (file_exists($targetXpdlFileName))
   $targetXpdlFileName = $targetDirectory . time() . $this->fileName;
move_uploaded_file($sourceXpdlFileName, $targetXpdlFileName);
$this->fileName = $targetXpdlFileName;
}

/*************************************************************************/

public function cleanXpdlFile()
{
$this->workflow = new Workflow($this->fileName);
```

```
$this->workflow->cleanWorkflowData();
}

/*********************************************************************************/

public function getWorkflow()
{
return $this->workflow;
}

/*********************************************************************************/

}

?>
```

C.1.6 Directorio /phpmailer

Contiene el componente empleado para el envío de los correos electrónicos de notificación.

C.1.7 Directorio /scaffold

Almacena la estructura básica para la aplicación resultante, con una disposición de directorios similar a la mostrada para el propio prototipo. Su contenido se detallará en la siguiente sección.

C.1.8 Directorio /view

C.1.8.1 applicationCreated.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Aplicación creada - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(4, 0); ?>
      </div>
      <div id="content">
         <h2>Aplicación creada</h2>
         <div id="form">
            <p>
               La aplicación ha sido creada con éxito. Si desea acceder a ella
               pulse <a href="<?= $rootDirectory; ?>"> este enlace</a>.
            </p>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
```

```
</html>
```

C.1.8.2 attachUsersAndFields.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Asignación de usuarios y campos - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(1, 2); ?>
      </div>
      <div id="content">
         <h2>
            Asignación de <?= $itemName; ?>
            al estado <?= $selectedTask->getName() ?>
         </h2>
         <div id="form">
            <p>
               Seleccione los <?= $itemDescription; ?> para
               alcanzar el estado <?= $selectedTask->getName() ?>.
            </p>
            <form
                  action="<?= $formTarget ?>" method="post"
                  enctype="multipart/form-data">
               <?php
                  renderItemsSelectors($items, $selectedItems);
               ?>
               <p class="center">
                  <input type="submit" name="assign" value="Asignar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.3 cimCreated.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Modelo CIM creado - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(1, 3); ?>
      </div>
      <div id="content">
         <h2>Modelo CIM creado</h2>
         <div id="form">
            <p>
```

```
            El modelo CIM ha sido creado con éxito. Si desea ver su contenido
            pulse <a href="../control/getCim.php">este enlace</a>.
         </p>
         <p>
            Para obtener
            el modelo PIM pulse
            <a href="../control/extendFields.php">este enlace</a>.
         </p>
      </div>
   </div>
   <div id="footer" class="wrap">
      <?php include ("../include/footer.php") ?>
   </div>
</body>
</html>
```

C.1.8.4 displayProperties.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Configuración de la presentación - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(3, 2); ?>
      </div>
      <div id="content">
         <h2>Configuración de la presentación
         </h2>
         <div id="form">
            <p>
               Si desea personalizar el estilo, agregue su propia hoja CSS.
               Después, indique qué campos de los registros aparecerán en el
               portal y en los emails.
            </p>
            <form action="../control/displayProperties.php" method="post"
                  enctype="multipart/form-data">
               <?php
                  renderDisplayFieldsSelector($fields, $fieldsInPortal,
                        $fieldsInEmails);
               ?>
               <p class="center">
                  <input type="submit" name="accept" value="Aceptar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.5 fieldsContraints.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Restricciones sobre los campos - Proc2Form
      </title>
```

```
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(2, 1); ?>
      </div>
      <div id="content">
         <h2>Restricciones sobre los campos</h2>
         <div id="form">
            <p>
               Especifique restricciones y valores admitidos para los campos
               especificados.
            </p>
            <?php
               if (count($errorList) != 0)
                  renderErrorList($errorList);
            ?>
            <form
                  action="../control/fieldsConstraints.php" method="post"
                  enctype="multipart/form-data">
               <?php
                  renderFieldsControls($fields, $errorFields);
               ?>
               <p class="center">
                  <input type="submit" name="accept" value="Aceptar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.6 pimCreated.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Modelo PIM creado - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(2, 2); ?>
      </div>
      <div id="content">
         <h2>Modelo PIM creado</h2>
         <div id="form">
            <p>
               El modelo PIM ha sido creado con éxito. Si desea ver su contenido
               pulse <a href="../control/getPim.php">este enlace</a>.
            </p>
            <p>
               Para obtener
               el modelo PSM pulse
               <a href="../control/usersEmails.php">este enlace</a>.
            </p>
         </div>
      </div>
      <div id="footer" class="wrap">
```

```
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.7 psmCreated.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Modelo PSM creado - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(3, 3); ?>
      </div>
      <div id="content">
         <h2>Modelo PSM creado</h2>
         <div id="form">
            <p>
               El modelo PSM ha sido creado con éxito. Si desea ver su contenido
               pulse <a href="../control/getPsm.php">este enlace</a>.
            </p>
            <p>
               Para generar
               la aplicación pulse
               <a href="../control/getApplication.php">este enlace</a>.
            </p>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.8 selectFile.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Selección del fichero XPDL - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(1, 1); ?>
      </div>
      <div id="content">
         <h2>Seleccione el fichero XPDL</h2>
         <div id="form">
            <p>
               Escoja el fichero XPDL que contiene la información básica
               del workflow.
            </p>
            <?php
               if (count($errorList) != 0)
```

```
                renderErrorList($errorList);
            ?>
            <form action="selectFile.php" method="post"
                  enctype="multipart/form-data">
               <p>
                  <? getFormFieldLabel($errorFields, "xpdlFile", "Fichero:"); ?>
                  <input type="file" id ="xpdlFile" name="xpdlFile" />
               </p>
               <p class="center">
                  <input type="submit" name="send" value="Enviar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.9 usersAndFieldsSummary.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Resumen de usuarios y campos asignados - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(1, 2); ?>
      </div>
      <div id="content">
         <h2>Resumen de usuarios y campos asignados a cada estado</h2>
         <div id="form">
            <p>
               Pulse sobre el número de usuarios o campos para modificar la
               asignación.
            </p>
            <?php
               if (count($errorList) != 0)
                  renderErrorList($errorList);
               renderTasksSummary($tasks, $errorFields);
            ?>
            <form action="../control/attachUsersAndFields.php" method="post"
                  enctype="multipart/form-data">
               <p class="center">
                  <input type="submit" name="accept" value="Aceptar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.8.10 usersEmails.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
      <title>
         Configuración del sistema de emails - Proc2Form
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
         <?php renderMenu(3, 1); ?>
      </div>
      <div id="content">
         <h2>Configuración del sistema de emails
         </h2>
         <div id="form">
            <p>
               Especifique una dirección desde la que se enviarán los emails de
               notificación. Además, debe indicar una dirección por cada usuario,
               donde recibirá los avisos.
            </p>
            <?php
               if (count($errorList) != 0)
                  renderErrorList($errorList);
            ?>
            <form action="../control/usersEmails.php" method="post"
                  enctype="multipart/form-data">
               <?php
                  renderEmailAccountFields($emailAccountName, $emailAccountPassword,
                        $errorFields);
                  renderEmailFields($users, $errorFields);
               ?>
               <p class="center">
                  <input type="submit" name="accept" value="Aceptar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.1.9 Directorio raíz

C.1.9.1 index.php

```
<?php

header("Location: control/selectFile.php");
exit;

?>
```

C.1.9.2 style.css

```
*
{
color: #000066;
background-color: #55AAFF;
font-family: "Times New Roman", serif;
font-size: 20px;
}

h1, h2
{
color: white;
font-family: Arial, sans-serif;
```

```
}

h1
{
float: left;
text-align: left;
width: 80%;
margin: 1%;
font-size: 1.75em;
}

h2
{
margin-left: 5px;
font-size: 1.15em;
}

#header
{
width: 100%;
float: left;
border: 1px solid #000099;
padding: 2px;
}

#header h2
{
text-align: right;
margin-right: 5%;
font-size: 1.5em;
color: #000066;
margin-bottom: 0px;
}

img
{
padding: 2px;
border: 1px solid red;
}

#header img
{
float: left;
padding: 2px;
border: 1px solid red;
}

#menu
{
width: 30%;
float: left;
margin-top: 20px;
margin-left: -20px;
}

#menu ul
{
list-style-type: none;
margin-left: -10px;
}

#menu li
{
padding-top: 0px;
margin-top: 10px;
}

#content
{
width: 65%;
float: left;
text-align: center;
```

```
}

#footer
{
text-align: center;
width: 90%;
margin: 5%;
float: left;
}

#footer p
{
margin: 3px 0;
font-size: 0.65em;
}

.wrap
{
border: 1px solid #000099;
padding: 2px;
}

li, p
{
font-size: 0.85em;
}

span, input, select, option
{
font-size: 1em;
}

a
{
font-size: 1em;
text-decoration: none;
}

a:hover
{
text-decoration: underline;
}

input , select, option
{
background-color: #99FFCC;
}

.label
{
font-size: 0.85em;
margin: 5px 10px 5px 15px;
}

#form
{
text-align: left;
margin-left: 20%;
}

.center
{
margin-left: 30px;
}

#errors
{
padding: 5px;
border: 1px solid red;
width: 75%;
}
```

```
#errors p, #errors li, #errors i
{
font-size: 0.65em;
margin: 0;
padding-bottom: 3px;
}

#errors ul
{
list-style-type: none;
margin: 0 0 0 3px;
}

#errors i
{
font-size: 1em;
}

.errorLabel
{
font-size: 0.85em;
margin: 5px 10px 5px 15px;
padding: 2px;
border: 1px solid red;
}

#content table
{
width: 65%;
border: 1px solid black;
}

#content td, #content th
{
border: 1px solid black;
}

#formulario2
{
text-align: left;
margin-left: 10%;
}

input[type="radio"], input[type="checkbox"]
{
background-color: #55AAFF;
}
```

C.2 CONTENIDO DE /SCAFFOLD

En esta sección se mostrará el contenido de este directorio. Los ficheros que contienen se emplearán como base para la posterior generación de la aplicación generada. En algunos casos, dichos archivos se encuentran vacíos o incompletos, a la espera de la actuación del generador.

C.2.1 Directorio /control

C.2.1.1 access.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();

$_SESSION["user"] = null;

$errorList = array();
$errorFields = array();
$user = new User();

if (isset($_POST["access"]))
   {
   $user->setEmail($_POST["email"]);
   $validationResult = $user->validate();
   $errorList = $validationResult->getErrors();
   $errorFields = $validationResult->getErrorFieldNames();
   if (count($errorList) != 0)
      include "../view/access.php";
   else
      {
      $_SESSION["user"] = serialize($user);
      header("Location: getRecords.php");
      exit;
      }
   }
else
   include "../view/access.php";

?>
```

C.2.1.2 getForm.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();

$user = $_SESSION["user"];
if ($user == null)
   {
   header("Location: access.php");
   exit;
   }

$user = unserialize($user);

$errorList = array();
$errorFields = array();
```

```
$activeFields = array();
$form = new Form();
$allStatus = Status::getAllStatus();
$status = null;
$userIsAuthorized = false;

$statusId = null;
$recordId = null;

if (isset($_GET["statusId"]))
   {
   $statusId = $_GET["statusId"];
   if (isset($_GET["recordId"]))
      $recordId = $_GET["recordId"];
   }
else if (isset($_POST["accept"]))
   {
   $statusId = $_POST["statusId"];
   if (isset($_POST["recordId"]))
      $recordId = $_POST["recordId"];
   }

if ($statusId != null)
   {
   $status = new Status($statusId);
   $userIsAuthorized = in_array($user->getIdInModel(),
         $status->getAuthorizedUsers());
   if (!$userIsAuthorized)
      {
      header("Location: getRecords.php");
      exit;
      }
   $activeFields = $status->getActiveFields();
   if ($recordId != null)
      $form = $status->getRecordById($recordId);
   }

if (isset($_POST["accept"]))
   {
   $validationResult = new ValidationResult();
   $form->formFromArray($_POST);
   $validateFunctionName = "validateTask_" . $status->getName();
   $saveFunctionName = "save_" . $status->getName();
   $form->$validateFunctionName($validationResult);
   $errorList = $validationResult->getErrors();
   $errorFields = $validationResult->getErrorFieldNames();
   if (count($errorList) != 0)
      include "../view/getForm.php";
   else
      {
      $form->$saveFunctionName();
      header("Location: getRecords.php");
      exit;
      }
   }
else
   include "../view/getForm.php";


?>
```

C.2.1.3 getRecords.php

```
<?php

include "../lib/helpers.php";
include "../include/classInclude.php";

session_start();

$user = $_SESSION["user"];
if ($user == null)
```

```
   {
   header("Location: access.php");
   exit;
   }

$user = unserialize($user);

$allStatus = Status::getAllStatus();
$status = null;
$records = null;
$statusId = null;
$userIsAuthorized = false;

if (isset($_GET["statusId"]))
   {
   $records = array();
   $statusId = $_GET["statusId"];
   $status = new Status($statusId);
   $records = $status->getRecords();
   $userIsAuthorized = in_array($user->getIdInModel(),
         $status->getAuthorizedUsers());
   include "../view/getRecords.php";
   }
else
   include "../view/getRecords.php";

?>
```

C.2.2 Directorio /images

Este directorio contiene el logotipo mostrado en el prototipo, bajo el nombre *logo.png*.

C.2.3 Directorio /include

C.2.3.1 classInclude.php

```
<?php

require_once "../model/DB_mysql.php";
require_once "../model/Error.php";
require_once "../model/Form.php";
require_once "../model/User.php";
require_once "../model/Status.php";
require_once "../model/ValidationResult.php";
require_once "../phpmailer/class.phpmailer.php";

?>
```

C.2.3.2 footer.php

```
<p>
   Aplicación generada a través de Proc2Form
</p>
```

C.2.3.3 header.php

```
<a href="../index.php">
   <img src="../images/logo.png" alt="Inicio" />
</a>
<h1>Formulario generado</h1>
<h2>Proc2Form</h2>
```

C.2.4 Directorio /lib

C.2.4.1 error.php

```
<?php

/*****************************************************************************/

function getErrorTextualDescription($errorCode, $errorField, $extraInfo)
{
$errorDescription = "";
switch($errorCode)
   {
   case 0:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "es obligatorio";
      break;
      }
   case 1:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "contiene un valor incorrecto";
      break;
      }
   case 2:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "tiene una longitud superior a la permitida";
      break;
      }
   case 3:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "no tiene un formato válido (dd/mm/aaaa)";
      break;
      }
   case 4:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "no es de tipo numérico";
      break;
      }
   case 5:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "no debe ser inferior al límite mínimo establecido";
      break;
      }
   case 6:
      {
      $errorDescription = "El campo <i>$errorField</i> " .
            "no debe ser superior al límite máximo establecido";
      break;
      }
   default:
      {
      $errorDescription = "Error sobre <i>$errorField</i> no identificado";
      break;
      }
   }
if ($extraInfo != "")
   $errorDescription = $errorDescription . " ($extraInfo)";
$errorDescription = $errorDescription . ".";
return $errorDescription;
}

/*****************************************************************************/

?>
```

C.2.4.2 helpers.php

```
<?php

include "../lib/error.php";

/*****************************************************************************/

function renderAccessFormFields($errorFields, $email)
{
$label = "Correo electrónico:";
echo "<p>";
getFormFieldLabel($errorFields, "email", "Correo electrónico:");
echo "<input type=\"text\" name=\"email\" value=\"$email\" " .
      "/></p>";
}

/*****************************************************************************/

function getFormFieldLabel($errorFields, $fieldName, $fieldLabel)
{
echo "<span class=\"";
if (in_array($fieldName, $errorFields))
   echo "errorLabel";
else
   echo "label";
echo "\">$fieldLabel</span>";
}

/*****************************************************************************/

function renderErrorList($errorList)
{
echo "<div id=\"errors\">\n";
echo "<p>Se han detectado los siguientes errores:</p>";
echo "<ul>\n";
foreach ($errorList as $error)
   echo "<li>" . getErrorDescription($error) . "</li>";
echo "</ul>\n";
echo "</div>";
}

/*****************************************************************************/

function getErrorDescription($error)
{
$errorCode = $error->getErrorCode();
$errorField = $error->getErrorFieldLabel();
$extraInfo = $error->getExtraInfo();
$errorDescription = getErrorTextualDescription($errorCode,
      $errorField, $extraInfo);
return $errorDescription;
}

/*****************************************************************************/

function renderDirectInputControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<input type=\"text\" name=\"$fieldName\" value=\"$fieldValue\" ";
if ($fieldIsActive == false)
   echo "disabled = \"disabled\" ";
echo "/></p>";
}

/*****************************************************************************/

function renderRadioButtonControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields, $values, $labels)
{
```

```
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);

$valuesArray = explode(";", $values);
$labelsArray = explode(";", $labels);

for ($i = 0; $i < count($valuesArray); $i++)
   {
   $value = $valuesArray[$i];
   $label = $labelsArray[$i];
   echo "<input type=\"radio\" name=\"$fieldName\" " .
         "value=\"$value\" ";
   if ($fieldIsActive == false)
      echo "disabled=\"disabled\" ";
   if ($fieldValue == $value)
      echo "checked=\"checked\" ";
   echo "/>$label";
   }

echo "</p>";
}

/*************************************************************************/

function renderDropDownControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields, $values, $labels)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);

$valuesArray = explode(";", $values);
$labelsArray = explode(";", $labels);

echo "<select name=\"$fieldName\" ";
if ($fieldIsActive == false)
   echo "disabled=\"disabled\"";
echo ">";
echo "<option value=\"\"></option>";

for ($i = 0; $i < count($valuesArray); $i++)
   {
   $value = $valuesArray[$i];
   $label = $labelsArray[$i];
   echo "<option value=\"$value\" ";
   if ($fieldValue == $value)
      echo "selected=\"selected\" ";
   echo ">$label</option>";
   }

echo "</select>";

echo "</p>";
}

/*************************************************************************/

function renderCheckBoxControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<input type=\"checkbox\" name=\"$fieldName\" value=\"1\" ";
if ($fieldIsActive == false)
   echo "disabled=\"disabled\" ";
if ($fieldValue == 1)
   echo "checked=\"checked\" ";
echo "/></p>";
}

/*************************************************************************/

function renderTextAreaControl($fieldName, $fieldLabel, $fieldValue,
```

```
      $fieldIsActive, $errorFields)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<textarea name=\"$fieldName\" rows=\"3\" cols=\"15\" ";
if ($fieldIsActive == false)
   echo "disabled = \"disabled\" ";
echo ">$fieldValue</textarea></p>";
}

/*****************************************************************************/

function renderStatusMenu($allStatus, $selectedStatus, $userIdInModel)
{
echo "<ul>";
foreach ($allStatus as $status)
   {
   $userIsAuthorized = in_array($userIdInModel, $status->getAuthorizedUsers());
   echo "<li>";
   if ($selectedStatus != null && $status->getId() == $selectedStatus->getId())
      echo "<span class=\"wrap\">";
   if ($status->getType() == "edit")
      {
      echo "<a href=\"../control/getRecords.php?statusId=" .
            $status->getId() . "\">" .
            $status->getName();
      if ($userIsAuthorized && $status->getRecordsCount() > 0)
         echo " (" . $status->getRecordsCount() . ")";
      echo "</a>";
      }
   if ($status->getType() == "create")
      {
      if ($userIsAuthorized)
         echo "<a href=\"../control/getForm.php?statusId=" .
               $status->getId() . "\">";
      echo $status->getName();
      if ($userIsAuthorized)
         echo "</a>";
      }
   if ($selectedStatus != null && $status->getId() == $selectedStatus->getId())
      echo "</span>";
   echo "</li>";
   }
echo "</ul>";
}

/*****************************************************************************/

function renderRecordsTable($fieldNames, $fieldLabels, $statusId,
      $records, $userIsAuthorized)
{
echo "<table>";
echo "<tr>";
foreach ($fieldLabels as $fieldLabel)
   echo "<th>$fieldLabel</th>";
if ($userIsAuthorized)
   echo "<th></th>";
echo "</tr>";
foreach ($records as $record)
   {
   echo "<tr>";
   foreach ($fieldNames as $fieldName)
      {
      $getter = "get_" . $fieldName;
      echo "<td>";
      echo $record->$getter();
      echo "</td>";
      }
   if ($userIsAuthorized)
      {
      echo "<td>";
      echo "<a href=\"../control/getForm.php?statusId=$statusId&amp;recordId=" .
```

```
            $record->getId() . "\">Ver...</a>";
      echo "</td>";
      echo "</tr>";
      }
   }
echo "</table>";
}

/*****************************************************************************/

function renderContent($statusId, $records, $userIsAuthorized)
{
if (!is_array($records))
   echo "<p>Seleccione un estado para ver sus registros pendientes.</p>";
else
   {
   if (count($records) == 0)
      echo "<p>Este estado no dispone de registros.</p>";
   else
      renderRecords($statusId, $records, $userIsAuthorized);
   }
}

/*****************************************************************************/
```

C.2.4.3 utils.php

```
<?php

/*****************************************************************************/

function dateToString($date)
{
$dateArray = array_reverse(explode("/", $date));
return implode($dateArray);
}

/*****************************************************************************/

function stringToDate($str)
{
$strArray = array_reverse(explode("-", $str));
return implode("/", $strArray);
}

/*****************************************************************************/

function is_valid_date($date)
{
if ($date == null)
   return false;
$dateArray = explode("/", $date);
if (count($dateArray) != 3)
   return false;
if (strlen($dateArray[1]) != 2 || strlen($dateArray[0]) != 2 ||
      strlen($dateArray[2]) != 4)
   return false;
return checkdate($dateArray[1], $dateArray[0], $dateArray[2]);
}

/*****************************************************************************/

?>
```

C.2.5 Directorio /model

C.2.5.1 DB_mysql.php

```
<?php
```

```
class DB_mysql
{

private $database;
private $server;
private $user;
private $password;
private $connectionId = 0;
private $queryId = 0;
private $errno = 0;
private $error = "";

/************************************************************************/

public function DB_mysql($database = "Form", $server = "localhost", $user = "root",
      $password = "")
{
$this->database = $database;
$this->server = $server;
$this->user = $user;
$this->password = $password;
}

/************************************************************************/

public function connect()
{
$this->connectionId =
      mysql_connect($this->server, $this->user, $this->password);
if (!$this->connectionId)
   {
   $this->error = "Intento de conexión fallido";
   return 0;
   }
$this->useDB();
}

/************************************************************************/

public function useDB()
{
if (!@mysql_select_db($this->database, $this->connectionId))
   {
   $this->error = "No se ha podido abrir " . $this->database;
   return 0;
   }
return $this->connectionId;
}

/************************************************************************/

public function disconnect()
{
if (!mysql_close($this->connectionId))
   {
   $this->error = "Intento de desconexión fallido";
   return 0;
   }
return 1;
}

/************************************************************************/

public function query($query)
{
mysql_query("SET names UTF8");
if ($query == "")
   {
   $this->error = "No se ha especificado una consulta SQL";
   return 0;
   }
```

```
$this->queryId = mysql_query($query, $this->connectionId);
if (!($this->queryId))
   {
   $this->errno = mysql_errno();
   $this->error = mysql_error();
   }
return $this->queryId;
}

/*************************************************************************/

public function numFields()
{
return mysql_num_fields($this->queryId);
}

/*************************************************************************/

public function numRecords()
{
return mysql_num_rows($this->queryId);
}

/*************************************************************************/

public function fieldName($numField)
{
return mysql_field_name($this->queryId, $numField);
}

/*************************************************************************/

public function resultArray()
{
return mysql_fetch_array($this->queryId);
}

/*************************************************************************/

public function getError()
{
return $this->error;
}

/*************************************************************************/

public function getId()
{
return mysql_insert_id();
}

/*************************************************************************/

}
?>
```

C.2.5.2 Error.php

```
<?php

include "../include/classInclude.php";

class Error
{

private $errorCode;
private $errorFieldName;
private $errorFieldLabel;
private $extraInfo;

/*************************************************************************/
```

```
public function Error($errorCode, $errorFieldName, $errorFieldLabel, $extraInfo)
{
$this->errorCode = $errorCode;
$this->errorFieldName  = $errorFieldName;
$this->errorFieldLabel = $errorFieldLabel;
$this->extraInfo = $extraInfo;
}

/*************************************************************************/

public function getErrorCode()
{
return $this->errorCode;
}

/*************************************************************************/

public function getErrorFieldName()
{
return $this->errorFieldName;
}

/*************************************************************************/

public function getErrorFieldLabel()
{
return $this->errorFieldLabel;
}

/*************************************************************************/

public function getExtraInfo()
{
return $this->extraInfo;
}

/*************************************************************************/

}

?>
```

C.2.5.3 Form.php

Este fichero se encuentra inicialmente vacío. El generador se encargará de especificar su contenido completo.

C.2.5.4 Status.php

```
<?php

class Status
{

private $id;
private $name;
private $authorizedUsers;
private $activeFields;
private $ingoing;
private $records;
private $type;

/*************************************************************************/

public function Status($id)
{
$this->id = $id;
$this->getStatusInformation();
}
```

```
/*****************************************************************************/

public function getId()
{
return $this->id;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getActiveFields()
{
return $this->activeFields;
}

/*****************************************************************************/

public function getIngoing()
{
return $this->ingoing;
}

/*****************************************************************************/

public function getRecords()
{
return $this->records;
}

/*****************************************************************************/

public function getType()
{
return $this->type;
}

/*****************************************************************************/

public function getRecordsCount()
{
return count($this->records);
}

/*****************************************************************************/

public function getRecordById($recordId)
{
foreach ($this->records as $record)
   if ($record->getId() == $recordId)
      return $record;
}

/*****************************************************************************/

public function getAuthorizedUsers()
{
return $this->authorizedUsers;
}

/*****************************************************************************/
```

C.2.5.5 User.php

```
<?php
```

```
include "../include/classInclude.php";

class User
{

private $id;
private $name;
private $email;
private $idInModel;
private $myDB;

/*****************************************************************************/

public function setEmail($email)
{
$this->email = $email;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getEmail()
{
return $this->email;
}

/*****************************************************************************/

public function getIdInModel()
{
return $this->idInModel;
}

/*****************************************************************************/

public function validate()
{
$validationResult = new ValidationResult();

if ($this->email == null)
   $validationResult->addErrorNameAndLabel(0, "email",
         "Correo electrónico", "");
else
   {
   $this->myDB->useDB();
   $this->myDB->connect();
   $query = "SELECT * FROM Users WHERE email = '$this->email'";
   $this->myDB->query($query);
   if ($this->myDB->numRecords() == 0)
      $validationResult->addErrorNameAndLabel(1, "email",
            "Correo electrónico", "");
   else
      {
      $row = $this->myDB->resultArray();
      $this->id = $row["id"];
      $this->name = $row["name"];
      $this->idInModel = $row["idInModel"];
      }
   $this->myDB->disconnect();
   }
return $validationResult;
}

/*****************************************************************************/

public function User()
```

```
{
```

C.2.5.6 ValidationResult.php

```
<?php

include "../include/classInclude.php";

class ValidationResult
{

private $errors;

/*****************************************************************************/

function ValidationResult()
{
$this->errors = array();
}

/*****************************************************************************/

function addErrorNameAndLabel($errorCode, $errorFieldName,
      $errorFieldLabel, $extraInfo)
{
array_push($this->errors,
      new Error($errorCode, $errorFieldName, $errorFieldLabel, $extraInfo));
}

/*****************************************************************************/

function getErrorFieldNames()
{
$errorFields = array();
foreach ($this->errors as $error)
   array_push($errorFields, $error->getErrorFieldName());
return $errorFields;
}

/*****************************************************************************/

function getErrors()
{
return $this->errors;
}

/*****************************************************************************/

}

?>
```

C.2.6 Directorio /phpmailer

Contiene el componente empleado para el envío de los correos electrónicos de notificación.

C.2.7 Directorio /view

C.2.7.1 access.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
```

```
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Acceso
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php") ?>
      </div>
      <div id="menu">
      </div>
      <div id="content">
         <h2>Acceso
         </h2>
         <div id="form">
            <p>
               Inserte su dirección de correo electrónico.
            </p>
            <?php
               if (count($errorList) != 0)
                  renderErrorList($errorList);
            ?>
            <form action="../control/access.php" method="post"
                  enctype="multipart/form-data">
               <?php renderAccessFormFields($errorFields, $user->getEmail()); ?>
               <p class="center">
                  <input type="submit" name="access" value="Acceder" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.2.7.2 getForm.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Formulario
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php"); ?>
      </div>
      <div id="menu">
         <?php renderStatusMenu($allStatus, $status, $user->getIdInModel()); ?>
      </div>
      <div id="content">
         <div id="user">
            Usuario: <?= $user->getName() ?>
                  <a href="../control/access.php">(Salir)</a>
         </div>
         <h2>Formulario
         </h2>
         <div id="form">
            <p>
               Rellene los campos activos del formulario.
            </p>
            <?php
               if (count($errorList) != 0)
                  renderErrorList($errorList);
            ?>
```

```
            <form action="../control/getForm.php" method="post"
                  enctype="multipart/form-data">
               <p>
                  <input type="hidden" name="recordId"
                        value="<?= $form->getId(); ?>" />
               </p>
               <p>
                  <input type="hidden" name="statusId"
                        value="<?= $status->getId(); ?>" />
               </p>
               <?php renderFormFields($form, $activeFields, $errorFields); ?>
               <p class="center">
                  <input type="submit" name="accept" value="Aceptar" />
               </p>
            </form>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.2.7.3 getRecords.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>
         Registros
      </title>
      <link rel="stylesheet" type="text/css" href="../style.css" />
   </head>
   <body>
      <div id="header" class="wrap">
         <?php include ("../include/header.php"); ?>
      </div>
      <div id="menu">
         <?php renderStatusMenu($allStatus, $status, $user->getIdInModel()); ?>
      </div>
      <div id="content">
         <div id="user">
            Usuario: <?= $user->getName() ?>
                  <a href="../control/access.php">(Salir)</a>
         </div>
         <h2>Registros
         </h2>
         <div id="form">
            <? renderContent($statusId, $records, $userIsAuthorized) ?>
         </div>
      </div>
      <div id="footer" class="wrap">
         <?php include ("../include/footer.php") ?>
      </div>
   </body>
</html>
```

C.2.8 Directorio raíz

C.2.8.1 index.php

```
<?php

header("Location: control/access.php");
exit;

?>
```

C.2.8.2 style.css

```
*
{
color: #000066;
background-color: #55AAFF;
font-family: "Times New Roman", serif;
font-size: 20px;
}

h1, h2
{
color: white;
font-family: Arial, sans-serif;
}

h1
{
float: left;
text-align: left;
width: 80%;
margin: 1%;
font-size: 1.75em;
}

h2
{
margin-left: 5px;
font-size: 1.15em;
}

#header
{
width: 100%;
float: left;
border: 1px solid #000099;
padding: 2px;
}

#header h2
{
text-align: right;
margin-right: 5%;
font-size: 1.5em;
color: #000066;
margin-bottom: 0px;
}

img
{
padding: 2px;
border: 1px solid red;
}

#header img
{
float: left;
padding: 2px;
border: 1px solid red;
}

#menu
{
width: 30%;
float: left;
margin-top: 20px;
}

#menu ul
{
list-style-type: none;
margin-left: -10px;
```

```
}

#menu li
{
padding-top: 0px;
margin-top: 10px;
}

#content
{
width: 65%;
float: left;
text-align: center;
}

#footer
{
text-align: center;
width: 90%;
margin: 5%;
float: left;
}

#footer p
{
margin: 3px 0;
font-size: 0.65em;
}

.wrap
{
border: 1px solid #000099;
padding: 2px;
}

li, p
{
font-size: 0.85em;
}

span, input, select, option, textarea
{
font-size: 1em;
}

*[disabled="disabled"]
{
color: gray;
}

a
{
font-size: 1em;
text-decoration: none;
}

a:hover
{
text-decoration: underline;
}

input , select, option, textarea
{
background-color: #99FFCC;
}

.label
{
vertical-align: top;
font-size: 0.85em;
margin: 5px 10px 5px 15px;
}
```

```
#form
{
text-align: left;
margin-left: 20%;
}

.center
{
margin-left: 30px;
}

#errors
{
padding: 5px;
border: 1px solid red;
width: 75%;
}

#errors p, #errors li, #errors i
{
font-size: 0.65em;
margin: 0;
padding-bottom: 3px;
}

#errors ul
{
list-style-type: none;
margin: 0 0 0 3px;
}

#errors i
{
font-size: 1em;
}

.errorLabel
{
vertical-align: top;
font-size: 0.85em;
margin: 5px 10px 5px 15px;
padding: 2px;
border: 1px solid red;
}

#content table
{
width: 65%;
border: 1px solid black;
border-spacing: 0;
}

#content td, #content th
{
border: 1px solid black;
}

input[type="radio"], input[type="checkbox"]
{
background-color: #55AAFF;
}

#user
{
font-size: 0.8em;
width: 100%;
text-align: right;
}
```

C.3 MODELOS GENERADOS

En esta sección se detalla el contenido de los modelos creados por la herramienta *Proc2Form*.

C.3.1 Modelo CIM

```
<?xml version="1.0" encoding="utf-8" ?>
<workflow>
   <fields>
      <field id="0" name="Input_String" dataType="String"
            fieldType="Input" tooltip="" grid="" />
      <field id="1" name="Input_Number" dataType="Number"
            fieldType="Input" tooltip="" grid="" />
      <field id="2" name="Input_Date" dataType="Date"
            fieldType="Input" tooltip="" grid="" />
      <field id="3" name="Radio_String" dataType="String"
            fieldType="Radio Button" tooltip="" grid="" />
      <field id="4" name="Radio_Number" dataType="Number"
            fieldType="Radio Button" tooltip="" grid="" />
      <field id="5" name="Radio_Date" dataType="Date"
            fieldType="Radio Button" tooltip="" grid="" />
      <field id="6" name="TextArea" dataType="String"
            fieldType="Text Area" tooltip="" grid="" />
      <field id="7" name="CheckBox" dataType="String"
            fieldType="CheckBox" tooltip="" grid="" />
      <field id="8" name="Drop_String" dataType="String"
            fieldType="ListBox" tooltip="" grid="" />
      <field id="9" name="Drop_Number" dataType="Number"
            fieldType="ListBox" tooltip="" grid="" />
      <field id="10" name="Drop_Date" dataType="Date"
            fieldType="ListBox" tooltip="" grid="" />
   </fields>
   <users>
      <user id="0" name="Raúl Castro" />
      <user id="1" name="Raúl Castro 2" />
   </users>
   <tasks>
      <task id="5" name="CREATION_1">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
            <user ref="0" />
         </users>
      </task>
      <task id="6" name="CREATION_2">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
            <user ref="0" />
         </users>
      </task>
      <task id="7" name="EDIT_1">
         <fields>
            <field ref="5" />
            <field ref="6" />
```

```
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="8" name="EDIT_2">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="9" name="EDIT_3">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="10" name="EDIT_4">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="15" name="FINISH_1">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="16" name="FINISH_2">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
```

```
         </users>
      </task>
   </tasks>
   <gateways>
      <gateway id="11" type="Parallel" />
      <gateway id="12" type="Exclusive" />
      <gateway id="13" type="Parallel" />
      <gateway id="14" type="Exclusive" />
   </gateways>
   <transitions>
      <transition id="20" sourceId="5" targetId="11" />
      <transition id="21" sourceId="6" targetId="12" />
      <transition id="22" sourceId="7" targetId="13" />
      <transition id="23" sourceId="8" targetId="13" />
      <transition id="24" sourceId="9" targetId="14" />
      <transition id="25" sourceId="10" targetId="14" />
      <transition id="26" sourceId="11" targetId="7" />
      <transition id="27" sourceId="11" targetId="8" />
      <transition id="28" sourceId="12" targetId="9" />
      <transition id="29" sourceId="12" targetId="10" />
      <transition id="30" sourceId="13" targetId="15" />
      <transition id="31" sourceId="14" targetId="16" />
   </transitions>
</workflow>
```

C.3.2 Modelo PIM

```
<?xml version="1.0" encoding="utf-8" ?>
<workflow>
   <fields>
      <field id="0" name="Input_String" dataType="String" fieldType="Input"
            tooltip="" grid="" checkMaxLength="100"/>
      <field id="1" name="Input_Number" dataType="Number" fieldType="Input"
            tooltip="" grid="" checkMinValue="1" checkMaxValue="10"/>
      <field id="2" name="Input_Date" dataType="Date" fieldType="Input"
            tooltip="" grid="" checkMinValue="01/01/2012"
            checkMaxValue="31/01/2012"/>
      <field id="3" name="Radio_String" dataType="String" fieldType="Radio Button"
            tooltip="" grid="" values="uno;dos" labels="uno;dos"/>
      <field id="4" name="Radio_Number" dataType="Number" fieldType="Radio Button"
            tooltip="" grid="" values="1;2" labels="uno_1;dos_2"/>
      <field id="5" name="Radio_Date" dataType="Date" fieldType="Radio Button"
            tooltip="" grid="" values="26/01/1983;29/01/1984"
            labels="fecha_1;fecha_2"/>
      <field id="6" name="TextArea" dataType="String" fieldType="Text Area"
            tooltip="" grid="" checkMaxLength="500"/>
      <field id="7" name="CheckBox" dataType="String" fieldType="CheckBox"
            tooltip="" grid="" />
      <field id="8" name="Drop_String" dataType="String" fieldType="ListBox"
            tooltip="" grid="" values="tres;cuatro" labels="tres;cuatro"/>
      <field id="9" name="Drop_Number" dataType="Number" fieldType="ListBox"
            tooltip="" grid="" values="3;4" labels="tres_3;cuatro_4"/>
      <field id="10" name="Drop_Date" dataType="Date" fieldType="ListBox"
            tooltip="" grid="" values="03/12/1953;17/12/1958"
            labels="fecha_3;fecha_4"/>
   </fields>
   <users>
      <user id="0" name="Raúl Castro" />
      <user id="1" name="Raúl Castro 2" />
   </users>
   <tasks>
      <task id="5" name="CREATION_1">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
```

```
            <user ref="0" />
         </users>
      </task>
      <task id="6" name="CREATION_2">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
            <user ref="0" />
         </users>
      </task>
      <task id="7" name="EDIT_1">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="8" name="EDIT_2">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="9" name="EDIT_3">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="10" name="EDIT_4">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="15" name="FINISH_1">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
```

```
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="16" name="FINISH_2">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
   </tasks>
   <gateways>
      <gateway id="11" type="Parallel" />
      <gateway id="12" type="Exclusive" />
      <gateway id="13" type="Parallel" />
      <gateway id="14" type="Exclusive" />
   </gateways>
   <transitions>
      <transition id="20" sourceId="5" targetId="11" />
      <transition id="21" sourceId="6" targetId="12" />
      <transition id="22" sourceId="7" targetId="13" />
      <transition id="23" sourceId="8" targetId="13" />
      <transition id="24" sourceId="9" targetId="14" />
      <transition id="25" sourceId="10" targetId="14" />
      <transition id="26" sourceId="11" targetId="7" />
      <transition id="27" sourceId="11" targetId="8" />
      <transition id="28" sourceId="12" targetId="9" />
      <transition id="29" sourceId="12" targetId="10" />
      <transition id="30" sourceId="13" targetId="15" />
      <transition id="31" sourceId="14" targetId="16" />
   </transitions>
</workflow>
```

C.3.3 Modelo PSM

```
<?xml version="1.0" encoding="utf-8" ?>
<workflow cssFile="" emailAccountName="raulcaster"
      emailAccountPassword="**********">
   <fields>
      <field id="0" name="Input_String" dataType="String" fieldType="Input"
            tooltip="" grid="" checkMaxLength="100"/>
      <field id="1" name="Input_Number" dataType="Number" fieldType="Input"
            tooltip="" grid="" checkMinValue="1" checkMaxValue="10"/>
      <field id="2" name="Input_Date" dataType="Date" fieldType="Input"
            tooltip="" grid="" checkMinValue="01/01/2012"
            checkMaxValue="31/01/2012"/>
      <field id="3" name="Radio_String" dataType="String" fieldType="Radio Button"
            tooltip="" grid="" values="uno;dos" labels="uno;dos"/>
      <field id="4" name="Radio_Number" dataType="Number" fieldType="Radio Button"
            tooltip="" grid="" values="1;2" labels="uno_1;dos_2"/>
      <field id="5" name="Radio_Date" dataType="Date" fieldType="Radio Button"
            tooltip="" grid="" values="26/01/1983;29/01/1984"
            labels="fecha_1;fecha_2"/>
      <field id="6" name="TextArea" dataType="String" fieldType="Text Area"
            tooltip="" grid="" checkMaxLength="500"/>
      <field id="7" name="CheckBox" dataType="String" fieldType="CheckBox"
            tooltip="" grid="" />
      <field id="8" name="Drop_String" dataType="String" fieldType="ListBox"
            tooltip="" grid="" values="tres;cuatro" labels="tres;cuatro"/>
```

```
      <field id="9" name="Drop_Number" dataType="Number" fieldType="ListBox"
            tooltip="" grid="" values="3;4" labels="tres_3;cuatro_4"/>
      <field id="10" name="Drop_Date" dataType="Date" fieldType="ListBox"
            tooltip="" grid="" values="03/12/1953;17/12/1958"
            labels="fecha_3;fecha_4"/>
   </fields>
   <users>
      <user id="0" name="Raúl Castro" email="raulcaster@gmail.com" />
      <user id="1" name="Raúl Castro 2" email="rcastro_83@hotmail.com" />
   </users>
   <tasks>
      <task id="5" name="CREATION_1">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
            <user ref="0" />
         </users>
      </task>
      <task id="6" name="CREATION_2">
         <fields>
            <field ref="0" />
            <field ref="1" />
            <field ref="2" />
            <field ref="3" />
            <field ref="4" />
         </fields>
         <users>
            <user ref="0" />
         </users>
      </task>
      <task id="7" name="EDIT_1">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="8" name="EDIT_2">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
         </users>
      </task>
      <task id="9" name="EDIT_3">
         <fields>
            <field ref="5" />
            <field ref="6" />
            <field ref="7" />
            <field ref="8" />
            <field ref="9" />
            <field ref="10" />
         </fields>
         <users>
            <user ref="1" />
```

```
      </users>
   </task>
   <task id="10" name="EDIT_4">
      <fields>
         <field ref="5" />
         <field ref="6" />
         <field ref="7" />
         <field ref="8" />
         <field ref="9" />
         <field ref="10" />
      </fields>
      <users>
         <user ref="1" />
      </users>
   </task>
   <task id="15" name="FINISH_1">
      <fields>
         <field ref="5" />
         <field ref="6" />
         <field ref="7" />
         <field ref="8" />
         <field ref="9" />
         <field ref="10" />
      </fields>
      <users>
         <user ref="1" />
      </users>
   </task>
   <task id="16" name="FINISH_2">
      <fields>
         <field ref="5" />
         <field ref="6" />
         <field ref="7" />
         <field ref="8" />
         <field ref="9" />
         <field ref="10" />
      </fields>
      <users>
         <user ref="1" />
      </users>
   </task>
</tasks>
<gateways>
   <gateway id="11" type="Parallel" />
   <gateway id="12" type="Exclusive" />
   <gateway id="13" type="Parallel" />
   <gateway id="14" type="Exclusive" />
</gateways>
<transitions>
   <transition id="20" sourceId="5" targetId="11" />
   <transition id="21" sourceId="6" targetId="12" />
   <transition id="22" sourceId="7" targetId="13" />
   <transition id="23" sourceId="8" targetId="13" />
   <transition id="24" sourceId="9" targetId="14" />
   <transition id="25" sourceId="10" targetId="14" />
   <transition id="26" sourceId="11" targetId="7" />
   <transition id="27" sourceId="11" targetId="8" />
   <transition id="28" sourceId="12" targetId="9" />
   <transition id="29" sourceId="12" targetId="10" />
   <transition id="30" sourceId="13" targetId="15" />
   <transition id="31" sourceId="14" targetId="16" />
</transitions>
<showInPortal>
   <field ref="0" />
   <field ref="1" />
   <field ref="2" />
   <field ref="3" />
</showInPortal>
<showInMail>
   <field ref="0" />
   <field ref="1" />
   <field ref="2" />
```

```
      <field ref="3" />
      <field ref="4" />
      <field ref="5" />
   </showInMail>
</workflow>
```

C.4 CÓDIGO FUENTE GENERADO

En esta sección se muestra el contenido definitivo de los ficheros almacenados en /scaffold, que son manipulados por *Form2Proc*.

C.4.1 Directorio /lib

C.4.1.1 helpers.php

```
<?php

include "../lib/error.php";

/*********************************************************************/

function renderAccessFormFields($errorFields, $email)
{
$label = "Correo electrónico:";
echo "<p>";
getFormFieldLabel($errorFields, "email", "Correo electrónico:");
echo "<input type=\"text\" name=\"email\" value=\"$email\" " .
      "/></p>";
}

/*********************************************************************/

function getFormFieldLabel($errorFields, $fieldName, $fieldLabel)
{
echo "<span class=\"";
if (in_array($fieldName, $errorFields))
   echo "errorLabel";
else
   echo "label";
echo "\">$fieldLabel</span>";
}

/*********************************************************************/

function renderErrorList($errorList)
{
echo "<div id=\"errors\">\n";
echo "<p>Se han detectado los siguientes errores:</p>";
echo "<ul>\n";
foreach ($errorList as $error)
   echo "<li>" . getErrorDescription($error) . "</li>";
echo "</ul>\n";
echo "</div>";
}

/*********************************************************************/

function getErrorDescription($error)
{
$errorCode = $error->getErrorCode();
$errorField = $error->getErrorFieldLabel();
$extraInfo = $error->getExtraInfo();
$errorDescription = getErrorTextualDescription($errorCode,
      $errorField, $extraInfo);
return $errorDescription;
}

/*********************************************************************/

function renderDirectInputControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields)
```

```
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<input type=\"text\" name=\"$fieldName\" value=\"$fieldValue\" ";
if ($fieldIsActive == false)
   echo "disabled = \"disabled\" ";
echo "/></p>";
}

/*************************************************************************/

function renderRadioButtonControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields, $values, $labels)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);

$valuesArray = explode(";", $values);
$labelsArray = explode(";", $labels);

for ($i = 0; $i < count($valuesArray); $i++)
   {
   $value = $valuesArray[$i];
   $label = $labelsArray[$i];
   echo "<input type=\"radio\" name=\"$fieldName\" " .
         "value=\"$value\" ";
   if ($fieldIsActive == false)
      echo "disabled=\"disabled\" ";
   if ($fieldValue == $value)
      echo "checked=\"checked\" ";
   echo "/>$label";
   }

echo "</p>";
}

/*************************************************************************/

function renderDropDownControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields, $values, $labels)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);

$valuesArray = explode(";", $values);
$labelsArray = explode(";", $labels);

echo "<select name=\"$fieldName\" ";
if ($fieldIsActive == false)
   echo "disabled=\"disabled\"";
echo ">";
echo "<option value=\"\"></option>";

for ($i = 0; $i < count($valuesArray); $i++)
   {
   $value = $valuesArray[$i];
   $label = $labelsArray[$i];
   echo "<option value=\"$value\" ";
   if ($fieldValue == $value)
      echo "selected=\"selected\" ";
   echo ">$label</option>";
   }

echo "</select>";

echo "</p>";
}

/*************************************************************************/

function renderCheckBoxControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields)
```

```
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<input type=\"checkbox\" name=\"$fieldName\" value=\"1\" ";
if ($fieldIsActive == false)
   echo "disabled=\"disabled\" ";
if ($fieldValue == 1)
   echo "checked=\"checked\" ";
echo "/></p>";
}

/*****************************************************************************/

function renderTextAreaControl($fieldName, $fieldLabel, $fieldValue,
      $fieldIsActive, $errorFields)
{
echo "<p>";
getFormFieldLabel($errorFields, $fieldName, $fieldLabel);
echo "<textarea name=\"$fieldName\" rows=\"3\" cols=\"15\" ";
if ($fieldIsActive == false)
   echo "disabled = \"disabled\" ";
echo ">$fieldValue</textarea></p>";
}

/*****************************************************************************/

function renderStatusMenu($allStatus, $selectedStatus, $userIdInModel)
{
echo "<ul>";
foreach ($allStatus as $status)
   {
   $userIsAuthorized = in_array($userIdInModel, $status->getAuthorizedUsers());
   echo "<li>";
   if ($selectedStatus != null && $status->getId() == $selectedStatus->getId())
      echo "<span class=\"wrap\">";
   if ($status->getType() == "edit")
      {
      echo "<a href=\"../control/getRecords.php?statusId=" .
            $status->getId() . "\">" .
            $status->getName();
      if ($userIsAuthorized && $status->getRecordsCount() > 0)
         echo " (" . $status->getRecordsCount() . ")";
      echo "</a>";
      }
   if ($status->getType() == "create")
      {
      if ($userIsAuthorized)
         echo "<a href=\"../control/getForm.php?statusId=" .
               $status->getId() . "\">";
      echo $status->getName();
      if ($userIsAuthorized)
         echo "</a>";
      }
   if ($selectedStatus != null && $status->getId() == $selectedStatus->getId())
      echo "</span>";
   echo "</li>";
   }
echo "</ul>";
}

/*****************************************************************************/

function renderRecordsTable($fieldNames, $fieldLabels, $statusId,
      $records, $userIsAuthorized)
{
echo "<table>";
echo "<tr>";
foreach ($fieldLabels as $fieldLabel)
   echo "<th>$fieldLabel</th>";
if ($userIsAuthorized)
   echo "<th></th>";
echo "</tr>";
```

```
foreach ($records as $record)
   {
   echo "<tr>";
   foreach ($fieldNames as $fieldName)
      {
      $getter = "get_" . $fieldName;
      echo "<td>";
      echo $record->$getter();
      echo "</td>";
      }
   if ($userIsAuthorized)
      {
      echo "<td>";
      echo "<a href=\"../control/getForm.php?statusId=$statusId&amp;recordId=" .
            $record->getId() . "\">Ver...</a>";
      echo "</td>";
      echo "</tr>";
      }
   }
echo "</table>";
}

/*****************************************************************************/

function renderContent($statusId, $records, $userIsAuthorized)
{
if (!is_array($records))
   echo "<p>Seleccione un estado para ver sus registros pendientes.</p>";
else
   {
   if (count($records) == 0)
      echo "<p>Este estado no dispone de registros.</p>";
   else
      renderRecords($statusId, $records, $userIsAuthorized);
   }
}

/*****************************************************************************/

function renderFormFields($form, $activeFields, $errorFields)
{
$fieldValue = $form->get_Input_String();
$fieldIsActive = in_array(0, $activeFields);
renderDirectInputControl("Input_String", "Input String:", $fieldValue,
$fieldIsActive, $errorFields);
$fieldValue = $form->get_Input_Number();
$fieldIsActive = in_array(1, $activeFields);
renderDirectInputControl("Input_Number", "Input Number:", $fieldValue,
$fieldIsActive, $errorFields);
$fieldValue = $form->get_Input_Date();
$fieldIsActive = in_array(2, $activeFields);
renderDirectInputControl("Input_Date", "Input Date:", $fieldValue, $fieldIsActive,
$errorFields);
$fieldValue = $form->get_Radio_String();
$fieldIsActive = in_array(3, $activeFields);
renderRadioButtonControl("Radio_String", "Radio String:", $fieldValue,
$fieldIsActive, $errorFields, "uno;dos", "uno;dos");
$fieldValue = $form->get_Radio_Number();
$fieldIsActive = in_array(4, $activeFields);
renderRadioButtonControl("Radio_Number", "Radio Number:", $fieldValue,
$fieldIsActive, $errorFields, "1;2", "uno_1;dos_2");
$fieldValue = $form->get_Radio_Date();
$fieldIsActive = in_array(5, $activeFields);
renderRadioButtonControl("Radio_Date", "Radio Date:", $fieldValue, $fieldIsActive,
$errorFields, "26/01/1983;29/01/1984", "fecha_1;fecha_2");
$fieldValue = $form->get_TextArea();
$fieldIsActive = in_array(6, $activeFields);
renderTextAreaControl("TextArea", "Text Area:", $fieldValue, $fieldIsActive,
$errorFields);
$fieldValue = $form->get_CheckBox();
$fieldIsActive = in_array(7, $activeFields);
```

```
renderCheckBoxControl("CheckBox", "Check Box:", $fieldValue, $fieldIsActive,
$errorFields);
$fieldValue = $form->get_Drop_String();
$fieldIsActive = in_array(8, $activeFields);
renderDropDownControl("Drop_String", "Drop String:", $fieldValue, $fieldIsActive,
$errorFields, "tres;cuatro", "tres;cuatro");
$fieldValue = $form->get_Drop_Number();
$fieldIsActive = in_array(9, $activeFields);
renderDropDownControl("Drop_Number", "Drop Number:", $fieldValue, $fieldIsActive,
$errorFields, "3;4", "tres_3;cuatro_4");
$fieldValue = $form->get_Drop_Date();
$fieldIsActive = in_array(10, $activeFields);
renderDropDownControl("Drop_Date", "Drop Date:", $fieldValue, $fieldIsActive,
$errorFields, "03/12/1953;17/12/1958", "fecha_3;fecha_4");
}

/*****************************************************************************/

function renderRecords($statusId, $records, $userIsAuthorized)
{
renderRecordsTable(array("Input_String", "Input_Number", "Input_Date",
"Radio_String"),
      array("Input String", "Input Number", "Input Date", "Radio String"),
      $statusId ,
      $records ,
      $userIsAuthorized);
}

/*****************************************************************************/

?>
```

C.4.2 Directorio /model

C.4.2.1 Form.php

```
<?php

include "../include/classInclude.php";
include "../lib/utils.php";

class Form
{

private $myDB;
private $phpmailer;
private $id;
private $Input_String;
private $Input_Number;
private $Input_Date;
private $Radio_String;
private $Radio_Number;
private $Radio_Date;
private $TextArea;
private $CheckBox;
private $Drop_String;
private $Drop_Number;
private $Drop_Date;

/*****************************************************************************/

public function Form()
{
}

/*****************************************************************************/

public function FormFromArray($fieldsArray)
{
```

```
$this->myDB = new DB_mysql("Form", "localhost", "root", "");
$this->phpmailer = new PHPMailer(true);
if (isset($fieldsArray["id"]))
   $this->id = $fieldsArray["id"];
if (isset($fieldsArray["Input_String"]))
   $this->Input_String = $fieldsArray["Input_String"];
if (isset($fieldsArray["Input_Number"]))
   $this->Input_Number = $fieldsArray["Input_Number"];
if (isset($fieldsArray["Input_Date"]))
   $this->Input_Date = $fieldsArray["Input_Date"];
if (isset($fieldsArray["Radio_String"]))
   $this->Radio_String = $fieldsArray["Radio_String"];
if (isset($fieldsArray["Radio_Number"]))
   $this->Radio_Number = $fieldsArray["Radio_Number"];
if (isset($fieldsArray["Radio_Date"]))
   $this->Radio_Date = $fieldsArray["Radio_Date"];
if (isset($fieldsArray["TextArea"]))
   $this->TextArea = $fieldsArray["TextArea"];
if (isset($fieldsArray["CheckBox"]))
   $this->CheckBox = $fieldsArray["CheckBox"];
if (isset($fieldsArray["Drop_String"]))
   $this->Drop_String = $fieldsArray["Drop_String"];
if (isset($fieldsArray["Drop_Number"]))
   $this->Drop_Number = $fieldsArray["Drop_Number"];
if (isset($fieldsArray["Drop_Date"]))
   $this->Drop_Date = $fieldsArray["Drop_Date"];
}

/*****************************************************************************/

private function sendEmail($statusName, $addresses)
{
$body = "Se requiere su participación para que " .
      "el flujo de un registro " .
      "continúe al estado $statusName.<br />";
$this->phpmailer->IsSMTP();
try
   {
   $this->phpmailer->SMTPAuth = true;
   $this->phpmailer->SMTPSecure = "ssl";
   $this->phpmailer->Host = "smtp.gmail.com";
   $this->phpmailer->Port = 465;
   $this->phpmailer->CharSet = "UTF-8";
   foreach ($addresses as $address)
      $this->phpmailer->AddAddress($address, $address);
   $this->phpmailer->Username = "raulcaster@gmail.com";
   $this->phpmailer->Password = "rauldixebra";
   $this->phpmailer->SetFrom("raulcaster@gmail.com", "Proc2Form");
   $this->phpmailer->Subject = "Requerimiento de participación";
   $fieldValue = $this->get_Input_String();
   $body .= "<br />Input String: $fieldValue";
   $fieldValue = $this->get_Input_Number();
   $body .= "<br />Input Number: $fieldValue";
   $fieldValue = $this->get_Input_Date();
   $body .= "<br />Input Date: $fieldValue";
   $fieldValue = $this->get_Radio_String();
   $body .= "<br />Radio String: $fieldValue";
   $fieldValue = $this->get_Radio_Number();
   $body .= "<br />Radio Number: $fieldValue";
   $fieldValue = $this->get_Radio_Date();
   $body .= "<br />Radio Date: $fieldValue";
   $this->phpmailer->MsgHTML($body);
   $this->phpmailer->Send();
   }
catch(Exception $e)
   {
   echo $e->getMessage();
   }
}
/*****************************************************************************/

public function getId()
```

```
{
return $this->id;
}

/*****************************************************************************/

public function get_Input_String()
{
return $this->Input_String;
}

/*****************************************************************************/

public function get_Input_Number()
{
return $this->Input_Number;
}

/*****************************************************************************/

public function get_Input_Date()
{
return stringToDate($this->Input_Date);
}

/*****************************************************************************/

public function get_Radio_String()
{
return $this->Radio_String;
}

/*****************************************************************************/

public function get_Radio_Number()
{
return $this->Radio_Number;
}

/*****************************************************************************/

public function get_Radio_Date()
{
return stringToDate($this->Radio_Date);
}

/*****************************************************************************/

public function get_TextArea()
{
return $this->TextArea;
}

/*****************************************************************************/

public function get_CheckBox()
{
return $this->CheckBox;
}

/*****************************************************************************/

public function get_Drop_String()
{
return $this->Drop_String;
}

/*****************************************************************************/

public function get_Drop_Number()
{
return $this->Drop_Number;
```

```
}

/*************************************************************************/

public function get_Drop_Date()
{
return stringToDate($this->Drop_Date);
}

/*************************************************************************/

private function isEnabled($transitionId)
{
$this->myDB = new DB_mysql("Form", "localhost", "root", "");
$this->myDB->connect();
$query = "SELECT * FROM ActiveTransitions
      WHERE recordId = $this->id AND transitionId = $transitionId";
$this->myDB->query($query);
$numRows = $this->myDB->numRecords();
$this->myDB->disconnect();
return $numRows > 0;
}

/*************************************************************************/

private function enableTransition($transitionId)
{
$this->myDB = new DB_mysql("Form", "localhost", "root", "");
$this->myDB->connect();
$query = "INSERT INTO ActiveTransitions(recordId, transitionId)
      VALUES ($this->id, $transitionId)";
$this->myDB->query($query);
$this->myDB->disconnect();
}

/*************************************************************************/

private function disableTransition($transitionId)
{
$this->myDB = new DB_mysql("Form", "localhost", "root", "");
$this->myDB->connect();
$query = "DELETE FROM ActiveTransitions
      WHERE recordId = $this->id AND transitionId = $transitionId";
$this->myDB->query($query);
$this->myDB->disconnect();
}

/*************************************************************************/

public function validateField_Input_String(&$validationResult)
{
if ($this->Input_String == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Input_String", "Input String", "");
   return;
   }
if (strlen($this->Input_String) > 100)
   {
   $validationResult->addErrorNameAndLabel(2, "Input_String", "Input String",
"100");
   return;
   }
}

/*************************************************************************/

public function validateField_Input_Number(&$validationResult)
{
if ($this->Input_Number == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Input_Number", "Input Number", "");
   return;
```

```
   }
if (!is_numeric($this->Input_Number))
   {
   $validationResult->addErrorNameAndLabel(4, "Input_Number", "Input Number", "");
   return;
   }
if ($this->Input_Number < 1)
   {
   $validationResult->addErrorNameAndLabel(5, "Input_Number", "Input Number", "1");
   return;
   }
if ($this->Input_Number > 10)
   {
   $validationResult->addErrorNameAndLabel(6, "Input_Number", "Input Number",
"10");
   return;
   }
}

/*************************************************************************/

public function validateField_Input_Date(&$validationResult)
{
if ($this->Input_Date == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Input_Date", "Input Date", "");
   return;
   }
if (!is_valid_date($this->Input_Date))
   {
   $validationResult->addErrorNameAndLabel(3, "Input_Date", "Input Date", "");
   return;
   }
if (dateToString($this->Input_Date) < '20120101')
   {
   $validationResult->addErrorNameAndLabel(5, "Input_Date", "Input Date",
"01/01/2012");
   return;
   }
if (dateToString($this->Input_Date) > '20120131')
   {
   $validationResult->addErrorNameAndLabel(6, "Input_Date", "Input Date",
"31/01/2012");
   return;
   }
}

/*************************************************************************/

public function validateField_Radio_String(&$validationResult)
{
if ($this->Radio_String == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Radio_String", "Radio String", "");
   return;
   }
}

/*************************************************************************/

public function validateField_Radio_Number(&$validationResult)
{
if ($this->Radio_Number == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Radio_Number", "Radio Number", "");
   return;
   }
}

/*************************************************************************/

public function validateField_Radio_Date(&$validationResult)
```

```
{
if ($this->Radio_Date == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Radio_Date", "Radio Date", "");
   return;
   }
}

/*************************************************************************/

public function validateField_TextArea(&$validationResult)
{
if ($this->TextArea == null)
   {
   $validationResult->addErrorNameAndLabel(0, "TextArea", "Text Area", "");
   return;
   }
if (strlen($this->TextArea) > 500)
   {
   $validationResult->addErrorNameAndLabel(2, "TextArea", "Text Area", "500");
   return;
   }
}

/*************************************************************************/

public function validateField_CheckBox(&$validationResult)
{
return;
}

/*************************************************************************/

public function validateField_Drop_String(&$validationResult)
{
if ($this->Drop_String == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Drop_String", "Drop String", "");
   return;
   }
}

/*************************************************************************/

public function validateField_Drop_Number(&$validationResult)
{
if ($this->Drop_Number == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Drop_Number", "Drop Number", "");
   return;
   }
}

/*************************************************************************/

public function validateField_Drop_Date(&$validationResult)
{
if ($this->Drop_Date == null)
   {
   $validationResult->addErrorNameAndLabel(0, "Drop_Date", "Drop Date", "");
   return;
   }
}

/*************************************************************************/

public function validateTask_CREATION_1(&$validationResult)
{
$this->validateField_Input_String($validationResult);
$this->validateField_Input_Number($validationResult);
$this->validateField_Input_Date($validationResult);
$this->validateField_Radio_String($validationResult);
```

```
$this->validateField_Radio_Number($validationResult);
}

/*********************************************************************************/

public function save_CREATION_1()
{
$this->myDB->connect();
$query = "INSERT INTO Records
(Input_String,
Input_Number,
Input_Date,
Radio_String,
Radio_Number)
VALUES
('$this->Input_String',
$this->Input_Number,
'" . dateToString($this->Input_Date) . "',
'$this->Radio_String',
$this->Radio_Number)";
$this->myDB->query($query);
$this->id = $this->myDB->getId();
$this->myDB->disconnect();
$this->enableTransition(20);
if (1==1 && $this->isEnabled(20))
   {
   $this->enableTransition(26);
   $this->sendEmail("EDIT_1", array("rcastro_83@hotmail.com"));
   $this->enableTransition(27);
   $this->sendEmail("EDIT_2", array("rcastro_83@hotmail.com"));
   $this->disableTransition(20);
   }
}

/*********************************************************************************/

public function validateTask_CREATION_2(&$validationResult)
{
$this->validateField_Input_String($validationResult);
$this->validateField_Input_Number($validationResult);
$this->validateField_Input_Date($validationResult);
$this->validateField_Radio_String($validationResult);
$this->validateField_Radio_Number($validationResult);
}

/*********************************************************************************/

public function save_CREATION_2()
{
$this->myDB->connect();
$query = "INSERT INTO Records
(Input_String,
Input_Number,
Input_Date,
Radio_String,
Radio_Number)
VALUES
('$this->Input_String',
$this->Input_Number,
'" . dateToString($this->Input_Date) . "',
'$this->Radio_String',
$this->Radio_Number)";
$this->myDB->query($query);
$this->id = $this->myDB->getId();
$this->myDB->disconnect();
$this->enableTransition(21);
$this->enableTransition(28);
$this->sendEmail("EDIT_3", array("rcastro_83@hotmail.com"));
$this->enableTransition(29);
$this->sendEmail("EDIT_4", array("rcastro_83@hotmail.com"));
$this->disableTransition(21);
}
```

```
/*************************************************************************/

public function validateTask_EDIT_1(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/*************************************************************************/

public function save_EDIT_1()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(26);
$this->enableTransition(22);
if (1==1 && $this->isEnabled(22) && $this->isEnabled(23))
   {
   $this->enableTransition(30);
   $this->sendEmail("FINISH_1", array("rcastro_83@hotmail.com"));
   $this->disableTransition(22);
   $this->disableTransition(23);
   }
}

/*************************************************************************/

public function validateTask_EDIT_2(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/*************************************************************************/

public function save_EDIT_2()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(27);
$this->enableTransition(23);
if (1==1 && $this->isEnabled(22) && $this->isEnabled(23))
   {
   $this->enableTransition(30);
   $this->sendEmail("FINISH_1", array("rcastro_83@hotmail.com"));
```

```
   $this->disableTransition(22);
   $this->disableTransition(23);
   }
}

/**************************************************************************/

public function validateTask_EDIT_3(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/**************************************************************************/

public function save_EDIT_3()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(28);
$this->disableTransition(28);
$this->disableTransition(29);
$this->enableTransition(24);
$this->enableTransition(31);
$this->sendEmail("FINISH_2", array("rcastro_83@hotmail.com"));
$this->disableTransition(24);
}

/**************************************************************************/

public function validateTask_EDIT_4(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/**************************************************************************/

public function save_EDIT_4()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(29);
$this->disableTransition(28);
$this->disableTransition(29);
$this->enableTransition(25);
```

```
$this->enableTransition(31);
$this->sendEmail("FINISH_2", array("rcastro_83@hotmail.com"));
$this->disableTransition(25);
}

/**********************************************************************************/

public function validateTask_FINISH_1(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/**********************************************************************************/

public function save_FINISH_1()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(30);
}

/**********************************************************************************/

public function validateTask_FINISH_2(&$validationResult)
{
$this->validateField_Radio_Date($validationResult);
$this->validateField_TextArea($validationResult);
$this->validateField_CheckBox($validationResult);
$this->validateField_Drop_String($validationResult);
$this->validateField_Drop_Number($validationResult);
$this->validateField_Drop_Date($validationResult);
}

/**********************************************************************************/

public function save_FINISH_2()
{
$this->myDB->connect();
$query = "UPDATE Records SET
Radio_Date = '" . dateToString($this->Radio_Date) . "',
TextArea = '$this->TextArea',
CheckBox = '$this->CheckBox',
Drop_String = '$this->Drop_String',
Drop_Number = $this->Drop_Number,
Drop_Date = '" . dateToString($this->Drop_Date) . "'
WHERE id = $this->id";
$this->myDB->query($query);
$this->myDB->disconnect();
$this->disableTransition(31);
$this->disableTransition(31);
}

/**********************************************************************************/

}

?>
```

C.4.2.2 Status.php

```
<?php

class Status
{

private $id;
private $name;
private $authorizedUsers;
private $activeFields;
private $ingoing;
private $records;
private $type;

/*****************************************************************************/

public function Status($id)
{
$this->id = $id;
$this->getStatusInformation();
}

/*****************************************************************************/

public function getId()
{
return $this->id;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getActiveFields()
{
return $this->activeFields;
}

/*****************************************************************************/

public function getIngoing()
{
return $this->ingoing;
}

/*****************************************************************************/

public function getRecords()
{
return $this->records;
}

/*****************************************************************************/

public function getType()
{
return $this->type;
}

/*****************************************************************************/

public function getRecordsCount()
{
return count($this->records);
}
```

```
/*****************************************************************************/

public function getRecordById($recordId)
{
foreach ($this->records as $record)
   if ($record->getId() == $recordId)
      return $record;
}

/*****************************************************************************/

public function getAuthorizedUsers()
{
return $this->authorizedUsers;
}

/*****************************************************************************/

private function calculateRecords()
{
$result = array();
if ($this->ingoing == null)
   return $result;
$myDB = new DB_mysql("Form",
      "localhost", "root", "");
$myDB->connect();
$query = "SELECT * FROM Records WHERE id IN
(SELECT recordId FROM ActiveTransitions
WHERE transitionId=$this->ingoing)";
$myDB->query($query);
while ($myDB->numRecords() > 0 && $row = $myDB->resultArray())
   {
   $newForm = new Form();
   $newForm->formFromArray($row);
   array_push($result, $newForm);
   }
$myDB->disconnect();
return $result;
}

/*****************************************************************************/

public function getStatusInformation()
{
switch($this->id)
   {
   case 5:
      {
      $this->name = "CREATION_1";
      $this->type = "create";
      $this->authorizedUsers = array(0);
      $this->activeFields = array(0, 1, 2, 3, 4);
      $this->ingoing = null;
      $this->records = $this->calculateRecords();
      break;
      }
   case 6:
      {
      $this->name = "CREATION_2";
      $this->type = "create";
      $this->authorizedUsers = array(0);
      $this->activeFields = array(0, 1, 2, 3, 4);
      $this->ingoing = null;
      $this->records = $this->calculateRecords();
      break;
      }
   case 7:
      {
      $this->name = "EDIT_1";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
```

```
      $this->ingoing = 26;
      $this->records = $this->calculateRecords();
      break;
      }
   case 8:
      {
      $this->name = "EDIT_2";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
      $this->ingoing = 27;
      $this->records = $this->calculateRecords();
      break;
      }
   case 9:
      {
      $this->name = "EDIT_3";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
      $this->ingoing = 28;
      $this->records = $this->calculateRecords();
      break;
      }
   case 10:
      {
      $this->name = "EDIT_4";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
      $this->ingoing = 29;
      $this->records = $this->calculateRecords();
      break;
      }
   case 15:
      {
      $this->name = "FINISH_1";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
      $this->ingoing = 30;
      $this->records = $this->calculateRecords();
      break;
      }
   case 16:
      {
      $this->name = "FINISH_2";
      $this->type = "edit";
      $this->authorizedUsers = array(1);
      $this->activeFields = array(5, 6, 7, 8, 9, 10);
      $this->ingoing = 31;
      $this->records = $this->calculateRecords();
      break;
      }
   }
}

/*****************************************************************************/

public static function getAllStatus()
{
$result = array();
$newStatus = new Status(5);
array_push($result, $newStatus);
$newStatus = new Status(6);
array_push($result, $newStatus);
$newStatus = new Status(7);
array_push($result, $newStatus);
$newStatus = new Status(8);
array_push($result, $newStatus);
$newStatus = new Status(9);
array_push($result, $newStatus);
```

```
$newStatus = new Status(10);
array_push($result, $newStatus);
$newStatus = new Status(15);
array_push($result, $newStatus);
$newStatus = new Status(16);
array_push($result, $newStatus);
return $result;
}

/*****************************************************************************/

}

?>
```

C.4.2.3 User.php

```
<?php

include "../include/classInclude.php";

class User
{

private $id;
private $name;
private $email;
private $idInModel;
private $myDB;

/*****************************************************************************/

public function setEmail($email)
{
$this->email = $email;
}

/*****************************************************************************/

public function getName()
{
return $this->name;
}

/*****************************************************************************/

public function getEmail()
{
return $this->email;
}

/*****************************************************************************/

public function getIdInModel()
{
return $this->idInModel;
}

/*****************************************************************************/

public function validate()
{
$validationResult = new ValidationResult();

if ($this->email == null)
   $validationResult->addErrorNameAndLabel(0, "email",
         "Correo electrónico", "");
else
   {
   $this->myDB->useDB();
   $this->myDB->connect();
   $query = "SELECT * FROM Users WHERE email = '$this->email'";
```

```
   $this->myDB->query($query);
   if ($this->myDB->numRecords() == 0)
      $validationResult->addErrorNameAndLabel(1, "email",
            "Correo electrónico", "");
   else
      {
      $row = $this->myDB->resultArray();
      $this->id = $row["id"];
      $this->name = $row["name"];
      $this->idInModel = $row["idInModel"];
      }
   $this->myDB->disconnect();
   }
return $validationResult;
}

/*****************************************************************************/

public function User()
{
$this->myDB = new DB_mysql("Form", "localhost", "root", "");
}

/*****************************************************************************/

}
?>
```