Una Aproximación Metodológica al Desarrollo de Flujos de Trabajo

Mª Carmen Penadés Gramaje

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Memoria presentada para optar al Grado de Doctor en Informática

Dirigida por: Prof. Dr. José Hilario Canós Cerdá
Prof. Dr. Gustavo Alonso García

Valencia, Enero de 2002

A Fernando

y

A María

Agradecimientos

Deseo manifestar mi más sincero agradecimiento a cuantas personas han hecho posible con su aportación, de forma directa o indirecta, la realización del presente trabajo.

En primer lugar, a mis directores el Prof. Hilario Canós y el Prof. Gustavo Alonso, quienes han sabido infundirme su constante entusiasmo por el trabajo realizado en esta tesis, al tiempo que han logrado despertar y mantener mi interés. Me gustaría agradecer la confianza que han depositado en mí y que ha supuesto un estímulo continuado durante todo este tiempo.

Al Prof. Isidro Ramos por su apoyo y consejos siempre acertados, tanto en lo académico como en lo personal.

A todos mis compañeros del grupo de investigación "Programación Lógica e Ingeniería del Software" del DSIC, por su amistad y aportaciones al trabajo en las numerosas reuniones que hemos tenido durante todos estos años, especialmente a Juan Sánchez, José A. Carsí y Patricio Letelier.

A los miembros del grupo "Information and Communication Systems" del ETH de Zürich, y en especial a Amaia Lazcano, por su agradable acogida y ayuda prestada durante mi estancia con ellos.

A Mari Paz Segura, Pau Franco e Inma Salesa por su trabajo e inestimable colaboración en las implementaciones.

A mis padres, José y Carmen, y a mis hermanos José y Julián, cuyo apoyo y cariño me ha acompañado siempre durante estos años.

Finalmente, a Fernando, mi marido, por su apoyo incondicional, por animarme siempre a seguir ofreciéndome todo su amor y paciencia. Y a María, mi hija, por regalarme los momentos más agradables con sus risas y juegos, a pesar de las muchas horas que no he podido estar con ella. Gracias a los dos por permitirme dedicar parte de vuestro tiempo a llegar hasta aquí.

Contenidos

PARTE II. DEFINICIÓN Y FORMALIZACIÓN DE UN METAMODELO DE FLUJO DE TRABAJO

PARTE III. UN MODELO DE PROCESO DE DESARROLLO DE FLUJOS DE TRABAJO

Índice de Figuras

Índice de Tablas

PARTE I

Preliminares

Capítulo 1.

Introducción

El trabajo presentado en esta tesis es una aproximación metodológica al desarrollo de Flujos de Trabajo. La principal contribución del trabajo es la definición de un Ciclo de Vida que, integrando en un mismo marco técnicas aplicadas con éxito en la Ingeniería del Software y la Extracción de Conocimiento, tiene como objetivo fundamental el desarrollo de Flujos de Trabajo de calidad. La definición y formalización de un metamodelo de referencia permite establecer un nexo entre las distintas fases o subprocesos de desarrollo. La aproximación que se presenta se basa en la utilización del modelo orientado a objetos, lenguajes formales y generación de código basada en modelos, así como de técnicas de análisis de información almacenada para extracción de conocimiento. El trabajo incorpora una capa de Ingeniería de Requisitos al proceso de definición del flujo de trabajo, que facilita la captura y validación de los requisitos del proceso de negocio. Esto, junto con la formalización del metamodelo de referencia, hacen que aumenten las garantías de generación de código de calidad a partir de los modelos construidos. Sin embargo, pueden existir errores o deficiencias no detectadas hasta que no se produzca la ejecución real del flujo de trabajo. Por lo tanto, se incluye también un proceso de mejora basado en la definición de un modelo de datos multidimensional para análisis de ejecuciones de proceso. Este modelo permite la extracción de conocimiento mediante técnicas de explotación de almacenes de datos y minería de datos.

El resto del capítulo se estructura como sigue. En la sección 1.1 se describe la motivación que ha llevado a la realización de este trabajo. En la sección 1.2 se presentan los objetivos que se persiguen en la tesis, y finalmente en la sección 1.3 se describe la estructura de la memoria, presentando el contenido de cada uno de los capítulos desarrollados.

1.1 Motivación

En los años 70, 80 y parte de los 90, el principal uso de los ordenadores en las organizaciones era la automatización de las actividades individuales. La situación actual es distinta, en los últimos años hay un interés cada vez más creciente por las organizaciones vistas como un todo. Existen métodos y notaciones de análisis y diseño consolidados, como UML [Boo99], herramientas de desarrollo cada vez más potentes (RAD, generadores de código, etc.) y modelos de ciclo de vida basados en el uso de las mismas. Todo ello ha motivado que se esté produciendo un cambio de orientación hacia los procesos organizacionales o procesos de negocio. El interés de las organizaciones no está limitado únicamente al desarrollo de software que automatice ciertas actividades individuales, sino que su objetivo final es la automatización de todo el proceso de negocio, puesto que de ello depende gran parte su competitividad. Surgen, por lo tanto, nuevas necesidades de capturar, modelar, ejecutar y monitorizar los procesos de negocio, vistos como un conjunto de procedimientos o actividades enlazadas, cuya realización permite alcanzar un cierto objetivo o meta en el contexto de una organización [Geo95, Hol95]. En este nuevo escenario, los *flujos de trabajo*[1] y la tecnología asociada ofrecen un marco adecuado para abordar el problema, puesto que cubre, al menos parcialmente, estas necesidades.

Un flujo de trabajo se define en [Hol95] como la automatización total o parcial de un proceso de negocio o, lo que es equivalente, la representación del mismo en un formato entendible por una máquina. En un flujo de trabajo, la información, tareas y documentos pasan de un participante a otro, para que se realicen una serie de acciones de acuerdo con un conjunto de reglas procedimentales; los participantes pueden ser personas o máquinas. Los sistemas que dan soporte a la definición del flujo de trabajo y a su posterior ejecución, se denominan Sistemas de Gestión de Flujos de Trabajo (SGFT).

La comunidad de flujos de trabajo ha detectado hace algún tiempo nuevos retos que hasta el momento no han sido cubiertos adecuadamente por los SGFT [She96]. Algunos de ellos están motivados por los avances en la tecnología, tales como ejecución distribuida de procesos e interoperabilidad. En cambio, otros provienen de aspectos que han sido sistemáticamente olvidados por los desarrolladores de SGFT, tales como modelado y metamodelado de flujos de trabajo, simulación, análisis de ejecuciones de flujos de trabajo, evolución y reutilización.

Sin embargo, en otras áreas tales como Ingeniería del Software e Ingeniería de Requisitos sí se han abordado problemas de modelado, validación e implementación, como parte del desarrollo de aplicaciones en general. En el área de Extracción de Conocimiento, se ha prestado atención a la búsqueda de información de interés a partir del análisis de datos existentes en las organizaciones. Por lo tanto, a pesar de que el modelado y ejecución de un flujo de trabajo, como disciplina emergente, es relativamente nueva, muchas de las ideas y

[1] En la literatura suele denotarse por el término en inglés *Workflow*.

conceptos que se manejan existen desde hace tiempo y provienen de áreas muy diversas. Además de las anteriores podemos también citar las Bases de Datos, la Ingeniería de Procesos, y otros temas ya más concretos como automatización de trabajos, asignación de rutas a documentos, procesos transaccionales, tratamiento de imágenes, sistemas operativos, computación móvil, etc.

Los SGFT surgieron primero en la industria y más tarde se convirtieron en área de investigación (como en el caso de las Bases de Datos Orientadas a Objetos [DeW91]). Esto ha motivado la aparición en el mercado de numerosos productos, catalogados como SGFT, que resuelven problemas concretos pero en los que, en muchas ocasiones, se echa en falta una base científica que permita resolver ciertas limitaciones existentes. Entre ellas podemos citar problemas de escalabilidad, interoperabilidad, robustez, falta de soporte metodológico al modelado y soporte al análisis de registros de ejecución. Todas ellas se detallarán en el siguiente capítulo. Respecto a los productos existentes en el mercado, no todos entran dentro de la misma categoría, ni todos ellos proporcionan las mismas prestaciones a la hora de modelar o ejecutar un flujo de trabajo. Existen diversos tipos de SGFT dependiendo de las características del flujo de trabajo que gestionen y que también se detallan en el siguiente capítulo. Algunos de estos productos son: MQSeries Workflow de IBM [MQS], InConcert [TIB], Staffware [STA], Lotus Notes [LOT], Bizflow 2000 [BIZ], Eastman Enterprise Workflow [EAS], Dolphin [DOL] y SERfloware [SER].

En cuanto a investigación, existen diferentes líneas relacionadas con los SGFT, que abarcan desde temas relacionados tanto con la definición de flujos de trabajo como con su ejecución. Precisamente, es en la ejecución donde se han invertido más esfuerzos. De entre los proyectos de investigación relacionados directamente con el campo de los SGFT destacamos el proyecto OPERA que se desarrolla en el ETH de Zurich [OPE], el proyecto *EXOTICA* de IBM Almaden Research Center y cuyos resultados se están aplicando directamente sobre MQSeries WF[2] [EXO], el proyecto *WIDE* desarrollado conjuntamente por el Politécnico de Milán, la empresa Sema Group, la Universidad de Twente y con participación de ING Bank y el Hospital General de Manresa [WID]. También podemos citar el proyecto *METEOR* de la Universidad de Georgia [MET], el proyecto *MENTOR* de la Universidad de Saaland [MEN] y el proyecto MOBILE de la Universidad de Erlangen-Nürnberg [MOB].

Finalmente, cabe destacar la aparición de una organización internacional denominada *Workflow Management Coalition*[3] (WfMC), cuyo objetivo es aunar esfuerzos para proponer estándares en el campo de los SGFT. Estos estándares van desde proponer un modelo básico para la definición de flujos de trabajo, hasta definir las interfaces de comunicación entre las distintas herramientas que componen el sistema.

[2] Anteriormente existía FlowMark que ahora IBM está sustituyendo por MQSeries WF.
[3] http://www.wfmc.org

1.1.1 Planteamiento del Problema y Solución Propuesta

Las distintas aproximaciones al desarrollo de SGFT han tenido hasta ahora un fuerte componente tecnológico. En la mayoría de los casos, todo el esfuerzo se ha centrado en aspectos de ejecución, dando lugar a sistemas que proporcionan entornos de ejecución eficientes y, en ocasiones distribuidos, con capacidad para ejecutar un gran número de procesos simultáneamente. Sin embargo, en la definición del flujo de trabajo se detectan limitaciones importantes [She96, Alo97]. La mayoría de los SGFT únicamente proporcionan una notación gráfica para el modelado del flujo de trabajo, sin una semántica precisa, siendo en ocasiones muy dependiente de las facilidades de ejecución soportadas. Aspectos de formalización y metodológicos no han sido tenidos en cuenta hasta la fecha más que tímidamente [Cas95, Ley98] y utilizando tecnologías no siempre apropiadas.

Esta cuestión es importante, puesto que la calidad final del flujo de trabajo que se ejecuta depende directamente de la calidad del modelo previamente especificado. Más concretamente, si este modelo no recoge todos los requisitos del proceso de negocio, la ejecución del mismo no será de utilidad para la organización, independientemente de otras buenas propiedades que pueda tener (eficiencia, robustez, distribución, etc.).

La solución que nosotros planteamos al problema descrito es considerar un flujo de trabajo como un software complejo, y por lo tanto, para su desarrollo aplicar técnicas, métodos y herramientas utilizadas en el campo de la Ingeniería del Software. En esta disciplina se han realizado numerosos avances en temas directamente relacionados con el modelado conceptual, puesto que es una de las fases críticas dentro del proceso de desarrollo de software. Los errores cometidos en esta fase son los que mayor incidencia negativa tienen en la calidad final del producto. Entre los trabajos realizados, destacamos varias líneas por ser de interés en la búsqueda de soluciones al problema planteado.

- Definición de métodos para la captura de requisitos [Jac92, Rol96, Boo99, Lei00].
- Uso de lenguajes formales para especificar, dotando a los modelos de una semántica precisa [Fee93, Dub93, Jun95, Pas95b, Obl99].
- Validación de modelos basada en prototipación automática [Har93, Dub94, Sid97].
- Generación de código basada en modelos [Bell98].

Teniendo en cuenta esto, el desarrollo de modelos de flujo de trabajo de calidad se puede abordar con el uso de lenguajes formales para construir las especificaciones correspondientes y obtener automáticamente un prototipo (de acuerdo con las ideas del paradigma de programación automática [Bal83]). Esto permite iniciar un proceso de validación de requisitos con el usuario para mejorar la calidad del modelo a nivel de definición. En concreto se propone el uso de OASIS [Pas95b, Can96b] como marco expresivo y de KAOS [Can95, Can96a] como entorno de prototipado.

Siguiendo con la solución planteada, se puede establecer un símil entre las fases típicas de desarrollo de software en general y el desarrollo de flujos de trabajo. La definición y ejecución de modelos de flujo de trabajo se correspondería siguiendo el paradigma de programación automática [Bal93], con las actividades a realizar desde el análisis a la implementación. Al igual que tras la fase de implementación hay una fase de pruebas que permite detectar errores, la inclusión de una fase de análisis de datos tras la ejecución real del modelo de flujo de trabajo puede aportar información útil que sirva de realimentación para mejorarlo. Esto permite que se incida no sólo en la definición y ejecución de flujos de trabajo, sino también en la mejora de los mismos a partir de datos reales. Para ello se propone el uso de técnicas específicas de extracción de conocimiento.

La solución propuesta al problema planteado determina los objetivos de la tesis. Antes de enumerarlos, hay que indicar que el trabajo se ha desarrollado dentro del grupo de investigación "Programación Lógica e Ingeniería del Software- Subgrupo *Modelado Conceptual Orientado a Objetos y Bases de Datos* (MCOOBD)", del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia, en colaboración con el grupo de investigación "*Information and Communication Systems* (IKS)" del ETH de Zürich. Las áreas de interés del grupo MCOOBD se han centrado históricamente en la definición de modelos y lenguajes de especificación formales y orientados a objetos, y en el desarrollo de entornos de prototipado y sistemas de gestión de bases de datos deductivas activas y orientadas a objetos. Los principales resultados se plasman en la definición del lenguaje de especificación formal y orientado a objetos OASIS [Pas95a, Pas95b, Can96a, Let98] y la construcción de entornos de ejecución o animación de especificaciones OASIS [Can95, Car99, Let99, San00].

Todas las actividades asociadas al desarrollo de este trabajo se han enmarcado en dos proyectos coordinados de la Comisión Interministerial de Ciencia y Tecnología (CICYT): MENHIR - Supbroyecto ESPILL (1997-2000, TIC97-0593-C05-01) y DOLMEN – Subproyecto SIGLO (2000-2003, TIC2000-1673-C06-01).

1.2 Objetivos

El objetivo principal de la tesis es definir un marco (*framework*) para el desarrollo de modelos de flujos de trabajo de calidad, que integre la definición, ejecución y mejora de los mismos. A nivel de definición y ejecución, el objetivo es aplicar métodos, técnicas y herramientas utilizadas con éxito en la Ingeniería del Software a este nuevo dominio. Para la mejora del flujo de trabajo se sigue una aproximación basada en técnicas de análisis de datos y extracción de conocimiento.

Este objetivo general se desglosa en los siguientes objetivos específicos:

- Conocer el estado del arte en el campo de los SGFT, tanto a nivel industrial como de investigación.
- Definir un modelo de proceso de desarrollo de flujos de trabajo que integre la definición, ejecución y mejora de flujo de trabajo, siguiendo una aproximación basada en la Ingeniería del Software.
- Establecer un metamodelo de flujo de trabajo de referencia que sirva de punto de unión de los distintos subprocesos, y formalizarlo en OASIS. Si la expresividad del lenguaje no es suficiente para representar flujos de trabajo, definir las extensiones sintáctico-semánticas necesarias.
- Proponer un método para la captura de requisitos de un proceso de negocio, que guíe la obtención del modelo de flujo de trabajo asociado.
- Proponer la validación automática de los modelos de flujo de trabajo a partir de especificaciones formales de flujos de trabajo.
- Estudiar la generación automática en SGFT eficientes a partir del modelo de flujo de trabajo construido.
- Estudiar y proponer una solución general para la mejora de un flujo de trabajo a partir de la información registrada por el SGFT tras cada ejecución del mismo.
- Construir un prototipo que dé soporte al modelo de proceso propuesto.

1.3 Estructura del Documento

El documento se ha estructurado en cuatro partes, cada una de las cuales consta de un conjunto de capítulos.

La **Parte I** establece las bases para el desarrollo del trabajo. Además de la introducción, comprende los capítulos 2 y 3.

En el **Capítulo 2** se presenta una visión general del estado del arte en los SGFT. Se introduce la terminología básica utilizada en este campo, tanto a nivel de definición como de ejecución de un flujo de trabajo. Se enumeran varias clasificaciones de SGFT, atendiendo a diversos criterios. Se citan las principales limitaciones que existen en estos sistemas, debido principalmente a la falta de un modelo de referencia común. Finalmente, se comentan los esfuerzos que se están realizando para la propuesta de estándares en este campo.

En el **Capítulo 3** se da una visión general del modelo de proceso de desarrollo de flujos de trabajo propuesto. Se describen las principales características de cada uno de los subprocesos que lo componen, enumerándose las actividades a realizar en cada caso y el

conjunto de herramientas necesarias para dar soporte a todas ellas. Finalmente, se presenta el caso de estudio utilizado a lo largo de la tesis.

La **Parte II** contiene la definición y formalización del metamodelo de flujo de trabajo, que servirá de base al proceso de desarrollo propuesto. Comprende los capítulos 4 y 5.

En el **Capítulo 4** se describe detalladamente el metamodelo tomado como referencia en la tesis. Se identifican y modelan los distintos elementos que componen un flujo de trabajo, siguiendo un enfoque orientado a objetos y utilizando UML como lenguaje de modelado.

En el **Capítulo 5** se presenta la formalización del metamodelo de referencia en OASIS. Antes de abordar la representación en OASIS de los conceptos del metamodelo, se resumen las principales características del marco conceptual proporcionado por OASIS, en particular, aquéllas que permiten utilizarlo como lenguaje de especificación de flujos de trabajo.

La **Parte III** da una descripción más detallada del modelo de proceso de desarrollo propuesto. Comprende los capítulos 6, 7 y 8, dedicados a la definición, ejecución y mejora de flujos de trabajo, respectivamente.

En el **Capítulo 6** se describe el subproceso de construcción y estabilización, que comprende la definición y validación de modelos de flujo de trabajo. Respecto a la definición del modelo, se presenta el lenguaje gráfico propuesto, los servicios genéricos que debe proporcionar un editor de modelos de flujo de trabajo y las pautas de generación de la especificación OASIS equivalente al modelo construido. Todo ello de acuerdo con el metamodelo de referencia y su formalización. La especificación obtenida es la entrada para iniciar un proceso de validación de la misma. Por último, se presenta un método de derivación de modelos de flujo de trabajo, siguiendo una aproximación ascendente basada en casos de uso del negocio, que sirve de ayuda en la fase de definición del modelo.

En el **Capítulo 7** se estudia la ejecución de los modelos de flujo de trabajo construidos en SGFT, siguiendo una aproximación basada en compiladores de modelos. En concreto, se establecen las correspondencias con el sistema OPERA, enumerándose las distintas plantillas de generación a seguir por el compilador de modelos.

En el **Capítulo 8** se describe el análisis de ejecuciones de flujos de trabajo, desde la perspectiva de las técnicas de extracción de conocimiento a partir de información almacenada. Se resumen los principales conceptos y técnicas utilizadas en este campo, y que son de utilidad en nuestro caso. Se enumeran los elementos de interés a analizar para la mejora del flujo de trabajo y se propone un modelo multidimensional que permite abordar el análisis de forma general. Finalmente se muestra cómo se integra la solución propuesta en el marco general del proceso de desarrollo de flujos de trabajo, y su aplicación al sistema OPERA.

En el **Capítulo 9** se presenta el prototipo desarrollado, describiéndose el conjunto de herramientas que implementan lo descrito en los capítulos anteriores.

La **Parte IV** está dedicada a las conclusiones generales de la tesis. Comprende únicamente el **Capítulo 10** en el cual se presentan las principales contribuciones y se describen futuras líneas de investigación que quedan abiertas y que pueden abordarse a partir del presente trabajo.

Finalmente, como complemento a los capítulos, se incluyen dos apéndices. El **Apéndice A** contiene la especificación OASIS completa del metamodelo de referencia definido. El **Apéndice B** recopila y documenta los servicios proporcionados por el editor de flujos de trabajo para la definición de modelos.

Capítulo 2.

Sistemas de Gestión de Flujos de Trabajo. Estado del Arte

En este capítulo se introduce una visión general al contexto tecnológico en el que se sitúa el trabajo desarrollado en esta tesis, los SGFT. Se hace una revisión del estado del arte, que abarca su definición, clasificación, limitaciones y esfuerzos actuales de estandarización.

El capítulo se estructura como sigue. En la sección 2.1 se introducen los conceptos básicos relacionados con los SGFT, enumerándose la terminología asociada. En la sección 2.2 se presentan diversas taxonomías de los SGFT, atendiendo a diferentes criterios. La sección 2.3 recopila las principales limitaciones de los productos actuales. En la sección 2.4 se muestran los esfuerzos realizados a nivel de estandarización, con la descripción de los distintos estándares propuestos por al WfMC, en especial, el Modelo de Referencia. Finalmente, se muestran las conclusiones en la sección 2.5.

2.1 Introducción

Un SGFT es "un sistema que define, crea y gestiona la ejecución de flujos de trabajo mediante el uso de software, siendo capaz de interpretar la definición del proceso, interactuar con los participantes y, siempre que se requiera, invocar el uso de herramientas y aplicaciones" [WFM99a].

De acuerdo con la definición anterior, en un SGFT existen dos actividades claramente diferenciadas, aunque con relaciones entre ellas. Por una parte está la definición del flujo de trabajo que implementa al proceso de negocio, en lo que llamaremos modelado del flujo de

trabajo, y por otra parte está la animación o ejecución de dicho modelo, también conocido en la literatura con el término inglés *enactment*. La Figura 2-1 muestra los conceptos básicos y la terminología asociada tanto para la fase de modelado como para la fase de ejecución.

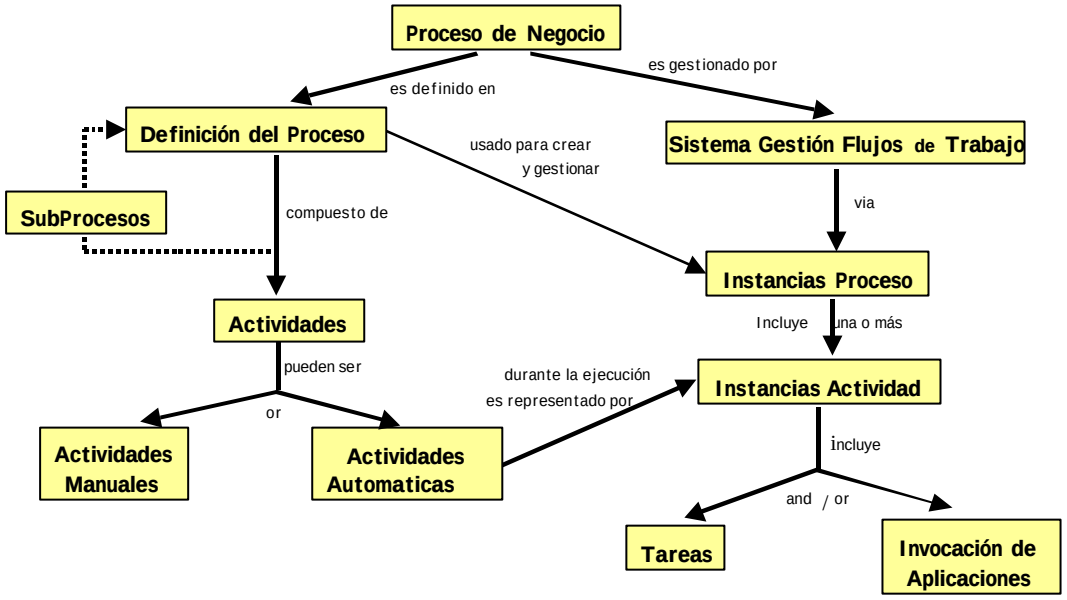


Figura 2-1. Conceptos básicos y terminología (fuente WfMC)

La especificación del flujo de trabajo consiste en definirlo como un conjunto de actividades relacionadas, más un conjunto de criterios que indican el comienzo y finalización del proceso y de sus componentes, e información adicional sobre cada actividad, tal como participantes, invocación de aplicaciones, datos, etc. Las actividades describen trozos de trabajo y constituyen pasos o tareas dentro del proceso. Éstas pueden ser manuales o automáticas, en el sentido de que puedan requerir o no recursos humanos para su realización, así como invocar aplicaciones externas que realicen parte o todo el trabajo asignado a dicha actividad.

La ejecución del flujo de trabajo en un SGFT consiste en la creación, manipulación y destrucción de instancias proceso, que representen al flujo de trabajo de acuerdo con la especificación previa. Cada instancia proceso tendrá una identidad visible externamente y un estado interno que representa su progreso hacia la finalización y su estado con respecto a las actividades que lo componen. Cada vez que la ejecución del proceso suponga la invocación de una actividad, según la definición del mismo, se crea una instancia actividad que la representa. Dicha instancia se encarga de ejecutar las acciones asignadas, accediendo a los datos que sean necesarios e invocando la aplicación externa correspondiente, si así lo requiere la actividad.

Una vez presentados los conceptos básicos relacionados con los SGFT, si profundizamos un poco más vemos que además del *proceso* o flujo de trabajo como componente más importante, existen otros dos aspectos que se pueden considerar. En primer lugar, puesto que algunas actividades pueden requerir participación humana, se puede incluir no sólo la información de quién realiza dicha tarea, sino también modelar toda la estructura organizativa del personal. En segundo lugar, y de forma similar a lo anterior, la utilización de datos por parte de las actividades y la posibilidad de invocar aplicaciones externas obliga, en cierto modo, a conocer y modelar la propia infraestructura de tecnologías de la información existente en la organización.

Los SGFT pueden ser implementados de una gran variedad de formas, como resultado de la utilización de diferentes tecnologías en diferentes entornos, que pueden ir desde un pequeño grupo de trabajo a una gran organización. En cuanto a la tecnología utilizada, son muchas las que pueden confluir en un SGFT. Destacamos gestión de bases de datos, interfaces gráficas de usuario, integración de sistemas (nuevos con los ya existentes), mensajes, gestión de documentos, cliente / servidor y distribución.

Además, en los SGFT confluyen diferentes áreas, de entre las cuales podemos mencionar como más significativas las siguientes:

- **Procesos**: Los SGFT intentan automatizar la coordinación de todos los elementos que componen un proceso, indicando qué hay que hacer, en qué orden, quién debe hacerlo y qué recursos se deben utilizar.
- **Reingeniería**: Entendiendo la reingeniería de procesos como una forma de definir, analizar y optimizar procesos empresariales, los SGFT también juegan un papel importante en dicho campo, mediante el análisis de la propia ejecución del flujo de trabajo.
- **Islas de Información**: Se llaman así a las redes o subredes de ordenadores existentes en una organización, que quedan aisladas unas de otras sin ofrecer una visión global. Los SGFT pueden servir para conectar estas islas entre sí utilizando la noción de proceso como forma de enlazar la información y proporcionar una visión global de la misma.
- **Sistemas basados en componentes:** En los SGFT también confluyen ideas de los sistemas basados en componentes, teniendo en cuenta que cada vez más, el desarrollo de aplicaciones se basa en combinar componentes que ya existen, siendo la unificación de dichos componentes el principal problema. En este sentido, los SGFT son un componente más, donde a la vez podemos utilizar la visión global que proporcionan como una posible forma de programación de grandes sistemas. A esta aproximación también se le denomina *megaprogramación* o *metaprogramación* [CHA].

Otras áreas que también abarcan los SGFT son: procesamiento de imágenes, gestión de documentos, aplicaciones colaborativas (*groupware*), correo electrónico y software de soporte para gestión de proyectos.

2.2 Taxonomía

Los elementos a considerar en la especificación de un flujo de trabajo son muchos y de naturaleza muy variada, como ya se ha comentado anteriormente. Por ello, no existe una única clasificación de los SGFT, sino varias dependiendo del criterio a considerar. A continuación se enumeran algunas de las clasificaciones que podemos encontrar en la literatura:

A. ([Geo95]) Según la complejidad del grafo que representa al flujo de trabajo, es decir, de los requisitos de coordinación entre las distintas actividades, tenemos dos tipos de flujos de trabajo:

- ***Ligeramente estructurado.*** En este caso, el grafo que representa la coordinación entre las actividades es prácticamente *lineal*, y éstas se van ejecutando una a continuación de otra.
- ***Altamente estructurado.*** El grafo que representa la coordinación entre las actividades ya no es lineal, sino que se representan ejecuciones en paralelo de actividades, sincronización de actividades, etc.

B. ([Geo95]) Según el grado de participación humana en el flujo de trabajo, éstos se pueden clasificar en:

- ***Orientados a personas.*** Cuando la participación humana en la ejecución del SGFT es importante. Estas personas serán agentes que inician o realizan las actividades. Por su naturaleza, la mayor parte de las actividades que conforman el flujo de trabajo son manuales.
- ***Orientados a sistemas.*** La participación humana suele ser menor y el proceso está altamente automatizado, por lo que la mayor parte de las actividades del flujo de trabajo son actividades automáticas.

C. De acuerdo con la tecnología en la que se basa el SGFT, éstos se pueden clasificar en:

- ***Centrados en el correo electrónico.*** En estos SGFT se utiliza el *e_mail* como medio de comunicación.
- ***Centrados en el documento.*** La principal característica es que los documentos circulan e interaccionan con aplicaciones externas. Predominan los aspectos de gestión de documentos.
- ***Centrados en el proceso.*** En este caso, lo importante es el propio proceso. Los mecanismos de comunicación entre las actividades y/o los datos se implementan por encima de la base de datos, proporcionándose interfaces de interacción.

D. Una clasificación de SGFT bastante aceptada hoy en día, es aquella que distingue entre SGFT de *producción*, *administrativos, ad-hoc* y de *colaboración*. Esta clasificación se establece en base a dos criterios: la complejidad de las actividades involucradas y la estructura de las mismas, por una parte, y por otra, en base a las similitudes de los procesos de negocio involucrados y su valor o importancia en la propia organización. La Figura 2-2 y la Figura 2-3 muestran cómo se sitúa cada tipo de SGFT de acuerdo a los criterios establecidos.

- ***Administrativos.*** Este tipo de SGFT es el que modela los procesos burocráticos de una organización. La función básica asociada al mismo es el *procesamiento de formularios*. El SGFT se encarga de activar la ejecución de las actividades (la mayor parte de las cuales son manuales e interviene un agente en ellas), recoger las respuestas (datos) y obtener el formulario (que normalmente es una o varias actividades automáticas, que recopilan y procesan la información obtenida). Por ejemplo, la matricula universitaria o la obtención de certificados.

  La principal característica de las actividades involucradas en el proceso es que se trata de actividades repetitivas y de baja complejidad. La estructura del grafo que representa el proceso puede varíar de baja a alta, aunque en muchas ocasiones el grafo de tareas es lineal.

  Algunos productos existentes en el mercado y que pertenecen a esta categoría son: Staffware [STA], Bizflow 2000 [BIZ] y COSA Workflow [COS].

- ***Ad Hoc.*** Estos SGFT son muy similares a los administrativos, pero con la característica de ser utilizados para tratar situaciones únicas o excepcionales, en lugar de procesos burocráticos perfectamente establecidos. El SGFT informa sobre el estado de ejecución de cada actividad y la participación humana es esencial.

  Las actividades involucradas en el proceso suelen ser únicas, en el sentido de que muchas de ellas sólo se realizan una vez y su complejidad oscila de baja a media. La principal dificultad en este tipo de SGFT estriba en construir el grafo de coordinación y cooperación entre actividades.

  Un ejemplo de este tipo es el proceso para la aceptación de artículos en una revista desde el punto de vista del autor. Cada revista tiene su propio proceso a seguir, y en este caso, seguir la pista por parte del autor de cada proceso de aceptación por separado no es complicado, pero sí en caso de estar todos juntos y ejecutándose simultáneamente (envío simultáneo a varias revistas).

  Podemos citar como SGFT ad-hoc a Dolphin [DOL] y a Panagon Workflow [FIL].

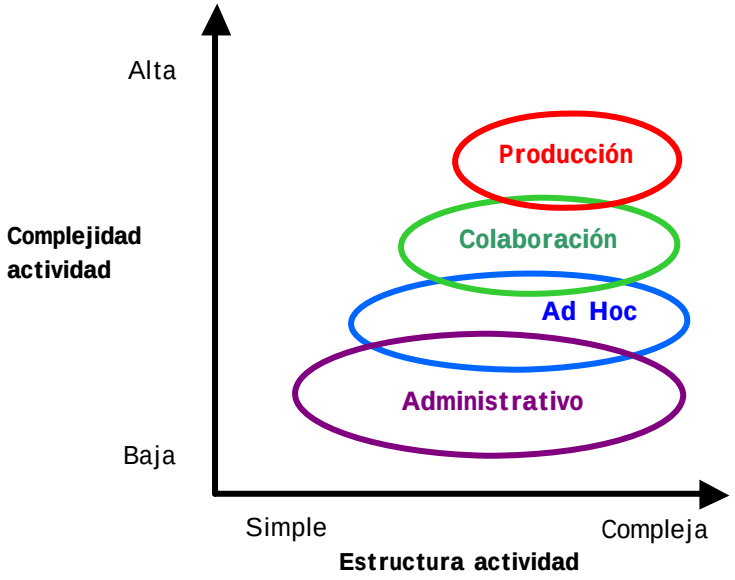


Figura 2-2. Clasificación de los SGFT según complejidad y estructura de las actividades involucradas

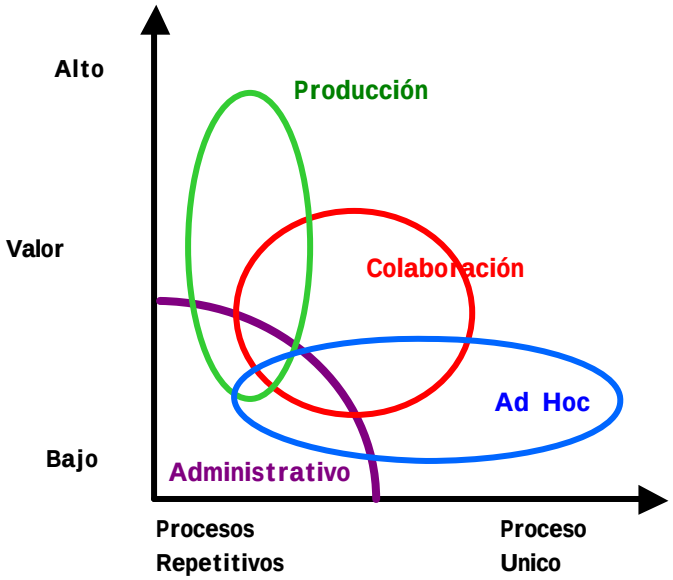


Figura 2-3. Clasificación de los SGFT según la similitud de los procesos de negocio y su valor en la organización

- ***Producción.*** Modelan e implementan los procesos de negocio críticos de la organización, es decir, los procesos directamente relacionados con la función principal de la misma.

  Los SGFT de este tipo son sistemas complejos que se ejecutan sobre entornos heterogéneos, y en los que suelen participar como agentes gran variedad de personas y organizaciones. Normalmente se requiere la ejecución de transacciones para el acceso a la información, que puede encontrarse en diferentes sistemas. Finalmente, cabe destacar que la monitorización del estado en el que se encuentra cada actividad es muy importante en este tipo de sistemas, así como el posterior análisis estadístico de toda esta información, que va a permitir, entre otras cosas, mejorar el proceso.

  Las actividades involucradas en el proceso son repetitivas (cientos o miles de instancias en ejecución), pero de alta complejidad. La estructura del grafo que representa el proceso suele ser más bien compleja.

  Como ejemplos podemos citar la integración de software preexistente (*legacy applications)*, los préstamos bancarios, préstamos y devoluciones en general, seguros, etc.

  Los siguientes productos existentes en el mercado pertenecen a este tipo: MQSeries WF [MQS], Eastman Enterprise Workflow [EAS], Visual and Panagon Workflow (es la evolución de FileNet) [FIL], InConcert [TIB], y SERfloware [SER].

- ***Colaboración.*** Este tipo de SGFT se caracteriza principalmente por la participación de distintas personas y las interacciones que tienen lugar entre ellas, de tal forma que la mayor parte de la coordinación en realidad la realiza el hombre. Otra característica importante es la existencia de ciertas actividades sobre las que se pueden realizar varias iteraciones hasta que se alcance un cierto consenso por parte de todos los participantes; una vez alcanzado, la ejecución continúa siempre hacia adelante. Son sistemas muy dinámicos que en muchas ocasiones se definen conforme se avanza en el proceso.

  Las características enunciadas anteriormente hacen que no exista un consenso total sobre si realmente se trata o no de SGFT, especialmente cuando las características anteriores se llevan al límite, ya que la mayor parte de la coordinación la realiza el hombre y el sistema se limita a proporcionar un buen interfaz para las interacciones (normalmente vía correo electrónico).

  Algunos de los productos existentes en el mercado son: Lotus Notes [LOT] y TeamWare Flow [TEM].

2.3 Limitaciones

Existen en el mercado numerosos productos comerciales catalogados como SGFT. No todos ellos reúnen las mismas características, sino que muchas veces se centran en proporcionar funcionalidades específicas, que encajan en alguna de las clasificaciones vistas en la sección anterior. Además cada uno de estos productos han evolucionado incorporando nuevas tecnologías y convirtiéndose muchas veces en productos para aplicaciones muy específicas.

En cuanto a la investigación, actualmente existen numerosos grupos trabajando en distintas líneas relacionadas con los SGFT. Pero, como se mencionó anteriormente, muchos productos comerciales han aparecido antes de que la investigación en ese área haya llegado a fundamentos teóricos serios, obteniéndose soluciones tecnológicas ad-hoc sin una base formal que las sustente.

Esta situación condiciona que el estado del arte venga marcado por los productos comerciales existentes en el mercado, más que por los resultados de los grupos de investigación, que trabajan intentando subsanar las limitaciones actuales de los SGFT.

Las principales limitaciones de los SGFT se pueden resumir en las siguientes[4]:

- Los SGFT existentes no son totalmente compatibles, haciéndose prácticamente imposible unir diferentes sistemas que proporcionan diferentes funcionalidades en uno único. Las incompatibilidades existentes no sólo tienen que ver con la sintaxis o la plataforma en la que se ejecutan, sino que muchas veces dichas incompatibilidades se refieren al propio modelo de ejecución del flujo de trabajo. Esta situación provoca que los diferentes productos existentes se consideren como "islas" de la automatización de procesos.
- En muchas ocasiones, estos productos fueron diseñados para pequeños grupos de usuarios. Cuando estos productos se han querido utilizar a gran escala, han aparecido numerosas restricciones derivadas del propio diseño del producto, tales como pobre soporte a la comunicación, una única base de datos donde se almacena toda la información, etc. Estas restricciones obligan en numerosas ocasiones a rediseñar por completo el SGFT.
- La falta de robustez de muchos productos también es una limitación importante. Ante un fallo, no siempre aseguran una recuperación correcta del sistema, sino que muchas veces queda en manos de los propios mecanismos de recuperación de la base de datos. En numerosas ocasiones, el problema se debe al igual que antes, al hecho de que se diseñaron para utilizarse a más pequeña escala, sin tenerse en cuenta qué ocurriría cuando hubieran numerosos componentes (usuarios, aplicaciones y datos) involucrados en la ejecución de un proceso y estos componentes estuvieran distribuidos por la red.

[4] Una descripción más detallada aparece en [Alo97]

- No existe hasta el momento una formalización de la noción de flujo de trabajo, sino que en la mayoría de las ocasiones simplemente se proporcionan descripciones informales de cuáles son sus componentes.
- No existe soporte metodológico para los procesos de modelado de flujos de trabajo, ni tampoco para el análisis de los mismos [She96].

La mayor parte de las limitaciones mencionadas se deben a la falta de un Modelo de Referencia común para todos los SGFT, que sea aceptado como un estándar por parte de todos los desarrolladores, e incorporado en dichos sistemas. Con el objetivo de subsanar esta deficiencia, nace la *Workflow Management Coalition* (WfMC[5]), cuyos principales resultados se comentan a continuación.

2.4 Estandarización

Los principales esfuerzos en el desarrollo y promoción de estándares en el campo de los SGFT han sido realizados por la WfMC. Es una organización internacional, sin ánimo de lucro, que se establece en agosto de 1993 y reúne a un grupo de vendedores, usuarios, analistas y grupos de investigación. Su principal objetivo es promover el uso de SGFT mediante el establecimiento de estándares que faciliten la creación, desarrollo y análisis de estos sistemas.

La WfMC ha definido una serie de estándares (ver Figura 2-4). De entre todos ellos, destacamos el *Modelo de Referencia* [Hol95] que describe la arquitectura básica de un SGFT y que junto con el de *Terminología y Glosario* [WFM99a], constituyen el material básico. En base a ellos se han definido otras especificaciones con una finalidad mucho más concreta, como son: interoperabilidad entre SGFT [WFM99c], modelo para la definición de procesos [WFM99b], invocación a través de API (*Aplication Programming Interface)* [WFM98a] y análisis de datos [WFM98b]. En las siguentes secciones se comentan más extensamente el modelo de referencia propuesto y se darán las principales características del resto.

Posteriormente, *Object Management Group*[6] (OMG) también une sus esfuerzos con la WfMC en la propuesta de estándares. OMG es una organización internacional sin ánimo de lucro fundada en 1989, cuyo principal objetivo se centra promover la teoría y práctica de la tecnología orientada a objetos en el desarrollo de software. Entre sus principales propuestas está la arquitectura OMA (*Object Management Architecture*), que establece el marco conceptual en el que se basan todas sus especificaciones, y CORBA (*Common Object Request Broker Architecture)* como respuesta a las necesidades de interoperabilidad entre el gran número de productos software y hardware existentes hoy en día, permitiéndose la

[5] http://www.wfmc.org/
[6] http://www.omg.org/

comunicación entre ellos. En 1997 se inicia un acercamiento hacia los SGFT, publicando en [OMG97] una propuesta para incluir funcionalidad de ejecución de los SGFT en entornos desarrollados siguiendo una arquitectura OMA, lo cual mejoraría las prestaciones de dichos entornos.

Este acercamiento de OMG conlleva la publicación de una primera especificación, basada en los estándares de la WfMC, que establece requisitos de interoperabilidad entre distintos SGFT en un entorno CORBA [OMG98]. A partir de ese momento, se inicia un proceso de revisión y mejora de esta especificación, siendo [OMG00] la última propuesta.

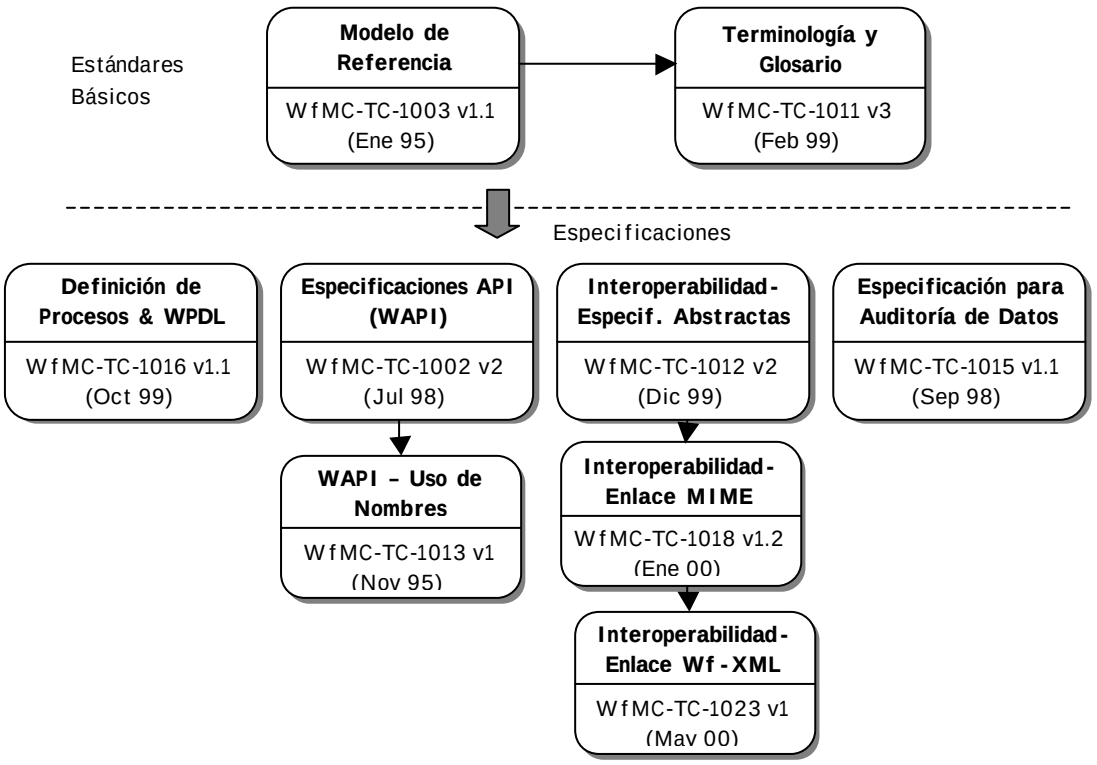


Figura 2-4. Estándares de la WfMC

2.4.1 Modelo de Referencia de la WfMC

El *Modelo de Referencia* propuesto por la WfMC intenta reunir las características comunes de cualquier producto para la gestión de flujos de trabajo, de manera que sea posible la interoperabilidad entre ellos, a través de estándares comunes para cada una de las funciones

que se puedan realizar. En primer lugar, se han identificado las distintas áreas funcionales y a continuación se han desarrollado especificaciones para la implementación de las mismas, asegurándose la interoperabilidad entre distintos SGFT y su integración con otras aplicaciones informáticas.

Este modelo ha sido desarrollado a partir de la *arquitectura de una aplicación de sistema de flujo de trabajo genérica*, identificando sus **componentes** y las diferentes **interfaces** que permiten comunicación a diferentes niveles (Figura 2-5).

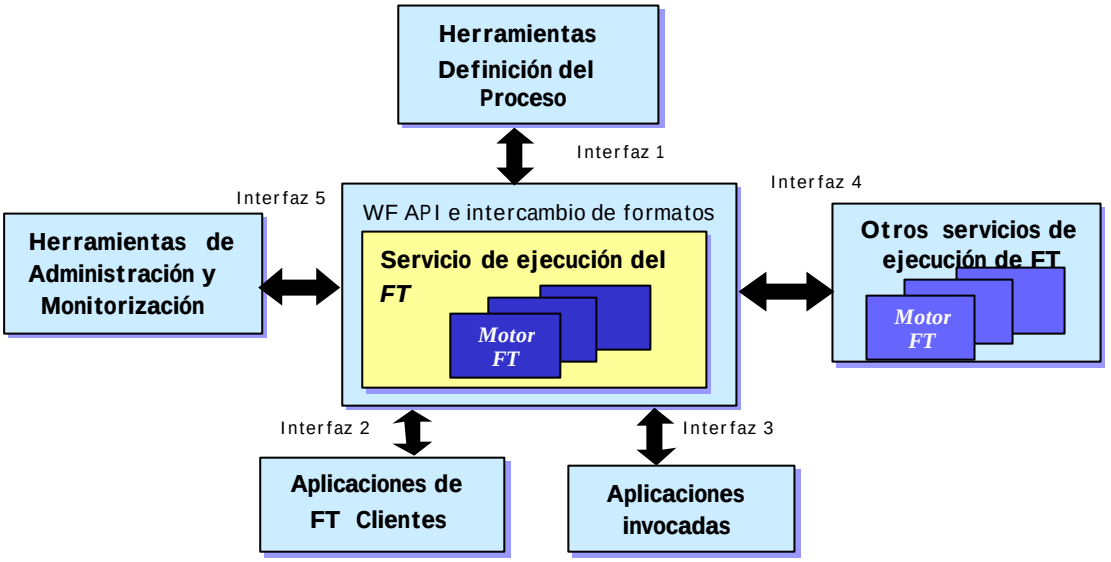


Figura 2-5. El modelo de referencia de un SGFT (fuente WfMC)

El componente central es el ***servicio de ejecución del flujo de trabajo*** que se encarga de crear, gestionar y ejecutar cada una de las instancias del modelo de flujo de trabajo. En este componente es donde se encuentra el motor del SGFT, que proporciona la ejecución, propiamente dicha, de cada instancia. En caso de estar en un entorno de ejecución de flujo de trabajo distribuido, pueden existir diferentes motores de flujo de trabajo que controlen distintas partes de la ejecución del proceso. La comunicación de este componente con el resto se realiza a través de lo que la WfMC denomina *WAPI* (*Workflow APIs*), es decir, la interfaz para la programación de aplicaciones de flujo de trabajo [WFM98a].

Otro de los componentes son ***las herramientas de definición del proceso*** o flujo de trabajo, las cuales permiten modelar, describir y documentar un determinado flujo de trabajo, proceso de negocio o proceso en general. Estas herramientas pueden ser totalmente informales (lenguaje natural, lápiz y papel) o mucho más sofisticadas y formalizadas (interfaces gráficas con un modelo subyacente bien definido). Se debe especificar la lógica del proceso, las actividades que lo componen, los participantes humanos, aplicaciones

invocadas, datos utilizados, etc. La ***interfaz 1*** permite la comunicación entre este componente y el servicio de ejecución del flujo de trabajo.

La distinción de ambos componentes aporta claros beneficios, entre los que destaca la separación entre el entorno de ejecución y el de definición. De este modo, es posible almacenar en un repositorio la información sobre la definición del proceso y ésta ser accedida por distintos entornos de ejecución para ejecutarlo completamente o de forma distribuida. La interfaz 1 se encarga por tanto del intercambio de información entre el componente que permite la definición del proceso y el propio servicio de ejecución del flujo de trabajo. Esta interfaz hace necesaria la definición de un metamodelo básico, en el que se identifica el conjunto mínimo de entidades para la definición de un proceso, permitiéndo el intercambio de información entre ambos componentes (ver sección 2.4.1.1).

Además del propio servicio de ejecución del flujo de trabajo y las herramientas para la definición del proceso, tenemos otro componente denominado ***aplicaciones clientes del flujo de trabajo,*** que representa las entidades software utilizadas por el usuario final en aquellas actividades que requieren participación humana para su realización. Si este componente se separa de lo que es el propio componente de ejecución, es necesaria una interfaz (**interfaz 2**) que defina y maneje claramente el concepto de lista de trabajos o *worklist*, como una cola de trabajo asignado a un usuario o a un grupo de usuarios por el propio motor de ejecución del flujo de trabajo.

El componente ***aplicaciones invocadas*** representa software o aplicaciones ya existentes que un SGFT puede utilizar para la realización de ciertas actividades, teniendo en cuenta que, en principio, dichas aplicaciones se pueden encontrar en cualquier plataforma o lugar de la red. La **interfaz 3** permite la comunicación entre este componente y el servicio de ejecución del flujo de trabajo, no sólo a nivel de invocación del mismo, sino de transformación de datos en formatos entendibles por ambos componentes. Una posible solución se obtiene a través de lo que se denomina *agente aplicación*, de modo que el servicio de ejecución del flujo de trabajo se comunica con las funciones estándar de dicho agente aplicación y éste define interfaces específicas para cada tipo de aplicación invocada.

La interoperabilidad entre SGFT está representada por el componente denominado ***otros servicios de ejecución de flujo de trabajo***, siendo la **interfaz 4** la que permite dicha comunicación. En este caso, la WfMC ha desarrollado un conjunto de escenarios de interoperabilidad que van desde la conexión a nivel de actividades simples hasta todo un completo intercambio de definición de procesos y datos [WFM99c].

Finalmente, el componente ***herramientas de administración y monitorización*** permite que distintos servicios de ejecución de flujo de trabajo compartan las mismas funciones de administración y monitorización del sistema, como pueden ser, por ejemplo, la gestión de usuarios, el control de los recursos y la supervisión del estado de todo el proceso.

2.4.1.1 Metamodelo de Flujo de Trabajo

La Figura 2-6 muestra una representación gráfica del metamodelo de flujo de trabajo de la WfMC, en notación UML. Como información adicional, hay que mencionar que un flujo de trabajo tiene como atributos un nombre, un número de versión, una condición de inicio y una condición de terminación, así como datos sobre seguridad, auditoría o cualquier otro tipo de control. Una actividad posee nombre, un tipo que indica si es una actividad atómica o una subactividad, una precondición, una postcondición y otras restricciones de planificación. Las condiciones de transición forman parte del flujo de control. Los datos del flujo de trabajo se caracterizan mediante un nombre, un camino (*path*) y un tipo. Los roles que representan la participación humana tienen un nombre y la unidad organizacional a la que pertenecen, además de una lista de trabajo que almacena las tareas a desarrollar por el actor humano. Finalmente, las aplicaciones invocadas poseen un nombre o tipo genérico, unos parámetros de ejecución y la ruta de acceso a la misma. Estas entidades están relacionadas entre sí y dichas relaciones están claramente etiquetadas en la Figura 2-6.

Este metamodelo básico puede enriquecerse con nuevas entidades y con más información sobre las mismas, de acuerdo con la funcionalidad añadida. Por ejemplo, en el caso de que el servicio de ejecución del flujo de trabajo sea distribuido, puede ser necesario incluir, como un atributo más de la actividad, la máquina o motor de flujo de trabajo que la va a ejecutar y su localización.

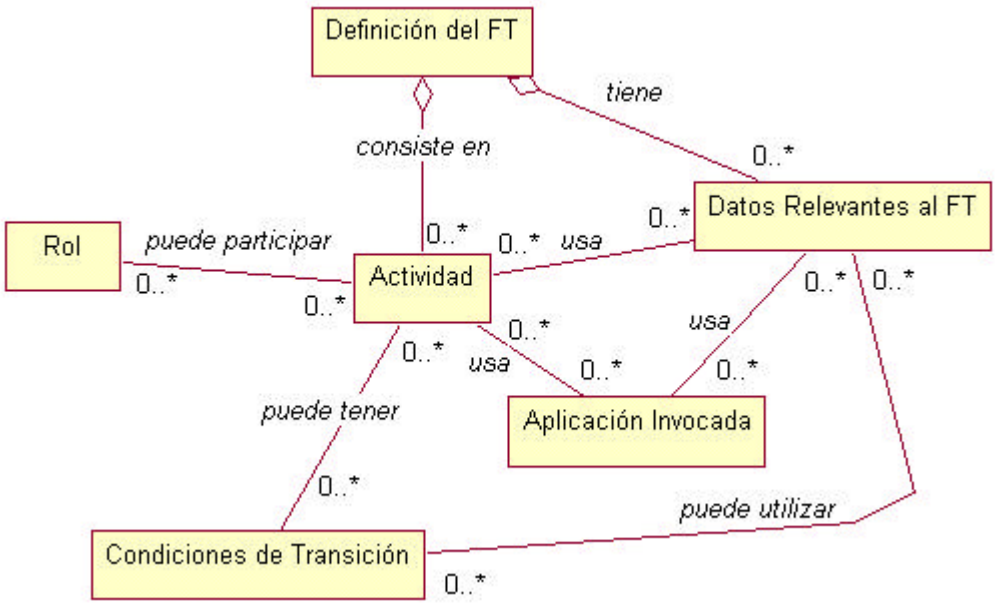


Figura 2-6. Metamodelo básico para la definición de un flujo de trabajo (fuente WfMC)

2.4.2 Otros Estándares

En esta sección se comenta brevemente el contenido de los distintos estándares de la WfMC. Estos estándares están en continuo proceso de revisión y actualización, citándose como referencia la última versión de los mismos.

- ✓ **WfMC-TC-1011. Terminología y glosario** [WFM99a]. El objetivo de este documento es definir los términos utilizados en las especificaciones de la WfMC, con la finalidad de establecer una consistencia en el uso de los mismos, tanto en la industria como en el mundo de los SGFT en general. En el documento se identifica la terminología usada para describir los conceptos y la estructura general de un SGFT, con sus principales componentes e interfaces. Para cada término se da una definición, posibles usos de los mismos y otros sinónimos.

- ✓ **WfMC-TC-1002. Workflow Client API Specifications (WAPI)** [WFM98a]. Este documento especifica la interfaz de comunicación del motor de flujo de trabajo con el resto de componentes que lo forman. Esta comunicación se realiza a través de APIs[7], definiéndose un conjunto de servicios para la programación de aplicaciones de flujo de trabajo, denominada WAPI. Este documento se complementa con un conjunto de convenciones para utilización de nombres en las WAPI [WFM95].

- ✓ **WfMC-TC-1012. Interoperabilidad entre SGFT** [WFM99c]. Este documento proporciona una especificación abstracta que define la funcionalidad necesaria para permitir un nivel definido de interoperabilidad entre dos o más motores de distintos SGFT. En el documento se presentan inicialmente distintos niveles de interoperabilidad y para cada uno de ellos un posible modelo que permita dicha integración. Finalmente se especifican los operadores para llevar a cabo distintas interacciones entre los SGFT. Este estándar de interoperabilidad se enriquece con otros documentos de la WfMC, tales como:

    - o El estándar de interoperabilidad "*Internet e-mail MIME Binding*" [WfM00a], donde se definen los tipos y formatos de mensajes concretos, de acuerdo con la especificación abstracta dada en el documento inicial, para poder permitir la interoperabilidad entre SGFT utilizando el correo electrónico en internet como medio de comunicación y la codificación MIME como mecanismo de transporte de información.

    - o Interoperabilidad utilizando XML 1.0 [W3C98]. Se trata de un borrador en el que se está trabajando y que aún no ha sido aprobado como estándar. Se denomina " Wf-XML Binding" [WFM00b], y su objetivo es utilizar XML como lenguaje de intercambio de mensajes entre SGFT. Cada mensaje es un

[7] Application Programming Interfaces.

documento XML que sigue un DTD prefijado. Se está trabajando en estos mensajes agrupados en tres grupos: definición del proceso, ejecución del mismo e interacción con el usuario.

- ✓ **WfMC-TC-1015. Análisis de datos** [WFM98b]. Este documento define cómo se representa la información a utilizar para la realización de auditorías de SGFT. En concreto, el estándar especifica la información que debe ser capturada y almacenada referente a los diversos eventos ocurridos durante la etapa de ejecución del flujo de trabajo. A dicha información se le denomina CWAD y se utilizará tanto para realizar análisis como para realizar pruebas de ejecución.

- ✓ **WfMC-TC-1016. Definición de Procesos & WPDL**[8] [WFM99b]. Este documento forma parte de la estandarización de la interfaz 1 del modelo de referencia definido por la WfMC. Se incluye un metamodelo para describir la definición de procesos y una propuesta de lenguaje textual, con la gramática asociada, para la definición de procesos denominado WPDL.
  - o Posteriormente, y debido al auge de XML [W3C98] se ha propuesto un lenguaje para la definición de procesos basado en XML, denominado XPDL. En [WFM01] se detalla una primera propuesta de DTD para el intercambio de definiciones de procesos.

2.5 Conclusiones

En este capítulo se han presentado las principales características de los SGFT, desde el punto de vista de su funcionalidad, detectándose que existen dos actividades básicas: definición y ejecución. La taxonomía presentada demuestra que existe gran variedad dentro de los SGFT, y que a pesar de que todos ellos permiten la automatización de procesos, siguen existiendo limitaciones importantes, relacionadas principalmente con la falta de un soporte metodológico y la falta de estándares que sean aceptados e implementados por los sistemas comerciales.

Precisamente, con éste último objetivo nace la WfMC, una organización internacional que está trabajando junto con OMG para la definición de estándares y su aceptación. De entre todos ellos, destaca el Modelo de Referencia, que establece un marco común a partir del cual distintos SGFT pueden interactuar. Esto determina la aparición de toda una familia de estándares para aspectos mucho más concretos y relacionados principalmente con temas de interoperabilidad. En los últimos años han ido apareciendo también propuestas relativas a la definición de flujos de trabajo, pero principalmente relacionadas con el establecimiento

[8] *Workflow Process Definition Language*

de formatos de intercambio de modelos de flujo de trabajo, para su exportación a otros SGFT, y no a nivel metodológico para la obtención y validación de dicho modelo.

Capítulo 3.

Visión General del Proceso de Desarrollo de Flujos de Trabajo

Un flujo de trabajo puede considerarse un producto software complejo que permite automatizar un cierto proceso. Según la Ingeniería del Software, en el desarrollo de todo producto software existen una serie de actividades que deben realizarse en un orden determinado y que abarcan no sólo su producción, sino también su explotación y mantenimiento. Esto se denomina *ciclo de vida* o *proceso de desarrollo del software* [Pre01]. Por lo tanto, y de acuerdo con esta idea, el punto de partida de la tesis es definir un modelo de proceso que proponga el uso de técnicas, métodos y herramientas en el desarrollo de flujos de trabajo de calidad, eliminándose algunas de las limitaciones existentes actualmente.

La estructura del capítulo es la que sigue. En la sección 3.1 se describe, de forma global, el modelo de proceso de desarrollo de flujos de trabajo propuesto. En la sección 3.2 se da una visión general del entorno que da soporte a dicho proceso de desarrollo, y en la sección 3.3 se introduce el caso de estudio que se utiliza como ejemplo en los siguientes capítulos para ilustrar las distintas actividades del proceso de desarrollo. Finalmente, en la sección 3.4 aparecen las conclusiones de este capítulo.

3.1 Un Modelo de Proceso de Desarrollo de Flujos de Trabajo

La definición de un modelo de proceso o ciclo de vida a seguir en el desarrollo de flujos de trabajo conlleva tener que identificar las actividades a realizar, su ordenación temporal y las

entradas y salidas de cada actividad. Aunque en la literatura de flujos de trabajo no existen propuestas explícitas de procesos de desarrollo, se asume con normalidad que existen dos actividades básicas: la definición del flujo de trabajo y la ejecución del mismo. En concreto, según la WfMC, a pesar de la gran variedad de SGFT existentes (vistos en el capítulo anterior), todos ellos se caracterizan por dar soporte a tres áreas funcionales, denominadas funciones de edición o definición (*build-time functions)*, funciones de control de ejecución (*run-time control functions)* y funciones de interacción en tiempo de ejecución (*run-time interactions)* [Hol95]. La Figura 3-1 muestra estas áreas funcionales y sus relaciones.

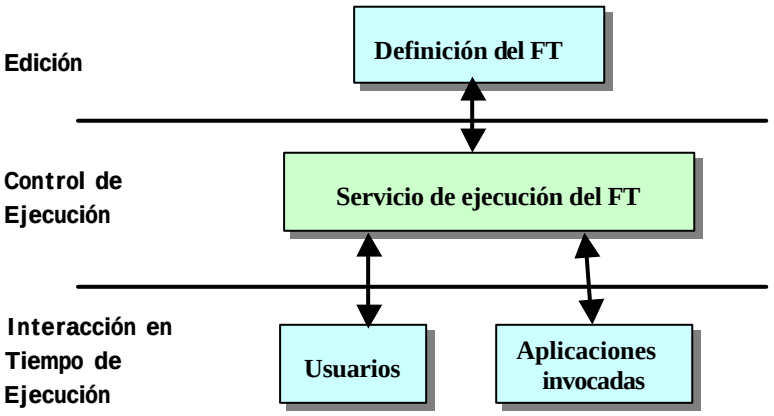


Figura 3-1. Funcionalidad común de los SGFT

La primera de ellas tiene que ver con la funcionalidad que ofrecen los SGFT de definir, modelar y especificar el flujo de trabajo de acuerdo con un modelo de flujo de trabajo ya establecido que incluye los conceptos de actividad, actor, datos, aplicaciones invocadas, etc.

Las funciones de control de ejecución son las que permiten generar una versión ejecutable del flujo de trabajo especificado, que pueda ser ejecutada sobre una máquina. Para ello, se creará la correspondiente instancia del flujo de trabajo y se controlará la creación y ejecución de las instancias actividad de acuerdo con el flujo de control preestablecido.

Finalmente, la interacción en tiempo de ejecución agrupa toda la funcionalidad que proporciona un SGFT para que sea posible la comunicación con usuarios, en aquellas actividades que requieran participación humana, así como la invocación de aplicaciones externas para la ejecución de ciertas actividades, y el acceso / almacenamiento de los datos de entrada / salida de las actividades.

Aparte de estas funciones básicas, existen otros servicios que podemos denominar servicios avanzados de soporte al proceso [Hag99], y que abarcan desde la gestión de

recursos (planificación y balanceado de cargas) hasta la monitorización del proceso, pasando por el almacenamiento de un registro de ejecución, análisis de las historias de ejecución, repositorio de modelos, evolución del modelo, aspectos de persistencia y sincronización en la ejecución del proceso, simulación, etc.

Ante la falta de propuestas concretas, se propone un modelo de proceso de desarrollo de flujos de trabajo que, además de incluir las actividades básicas de definición y ejecución, añade otras actividades complementarias. En concreto, se proponen las siguientes:

- captura de requisitos del proceso de negocio,
- prototipación automática de especificaciones formales, para la validación de requisitos,
- generación de código basada en modelos, y
- extracción de conocimiento a partir del análisis de los registros de ejecución.

Las tres primeras actividades están directamente relacionadas con la disciplina de Ingeniería del Software, haciendo uso de las técnicas, métodos y herramientas que ésta propone para dar soporte a dichas actividades. La última se apoya en el campo de Extracción de Conocimiento, en concreto en la tecnología OLAP y de minería de datos.

El modelo de proceso de desarrollo se muestra en la Figura 3.2, y consta de una serie de etapas que dan cuenta de tres procesos principales: construcción y estabilización, ejecución y prueba, y finalmente, evaluación del proceso de negocio [Can99]. A continuación se describen, de forma general, cada uno de estos procesos, para abordarse con mayor nivel de detalle en los capítulos siguientes.

3.1.1 Construcción y Estabilización de Flujos de Trabajo

Este subproceso inicial comprende el modelado y prototipado del flujo de trabajo. Para una mayor claridad, la Figura 3.3 resalta las actividades del ciclo de vida involucradas en este proceso. En la fase de modelado se identifican los procesos, actividades, recursos utilizados (datos, aplicaciones invocadas y participantes), condiciones de inicio y finalización, y orden de realización de las actividades, de acuerdo con el modelo de proceso de negocio del que se parte. Este modelo del proceso de negocio se ha obtenido en una fase de análisis previa al desarrollo del flujo de trabajo, partiendo del supuesto de que existía en la compañía antes de que ésta decidiera utilizar un SGFT para automatizar dicho proceso. Si esto no es así, el análisis del proceso de negocio y el modelado del flujo de trabajo serían parte de una misma actividad global. En esta situación se pueden utilizar técnicas específicas de Ingeniería de Requisitos como son los Casos de Uso, los cuales permiten capturar los requisitos del proceso y facilitan la obtención de un modelo del mismo.

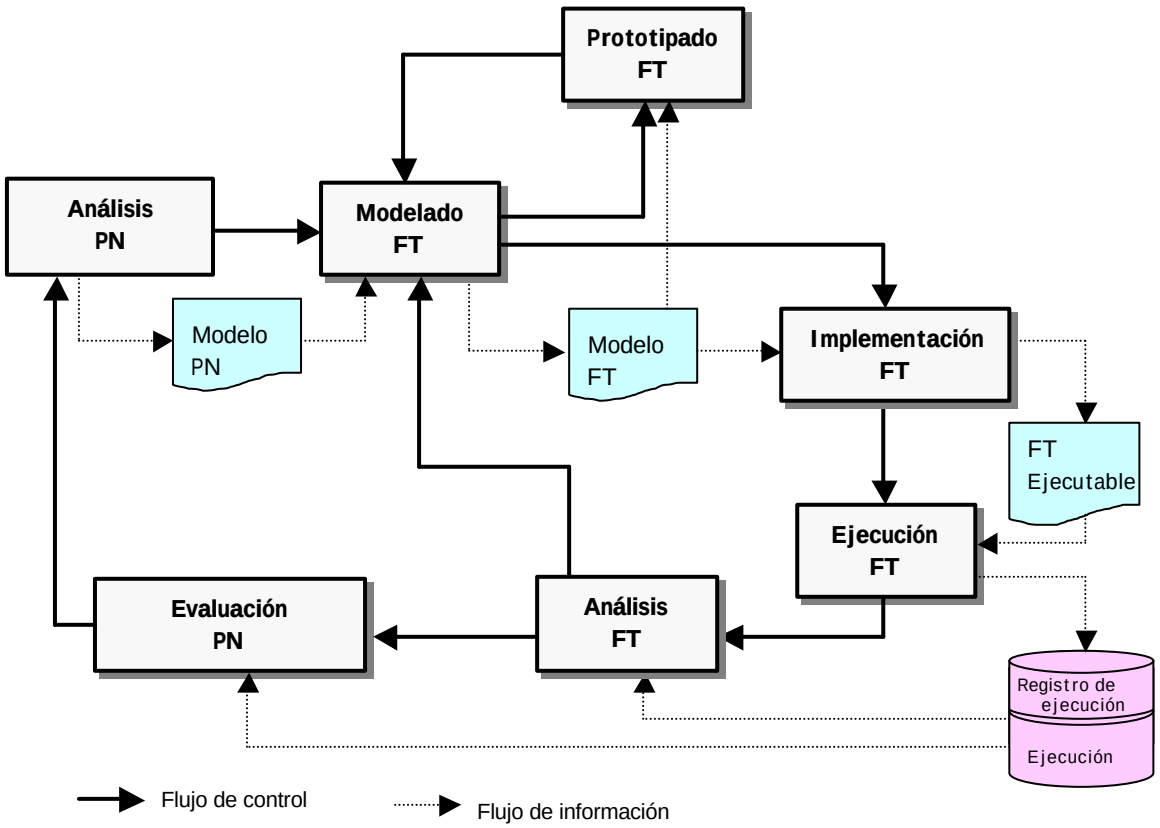


Figura 3-2. Modelo de proceso de desarrollo de flujos de trabajo

Con toda esta información se construye un modelo de flujo de trabajo, utilizando un lenguaje gráfico o un lenguaje textual. El siguiente paso sería la implementación y ejecución de dicho modelo, pero al igual que ocurre en el desarrollo de cualquier software, el modelo obtenido puede ser incompleto y contener errores. Dicho modelo debe estar lo más cerca posible del sistema real que deseamos implementar, y para lograrlo nos puede servir la técnica del prototipado [Agr86]. A partir del modelo construido, se obtiene un prototipo que permite simular la ejecución del flujo de trabajo con la finalidad de detectar y corregir errores en el modelado. Este proceso será iterativo, repitiéndose el número de veces que sea necesario hasta que podamos garantizar que la lógica del flujo de trabajo modelado está de acuerdo con la del proceso de negocio del que partimos. Esta fase de prototipado no necesita de almacenamiento persistente, ni que se ejecuten las llamadas a aplicaciones externas, puesto que como ya se ha comentado se trata de ver que el modelo construido refleja fielmente la lógica del proceso de negocio. Al finalizar este primer proceso de construcción y estabilización, el producto obtenido es un modelo estable del flujo de trabajo.

Las fases de captura de requisitos, modelado y prototipado del flujo de trabajo, que constituyen este subproceso denominado construcción y estabilización, son las que proporcionan la funcionalidad de edición citada en la Figura 3-1.

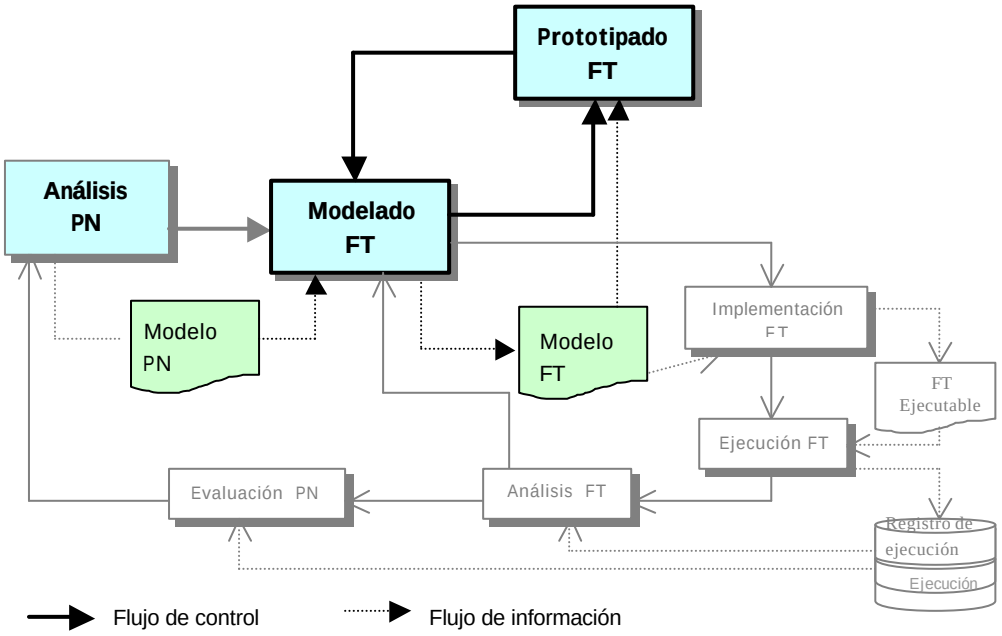


Figura 3-3. Subproceso de construcción y estabilización de un flujo de trabajo

3.1.2 Ejecución y Prueba de Flujos de Trabajo

Una vez validado el modelo, el siguiente subproceso consiste en la ejecución del mismo por parte de un SGFT y el posterior análisis de los resultados de ejecución, con el fin de mejorar el modelo de flujo de trabajo construido (ver Figura 3-4).

En la fase de implementación, los conceptos de modelado se representan en un formato entendible por un SGFT. Dado que se trata de una ejecución real, en este punto se tienen en cuenta todos los aspectos de eficiencia y robustez que se obviaron en el prototipado. Su posterior ejecución, aparte de proporcionar la funcionalidad que se espera del flujo de trabajo, generará información de ejecución diversa, tal como actividades ejecutadas, duración de cada actividad y recursos utilizados. Esta información se almacenará de forma persistente para su posterior análisis.

El análisis de de la información de ejecución nos debe servir tanto para detectar situaciones anómalas (bloqueos y cuellos de botella), como para poder analizar cualquier otro parámetro de ejecución que se desee (duración media de una actividad, número de veces que se invoca una aplicación externa, actividades que se ejecutaron en un intervalo de tiempo, etc.). La información obtenida en el proceso de análisis puede modificar el modelo de flujo de trabajo construido en el proceso anterior, si así se cree conveniente. Esto implicaría volver, si es necesario, al proceso inicial de estabilización.

Esta fase soporta las funcionalidades de control de ejecución y interacciones en tiempo de ejecución de la Figura 3-1.

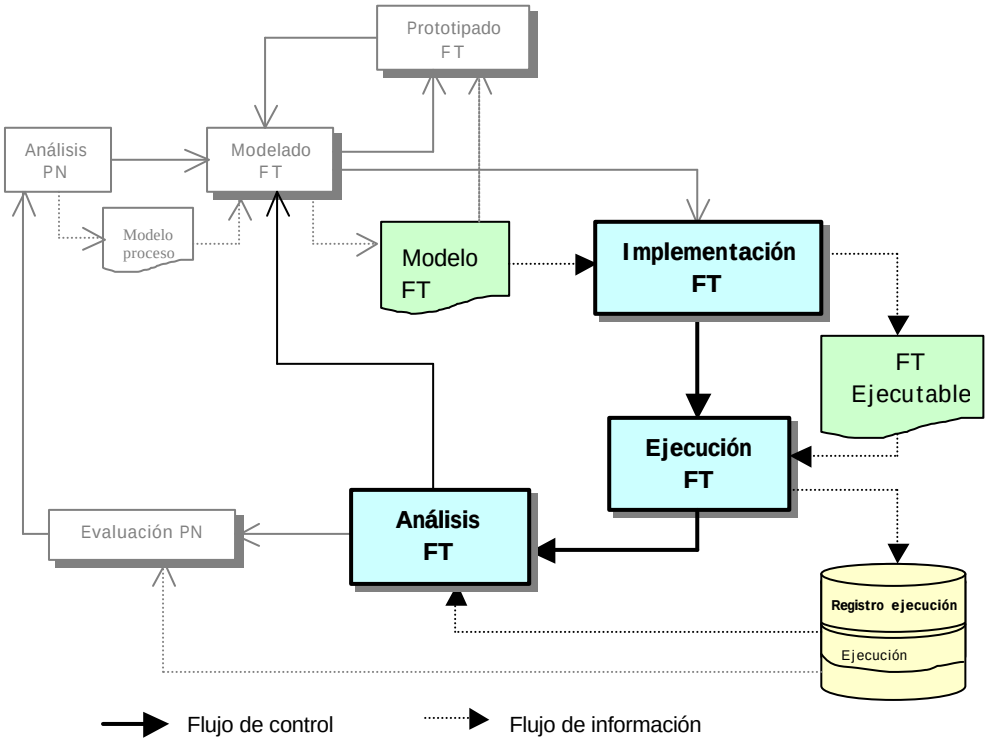


Figura 3-4. Subproceso de ejecución y prueba de un flujo de trabajo

3.1.3 Evaluación de Procesos de Negocio

Los subprocesos de construcción y estabilización, así como los de ejecución y prueba descritos anteriormente, parten del supuesto de que el modelo de proceso de negocio es fijo y lo que se está ejecutando y mejorando es sólo el diseño del flujo de trabajo. Ahora bien, el

proceso de desarrollo que estamos describiendo también incluye mejorar el modelo de proceso de negocio del que partimos. La ejecución del flujo de trabajo hace que quede almacenado persistentemente, no sólo la información descrita en el punto anterior, sino también los datos propios que manejan las distintas actividades involucradas en el flujo de trabajo. Estos datos podrían analizarse en lo que sería un proceso de reingeniería, y chequear si el modelo de proceso de negocio del que partimos es el adecuado para los objetivos de la organización; si no es así, habría que revisar dicho modelo. La Figura 3-5 resalta la parte del ciclo de vida que constituye este subproceso denominado evaluación del proceso de negocio.

Si nos centramos en los objetivos de la organización pueden encontrarse metas tales como aumento del beneficio, aumento de la capacidad de producción o disminución de los tiempos de entrega del producto. El proceso de negocio que siga la organización en un momento dado, no tiene por qué ser el más adecuado para el objetivo principal de la empresa, o puede que sí lo sea pero dicho objetivo cambie, con lo cual, probablemente habría que cambiar el proceso seguido. Para la optimización del proceso de negocio de una organización podemos utilizar los datos almacenados persistentemente tras la ejecución del flujo de trabajo que representa al proceso actual. En base a ellos, vamos modificando el modelo del flujo de trabajo de acuerdo con el nuevo proceso de negocio, utilizando el SGFT para simular qué resultados obtendríamos con el proceso mejorado.

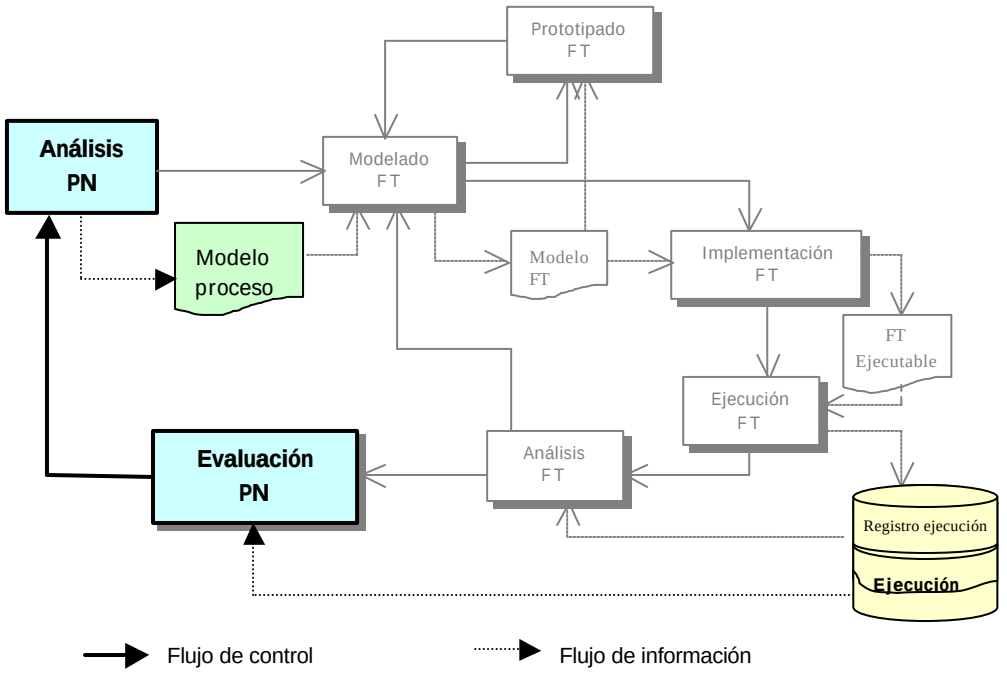


Figura 3-5. Subproceso de evaluación del proceso de negocio

3.2 Un Entorno de Soporte al Proceso de Desarrollo

Una vez definido el proceso de desarrollo de flujos de trabajo, el objetivo es disponer de un entorno que dé soporte a dicho proceso. Concretamente, este entorno debe proporcionar un conjunto de herramientas que permitan realizar las actividades concretas asociadas a cada subproceso. Desde esta perspectiva, la Figura 3-6 muestra una primera visión global del entorno como una unión de herramientas para:

- ✓ La construcción y estabilización de flujos de trabajo.
- ✓ La ejecución de los flujos de trabajo definidos.
- ✓ El análisis de las ejecuciones de flujos de trabajo.
- ✓ La evaluación de los procesos de negocio.

Además, es necesario disponer de un metamodelo de flujo de trabajo claramente definido que identifique los componentes básicos y sus relaciones. Este metamodelo es el nexo entre los distintos subprocesos.

El entorno propuesto interactúa con otros sistemas, que van a realizar en todo o en parte ciertas actividades dentro del proceso de desarrollo. En concreto, para la validación del modelo de flujo de trabajo se hace uso de un entorno de prototipación, la ejecución eficiente de dicho modelo se realiza en un SGFT, y finalmente, para el análisis se hace uso de tecnología existente, relacionada con los almacenes de datos y minería de datos.

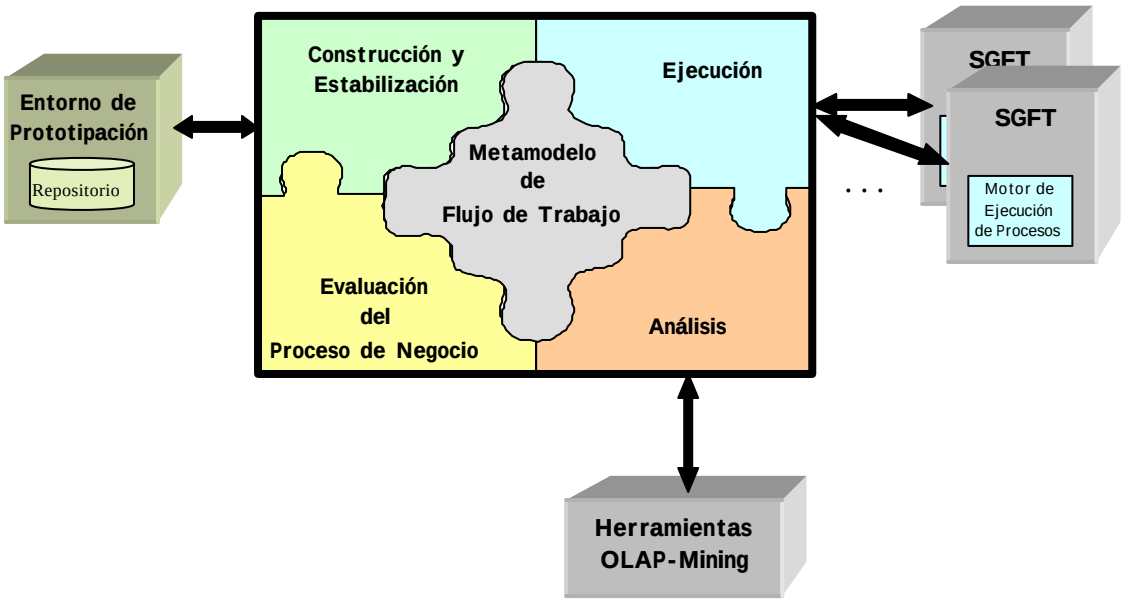


Figura 3-6. Visión global del entorno

A continuación se describe, de forma general, cada una de las partes que componen este entorno, enumerándose las distintas herrramientas que lo forman y dan soporte a las distintas actividades. También se describe la interacción con los sistemas externos mencionados anteriormente.

- En el *subproceso de construcción y estabilización*, el analista debe construir un modelo de flujo de trabajo que represente el proceso de negocio seguido por la organización. Para ello, el entorno proporciona un ***editor de flujos de trabajo*** que, haciendo uso de un lenguaje gráfico, permite modelar dicho proceso de negocio. El lenguaje gráfico se define en base al metamodelo de flujo de trabajo de referencia y permite que el analista no tenga que conocer en detalle la sintaxis del lenguaje específico que cada SGFT elegido acepta como entrada. Toda la información del modelo gráfico construido queda almacenada en un ***repositorio***, cuya estructura está basada en el metamodelo de referencia.

  Sin embargo, en muchas ocasiones, especialmente cuando el proceso de negocio es complejo y no totalmente conocido, el analista necesita descubrir dicho proceso antes de modelarlo, siendo necesaria una mayor interacción entre el analista y los miembros de la organización, quienes conocen todo o parte del proceso seguido por la misma. Para dar soporte a esta situación, se ha incluido en el entorno una capa de Ingeniería de Requisitos que permite ayudar al analista a capturar los requisitos del proceso de negocio, expresados mediante casos de uso extendidos; un ***editor de casos de uso*** permite construir el modelo asociado, a partir del cual se obtiene automáticamente una versión preliminar del modelo flujo de trabajo de la organización. Este modelo se puede refinar desde el editor de flujos de trabajo hasta completar el modelo definitivo. El ***conversor del modelo de casos de uso extendido al modelo de flujo de trabajo*** está basado en las equivalencias entre ambos modelos, junto con una serie de patrones de proceso definidos a partir de las relaciones de uso y extensión entre casos de uso.

  Una vez construido el modelo de flujo de trabajo, el entorno permite animar el modelo obtenido con la finalidad de iniciar un proceso de validación del mismo, basado en el paradigma de programación automática [Bal83]. Este paradigma se basa en el uso de lenguajes formales para construir especificaciones ejecutables que pueden ser animadas automáticamente. En concreto, el lenguaje de especificación formal elegido es OASIS y un ***módulo de generación de código OASIS*** permite, a partir del modelo construido, generar automáticamente la especificación formal equivalente, que puede ejecutarse en un entorno de animación de OASIS, como es el caso de KAOS. Este generador aplica un conjunto de plantillas de traducción definidos en base a la formalización OASIS del metamodelo de referencia.

  Llegados a este punto, se dispone de un modelo de flujo de trabajo validado que puede ser ejecutado, posteriormente.

- Para el *subproceso de ejecución*, puesto que existen herramientas que permiten la ejecución de flujos de trabajo de forma eficiente, robusta y en ocasiones distribuida, no es objetivo de este entorno proporcionar herramientas específicas para la ejecución, sino que se hace uso de SGFT existentes que ya proporcionan este servicio. Para ello, el entorno incluye un conjunto de compiladores de modelos que permiten generar semiautomáticamente una versión ejecutable del modelo de flujo de trabajo en diferentes SGFT[9], para su ejecución de forma eficiente. Estos compiladores de modelos están basados en las equivalencias establecidas entre el metamodelo de referencia y el que soporta cada sistema destino concreto, como por ejemplo el sistema OPERA [OPE] o el sistema MQSeries WF.
- Para el subproceso de *análisis*, se parte de la premisa de que la ejecución del flujo de trabajo por parte de un SGFT permite crear un registro de ejecución que da cuenta de cierta información relacionada con la ejecución del mismo, tal como instante de inicio, de finalización y estado alcanzado (éxito o fallo) para cada proceso y actividad, instante de asignación y liberación de recursos, etc. Teniendo en cuenta esto, este registro de ejecución[10] se puede analizar para obtener nueva información acerca del flujo de trabajo que se ha modelado. El análisis comprende tanto la obtención de información puramente estadística como el descubrimiento de conocimiento interesante acerca del proceso, que nos permita optimizar el flujo de trabajo en base a la información histórica de sus distintas ejecuciones. Dicho análisis se lleva a cabo con la construcción de un ***almacén de datos*** que recoge toda la información de ejecución, en el formato adecuado. La estructura del almacén de datos se define en base a los ítems que son interesantes analizar en un flujo de trabajo y que han sido establecidos previamente. Un ***módulo de recolección, transformación y carga*** deja el almacén de datos preparado para ser utilizado. En este punto, al igual que ocurre en el subproceso de ejecución, se hace uso de la tecnología existente para la explotación del almacén, en concreto se aplican ***técnicas OLAP y de minería de datos*** (*OLAP-Mining* [Han01]). Esto genera automáticamente un conjunto de informes preestablecidos que sirven como realimentación para poder mejorar el modelo de flujo de trabajo.
- Finalmente, el subproceso de *evaluación del proceso de negocio* cierra el proceso de desarrollo. En esta tesis no se da a un soporte explicíto a esta parte, pero a continuación se describe cómo realizarlo. Siguiendo un esquema similar al utilizado para el análisis de ejecuciones de flujo de trabajo, pero variando la información de entrada, se obtienen informes que tienen que ver no con el modelo de flujo de trabajo construido, sino con información propia del proceso de negocio concreto que ha modelado y ejecutado. La información de partida son los propios datos almacenados y manipulados por las actividades del proceso. Además, los informes a generar no están predeterminados, sino

[9] Dependiendo del SGFT destino habrán fases totalmente automáticas y otras semiautomáticas, puesto que necesitarán de ciertas decisiones que debe tomar el analista en su momento (ver Capítulo 7).
[10] En los SGFT, se denomina información de auditoría (*audit trail*).

que se deja en manos del analista del negocio que realice la explotación de dichos datos, en función de cuáles sean los objetivos a mejorar en la organización y las políticas de optimización a seguir. Para ello, se puede hacer uso igualmente de la tecnología OLAP y de minería de datos.

3.3 Caso de Estudio: Gestión de Proyectos Final de Carrera en una Biblioteca Digital

En esta sección se describe el caso de estudio que se utilizará como ejemplo conforme se vaya detallando cada una de las fases del proceso de desarrollo propuesto, en los siguientes capítulos. Se trata de la gestión de la asignatura Proyecto Final de Carrera (PFC) de la Facultad de Informática (FI) de la Universidad Politécnica de Valencia (UPV) y cuyo proceso forma parte de la Biblioteca Digital de PFC que se está desarrollando actualmente[11].

El PFC es una asignatura troncal de la titulación de Ingeniero en Informática y constituye la última etapa de formación de los alumnos que finalizan sus estudios en la FI de la UPV. Existe una normativa que reglamenta el desarrollo y evaluación de la asignatura, que resumimos a continuación.

El alumno realiza la matrícula del proyecto eligiendo una modalidad determinada. Existen tres tipos distintos:

a) *Proyectos de tipo A*. Son proyectos ofertados previamente por los departamentos con docencia en la FI. Para desarrollar uno de estos proyectos, el alumno debe realizar previamente una "Solicitud de Proyecto", pudiéndo indicar hasta un máximo de cinco. La Comisión de Proyectos de la FI analizará las solicitudes y asignará los proyectos a los alumnos de acuerdo, básicamente, al cumplimiento de las recomendaciones especificadas en los proyectos seleccionados y a su expediente académico.

b) *Proyectos de tipo B*. El alumno puede desarrollar un proyecto sobre un tema específico de su interés, debiendo contactar directamente con el profesor que puede estar interesado en actuar como su director. En este caso, se deberá realizar una “Solicitud de Aprobación del Proyecto” dirigida a la Comisión de Proyectos, en la que se hará constar los datos del proyecto. El director podrá ser, en general, cualquier profesor de la UPV. Si dicho profesor no imparte docencia en la Facultad, será necesario que un profesor con docencia en la Facultad realice la función de tutor.

c) *Proyectos tipo C*. Son proyectos desarrollados en el marco de un convenio, bien con empresas, organizaciones o universidades españolas, o bien con universidades extranjeras dentro de los programas de intercambio que la FI tiene suscritos (programa

[11] http://bdpfc.inf.upv.es

SOCRATES, ERASMUS, etc.). En ambos casos, para que el proyecto sea autorizado, se deberá seguir el mismo proceso que para los de tipo B.

En los proyectos de tipo B y C deberán transcurrir al menos 3 meses desde la presentación de la Propuesta de Proyecto hasta la presentación de la Memoria.

La Comisión de Proyectos asigna los PFC de tipo A y aprueba las solicitudes de PFC para los tipos B y C. Por su parte, el alumno debe presentar el PFC dentro del curso adémico correspondiente. En caso de no ser así, se debe presentar una Solicitud de Prórroga, con el visto bueno del director del proyecto, para extender el plazo de asignación del PFC durante un curso académico adicional.

La evaluación de la asignatura se efectuará en las convocatorias ordinaria y extraordinaria de cada semestre. Los proyectos podrán ser evaluados por el director del mismo o tutor en su caso, o bien, el alumno podrá solicitar a la Comisión de Proyectos la evaluación del proyecto por un Tribunal. Sólo en este último caso se podrá optar a una calificación de Matrícula de Honor, así como a la obtención de Premio Extraordinario.

La evaluación de un proyecto por un Tribunal requiere los siguientes pasos: cumplimentar la Solicitud de Tribunal, nombramiento del Tribunal por parte de la Comisión de Proyectos y presentación del proyecto en sesión pública. Una vez concluida la defensa del PFC, el Tribunal delibera en sesión cerrada y se otorga una calificación. La calificación de Matrícula de Honor, así como los Premios Extraordinarios, serán otorgados directamente por la Comisión de Proyectos, a instancia de los Tribunales que evalúan los proyectos. En el caso de evaluación por director la calificación máxima será 9.

Una vez terminado el trabajo, y dentro de los plazos de presentación establecidos, el alumno deberá presentar en la Secretaría del Centro dos copias de la memoria (si la evaluación la realiza el director/tutor) o tres copias (si es mediante tribunal) que reflejen el trabajo desarrollado. Las memorias podrán presentarse siempre y cuando el alumno haya superado todas las asignaturas troncales del segundo ciclo (a excepción de la asignatura PFC) y le resten no más de 24 créditos (excluidos los 15 créditos del PFC) para acabar la carrera.

Si un alumno con PFC asignado desea cambiar o renunciar a un tema deberá solicitarlo a la Comisión de Proyectos. Los PFC de cualquier modalidad asignados a alumnos cuya calificación haya sido NO PRESENTADO tanto en la convocatoria ordinaria como extraordinaria, causarán baja de forma automática, excepto aquellos para los que el director del mismo remita a la Comisión una solicitud de prórroga indicando su continuación para el siguiente curso académico. En caso de baja, el alumno tendrá que solicitar otro proyecto en el curso siguiente. La Figura 3-7 muestra una visión global del proceso.

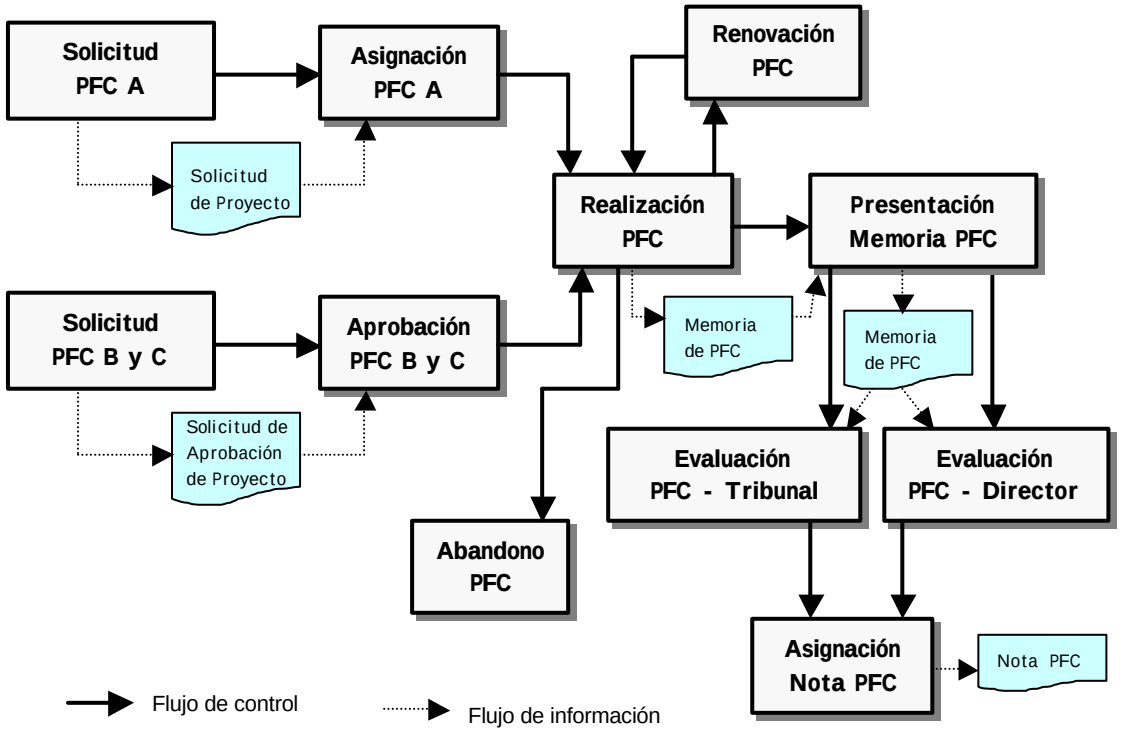


Figura 3-7. Visión global de la gestión de PFC

3.4 Conclusiones

En este capítulo se ha presentado de forma global la propuesta de *ciclo de vida* o *modelo de proceso de desarrollo* de flujos de trabajo. Consta de tres subprocesos que se denominan construcción y estabilización, ejecución y prueba, y evaluación del proceso de negocio. En cada uno de ellos se han definido un conjunto de actividades a realizar que están directamente relacionadas con la producción de software, al considerar que un flujo de trabajo puede ser visto como un producto software.

Además de definir el proceso de desarrollo, se ha presentado una visión general del entorno de soporte a las actividades que lo componen. Métodos, técnicas y herramientas provinientes de la Ingeniería del Software en general, y de la Ingeniería de Requisitos en particular, se han aplicado en las fases de construcción y estabilización, así como en la de ejecución; mientras que para la fase de prueba se utilizan técnicas de Minería de Datos.

Finalmente ha sido presentado un ejemplo o caso de estudio a seguir a lo largo del trabajo. Se trata de la gestión de Proyectos Final de Carrera en al Biblioteca Digital de la FI-

UPV. El flujo de trabajo que representa al proceso de gestión, en sí mismo no es demasiado complejo, pero nos permite aplicar el modelo de proceso de desarrollo a un caso real que está en desarrollo dentro de la Universidad, pudiéndo disponer de mayor información.

PARTE II

Definición y Formalización de un Metamodelo de Flujo de Trabajo

Capítulo 4.

Metamodelo de Flujo de Trabajo

En este capítulo se describe un metamodelo que especifica los diferentes elementos que componen un flujo de trabajo. Para su definición se ha utilizado el paradigma orientado a objetos, modelando los distintos componentes del mismo como clases e identificado las relaciones entre ellos. Se ha elegido la notación UML, en concreto el diagrama de clases, para representar sus distintas partes. El metamodelo propuesto sirve como base para el proceso de desarrollo de flujos de trabajo que se expone en la tercera parte de la memoria, siendo el punto de unión entre los diferentes subprocesos que lo forman. Además, este metamodelo se ha formalizado utilizando un lenguaje de especificación formal y orientado a objetos, como se muestra en el capítulo siguiente.

La estructura del capítulo es la que sigue. En la sección 4.1 se describen cuáles son los requisitos que debe cumplir el metamodelo, partiendo de las tres dimensiones típicas que se consideran en todo flujo de trabajo. En las secciones 4.2, 4.3 y 4.4 se pasa a describir detalladamente cada uno de los componentes del flujo de trabajo y cómo son modelados en un enfoque orientado a objetos, para finalmente presentar el metamodelo completo en la sección 4.5. Por último, en la sección 4.6 se presentan las conclusiones del capítulo, haciendo una comparativa con el metamodelo propuesto por la WfMC.

4.1 Requisitos

Con el término *metamodelo* denotamos a aquel modelo que se utiliza para definir nuevos modelos en un cierto dominio, puesto que establece la ontología asociada. En nuestro caso, un metamodelo de flujo de trabajo debe expresar qué es un flujo de trabajo; en definitiva, debe captar todos los aspectos o componentes del mismo. Una vez definido el metamodelo de referencia podremos especificar y definir flujos de trabajo concretos, que posteriormente se ejecuten en un SGFT. Además de los requisitos funcionales, existen otras propiedades que se le pueden exigir al metamodelo y que entran dentro de los denominados requisitos no funcionales, tales como ejecutabilidad, facilidad de uso, reutilización y otros.

El metamodelo que se presenta en este capítulo recoge todos aquellos aspectos o componentes básicos y esenciales de un flujo de trabajo. Estos componentes están de acuerdo con las especificaciones de la WfMC y la de otros trabajos relevantes en este campo, tales como [Cur92], [Cas95], [Alo98] y [Sad99]. Es importante destacar que, aunque este metamodelo recoge los aspectos más relevantes de un flujo de trabajo, pueden existir ciertos dominios de aplicación en los cuales puedan aparecer nuevos requisitos específicos que hagan necesario enriquecer dicho metamodelo con nuevas propiedades o incluso nuevas clases. Por lo tanto, es importante que este metamodelo sea también adaptable y extensible.

4.1.1 Dimensiones de un Flujo de Trabajo

De acuerdo con la definición de flujo de trabajo vista en el capítulo 1, se considera que un flujo de trabajo tiene tres dimensiones básicas, que son ortogonales entre sí [Ley00]. Estas son: lógica del proceso, estructura organizacional e infraestructura de tecnologías de la información (ver Figura 4-1).

En primer lugar, la lógica del proceso describe qué tareas se realizan y en qué orden. Las tareas pueden ser actividades concretas o subprocesos, que se pueden invocar de forma local o remota, dentro de la misma compañía o en otra distinta. En cuanto al flujo de control, éste puede ser secuencial o paralelo, pudiéndose establecer condiciones que determinen el camino a seguir dentro del proceso. La segunda dimensión hace referencia a la estructura organizativa de la compañía: unidades organizacionales o departamentos, relaciones entre ellas, usuarios que forman parte de las mismas y roles que éstos pueden desempeñar. Esta dimensión nos da cuenta de quién realiza cada tarea. La tercera dimensión describe la infraestructura de tecnologías de la información disponible en la compañía, tal como programas o aplicaciones que se utilizan para realizar una determinada tarea; en definitiva, representa los recursos asociados al flujo de trabajo.

Resumiendo, en un flujo de trabajo confluyen tres dimensiones, que dan cuenta de: qué se ejecuta, quién lo ejecuta y con qué se ejecuta. Teniendo en cuenta estas dimensiones, el resto del capítulo se dedica a describir el metamodelo propuesto.

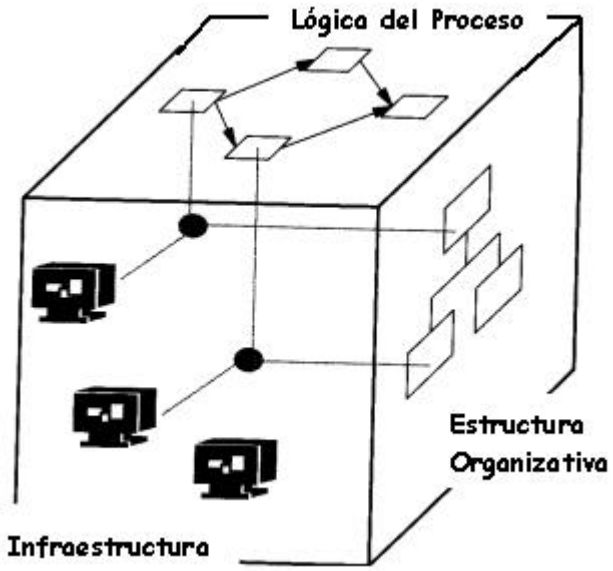


Figura 4-1. Dimensiones de un flujo de trabajo (fuente [Ley00])

4.2 Flujo de Trabajo

En nuestra propuesta, un *flujo de trabajo* está formado por un *proceso* y tiene además asociado un conjunto de *recursos*[12]. Lógicamente existe una relación entre el proceso que constituye el flujo de trabajo y los recursos asociados al mismo. El proceso está formado por un conjunto de *tareas* y son éstas las que realmente utilizan los recursos.

La Figura 4-2 muestra una visión global de los principales componentes de un flujo de trabajo y sus relaciones, en notación UML[13]. Podemos observar que el flujo de trabajo es modelado como una clase (denominada workflow). De la misma forma, sus componentes también son clases. Concretamente, la clase process representa el proceso, la clase resource representa los recursos y la clase task representa las tareas del proceso. La utilización de los recursos por parte de las tareas que forman el proceso queda representado mediante la asociación entre la clase resource y la clase task.

[12] El concepto de recurso engloba dos de las dimensiones vistas en la sección anterior, la estructura organizativa y la infraestructura de tecnologías de la información.
[13] Todos los componentes del metamodelo de flujos de trabajo se denotan por términos en inglés.

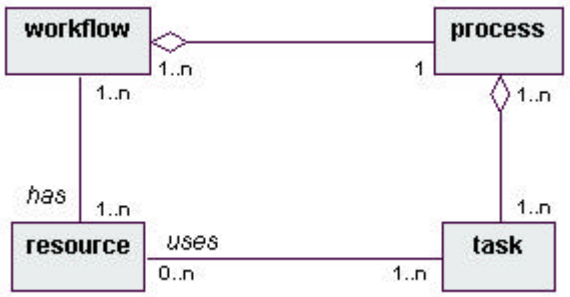


Figura 4-2. Componentes principales de un flujo de trabajo

En las siguientes secciones, y siguiendo esta misma estructura, se detalla cada uno de los componentes del flujo de trabajo, con la finalidad de ir refinando el metamodelo propuesto.

4.3 Proceso

Un proceso está formado por un conjunto de *tareas* conectadas mediante *flujos de control*. Este flujo de control establece el orden correcto de ejecución de las distintas tareas que componen el proceso (descrito en sección 4.3.4). Las tareas pueden ser de tres tipos: *actividades*, *subprocesos* y *condiciones de transición*. Las actividades y subprocesos especifican cada uno de los pasos que componen el proceso, a distinto nivel de granularidad. La condiciones de transición permiten establecer puntos de bifurcación y/o de reunión en el flujo de control. Además de las tareas conectadas mediante flujos de control, todo proceso tiene asociada la siguiente información: un identificador, un nombre, una descripción, una condición de inicio, una condición de finalización y un estado. La condición de inicio del proceso es una condición que sólo cuando se evalúa a cierto[14] permite que el proceso se inicie realmente, es decir, que el control pase a la primera tarea del mismo. La condición de finalización, de igual forma, determina el estado de finalización del proceso; de modo que si se evalúa a cierto, el proceso finaliza con éxito y si se evalúa a falso[15], el proceso finaliza sin éxito. Finalmente, todo proceso tiene información acerca del estado en que se encuentra[16]. En concreto, interesa conocer cuándo un proceso ha sido creado, cuándo se cumple la condición de inicio y el proceso pasa a estar activo, o por el contrario, si ésta no se cumple y el proceso no se ejecuta, cuándo el proceso inicia realmente su ejecución y cuándo finaliza, teniendo en cuenta si la

[14] Valor *true* del tipo bool.
[15] Valor *false* del tipo bool.
[16] Esta información sólo es relevante una vez el proceso ha iniciado su ejecución.

finalización del mismo se ha producido con éxito, sin éxito, o bien si ha sido abortado por el usuario.

La Figura 4-3 muestra la jerarquía de generalización correspondiente a las tareas. La clase que las representa (task) es una clase abstracta que define la estructura y el comportamiento común a todas ellas. En la Figura 4-3 sólo se muestran los atributos de la clase task y la clase process. Los del resto de clases se irán mostrando conforme se describan dichos componentes en las secciones siguientes.

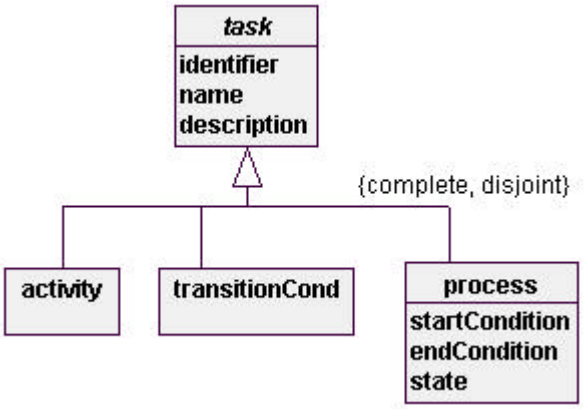


Figura 4-3. Jerarquía de generalización para las tareas

Las propiedades del proceso contienen la información indicada anteriormente, es decir, todo proceso tiene un identificador (identifier), un nombre (name), una descripción (description), una condición de inicio y una de finalización (startCondition y endCondition, respectivamente) y un estado (state).

Los posibles estados en los que se puede encontrar un proceso, junto con las transiciones válidas entre ellos, permiten definir el diagrama de estados para la clase process (ver Figura 4-4). La propiedad state da cuenta del estado en que se encuentra todo proceso y toma valores de acuerdo con el diagrama de estados definido. En concreto, los posibles estados de un proceso son: creado (created), activo (active), en ejecución (running), fin de ejecución (executed), finalización con éxito (finished), finalización sin éxito (terminated), finalización por interrupción del usuario (aborted) y muerto (dead). Como se puede apreciar en la Figura 4-4, tras el evento de creación se alcanza el estado creado, en el cual se evalúa la condición de inicio y sólo en el caso que se evalúe a cierto pasará al estado activo. Por el contrario, si la condición de inicio no se cumple, el proceso no puede seguir con su ejecución y pasa al estado muerto. Cuando el proceso inicia realmente su ejecución, lo cual implica que se empiece a ejecutar la primera tarea que lo compone según el orden marcado por el flujo de control, el proceso pasa a estar en ejecución. En este estado pueden ocurrir dos cosas: que la ejecución del proceso finalice, con lo cual se alcanza el estado fin de ejecución, o bien que el usuario decida interrumpir la ejecución del proceso, alcanzándose en este caso el estado de

finalización por interrupción del usuario. En el primer caso, se evalúa la condición de finalización, y el resultado de misma determina el siguiente estado a alcanzar. Si la condición se evalúa a cierto, el estado alcanzado es de finalización con éxito; en caso contrario se alcanza el estado finalización sin éxito. Como se puede apreciar, el cambio de estado no queda determinado únicamente por la ocurrencia de eventos, sino también por el cumplimiento o no de las condiciones de inicio y finalización. Éstas constituyen una parte importante de la definición y ejecución del proceso, determinando la mayor parte de los estados finales que dicho proceso puede alcanzar antes de que se destruya la instancia que lo representa

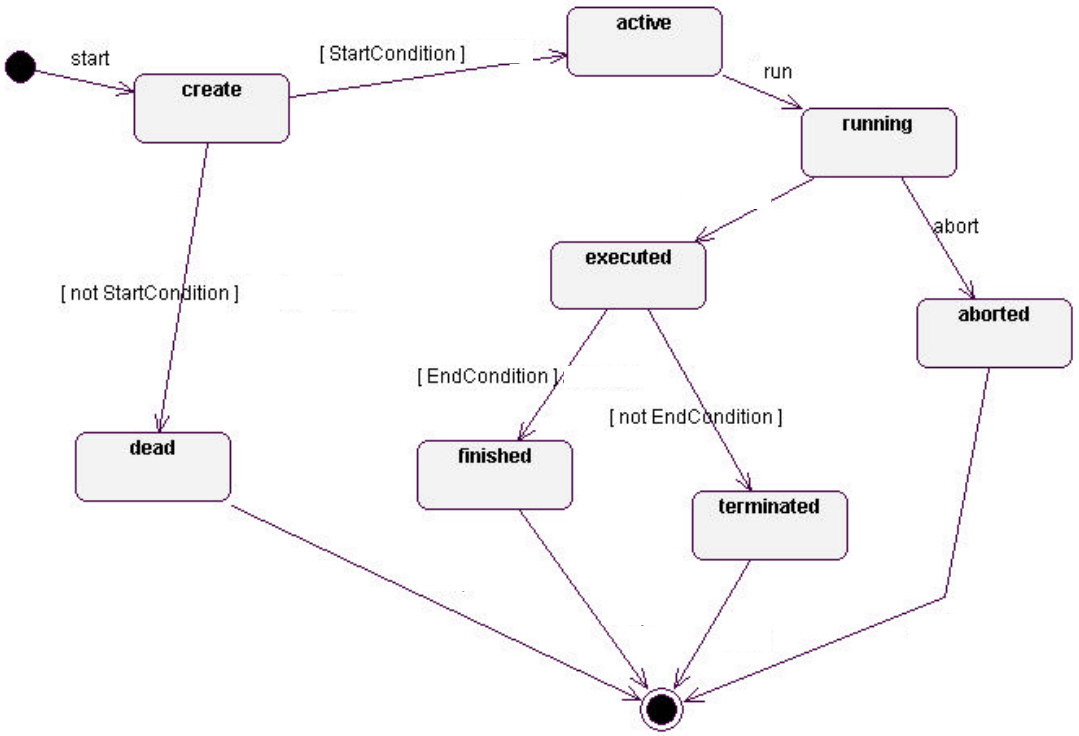


Figura 4-4. Diagrama de transición de estados de un proceso

4.3.1 Actividad

Una actividad representa cada uno de los pasos o tareas básicas a realizar dentro de un proceso. Desde la perspectiva del proceso que se está modelando, toda actividad se considera atómica, es decir, que no se puede descomponer en otras actividades. En otro caso, considera un subproceso, como se verá en la sección 4.3.2.

Toda actividad tiene un identificador, un nombre, un estado, una condición de inicio, una condición de finalización y una serie de acciones específicas a realizar, que constituyen

la propia actividad. La condición de inicio y la de finalización, tal como ocurre en el caso de un proceso, son condiciones que determinan cuándo una actividad realmente se inicia o finaliza, respectivamente. Además, al igual que con los procesos, nos interesa saber cuándo una actividad está activa, está en ejecución o ha finalizado, pudiendo finalizar con éxito, sin éxito o bien haber sido interrumpida por el usuario. Por ello, los estados por los que puede pasar una actividad coinciden con los definidos para el proceso.

Existen dos tipos de actividades. Una actividad *manual* es aquella que necesita al menos un participante humano para su realización; las actividades manuales tienen que ver con la toma de decisiones, rellenar formularios de datos, proporcionar información, etc. Una actividad *automática* es aquella que normalmente no requiere un participante humano para su realización, sino que es una determinada operación o conjunto de operaciones sobre un conjunto de datos, o bien, la invocación de una aplicación externa. Algunos ejemplos de actividades automáticas pueden ser, por ejemplo, lanzar una transacción sobre los datos o realizar cálculos estadísticos. Por otra parte, en todo proceso consideramos dos actividades diferenciadas que marcan el inicio y final del mismo y a las que denominamos actividad inicial y actividad final, respectivamente.

La Figura 4-5 muestra el modelo conceptual de una actividad, de acuerdo con la descripción previa. La actividad es una clase (activity) que se especializa en automática (automated_activity) y manual (manual_activity). En este modelo no se expresan las relaciones existentes entre las actividades y los recursos, ya que estas relaciones se presentarán tras definirlos (sección 4.5).

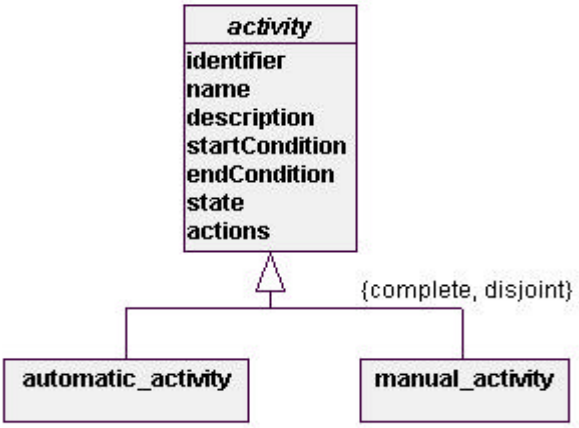


Figura 4-5. Jerarquía de generalización de las actividades

El modelado de las actividades es muy similar al de los procesos, puesto que gran parte de la información coincide en ambos. La clase actividad tiene como propiedades, al igual que el proceso, un identificador (identifier), un nombre (name), una descripción (description)[17],

[17] Las propiedades identifier, name y description son, en realidad, heredadas de la clase task.

una condición de inicio y de finalización (startCondition y endCondition, respectivamente) y un estado (state). Además de las anteriores, toda actividad tiene una propiedad que contiene las acciones concretas asociadas a la misma y que denominamos actions.

Por otra parte, el diagrama de transición de estados de la clase actividad coincide con el definido para la clase proceso. Esto supone que el objeto que representa a una actividad concreta pasa al estado activo (active), inmediatamente después de haber sido creado y cumplirse la condición de inicio de la actividad, tras lo cual pasa a ejecución (running). Una vez finaliza su ejecución (executed), se comprueba la condición de finalización. El estado alcanzado es distinto dependiendo de si dicha condición se satisface o no (finished o terminated, respectivamente). Además, también se considera la posibilidad de que la actividad pueda ser interrumpida por el usuario (aborted).

4.3.2 Subproceso

En la definición de un proceso pueden existir pasos complejos que no se puedan representar mediante actividades, puesto que éstas se consideran atómicas. En este caso, cada paso complejo se representa como un subproceso. Un subproceso es pues un proceso que forma parte de otro proceso y su definición coincide con la vista en la sección 4.3, con la particularidad de que siempre existirá un proceso de nivel más alto al que pertenezca. La noción de subproceso introduce modularidad en la definición de un flujo de trabajo, facilitando el modelado de procesos complejos.

La noción de subproceso o composición de un proceso en otros procesos, se modela con una relación recursiva de la clase process consigo misma, denominada subprocess. La Figura 4-6 muestra dicha relación.

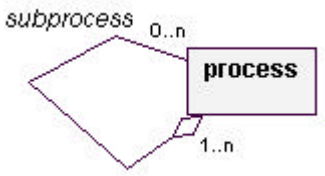


Figura 4-6. Composición de procesos

4.3.3 Condiciones de Transición

En la definición de un proceso se pueden añadir condiciones de transición que permiten establecer distintas alternativas o caminos a seguir, según se cumplan o no una serie de requisitos. En nuestro modelo, estas condiciones de transición pueden ser de dos tipos: de bifurcación y de unión. A continuación se describe cada una de ellas.

- ***Condiciones de bifurcación***: Son puntos de bifurcación dentro del proceso, a partir de los cuales se pueden iniciar en paralelo una o más tareas. Se caracterizan por tener un único punto de entrada y varios puntos de salida. Esto implica tener una única tarea como entrada (TEnt) y varias tareas como salida ($TSal_i$), que definen los distintos caminos a seguir tras la bifurcación (ver Figura 4-7). Los caminos de salida pueden tener asociadas condiciones sobre los datos que maneja el proceso o sobre su propia ejecución, como por ejemplo, el estado alcanzado por una actividad.

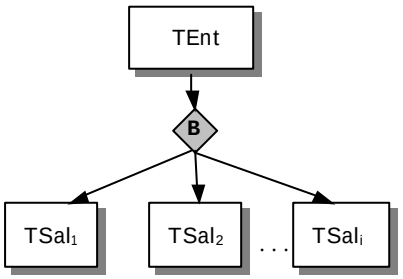


Figura 4-7. Condición de bifurcación

El comportamiento general de la condición de bifurcación es el siguiente: cuando finaliza la ejecución de la tarea de entrada, el control llega a la condición de bifurcación; en ese momento, ésta decide a qué caminos de salida se envía el control para que se inicie la ejecución de las correspondientes tareas de salida. Concretamente, sólo se enviará a aquellos cuya condición asociada se evalúe a cierto.
Dentro de las condiciones de bifurcación se distingue entre bifurcación total y bifurcación parcial. Esta clasificación depende fundamentalmente del número de tareas de salida que pueden iniciar su ejecución tras la bifurcación:

- *Bifurcación total.* Se inicia la ejecución en paralelo de todas las tareas de salida asociadas. No hay condiciones asociadas a los caminos de salida[18].
- *Bifurcación parcial.* En este caso, sólo se inicia la ejecución de aquellas tareas de salida cuya condición asociada al camino de salida correspondiente se evalúa a cierto.

- ***Condiciones de unión***: Son puntos de reunión, en los que, para ejecutar el siguiente paso dentro del proceso, es necesario que hayan finalizado uno o más de los pasos previos que se están ejecutando de forma concurrente. Las condiciones de unión, por lo tanto, tienen el papel de puntos de sincronización dentro del proceso. Se caracterizan por tener varios puntos de entrada y un único punto como salida, lo que supone tener varias tareas como entrada, $TEnt_i$, y una única tarea como salida, TSalida (ver Figura 4-8). Los

[18] En realidad, los caminos de salida pueden tener asociada una condición, pero en este caso se asume que es *cierto*.

caminos de entrada pueden tener asociadas condiciones sobre los datos que maneja el proceso o sobre su propia ejecución, como por ejemplo, el estado alcanzado por una actividad.

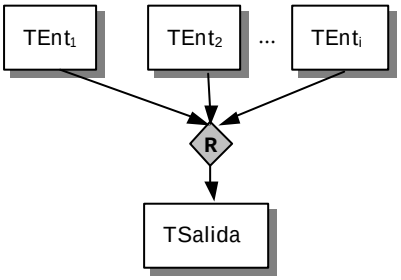


Figura 4-8. Condición de unión

El comportamiento general de estas condiciones es el siguiente: cada vez que acaba la ejecución de una tarea de entrada, el control llega a la condición de unión, siempre y cuando la condición asociada a dicho camino de entrada ha sido evaluada a cierto; en ese caso, la condición de unión decide si se envía el control al camino de salida para iniciarse la ejecución de la tarea de salida o se espera la finalización de más tareas de entrada, cuya condición asociada sea cierto.
Dentro de las condiciones de unión se distingue entre unión total y unión parcial. Esta clasificación depende del número de tareas de entrada que deben haber finalizado antes de que se inicie la ejecución de la tarea de salida.

- *Unión total*. Para que se inicie la ejecución de la tarea de salida es necesario que todas las tareas de entrada hayan finalizado. Al igual que en el caso de la bifurcación total, no hay condiciones asociadas a los caminos de salida[19].
- *Unión parcial*. En este caso, no es necesario que haya finalizado la ejecución de todas las tareas de entrada, sino que es posible especificar el número concreto de tareas de entrada que deben haber finalizado para iniciar la ejecución de la tarea de salida, siempre y cuando las condiciones asociadas sean ciertas.

Las condiciones de transición, tanto de bifurcación como de unión, abren la posibilidad de tener distintos caminos dentro del proceso y que la ejecución de las actividades y subprocesos que lo componen no sea únicamente secuencial. La clasificación aquí presentada se ampliará con la definición de nuevos subtipos cuando definamos un lenguaje gráfico para el modelado de flujos de trabajo (sección 6.1.2).

[19] Realmente se asume que la condición asociada a todos los flujos de entrada es *cierto*.

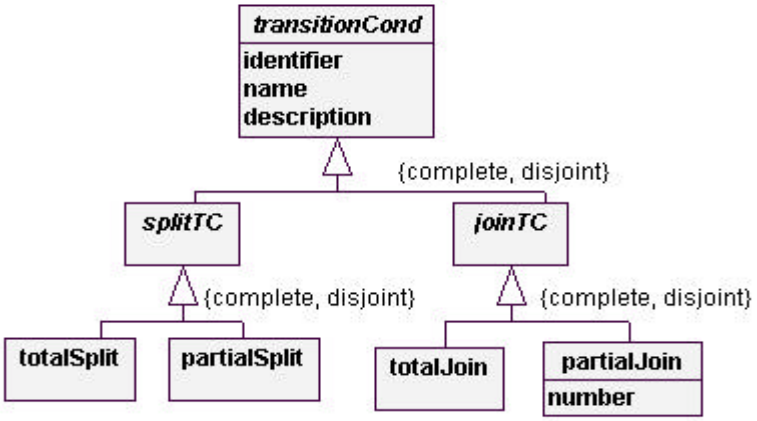


Figura 4-9. Jerarquía de generalización para las condiciones de transición

La Figura 4-9 muestra el metamodelo correspondiente a las condiciones de transición, de acuerdo con la descripción previa. Las condiciones de transición, como el resto de tareas que componen el proceso, son modeladas como clases, estableciéndose una jerarquía de generalización para modelar los distintos tipos de condiciones que pueden existir. En esta jerarquía, las clases que representan genéricamente a las condiciones de transición, tanto de unión como de bifurcación, son clases abstractas, denominadas tansitionCond, joinTC y splitTC, respectivamente. El resto de clases de la jerarquía no son abstractas y se corresponden con cada uno de los distintos tipos de condiciones consideradas en el modelo. Las subclases totalSplit y partialSplit representan la bifurcación total y parcial, respectivamente; y las clases totalJoin y partialJoin la unión total y parcial, respectivamente. Las propiedades de la clase transitionCond que aparecen en el diagrama son, en realidad, heredadas de la clase task.

Recopilando lo visto hasta ahora, la Figura 4-10 muestra el metamodelo de un proceso, sin tener en cuenta ni el flujo de control que conecta las tareas, ni los recursos utilizados por las mismas. De acuerdo con dicho metamodelo, vemos que un proceso está formado por al menos una actividad[20], por cero o más subprocesos y por cero o más condiciones de transición[21].

[20] No se tiene en cuenta la actividad inicial y la actividad final.

[21] Para facilitar la comprensión de la figura sólo se muestra para las condiciones de transición las subclases joinTC y splitTC (Condiciones de unión y bifurcación, respectivamente).

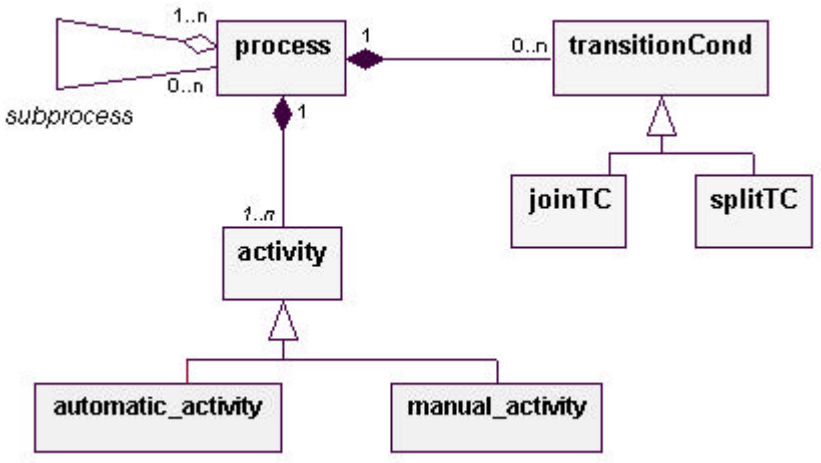


Figura 4-10. Metamodelo de proceso (*sin flujo de control*)

4.3.4 Flujo de Control

Un flujo de control conecta dos tareas del proceso, especificando la secuencia correcta de ejecución. Siempre tiene una tarea origen y otra destino. Las actividades inicial y final, puesto que delimitan el inicio y final de un proceso, sólo admiten un flujo de salida o uno de entrada, respectivamente. El resto de actividades, al igual que los subprocesos, admiten dos flujos: uno de entrada y otro de salida, puesto que siempre van precedidos y sucedidos por una sola tarea. Las tareas que sí pueden tener varios flujos de salida o varios flujos de entrada son las condiciones de transición, según los distintos tipos comentados en la sección anterior. En este caso, los flujos de control pueden tener asociadas condiciones para el caso de unión o bifurcación parcial.

El flujo de control, puesto que conecta dos tareas, se modela como una relación recursiva en la clase task, en lugar de como clases (ver Figura 4-11). Se etiquetan los roles de la asociación para distinguir entre la relación de precedencia (prior) y la de sucesión (next). La cardinalidad de estas relaciones siempre debe ser de 0..1, salvo para las condiciones de unión o de bifurcación, donde puede ser de 0..n en algunos casos (dependiendo del tipo de condiciones de transición, pueden haber varios flujos de entrada o de salida). En el modelo, la cardinalidad de la relación es de cero a muchos en ambos sentidos, pero se restringe con la siguiente especificación en OCL[22] [War99]. También se restringue para el caso de una actividad inicial y una final.

[22] *Object Constraint Language*

// restricción OCL

{context task

inv: ¬ self.oclIsTypeOf (splitTC) implies self.next → size ≤ 1

inv: ¬ self.oclIsTypeOf (joinTC) implies self.prior → size ≤ 1}

{context activity

inv: self.initial → notEmpty implies self.prior → isEmpty

inv: self.end → notEmpty implies self.next → isEmpty}

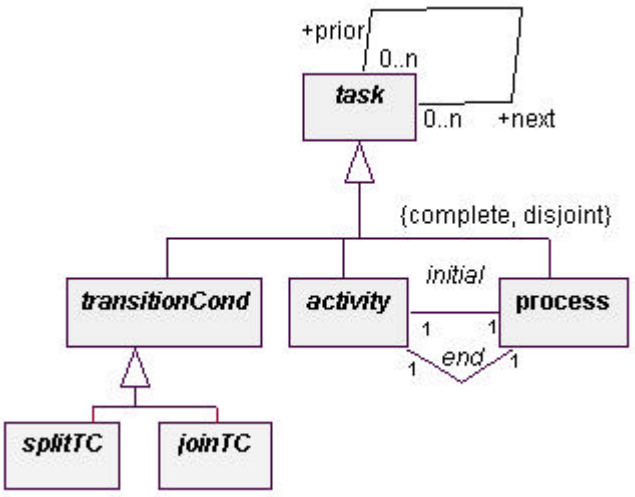


Figura 4-11. Metamodelo de flujo de control

4.4 Recursos

Los *recursos* se pueden asociar a las tareas que componen un proceso. En nuestro modelo, el término recurso engloba tanto la estructura organizativa como la infraestructura de tecnologías de la información de la organización para la cual se está modelando el flujo de trabajo. Concretamente, los recursos pueden ser de tres tipos: *actores*, *datos* y *aplicaciones*. El recurso *actor* representa la participación humana en el flujo de trabajo y se corresponde con la dimensión denominada estructura organizativa. La dimensión relativa a la infraestructura de la organización, en nuestro modelo está representada por los recursos denominados *datos* y *aplicaciones*. Los *datos* representan la información que maneja el flujo de trabajo, mientras

que las *aplicaciones* representan los programas invocados como parte de la ejecución del flujo de trabajo.

Los recursos se modelan como clases, al igual que el resto de componentes del flujo de trabajo exceptuando al flujo de control. Puesto que los recursos pueden ser de tres tipos, existe una jerarquía de generalización tal como se muestra en la Figura 4-12. La clase denominada resource es una clase abstracta que representa de forma genérica a los recursos. En dicha figura se muestran las propiedades comunes a todos ellos, en concreto, un identificador (identifier), un nombre (name) y una descripción (description). Utilizando el operador de especialización definimos las clases que representan a recursos concretos. La clase actor representa a los actores, la clase data representa a los datos, y la clase application representa a las aplicaciones. En las siguientes secciones se da una descripción más detallada de cada uno de los ellos.

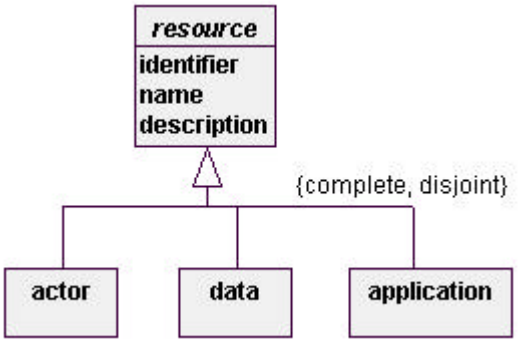


Figura 4-12. Jerarquía de generalización para los recursos

4.4.1 Aplicaciones

Las aplicaciones representan los programas que se invocan como parte de la ejecución de una o más actividades automáticas. Estos programas externos pueden ser aplicaciones informáticas genéricas que se utilizan para la realización de ciertas tareas, o incluso podría tratarse de software ya existente en la organización (*legacy systems*) que se sigue utilizando para ciertas tareas muy concretas.

Las aplicaciones externas que vayan a ser invocadas desde el flujo de trabajo, deben ser registradas previamente en el propio sistema, almacenándose la información necesaria para que puedan ser invocadas local o remotamente de forma correcta. Básicamente esta información es el identificador, nombre, descripción, ubicación física (ruta de acceso a la aplicación, ya sea estructura de directorios local o en otras máquinas), llamada y parámetros para su invocación, permisos de acceso necesarios, y finalmente, también interesa conocer el estado de una aplicación, es decir, si la aplicación se encuentra en ejecución o ya ha

finalizado. Al igual que para un proceso o una actividad, la información del estado sólo es relevante una vez el flujo de trabajo esté en ejecución.

La clase application (ver Figura 4-14) es una clase que representa genéricamente a una aplicación que va a ser invocada desde una actividad para su realización. De acuerdo con la descripción anterior, todo objeto de la clase application tiene un identificador (identifier), un nombre (name), una descripción (description)[23], una ubicación física (path), una forma de ser invocada que comprende tanto la llamada como parámetros necesarios (propiedad call), permisos de acceso en caso de ser necesarios (permissions) y estado de la aplicación (state). Precisamente, el valor de está última propiedad viene determinado por los posibles estados en los que se puede encontrar un objeto aplicación.

La Figura 4-13 muestra el diagrama de transición de estados definido para la clase application. En él se pueden observar tres estados relevantes que aparecen denotados por active, running y executed. El objeto que representa la aplicación que se va a invocar se encuentra en estado active cuando acaba de ser creado, se ha invocado la aplicación pero aún no ha iniciado su ejecución. El estado running denota que la aplicación invocada se está ejecutando y finalmente, el estado executed indica que la aplicación ha finalizado su ejecución.

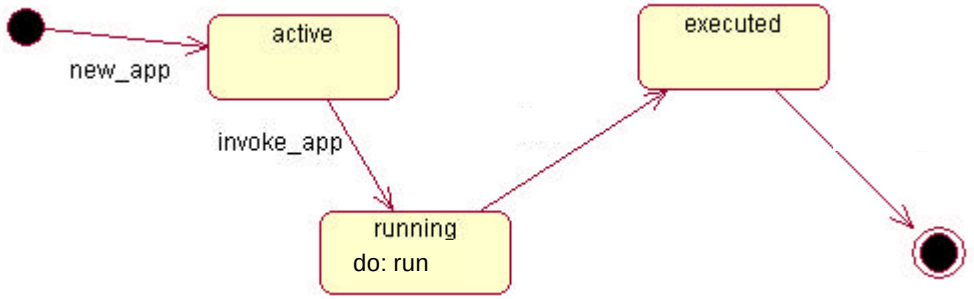


Figura 4-13. Diagrama de transición de estado de la clase application

Puede ocurrir que dependiendo del tipo de aplicación concreta que se vaya a ejecutar y del entorno en que se ejecute, se necesite más información específica. En dicho caso, es posible definir toda una jerarquía de generalización que recoja dichas particularidades. La Figura 4-14 muestra la clase application y una posible jerarquía de herencia, para distintos tipos de aplicaciones. Concretamente, aparece una clase denominada corbaProgram, en la cual se incluye información específica adicional que se necesita para invocar la ejecución de aplicaciones CORBA, tal como nombre del servidor (serverName) y código de retorno de la aplicación, que indique si ha habido un error durante la ejecución (exceptionCode). También aparece la clase sapr3Program, para los programas de SAP R/3 [SAP], que incluye nuevas propiedades, puesto que en este caso, los permisos de acceso vienen dados para un usuario

[23] Las propiedades identifier, name y description son, en realidad, heredadas de la clase resource.

concreto con una palabra de paso asignada y un cliente (propiedades user, password y client respectivamente). Esta jerarquía se puede extender todo lo que se necesite en cada dominio particular, con información acerca de las aplicaciones que se van a invocar mucho más específica y dependiente del tipo de aplicación.

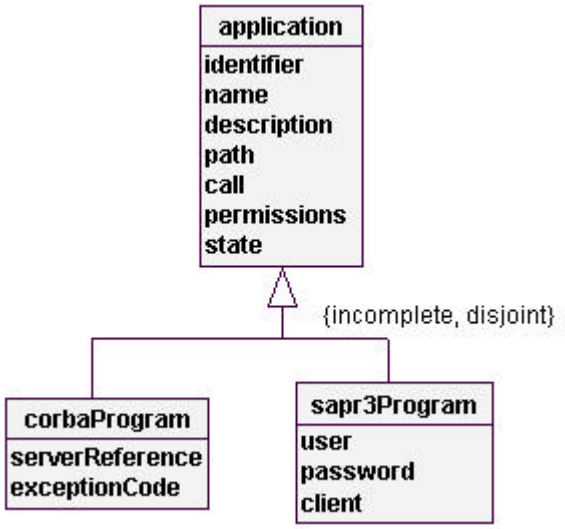


Figura 4-14. La clase application

4.4.2 Datos

Los datos representan la información que necesita el flujo de trabajo. En la definición del mismo, los datos van asociados a las tareas que componen un proceso, que los utilizan en su ejecución. Más concretamente, los datos son utilizados por actividades concretas o como parte de la evaluación de las condiciones de transición. En el caso de los subprocesos el tratamiento respecto a los datos es el mismo que para un proceso.

Los datos pueden ser tanto de entrada como de salida y normalmente son persistentes, estando almacenados en lo que genéricamente podemos llamar *repositorio*. Cuando una actividad comienza su ejecución, consulta del repositorio los datos de entrada que necesite. Durante la ejecución de la actividad o al finalizar la misma, se almacenan en dicho repositorio cualquier dato de salida que se haya podido generar.

Aquellos datos que vayan a ser utilizados en el flujo de trabajo, al igual que ocurre con las aplicaciones, deben ser registrados en el propio sistema, almacenándose la información necesaria para que se pueda acceder a ellos correctamente. Básicamente esta información es el nombre, descripción y ubicación física (ruta de acceso a los datos, ya sea estructura de

directorios local o en otras máquinas). En el caso concreto de los datos, en muchas ocasiones ocurre que el flujo de trabajo no utiliza solamente datos ya definidos y por lo tanto, existentes en una base de datos, un fichero de texto o cualquier otro formato, sino que es necesario definirlos al definir el flujo de trabajo. Por ejemplo, puede ocurrir que ciertas actividades o condiciones que componen el flujo de trabajo utilicen desde simples variables que almacenen un valor hasta objetos mucho más complejos, por lo que el metamodelo debe permitir definir dichos datos.

Los datos también se modelan como clases. La clase data es una clase abstracta que representa genéricamente todo dato utilizado por las tareas del proceso. De acuerdo con la descripción anterior, todo objeto de la clase data tiene un identificador (propiedad identifier), un nombre (name), una descripción (description)[24] y una ubicación física (path).

De forma similar a como ocurre en las aplicaciones, utilizando el operador de especialización creamos una jerarquía a partir de esta clase abstracta, definiendo nuevas clases que representen el tipo de datos concretos que utiliza un cierto proceso. En esta especialización se añaden nuevos atributos y métodos, que permiten modelar la estructura y comportamiento concreto del dato utilizado.

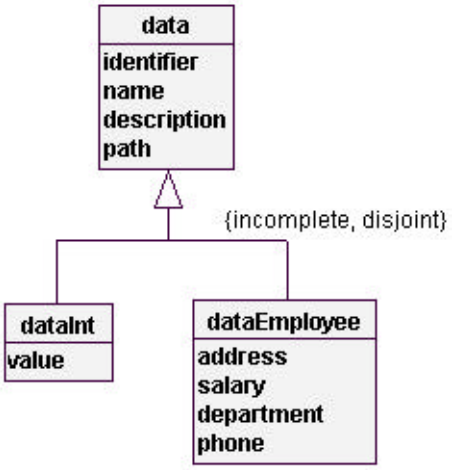


Figura 4-15. La clase data

La Figura 4-15 muestra la clase que representa genéricamente a los datos y algún ejemplo de clases especializadas. Por una parte aparece la clase que representaría los enteros que utilice el flujo de trabajo (dataInt), y por otra aparece una clase que representa entidades de un sistema de información con el que interactúa el flujo de trabajo, para extraer información o para actualizarla (dataEmployee, por ejemplo). La definición de nuevas clases

[24] Las propiedades identifier, name y description son, en realidad, heredadas de la clase resource.

especializadas depende de los tipos de datos que maneje el flujo de trabajo que opera sobre un determinado sistema de información; en cualquier caso, el operador de especialización asegura poder ir definiendo estas nuevas clases en función de los datos utilizados.

4.4.3 Actores y Modelo Organizacional

Con la noción de *actor* denotamos la participación humana en el flujo de trabajo. En este sentido, un proceso puede ser iniciado (o no) por un actor, de la misma forma que una actividad puede ser llevada a cabo por cero o más actores. Los actores que participan en el flujo de trabajo deben ser registrados, es decir, modelados y representados en el proceso que lo define, dando su nombre y una descripción del mismo.

El concepto de actor engloba tanto a cada uno de los *usuarios* que individualmente participan en el proceso como a los posibles grupos de usuarios que puedan participar en el mismo, tales como departamentos, grupos de trabajo, etc. Por ello, se registrará individualmente a cada uno de los usuarios y además se podrán registrar los grupos de usuarios que existan, a los que llamamos de forma genérica *unidades organizacionales*. Una unidad organizacional puede estar formada por otras unidades organizacionales. Para registrar a cada unidad organizacional se indicará cuáles son los usuarios que la forman, uno de los cuales será el responsable de la misma. Cuando a una actividad o a un proceso se le asocie como actor una unidad organizacional, cualquier componente de la misma puede ser el que realmente participe en la ejecución de la actividad o del proceso. En caso de que el actor sea un usuario concreto, sólo éste será el que participe.

Otra noción que tiene que ver con el concepto de actor es la de *rol*. Esta noción representa el hecho de que un usuario pueda ser sustituido por otro (que pasa a desempeñar la función de aquél). Esto significa que para cada unidad organizacional, que se registre en el sistema, se debe indicar, quién o quienes van a ser los sustitutos de sus miembros, si existen.

Modelar la participación humana en el flujo de trabajo, supone tener, en definitiva, un modelo organizacional que permita representar una jerarquía de actores que van a realizar diversas actividades dentro de la organización a la que pertenecen. La Figura 4-16 muestra el modelo organizacional propuesto, que es similar a otros modelos organizacionales utilizados en literatura de flujos de trabajo [Hol95, Ley00]. Si nos fijamos en el modelo, toda organización (representada por la clase organization) consta de un conjunto genérico de actores (representado por la clase abstracta actor). Una jerarquía de herencia define los distintos tipos de actores que pueden existir. En este caso, existen dos clases que son relevantes. Por una parte, la clase user que representa a los usuarios individuales, y por otra parte, la clase organizational_unit que representa a los grupos de usuario, que denominamos unidades organizacionales. La sustitución de un usuario por otro, en la realización de una cierta actividad, es modelada mediante una relación recursiva, que tiene asociada una clase (role).

La jerarquía existente en la organización es modelada mediante tres relaciones. En primer lugar, dos relaciones de composición (denominadas has_units y has_users) modelan, respectivamente, el hecho de que una unidad organizacional puede estar formada por otras unidades organizacionales, y que una unidad organizacional está formada por usuarios. Los usuarios pueden pertenecer a una o más unidades organizacionales, mientras que una unidad organizacional sólo puede formar parte de otra unidad organizacional de nivel más alto. En segundo lugar, la relación denominada managed_by modela el hecho de que toda unidad organizacional tiene, entre los usuarios que la componen, un responsable que es quien la gestiona.

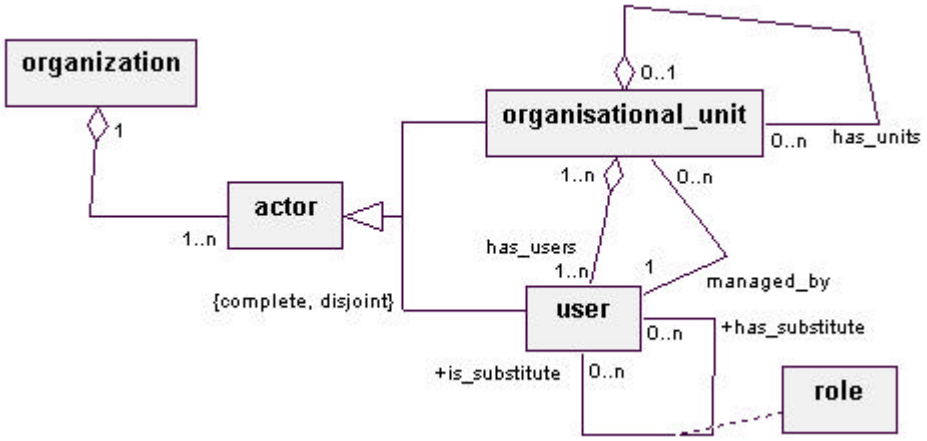


Figura 4-16. Modelo organizacional (Actores)

4.5 Metamodelo de Flujo de Trabajo de Referencia

En esta sección se resume e integra lo presentado en las secciones anteriores mostrándose el metamodelo de flujo de referencia completo. Para ello, y de acuerdo con lo visto en la sección 4.2, en la cual se indica que un flujo de trabajo está formado por un proceso y tiene asociado un conjunto de recursos que son utilizados por las tareas que componen el proceso, la Figura 4-17 muestra la utilización de los recursos por las distintas tareas, sin tener en cuenta las relaciones que aparecen al modelar el flujo de control. No se han incluido ni los atributos de las clases ni las jerarquias de herencia de los recursos para una mayor claridad.

A continuación analizamos las relaciones existentes entre las tareas y los recursos. Las aplicaciones son utilizadas por las actividades automáticas, de forma que toda actividad automática invoca en sus acciones a una aplicación (relación invokes).

Los datos son utilizados por las actividades y por las condiciones de transición. En la utilización de los datos por parte de las actividades se distingue si éstos son de entrada, es decir, la actividad consulta dichos datos para realizar con ellos cualquier operación (relación uses), o bien son de salida, siendo generados o actualizados como consecuencia de la ejecución de dicha actividad (relación returns). Estas dos relaciones definen el flujo de datos dentro del flujo de trabajo. En cuanto a las condiciones de transición, éstas también pueden utilizar los datos como parte de las condiciones que se deben evaluar para determinar por dónde continua el flujo de control. En este caso, los datos siempre son de entrada puesto que se consultan para evaluar las fórmulas, pero no se actualizan. La utilización de los datos por las condiciones se modela con la relación uses.

Finalmente, los actores pueden estar relacionados con las actividades y con el propio proceso asociado al flujo de trabajo. La relación entre un actor y un proceso representa que dicho actor es el responsable de dicho proceso; esto implica que cualquier incidencia que ocurra durante la ejecución del proceso debe ser notificada al actor. Esta relación se denomina en el modelo responsible. Por otra parte, también puede existir una relación entre uno o más actores y una actividad que compone el proceso. Esto significa que dichos actores son responsables de dicha actividad (relación may_have). En este caso se distingue si la actividad es manual o automática. Si la actividad es manual, la relación se redefine, entendiéndose que dicho actor no sólo es el responsable de la actividad, sino que también es quien debe realizarla. Necesariamente toda actividad manual debe tener asociado al menos un actor (relación has).

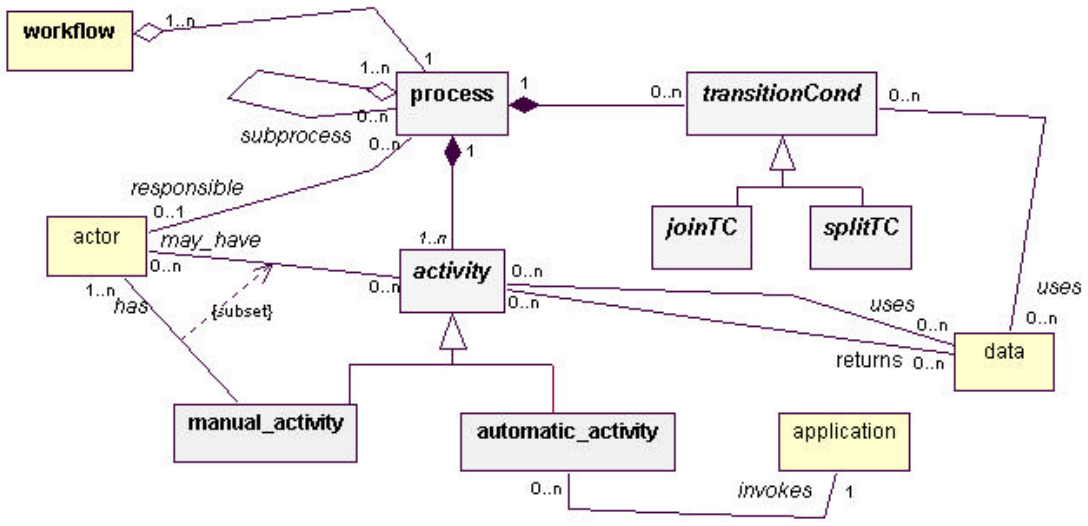


Figura 4-17. Metamodelo de flujo de trabajo (*sin flujo de control*)

4.6 Conclusiones

En este capítulo se ha presentado el metamodelo que se toma como referencia en los distintos subprocesos de desarrollo de flujos de trabajo. De acuerdo con dicho metamodelo, un flujo de trabajo está formado por un proceso que consta de un conjunto de tareas relacionadas mediante flujos de control; este flujo determina el orden de ejecución de las tareas. Además las tareas pueden utilizar recursos en su ejecución; estos recursos pueden ser datos, aplicaciones y actores.

El metamodelo presentado está de acuerdo con el propuesto por la WfMC (comentado en la sección 2.4.1). En ambos casos, los elementos básicos a considerar en el modelado de procesos o flujos de trabajo son clases; sin embargo, en el metamodelo definido se han refinado más estos elementos. Por ejemplo, se introduce el concepto de subproceso para representar aquellas actividades no atómicas. Se detallan los tipos de condiciones de transición y se establece una clasificación en condiciones de unión y de bifurcación, similar a la que aparece en [Cas95], considerándose tareas, al igual que las actividades y distinguiéndose del propio flujo de control. Éste aparece de forma explícita en el metamodelo, dando cuenta de las distintas restricciones existentes al unir las tareas. Finalmente, en cuanto a la participación humana, el concepto de rol que aparece en el metamodelo de la WfMC también se ha refinado definiendo un modelo organizacional, que sigue una aproximación centrada en unidades organizacionales [Ley00].

Capítulo 5.

Formalización del Metamodelo de Flujo de Trabajo

En este capítulo se presenta la formalización del metamodelo de flujo de trabajo de referencia. Esta formalización proporciona una semántica precisa que permite abordar el desarrollo de modelos de flujo de trabajo de calidad con garantías de éxito, tanto en la definición del mismo, como en su ejecución y posterior análisis. El metamodelo se ha formalizado en OASIS, un lenguaje de especificación formal y orientado a objetos [Pas95a, Can96a, Let98] .

La estructura del capítulo es la siguiente. En la sección 5.1 se justifica la necesidad de disponer de un metamodelo con una semántica formal bien definida y la elección de OASIS como marco de formalización. En la sección 5.2 se resumen las principales características de OASIS que son relevantes para este trabajo, tanto respecto al modelo objetual soportado como al lenguaje asociado. La sección 5.3 muestra la representación OASIS del metamodelo de referencia y cómo la expresividad de dicho lenguaje permite abordar el modelado conceptual de flujos de trabajo desde una perspectiva formal. Finalmente, en la sección 5.4 se muestran las conclusiones.

5.1 Justificación

Una de las principales limitaciones de los SGFT (ya mencionada en el capítulo 2) es la falta de un metamodelo de flujo de trabajo aceptado e implementado por la mayor parte de las herramientas existentes en el mercado. Esta falta de normalización en el metamodelo incide

de forma importante en las prestaciones que los SGFT ofrecen a otros niveles, como puede ser en la propia definición del flujo de trabajo. El modelo que se construye y las herramientas que proporciona el sistema son, en la mayoría de las ocasiones, muy dependientes de las propias facilidades de ejecución soportadas por cada sistema. Además, la falta de una semántica precisa asociada al metamodelo incide de forma directa en otros problemas importantes de los SGFT, como son la falta de compatibilidad entre diferentes sistemas (lo cual dificulta enormemente la interoperabilidad) [Alo98].

Uno de los objetivos del presente trabajo es, no sólo definir un metamodelo de referencia, sino dotarlo de una semántica formal. Esto permite abordar el desarrollo de modelos de flujo de trabajo de calidad, con ciertas garantías de éxito, a lo largo de un ciclo de vida concreto que abarca desde su definición hasta su mejora, pasando por la ejecución del mismo.

El metamodelo de referencia se ha definido utilizando UML. Este lenguaje se ha convertido en un estándar a utilizar en el campo del modelado conceptual, siendo cada vez mayor su implantación a todos los niveles, tanto en el ámbito comercial como a nivel de investigación. Pero un inconveniente importante de UML es la falta de una semántica precisa asociada al mismo, y a pesar de la gran cantidad de trabajos de investigación que se están realizando en este sentido [PUM, Fer00], no existe un consenso o una formalización del mismo ampliamente aceptada, que abarque la expresividad de proceso.

Para garantizar una semántica precisa al metamodelo de referencia necesitamos formalizarlo. En este sentido, OASIS se plantea como marco de formalización puesto que reune dos requisitos importantes: es formal y además orientado a objetos. Ahora bien, la primera cuestión que surge es si la expresividad del lenguaje OASIS y del modelo objetual subyacente permiten expresar qué es un flujo de trabajo, sus componentes y las relaciones existentes entre ellos. El resto del capítulo da respuesta a esta cuestión.

5.2 OASIS

OASIS[25] es una aproximación formal, declarativa y orientada a objetos a la especificación de sistemas de información. En esta sección se da una visión general de OASIS con la finalidad de conocer los aspectos más relevantes del mismo que hacen posible la formalización del metamodelo de flujo de trabajo propuesto. En concreto, se resumen las principales características del modelo de objetos que soporta OASIS, el marco formal elegido, la formalización de los conceptos básicos, la visión de OASIS como lenguaje, no sólo de definición, sino también de manipulación y programación de aplicaciones, y finalmente, la introducción de un meta nivel que da cuenta de aspectos de evolución.

[25] *pen and **A**ctive **S**pecification of **I**nformation **S**ystems*

5.2.1 El Modelo de Objetos OASIS

En OASIS, los bloques básicos de construcción de los sistemas de información son los *objetos*. Un **objeto** OASIS es un proceso observable [Ram93] que encapsula estado y comportamiento. Cada objeto tiene un identificador (oid)[1] que será único y lo distingue de cualquier otro objeto que haya existido, exista o pueda existir. Los objetos poseen propiedades, representadas en el modelo a través de la noción de *atributo*.

El estado de un objeto viene dado por el conjunto de valores de sus atributos, de forma que un cambio en alguno de dichos valores representará un cambio de estado. Básicamente se distinguen entre cambios de estado fuertes (aquellos que suponen la creación / destrucción del objeto) y cambios de estado débiles (cambia el valor de los atributos del objeto sin que esto implique su destrucción). La causa de que se produzca un cambio de estado en un objeto se denomina *evento*. Un evento se define como la abstracción de un cambio de estado; es discreto, no tiene duración y ocurre en un cierto instante de tiempo. En el modelo OASIS, distinguimos entre eventos propios y eventos compartidos, según que participen en la vida de uno o más objetos.

La ocurrencia de un evento en la vida de un objeto sólo tendrá éxito si el objeto se encuentra en un estado que verifica la *precondición* asociada a dicho evento. Además no todos los estados de los objetos van a ser estados permitidos, las *restricciones de integridad* definen precisamente qué estados serán aceptables en la vida de los objetos, en términos de valores permitidos para los atributos.

Los objetos no están aislados. La interacción entre objetos se modeliza en OASIS de dos formas: mediante compartición de eventos y relaciones de disparo. Las relaciones de disparo, representadas en el modelo OASIS por *triggers*, introducen actividad en la sociedad de objetos, permitiendo que un objeto actúe como agente de un evento de otro objeto cuando se satisfagan en él determinadas condiciones.

En el modelo OASIS, podemos definir *transacciones*, de manera que un conjunto de eventos funcionen como una unidad de ejecución de mayor granularidad. Siguen los dos principios básicos de toda transacción, a saber: todo o nada y no observabilidad de estados intermedios. Además también podemos definir *procesos* usando los eventos y transacciones como acciones atómicas, permitiéndonos especificar las posibles vidas correctas de los objetos.

Los objetos no están aislados, sino que tendemos a agrupar en *clases* aquellos objetos que tienen la misma estructura y comportamiento. Un objeto individual es una *instancia* o ejemplar de una clase. La sociedad de objetos del modelo OASIS está compuesta de las siguientes clases de objetos:

[1] En inglés, ***o**bject **id**entifier*.

- Las **clases primitivas** o dominios están constituidas por objetos que tienen un único estado y que siempre existen. En general, abarcan el dominio de los tipos abstractos de datos (TAD). La sociedad de objetos se construirá tomándolos como nivel estructural básico sobre el que se declaran las demás clases de objetos. En toda especificación OASIS habrá varios dominios predefinidos: los números naturales (nat), el tipo *bool* y las cadenas de caracteres (string, char). Además, pueden definirse explícitamente otros, a través de sus correspondientes TAD.
- A partir de los dominios se construyen las **clases elementales**. Estas vienen caracterizadas por el tipo que una colección de objetos comparte. En el tipo se definen los atributos, los eventos, las restricciones de integridad, las precondiciones, las relaciones de disparo, las transacciones y la parte proceso que caracterizan a los objetos de la clase.
- Mediante un mecanismo constructivo, se definen las **clases complejas** a partir de las clases elementales o de otras clases complejas definidas de antemano, aplicando sobre ellas los operadores de clases**.** Los operadores más utilizados son la especialización y la agregación, aunque existen otros como la generalización (operador inverso a la especialización), la asociación (un tipo particular de agregación que abarca la noción de colección) y la composición paralela (un operador que viene de la Teoría de Procesos y nos permite definir un sistema organizacional como el resultante de la reunión concurrente de las clases definidas con anterioridad). Todos los operadores de clases mencionados anteriormente son *ortogonales*, es decir, son independientes unos de otros y es posible cualquier combinación entre ellos.

El modelo objetual descrito está soportado por un lenguaje denominado también OASIS. El lenguaje está basado en la lógica y permite la definición de esquemas conceptuales según el modelo orientado a objetos presentado. Existen diferentes versiones del lenguaje dependiendo de la expresividad lógica que se utilice. Las más destacadas son R-OASIS (expresividad clausal), F-OASIS (expresividad funcional) y L-OASIS (expresividad lógico-ecuacional), todas ellas definidas en [Pas92]. En este trabajo, utilizaremos la versión de OASIS basada en la lógica clausal dinámica (R-OASIS). Una definición completa y detallada de R-OASIS se puede encontrar en [Can96].

5.2.2 Marco Formal: La Lógica Dinámica

La Lógica Dinámica fue propuesta por D. Harel [Har84]. Es una lógica modal cuyo objetivo es razonar acerca de los programas que observan y cambian un entorno, de modo que el efecto de la ejecución de un programa pueda ser descrito formalmente como paso previo a procesos de razonamiento formal acerca de los mismos.

Una fórmula dinámica puede escribirse en la forma ϕ [p] ψ , donde ϕ y ψ son fórmulas de primer orden, p representa a un programa (en su sentido más amplio) y [.] es el operador de necesidad. El significado intuitivo de la fórmula puede enunciarse de la manera siguiente: "si la fórmula ϕ es válida, entonces tras la ejecución del programa p necesariamente ψ ha de ser válida".

Las fórmulas de primer orden se evalúan en un único mundo o estructura de interpretación, mientras que las fórmulas de la lógica dinámica se evalúan en más de uno, puesto que son fórmulas que describen una evolución, un cierto cambio de estado. Una teoría formal dinámica se interpreta sobre una *estructura de Kripke* K = (Ω, ω_0, ρ), donde: Ω es un conjunto de mundos posibles, siendo cada uno de ellos una estructura de interpretación de primer orden; ω_0 es un mundo inicial, y ρ es la relación de accesibilidad entre mundos (ver Figura 5-1).

Este lenguaje permite describir con detalle los efectos de la ejecución de un programa. En concreto, esta expresividad es utilizada para dar cuenta de los cambios de estado de los objetos producidos en general como respuesta a ciertos estímulos externos (que podríamos llamar mensajes, eventos, transacciones, etc., dependiendo del modelo utilizado).

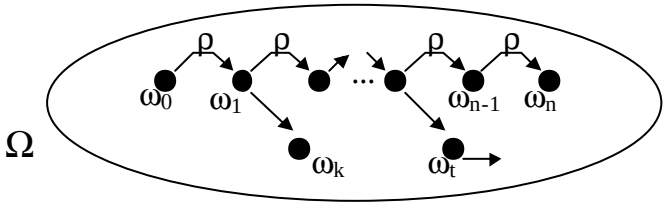


Figura 5-1. Estructura de Kripke simple

5.2.3 Formalización de los Conceptos Básicos OASIS

En esta sección se presenta la formalización de los principales conceptos OASIS de acuerdo con la Lógica Dinámica.

a) ***Noción de Tipo***. Llamamos tipo a un conjunto de propiedades de estructura (características estáticas) y comportamiento (características dinámicas) que son compartidas por una colección de objetos.

Un *tipo* es una tupla (*A, X, F, P*), donde:

- *A* es un conjunto de *atributos* definido como $A = A_c \ È \ A_v \ È \ A_d$, siendo A_c los *atributos constantes*, que toman un valor en el momento de creación del objeto y

éste no cambia durante toda la vida del mismo; A_v son los *atributos variables*, cuyo valor viene definido por *reglas de valuación* que expresan cómo varía su valor por la ocurrencia de eventos; y finalmente A_d son los *atributos derivados*, cuyo valor se obtiene a partir de otros atributos mediante ciertas *reglas de derivación*.

- *X* es un conjunto de *eventos*, que se clasifican en *propios* y *compartidos*.
- Φ es un conjunto de fórmulas de la lógica dinámica, que incluye:
  - Φ_v , *reglas de valuación*, fórmulas que expresan cómo los atributos cambian de valor con la ocurrencia de eventos. La expresión en lógica dinámica tiene la forma: $\psi\,[e]\,\zeta$
  - Φ_d , *reglas de derivación*, fórmulas que expresan cómo el valor de los atributos derivados se obtiene a partir del resto de atributos del objeto. Se representa como: $\psi \rightarrow \zeta$
  - Φ_p , *precondiciones*, fórmulas que expresan bajo qué condiciones un evento va a ser relevante en la vida de un objeto. La expresión queda: $\neg\psi\,[e]\,\mathrm{false}$
  - Φ_t , *fórmulas de disparo*, que describen la actividad del sistema. La expresión en lógica dinámica sería: $\psi\,[\neg e]\,\mathrm{false}$
  - Φ_r, *restricciones de integridad*, que deben cumplirse en cualquier estado (o secuencia de estados) de los objetos. Se expresan mediante fórmulas de 1er orden y temporales.
- Π es un término del Álgebra Básica de Procesos (ABP) definida en [Wie93] que nos describe cuáles son las secuencias válidas de eventos.

b) ***Noción de Plantilla.*** Una *plantilla de objeto* es una tupla (*I, T, P*), donde *I* es la *función de identificación* de objetos, que será un tipo abstracto de datos (TAD) que cumple las propiedades de cardinalidad infinita y relación de igualdad definida; *T* es el tipo, y *P* es la *parte pública* de *T*, que representa la parte de los atributos y eventos definidos en el tipo que son accesibles desde fuera del objeto.

c) ***Noción de Objeto.*** Un objeto es un par (Σ, E), donde Σ es la plantilla y E representa el estado del objeto descrito mediante fórmulas de primer orden (concretamente, fórmulas atómicas básicas que expresan pares (atributo,valor)).

d) ***Noción de Clase.*** Una *clase* OASIS consiste en una *plantilla* de objetos Σ, una *factoría* de objetos que aporta los mecanismos necesarios para la creación y destrucción de objetos.

Además, una vez los objetos han sido creados pasan a formar parte de lo que se denomina la *población* de la clase hasta su destrucción.

Las clases son consideradas como objetos, y en la plantilla que representa una clase se incluyen atributos y eventos que permiten a la clase crear objetos. Más adelante, en la sección 5.2.5 se da una descripción más amplia de este aspecto (metanivel).

Como ya se ha mencionado, las clases OASIS pueden ser: primitivas, elementales o complejas. A continuación se incluye sólo la formalización de las clases complejas especialización y agregación por ser de interés para el trabajo.

- **Agregación**: La plantilla de una clase compuesta C construida como agregación de las clases componentes Ci, se define como (I, T, P), donde:

  I : a partir de las I_i

  T = ($A_{ci} \cup A_c$, $X_{ci} \cup X_c$, $\Phi_{ci} \cup \Phi_c$, Π)

  P subconjunto de T

  En el caso de la agregación es posible etiquetar a cada una de las clases componentes Ci, indicando el significado exacto con que se agrega a la clase C. Tenemos:

  → *Inclusivo/Relacional*, dependiendo de si los objetos componentes tienen una relación de inclusión fuerte o débil en el compuesto.

  → *No Disjunta/Disjunta*, dependiendo de si los objetos compuestos pueden o no compartir objetos componentes.

  → *Dinámica/Estática*, dependiendo de si tras la creación de los objetos compuestos, éstos pueden variar o no sus objetos componentes.

  → *Flexible/Estricta,* dependiendo de si los objetos componentes pueden existir o no sin formar parte de ningún objeto compuesto.

  → *Univaluada/Multivaluada,* dependiendo de si pueden formar parte del objeto compuesto uno o más de un objetos de la misma clase componente.

  → *Nulo/No Nulo,* dependiendo de si en algún instante se permite o no que el objeto compuesto no tenga ningún objeto de una clase componente asociado.

- **Especialización**: La plantilla de una clase E (clase hija o especializada) construida como especialización de una clase C (clase padre), se define como (I, T, P), donde:

  $I \sim I_c$

  $T = Tc \cup Te$

  $P = Pc \cup Pe$

Este operador da soporte a la noción de herencia. En OASIS se definen dos tipos de especialización:

→ *Temporal o rol*, permite que los objetos cambien su estado y comportamiento de forma dinámica. Cada plantilla de una clase especializada posee un evento propio de creación de instancias, que debe pertenecer a la signatura de eventos de la plantilla de la clase padre. Cuando dicho evento ocurre en la vida de un objeto, éste juego el rol descrito en la plantilla de la clase especializada.

→ *Permanente*, en este caso el objeto es instancia de la clase especializada desde el mismo momento de su creación debido al cumplimiento de una condición de especialización sobre el valor que toman algunos de los atributos constantes; el evento de creación de la clase padre e hija coinciden.

De acuerdo con todo ello, la Figura 5-4 muestra la plantilla a seguir para la especificación de una clase elemental OASIS, con las diferentes secciones del lenguaje que recogen la noción de tipo (Tipo=(*A, X, F, P*)).

Las clases complejas se especifican siguiendo esta misma plantilla, tan sólo varía la cabecera, en la que se indica el operador de clase utilizado en cada caso. La Figura 5-2 y Figura 5-3 recogen esquemáticamente la declaración de la cabecera de una clase especializada y una agregada, respectivamente; los puntos suspensivos indican que ahí aparece la plantilla de especificación de una clase OASIS, vista en Figura 5-4.

```
complex_class <nombre_clase >
    specialization_of {<nombre_superclase> [where <condición>][,]}.
  …
end_complex_class
```

Figura 5-2. Especificación de la cabecera de una clase especializada OASIS

```
complex_class <nombre_clase> aggregation_of
    {nombre_clase ([inclusive|relational] [static|dynamic]
                   [disjoint|nondisjoint] [strict|flexible]
                   [univalued|multivalued [(n)]] [null|not_null])}
  …
end_complex_class
```

Figura 5-3. Especificación de la cabecera de una clase agregada OASIS

```
class <nombre_clase>
        constant_attributes
            key <nombre_atributo>(<tipo_atributo>);
                ...
                <nombre_atributo>(<tipo_atributo>).
        constraints <declaracion_de_variables>.
            Static
            <restricciones_estaticas>.
            ...
        variable_attributes
            <nombre_atributo>(<tipo_atributo>)
            valuation <declaracion_de_variables>.
                <formulas_observabilidad_atributo>.
                …
            end_valuation
            …
        derived_attributes
            <nombre_atributo>(<tipo_atributo>)
            derivation <declaracion_de_variables>.
                <formulas_derivación_atributo>.
                ...
            end_derivation
            ...
        private_events <declaracion_de_variables>.
            New          <nombre_evento>(<arg_evento>);
            destroy<nombre_evento>(<arg_evento>);
                         ...
                         <nombre_evento>(<arg_evento>).
        shared_events <declaracion_de_variables>.
            <nombre_evento>(<arg_evento>) with <lista_clases>;
            ...
            <nombre_evento>(<arg_evento>) with <lista_clases>.
        preconditions <declaracion_de_variables>.
            <formula_ nstante ón _evento>.
            ...
        triggers <declaracion_de_variables>.
            <formula_disparo_evento>.
            ...
        transactions <declaracion_de_variables>.
            <formula_ nstante ón>.
            ...
        processes <declaracion_de_variables>.
            <formula_procesos>.
            ...
    end_class
```

Figura 5-4. Plantilla de especificación de una clase elemental OASIS

5.2.4 OASIS como Lenguaje Único para Bases de Datos Orientadas a Objetos

En [Can96] se extiende el lenguaje OASIS, que pasa de ser un lenguaje de especificación formal orientado a objetos y ejecutable, dentro de un paradigma de ciclo de vida basado en el prototipado rápido y la programación automática, a ser un lenguaje único de bases de datos orientadas a objetos, activas y deductivas. La versión del lenguaje denominada R-OASIS es tanto lenguaje de definición de objetos, como lenguaje de actualizaciones y consultas como lenguaje de programación de aplicaciones.

La extensión de OASIS como lenguaje de manipulación de objetos, supone la definición de dos nuevos símbolos de predicado para acceder a los mismos, distinguiendo entre observaciones y actualizaciones. Se sigue el patrón *destino-mensaje-argumentos.*

‘ **:: / 2** ‘ se utiliza para cualquier *actualización* que queramos realizar sobre un objeto de la base de datos.

‘ **?? / 2** ‘ se utiliza para cualquier *observación* que queramos realizar sobre un objeto de la base de datos.

5.2.4.1 Actualizaciones

Llevar a cabo una actualización sobre un objeto equivale a disparar en el objeto un evento de su signatura; la ocurrencia del evento provocará un cambio de estado en el objeto.

Una actualización de un objeto es un átomo de la forma: A::B

siendo A el oid del objeto y B un término cuyo funtor es el símbolo del evento que se desea disparar. Las actualizaciones deben estar completamente instanciadas y no se resatisfacen con la vuelta atrás o backtracking.

Ejemplo: Si se desea disparar el evento ev con argumentos arg1,... ,argn en un objeto obj, se puede construir la actualización siguiente:

obj :: ev(arg1, …, argn)

5.2.4.2 Observaciones

El objetivo de cualquier observación es obtener el valor de un atributo perteneciente a la signatura del objeto. Cuando se realiza una observación de un objeto, el estado del mismo no cambia.

Una observación sobre un objeto es un átomo de la forma: A ?? B

siendo A el oid del objeto y B un término cuyo funtor es el símbolo del atributo que se desea observar. En este caso, A debe estar siempre instanciado y B puede estar o no instanciado. Una observación no se resatisface con la vuelta atrás o backtracking.

Ejemplo: Si se desea conocer el valor del atributo atr de un objeto obj, se puede construir la siguiente observación, en la que la variable V se instanciará al valor actual del atributo. En caso de que V este instanciada, la consulta realizada es si el valor del atributo atr del objeto obj es V.

obj ?? atr(V)

Los predicados de observación y actualización pueden aparecer en cualquier fórmula, de modo que con ellos y el resto de predicados asociados a la descripción de un objeto se pueden construir consultas complejas, no consistentes tan sólo en el acceso a un atributo de un objeto, sino en un cierto procesamiento de la información obtenida.

5.2.4.3 Programas

Finalmente, OASIS es también lenguaje de programación de aplicaciones. Básicamente, esto supone enriquecer el apartado processes de la especificación de la clase. En versiones previas de OASIS, en este apartado se definen las posibles vidas de un objeto OASIS desde el punto de vista pasivo (sólo aparecen eventos de su signatura). En R-OASIS ésto se enriquece con la incorporación de una dimensión activa (la ocurrencia de un evento, supone que necesariamente ocurra otro).

De este modo, la descripción del proceso que es un objeto permite especificar las posibles vidas del mismo utilizando un lenguaje. Las accciones atómicas de este lenguaje son de dos tipos: eventos de la signatura del propio objeto, y aquéllos que el objeto puede solicitar de otros objetos del sistema. Para conectar las acciones se utilizan los operadores del álgebra básica de procesos [Wie93]. En la descripción del proceso también se pueden incluir condiciones que se evalúan sobre el estado del objeto.

De acuerdo con esto, la definición de proceso OASIS es la siguiente:

I. Un evento e y su negación ¬e son procesos.

II. Una observación oid??atributo es un proceso.

III. Una actualización oid::evento es un proceso.

IV. check(φ), donde φ es una fórmula de primer orden, es un proceso.

V. Si p1 y p2 son procesos, entonces p1 & p2 es un proceso. (Secuencia)

VI. Si p1 y p2 son procesos, entonces p1 or p2 es un proceso. (Alternativa)

VII. Si p es un procesos, entonces comTrans p finTrans es un proceso. (Transacción)

VIII. Sólo son procesos los definidos por I)..VII).

La semántica de un proceso OASIS aparece descrita en [Ram93, Can98], pero para un mayor entendimiento de lo que se va ha exponer a continuación, intuitivamente la semántica asociada es:

- Una secuencia de la forma e_i & oid::e_k, tiene el siguiente significado: "tras la ocurrencia del evento e_i, el objeto debe enviar un mensaje al objeto identificado como oid, invocando el evento e_k". Esto correspondería a un patrón de comportamiento reactivo que podemos calificar como *evento-acción*.
- Una secuencia de la forma c_i & oid::e_k, tiene el siguiente significado: "si se cumple la condición c_i, el objeto debe enviar un mensaje al objeto identificado como oid, invocando el evento e_k". Esto correspondería a un patrón de comportamiento reactivo que podemos calificar como *condición-acción*.
- Una secuencia de la forma e_i & c_i & oid::e_k, tiene el siguiente significado: "tras la ocurrencia del evento e_i, si se cumple la condición c_i, el objeto debe enviar un mensaje al objeto identificado como oid, invocando el evento e_k". Esto correspondería a un patrón de comportamiento reactivo que podemos calificar como *evento-condición-acción*.

En el caso en el que un objeto tenga su sección processess descrita mediante un término que sólo contenga acciones, es decir, que carezca de parte pasiva, nos encontramos ante lo que a todos los efectos puede considerarse como un programa; de hecho, su vida consistirá en el disparo, en el orden especificado por los operadores del álgebra de procesos, de eventos de otros objetos en los que se apoye para realizar los servicios para los que fue diseñado.

Un programa en OASIS es un objeto activo. La vida de tal objeto siempre comenzará por un evento de creación (start) y terminará por uno de destrucción (stop). Entre ambos, el código o cuerpo (body) del programa contendrá las acciones a realizar y que constituyen la vida del programa. La Figura 5-5 muestra la especificación OASIS de la clase que caracteriza en abstracto los programas OASIS y que se denomina class program.

En la definición de la clase programa, el subproceso body es un proceso trivial, no hace nada; este subproceso adquiere significado cuando se define un programa o aplicación. Cualquier nueva aplicación que se defina se hará especializando la clase programa y redefiniendo el apartado processess, en concreto, redefiniendo el subproceso body con las acciones a realizar.

Cada instancia de la clase program representará una ejecución distinta del mismo programa. Cada vez que un agente solicita a la clase programa que ocurra el evento de creación, se crea una nueva instancia de dicha clase, con su oid propio. El nuevo objeto creado, de acuerdo con la vida especificada en el apartado processes, va a actuar de agente y ejecutar el cuerpo del programa (parte activa). Una vez finaliza, el propio objeto se destruye.

```
class program
        constant_attributes
                key id(string).
        private_events
                        new     start;
                        destroy stop.
        processes
                program ← start & body & stop.
                Body.
end_class
```

Figura 5-5. Especificación OASIS de la clase program

En la Figura 5-6 se puede ver gráficamente la jerarquía que existe. Cómo a partir de la clase abstracta program, obtenemos una clase por cada uno de los distintos programas que deseemos definir, redefiniendo del apartado processes. En este caso, definimos programX, programY y programZ. La creación de un objeto de estas clases representa la ejecución del programa asociado a cada una de ellas (definido en la sección processes). Por lo tanto, las distintas instancias de cada clase especializada representa las distintas ejecuciones de dicho programa. Por ejemplo, para la clase programX, la acción programX::start crea una instancia Oid1, que representa la ejecución de dicho programa.

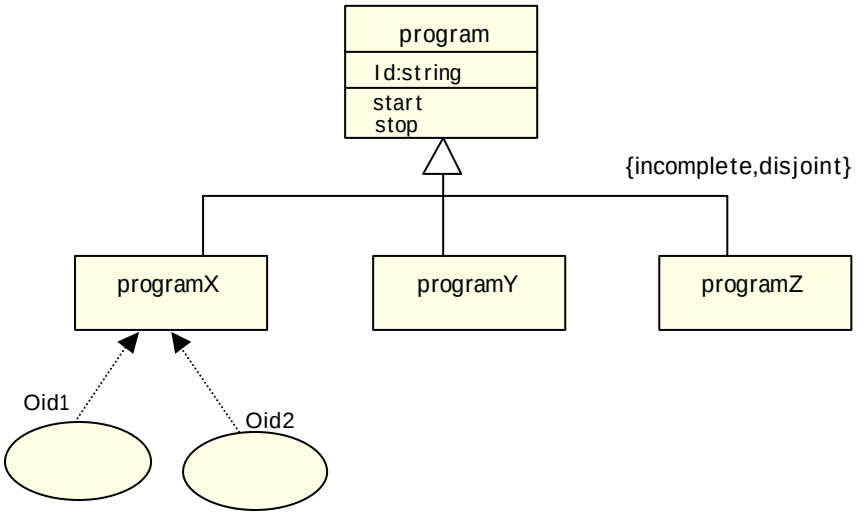


Figura 5-6. Jerarquía de la clase programa

5.2.5 La Metaclase OASIS

En [Ram95a, Ram95b, Can96] se introduce la noción de metaclase y se especifican los atributos y servicios de las clases OASIS vistas como objetos. En [Car99] se utiliza la noción de metaclase para dar cuenta de la evolución de los sistemas de información organizacionales dentro del propio modelo, formalizándose en TF-Logic [Bon95]. A continuación se da una visión general.

5.2.5.1 Metaobjetos

Una clase OASIS consiste en una plantilla, una factoría de objetos y un almacén de objetos. Con el objetivo de formalizar la funcionalidad de factoría y almacén de objetos se introduce en el modelo la noción de *metaobjeto*. Un metaobjeto es un objeto que puede ser visto desde dos puntos de vista diferentes pero complementarios: como *clase* y como *objeto*.

- Como *clase* ofrece los servicios de creación y destrucción de objetos y mantiene vivo el censo de objetos que han sido creados y todavía no han sido destruidos (la población).
- Como *objeto* ofrece los servicios que permiten definir el conjunto de propiedades que constituyen el tipo que compartirán todas sus instancias. Entre estos servicios se encuentran: *añadir un atributo, añadir un evento, añadir una transacción, añadir un disparo, cambiar una restricción, cambiar una precondición, cambiar la fórmula de evaluación de un evento, borrar un atributo, borrar un evento, borrar un disparo*, etc. La invocación de estos servicios modifica el estado del metaobjeto, representado por un conjunto de atributos, tales como: *conjunto de atributos, conjunto de eventos, conjunto de evaluaciones, conjunto de precondiciones*, etc.

En [Ram95b] aparece detallados todos los servicios y atributos de los metaobjetos. La Figura 5-7 muestra la representación gráfica del metaobjeto *usuario*, siguiendo esta aproximación.

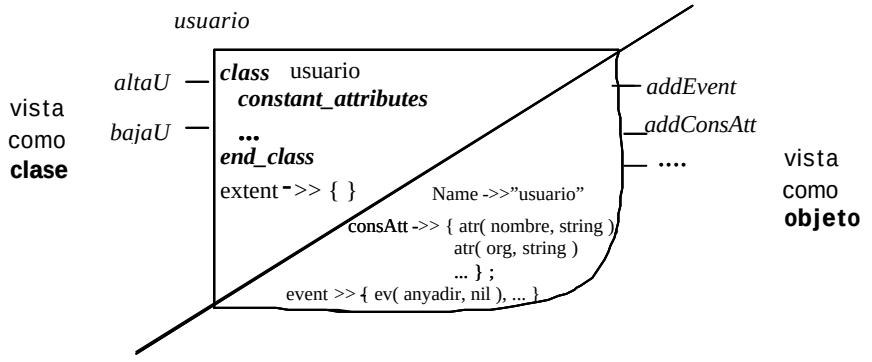


Figura 5-7. Doble visión de los metaobjetos

5.2.5.2 Metaclases

El concepto de metaclase se utiliza para denominar a aquellas clases cuyas instancias son clases. Bajo este punto de vista, los metaobjetos son instancias de metaclases.

En OASIS, las plantillas de las metaclases definen los atributos necesarios para almacenar el conjunto de fórmulas que definen la plantilla de una clase OASIS, los eventos para manipular dichos atributos, las precondiciones, las restricciones de integridad, las evaluaciones asociadas, etc. La Figura 5-8 muestra la jerarquía de generalización de metaclases de la metaclase OASIS. Esta jerarquía tiene en cuenta las unidades de diseño existentes y comentadas anteriormente (clases elementales, complejas y primitivas).

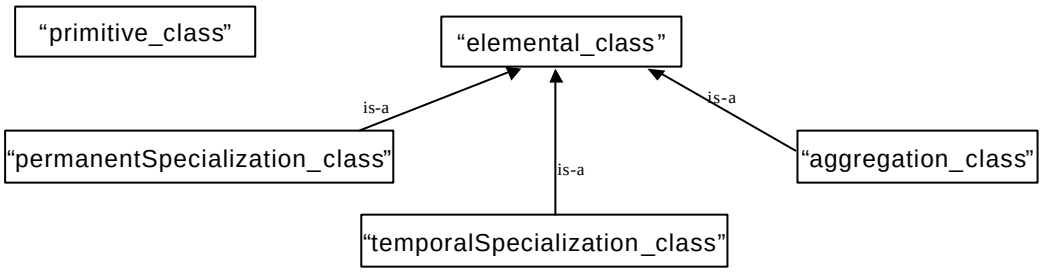


Figura 5-8. Jerarquía de generalización de metaclases

5.2.5.3 La Metaclase OASIS

Se denomina *metaclase OASIS* al esquema conceptual[26] que define las plantillas de las metaclases que permiten crear metaobjetos que con su estado definen las características de clases primitivas, elementales y complejas dentro modelo OASIS.

La metaclase OASIS ofrece únicamente dos servicios: creación y destrucción de instancias[27]. Además tiene como población el conjunto de esquemas conceptuales definidos y como plantilla la composición de las plantillas de las metaclases. Además, para eliminar la ciclicidad del modelo y no tener una jerarquía de potencialmente infinita de metaclases se realiza el cierre reflexivo de la metaclase. Se considera que la metaclase OASIS es instancia

[26] Un esquema conceptual es una colección de plantillas que definen totalmente un sistema de información (codificado en el estado de unos metaobjetos), mientras que instancia de un esquema conceptual será un prototipo del sistema que los diseñadores validarán por animación y con el que los usuarios finales pueden trabajar. La primera noción corresponde a lo que se conoce como *tiempo de edición* y la segunda al *tiempo de ejecución o animación*.

[27] Cada instancia de la metaclase es un esquema conceptual.

de sí misma. Dicha instancia contiene toda la información referente a cómo se ha de definir el tipo de los esquemas, crear metaobjetos y hacerlos evolucionar. La Figura 5-9 muestra doble visión de la metaclase OASIS.

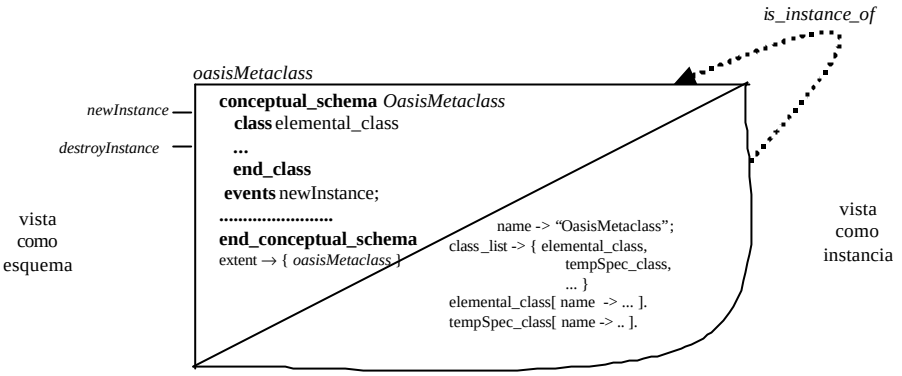


Figura 5-9. Doble visión de la metaclase OASIS

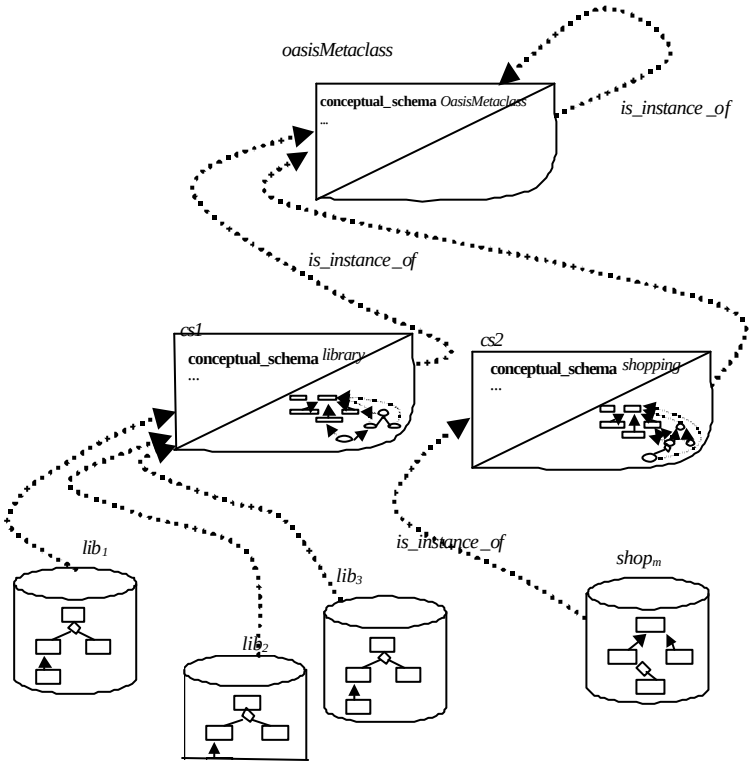


Figura 5-10. Jerarquía *is_instance_of* de la metaclase OASIS (fuente [Car99])

El proceso de creación de especificaciones OASIS haciendo uso de la metaclase OASIS se describe en [Car99]. Éste siempre se inicia con la invocación del servicio newInstance que crea una instanciación de la metaclase OASIS, inicialmente vacia[28] pero lista para definir nuevas clases invocando los servicios de las metaclases correspondientes, según se desee crear una clase elemental, compleja o primitiva. Una vez definido el esquema conceptual, es posible crear instancias de dicho esquema, que se corresponderán con distintas animaciones o ejecuciones del mismo. La Figura 5-10 muestra la jerarquía de la relación *is-intance_of* de la metaclase OASIS (raíz de la jerarquía), dos esquemas conceptuales y diferentes instanciaciones de los esquemas.

5.3 Representación OASIS del Metamodelo de Flujo de Trabajo

La primera cuestión que se plantea a la hora de formalizar en OASIS el metamodelo de flujo de trabajo definido en el capítulo anterior es si la expresividad del lenguaje y del modelo objetual subyacente lo permite. La respuesta es afirmativa, ya que el marco proporcionado por OASIS es lo suficientemente expresivo como para permitir abordar el modelado conceptual de flujos de trabajo. En concreto, la visión proceso de todo objeto OASIS permite especificar flujos de trabajo sin añadir ninguna característica nueva, sino haciendo especial hincapié en la sección processes del lenguaje [Pen99b, Pen00].

En el marco conceptual definido por OASIS, los flujos de trabajo y sus componentes se tratan como objetos. Los componentes del flujo de trabajo pueden ser totalmente pasivos (por ejemplo, los datos que utiliza un proceso o una actividad concreta), totalmente activos (por ejemplo, el propio proceso que representa al flujo de trabajo), o bien, tener parte activa y parte pasiva (por ejemplo, una actividad que requiera la participación humana).

Para la representación en OASIS del metamodelo de flujo de trabajo de referencia, se han identificado las correspondencias entre dicho metamodelo, en notación UML, y el modelo orientado a objetos OASIS presentado. De este modo, los principales elementos de un flujo de trabajo, como son el propio proceso, las distintas actividades a realizar, los datos utilizados, las aplicaciones invocadas y la participación humana se consideran clases OASIS y reciben el mismo nombre que en el metamodelo de referencia. Las relaciones entre clases se modelan en OASIS como agregaciones; al igual que antes las relaciones que subsisten mantienen el mismo nombre que en el metamodelo de referencia. Las jerarquías de generalización se definen mediante el operador OASIS de especialización, pudiéndose distinguir entre especialización temporal y permante. La Figura 5-11 muestra el metamodelo de flujo de trabajo desde el marco OASIS[29].

[28] Valor por defecto en los atributos y sin instancias creadas en las metaclases.
[29] Se utiliza notación UML, pero se están representando conceptos OASIS (Clases y relaciones).

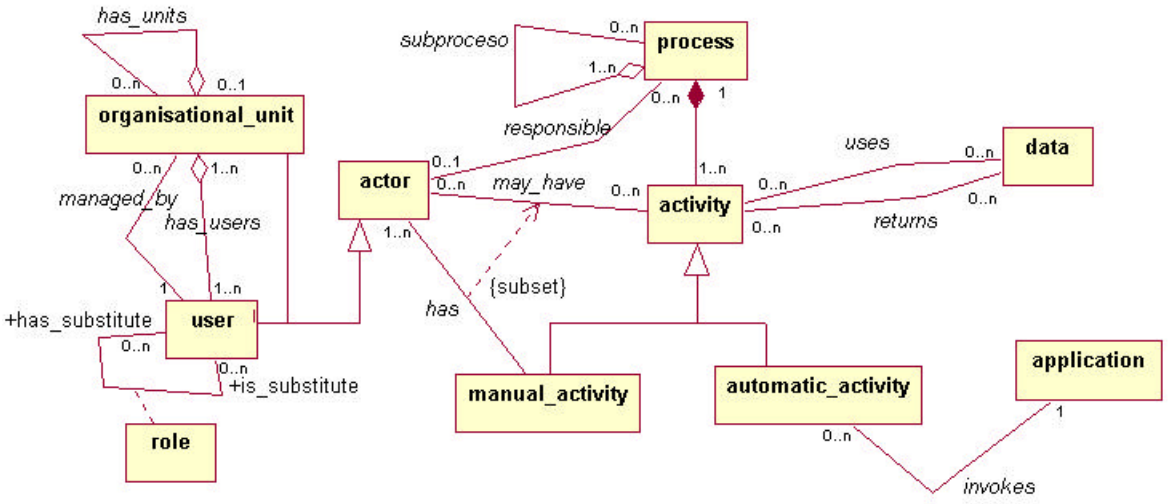


Figura 5-11. Modelo OASIS de un flujo de trabajo

Al representar en OASIS el metamodelo de referencia, podemos observar que un flujo de trabajo es un proceso[30] que puede estar compuesto por cero o más subprocesos y por cero o más actividades. Esto supone que un proceso (clase process) es una clase compleja agregada de la clase activity por una parte y de la propia clase process, por otra (esta última modela la composición de procesos).

Las clases automated_activity y manual_activity son especializaciones permanentes de la clase activity, al igual que las clases user y organizational_unit lo son de la clase actor. Respecto al resto del metamodelo de referencia y su representación en OASIS, las principales diferencias se centran en las condiciones de transición y el propio flujo de control.

Las condiciones de transición entre las tareas del flujo de trabajo no son clases, sino que quedan dentro de la definición del propio proceso; del mismo modo, el flujo de control entre tareas no se modela en OASIS como relaciones entre clases, sino que también queda dentro de la definición del propio proceso. Esto se detalla más en la siguiente sección dedicada a comentar la clase process.

En el apéndice A se incluye la especificación completa en R-OASIS del metamodelo de flujo de trabajo de referencia. No obstante, a continuación se presentan las principales clases del metamodelo y sus características más relevantes.

[30] En lo sucesivo se utiliza el término proceso.

5.3.1 La Clase *process*

En OASIS, un proceso es un objeto activo. La Figura 5-12 muestra la especificación de la clase process. Esta especificación toma como punto de partida la clase program definida (ver pág. 91), la cual incorpora la noción de objeto activo y permite la programación de aplicaciones utilizando objetos OASIS. La clase proceso modifica y enriquece dicha especificación, quedando como se muestra en la Figura 5-12.

La estructura de un proceso queda definida por los siguientes atributos: como atributos constantes, es decir, que no cambian de valor durante toda la vida del objeto, tiene definido un identificador del proceso (identifier), un nombre (name) y una descripción (description); y como atributos variables, uno denominado state cuyo valor cambiará para reflejar el estado del proceso según los eventos que vayan aconteciendo y de acuerdo con los posibles estados en los que se puede encontrar un proceso.

```
complex_class process aggregation_of
   activity(inclusiva,static,multivalued,disjoint,strict,not null),
   process(relational,static,multivalued,nodisjoint,flexible, null).
  constant_attributes
     key identifier(string);
         name(string);
         description(string).
  variable_attributes
     state(string).
       valuation
        [start]      state(create).
        [run]        state(running).
        [end_run]    state(executed).
        [finish]     state(finished).
        [terminate]  state(terminated).
        [kill]       state(dead).
       end_valuation
  private_events
      new       start;
    destroy     stop;
              run;
              end_run;
              finish;
              terminate;
              kill.
 processes
   process <-start & body & stop.
   body<- (check(start_condition) &  run & actions & end_run  &
          ((check(end_condition) & finish) or terminate))or kill.
   actions <- true.
end_complex_class
```

Figura 5-12. Especificación OASIS de la clase proceso

El comportamiento de un objeto proceso viene definido por un conjunto de eventos, especificados como eventos privados y denotados por: start, kill, stop, run, end_run, finish y terminate. Finalmente, la vida del objeto que representa al proceso viene especificada en el apartado processes de la especificación OASIS. Ésta siempre empieza por el evento de creación (start) y termina por el evento de destrucción (stop). Entre ambos eventos, la traza del proceso es la siguiente. Una vez creado el objeto, su estado pasa a ser create. Si se cumplen las condiciones de inicio del proceso (check(start_condition)), se dispara el evento run pasando su estado a running, en caso contrario el proceso pasa al estado dead. Si el proceso puede ejecutarse (se ha alcanzado el estado running), se inicia inmediatamente el subproceso actions, que contendrá el conjunto de actividades o subprocesos específicos que forman el proceso concreto con el que se está trabajando. Una vez éste finalice, se comprueba la condición de finalización del proceso (check(end_condition)). Si ésta se cumple, el estado alcanzado por el proceso, tras el disparo del evento finish, es de terminación con éxito (estado finished); mientras que en caso contrario, se dispara el evento terminate alcanzándose el estado terminated que indica terminación sin éxito. Como se puede apreciar, las condiciones de inicio y finalización forman parte de la especificación del apartado processes. Un proceso p inicia su ejecución cuando se le envía el mensaje p::start y finaliza su ejecución cuando su vida alcanza stop.

La especificación de la Figura 5-12 muestra una clase proceso genérica, es decir, una clase que proporciona el patrón genérico de comportamiento y estructura que debe tener cualquier proceso que se desee especificar. Si nos fijamos, el subproceso actions es un proceso trivial, que no hace nada y por ello aparece denotado por *true*. En la práctica nunca se crearán instancias de dicha clase (es una clase abstracta). Si lo que deseamos es especificar un proceso concreto, utilizamos el operador de especialización para crear una clase especializada de dicha clase process y redefinir aquello que se considere oportuno. En concreto, debemos redefinir el subproceso actions, de modo que se especifiquen las acciones o actividades concretas asociadas al nuevo proceso definido, incluyéndose la lógica del proceso y las posibles condiciones de transición que puedan existir. También debemos redefinir las condiciones de inicio y finalización del proceso, si es necesario. La especificación del proceso se realiza de acuerdo con la definición de proceso OASIS dada la sección 5.2.4.

5.3.2 La Clase *activity*

Al igual que ocurre con los procesos, en OASIS una actividad es un objeto activo. La especificación OASIS de la clase activity es muy similar a la clase process. En cuanto a la definición de la clase, la principal diferencia estriba en que la clase process es una clase compleja agregada, mientras que la clase activity es una clase elemental. La plantilla de especificación, sin embargo coincide. La Figura 5-13 muestra la clase activity especificada en OASIS.

```
class activity
constant_attributes
  key  identifier(string);
       name(string);
       description(string).
variable_attributes
   state(string).
     valuation
         [start]     state(create).
         [run]       state(running).
         [end_run]  state(executed).
         [finish]   state(finished).
         [terminate]  state(terminated).
        [kill]      state(dead).
     end_valuation
private_events
      new      start;
    destroy    stop;
               run;
               end_run;
               finish;
               terminate.
 processes
  activity <- start & body & stop.
  body <- (check(start_condition) &  run  &  actions  &
           end_run & (check(end_condition) & finish
           or terminate)) or kill.
  actions <- true.
end_class
```

Figura 5-13. Especificación de la clase actividad

El apartado processes de la especificación define el comportamiento activo del objeto que representa a una actividad, al igual que ocurre con los procesos. Su especificación coincide con la de la clase proceso, y al igual que antes, el subproceso etiquetado como actions es un proceso trivial que no hace nada. Definir una actividad concreta supone especializar la clase activity y redefinir el subproceso actions. En dicha redefinición se le asociarán las acciones concretas y específicas que tienen que realizarse para que dicha actividad pueda llevarse a cabo. En este caso, no se especificarán llamadas a otras actividades o subprocesos, sino que serán tareas concretas de acceso a los datos de entrada que se necesiten, realización de una cierta operación de actualización, y si así lo requiere la actividad, almacenamiento de nuevos datos generados como salida.

Puesto que las actividades de un flujo de trabajo pueden ser automáticas o manuales, su especificación como clases OASIS se muestra en Figura 5-14. En ambos casos, su especificación es la misma que la de una actividad genérica, utilizándose el operador de especialización para definirlas, pero sin redefinir nada. Son especializaciones permanentes.

La aplicación que invoca una actividad automática no queda reflejado explícitamente en la clase aplicación, sino en la agregación entre ambas clases. Para las actividades manuales ocurre lo mismo en el caso del actor que necesariamente participa en dicha actividad (en el apéndice A, aparecen especificadas todas ellas).

```
complex_class manual_activity specialization_of activity
end_complex_class

complex_class automatic_activity specialization_of activity
end_complex_class
```

Figura 5-14. Especificación OASIS de las clases actividad manual y automática

El diagrama de transición de estados de la clase process coincide con el de la clase activity. Es decir, el objeto que representa una actividad concreta está en un estado denominado create, inmediatamente después de haber sido creado y no pasa a ejecución hasta que no se cumpla la condición de inicio. Una vez finaliza su ejecución, se comprueba igualmente la condición de finalización y el estado alcanzado es distinto dependiendo de si dicha condición se satisface o no.

5.3.3 La Clase *application*

Las aplicaciones se modelan en OASIS como clases elementales cuyos ejemplares son objetos activos que representan la invocación de la aplicación para que se ejecute (ver Figura 5-15). El objeto activo que representa la aplicación puede modificar los datos con los que estamos trabajando, pero esto no aparece modelado explícitamente, sino que queda dentro de la ejecución de la aplicación. Lo que sí se modela como un atributo de la clase, aparte del estado del objeto, es el posible valor de retorno de la aplicación (atributo result_value). Una vez creado el objeto, siguiendo la especificación del apartado processes se irán disparando los diferentes eventos que forman parte de su signatura. En concreto, se invoca a la aplicación con los datos de entrada necesarios con el evento invoke_app(Call,Data,Result)[31] y cuando este evento finalice, significa que la ejecución de la aplicación ha finalizado. En ese mismo momento se dispara el evento result_app(Result) que almacena en el objeto el valor de retorno de la aplicación.

[31] En OASIS, los parámetros de los eventos siempre están todos instanciados antes de que éste ocurra. En este caso, el parámetro Result no está instanciado, sino que representa el valor de retorno de la aplicación una vez se ha ejecutado.

```
class application
  constant_attributes
     key   identifier(string);
          app_name(string);
          execution_params(string);
          path(string);
          data(list);
          call(string).
 variable_attributes
    state(string)
 valuation
     R,Call: string.
        [new_app] state(active).
        [invoke_app(Call,Data,Result)] state(running).
        [result_app(Result)] state(executed).
 end_valuation
    return_value(string)
  valuation
     R,Call: string.
        [new_app] return_value(‘’).
        [result_app(Result)] return_value(Result).
  end_valuation

private_events
   Call, Result:string; Data:list.
        new          new_app;
       destroy    destroy_app;
              invoke_app(Call,Data,Result);
              result_app(Result);
              end_app;
 processes
       application <- new_app & application1.
       application1<- call(Call) & data(Data) &
              invoke_app(Call,Data,Result)& result_app(Result) &
               end_app & application2.
       application2<- application1 or destroy_app.
end_class
```

Figura 5-15. Especificación OASIS de la clase aplicación

5.3.4 La Clase *data*

En todo flujo de trabajo, el proceso y las actividades que lo componen manejan información que tiene que ver con el dominio del problema. Esta información se modela como clases OASIS también. La clase data es la clase genérica a partir de la cual podemos definir, por especialización, todas las demás clases que componen el sistema de información. No se trata de objetos activos. La especificación aparece en la Figura 5-16.

```
class data
constant_attributes
    key  identifier(string).
private_events
   new     new_data;
  destroy   destroy_data.
processes
      data <- new_data & destroy_data.
end_class
```

Figura 5-16. Especificación OASIS de la clase dato

5.3.5 La Clase *actor*

La clase actor es la generalización de la clase user y de la clase organisational_unit. La Figura 5-17 muestra la especificación OASIS de estas clases. Destacar el hecho de que no se trata de objetos activos, sino pasivos.

```
class user
 private_events
     new         new_user;
    destroy  destroy_user;
 processes
       user <- new_user & destroy_user.
end_class


complex_class organisational_unit aggregation_of
   user(relacional,dynamic,multivalued,nodisjoint,strict,not null),
   organisational_unit(relacional,static,multivalued,disjoint,
                       flexible, null).
 private_events
      new     new_orgUnit;
   destroy    destroy_orgUnit;
 processes
       organisational_unit <- new_orgUnit & destroy_orgUnit.
end_complex_class


complex_class actor  generalization_of  user, organisational_unit.
 constant attributes
 key  identifier(string);
       name(string);
       description(string).

end_complex_class
```

Figura 5-17. Especificación OASIS de la clase actor

5.4 Conclusiones

La expresividad proporcionada por el lenguaje de especificación formal y orientado a objetos OASIS es suficiente para abordar la especificación de flujos de trabajo. La noción de objeto activo ha sido utilizada para modelar procesos, actividades y aplicaciones, mientras que el resto de clases se corresponden con clases pasivas.

En base a esta aproximación se ha definido el modelo OASIS de un flujo de trabajo. Sus principales características se pueden resumir en dos. Por una parte, las condiciones de transición y el flujo de control quedan incluidas como parte de la especificación OASIS de la propia clase que representa al proceso. Por otra parte, se pueden especificar flujos de trabajo concretos a partir de las clases del modelo OASIS definido, utilizando el operador de especialización. La aproximación seguida ha permitido formalizar el metamodelo de referencia propuesto en el capítulo anterior.

PARTE III

Un Modelo de Proceso de Desarrollo de Flujos de Trabajo

Capítulo 6.

Construcción y Estabilización de Flujos de Trabajo

En este capítulo se describe el subproceso de construcción y estabilización de flujos de trabajo. Su objetivo es la obtención de un modelo de flujo de trabajo que represente fielmente el proceso de negocio seguido por una organización, tal como ya se mencionó en el capítulo 3. Dicho modelo queda almacenado en un repositorio para su ejecución.

La estructura del capítulo es la siguiente. La sección 6.1 describe la construcción de especificaciones de flujos de trabajo de forma textual o gráfica. Se define un lenguaje gráfico y se describe un editor de flujos de trabajo, relacionándose las plantillas de generación de código a OASIS. En la sección 6.2 se ilustra el proceso de prototipación automática a partir de la especificación generada. En la sección 6.3 se presenta un método para la obtención automática de un modelo inicial de flujo de trabajo basado en la captura de requisitos del proceso de negocio mediante casos de uso. Finalmente, en la sección 6.4 aparecen las conclusiones de este capítulo.

6.1 Especificación de un Modelo de Flujo de Trabajo

Aquellas organizaciones cuyo objetivo sea la automatización de sus procesos de negocio utilizando un SGFT deben, en primer lugar, especificar dichos procesos de negocio como flujos de trabajo, en base a un metamodelo de referencia; un editor puede asistir el proceso de construcción de dicha especificación teniendo en cuenta dicho metamodelo.

En nuestro caso, utilizaremos el metamodelo de referencia que se ha definido y formalizado en los capítulos 4 y 5, respectivamente. En base al mismo, definimos modelos de flujos de trabajo que representen procesos concretos, como por ejemplo, el seguido para la gestión de PFC en la universidad, el seguido por una entidad bancaria para la concesión de un préstamo o el definido por una agencia para la reserva de viajes a realizar por sus clientes.

El proceso de construcción es asistido por un editor que proporciona un conjunto de servicios para la construcción de los modelos, teniendo en cuenta los elementos que componen un flujo de trabajo y las restricciones definidas en el metamodelo. Además, al igual que ocurre en la mayor parte de las herramientas de modelado existentes, junto con el lenguaje textual se proporciona un lenguaje o notación gráfica para modelar un sistema, haciendo más sencilla e intuitiva la interacción del usuario con la herramienta de modelado. Existe una correspondencia entre el lenguaje gráfico definido y la especificación textual.

A continuación se da cuenta de ambas, así como del editor de modelos de flujo de trabajo. En nuestro caso, el objetivo es obtener una especificación OASIS que represente al flujo de trabajo, por lo que también se relacionan las plantillas de generación de código que aplica el editor para la obtención de la especificación textual [Pen01a].

6.1.1 Especificación Textual

Utilizaremos como lenguaje textual de especificación de flujos de trabajo el propio lenguaje OASIS. Más concretamente la versión denominada R-OASIS, que utiliza como formalismo la lógica clausal dinámica. La construcción de las especificaciones debe hacerse de acuerdo con el modelo OASIS de un flujo de trabajo visto en el capítulo anterior. Esto obliga a conocer no sólo el lenguaje sino también la formalización realizada. Por ejemplo, si deseamos especificar el subproceso que sigue la realización de un PFC, según el caso de estudio planteado, debemos especializar la clase proceso del modelo OASIS y en el apartado processes definir las acciones oportunas. De forma similar, las actividades que se han definido son también especializaciones de la clase actividad, a las que debemos asociar los recursos e indicar las acciones asociadas. La Figura 6-1 muestra un fragmento de la especificación OASIS correspondiente a este subproceso. En este caso, el proceso se denomina realizacionPFC y consta de dos actividades, la elaboración de la memoria (elaborarMemoria) y su revisión por parte del director (revisarPFC). Además, este proceso es iterativo, finalizando cuando el resultado de la revisión es positivo (condición revResult).

La utilización de OASIS como lenguaje de modelado puede resultar complicada y poco intuitiva si el usuario no es un experto conocedor del mismo. Por esta razón, se ve la conveniencia de definir un lenguaje gráfico que permita al usuario construir los modelos de flujo de trabajo de una forma más intuitiva, y que la especificación OASIS correspondiente se obtenga de forma automática. Esto es lo que se describe en las secciones siguientes.

```
conceptual_schema gestionPFC
domains string; bool; nat.
  . . .
  complex_class realizacionPFC specialization_of process
  processes
    realizacionPFC <- start & body & stop.
    body <- (check(true) & run & actions & end_run &
            (check(true) & finish or terminate)) or kill.
    actions <- elaborarMemoria :: start(_Oid,'elaborarMemoria') &
               revisarPFC :: start(_Oid,'revisarPFC') &
               (check(revResult) or actions).
  end_complex_class

  complex_class elaborarMemoria specialization_of manual_activity
  processes
    elaborarMemoria <- start & body & stop.
    body <- (check(true) & run & actions & end_run &
            (check(true) & finish or terminate)) or kill.
    actions <- resources & actions_activity.
    resources <- . . .
    actions_activity <- . . .
  end_complex_class
. . .
end_conceptual_schema
```

Figura 6-1. Fragmento de la especificación del subproceso de realización de un PFC

6.1.2 Especificación Gráfica

En esta sección se describe el lenguaje gráfico definido para el modelado de flujo de trabajo y se muestra un ejemplo de utilización del mismo.

6.1.2.1 Lenguaje Gráfico

El lenguaje gráfico definido asocia un icono a cada uno de los componentes del flujo de trabajo, a partir del metamodelo de referencia. La semántica asociada viene proporcionada por la formalización OASIS del mismo. El analista construye el grafo que representa el flujo

de trabajo que desea modelar utilizando dichos iconos. La Figura 6-2 muestra la notación gráfica que a continuación se describe brevemente.

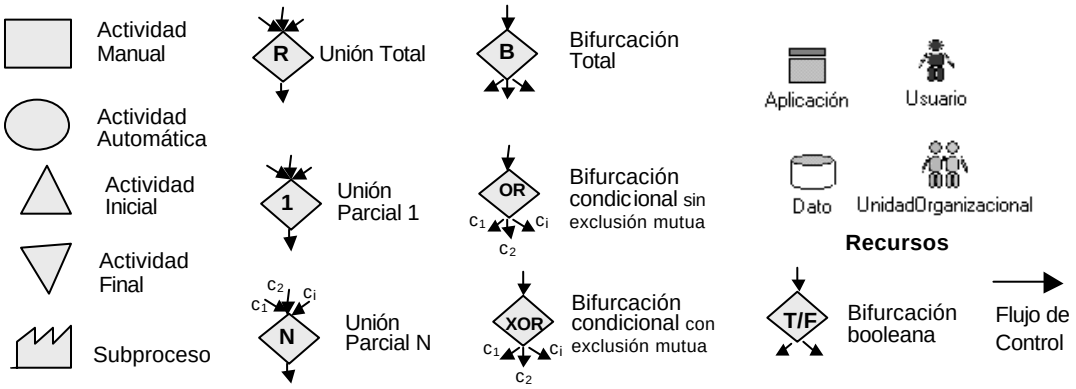


Figura 6-2. Notación gráfica para la definición de flujos de trabajo

- **Actividades**: Se han definido distintos iconos con los que el usuario podrá representar los distintos tipos de actividades a realizar dentro de un proceso. Además de la distinción entre actividades manuales y automáticas, se dispone de dos iconos que representan las actividades de inicio y fin, que nos sirven para poder marcar el inicio y el final del proceso, respectivamente. Obviamente, sólo podrá existir una actividad de inicio y de finalización en cada proceso.
- **Subproceso**: Se ha definido un icono para representar los subprocesos, es decir, aquellas tareas dentro de un proceso que son complejas y representan en sí mismas un nuevo proceso.
- **Condiciones**: Al definir el metamodelo de referencia (capítulo 4) se identificaron dos tipos generales de condiciones que se podían establecer al modelar un flujo de trabajo (condiciones de unión y condiciones de bifurcación) y se señaló que para cada una de ellas se podían distinguir varios casos. A continuación se detallan los diferentes tipos de condiciones de unión y bifurcación que se pueden especificar y la notación empleada.

  En cuanto a **condiciones de unión** tenemos:

  - *Unión total* (punto de reunión). Hasta que no finalicen todas las tareas de entrada no se inicia la tarea de salida. En este caso el icono correspondiente a la condición se etiqueta con el símbolo R.

- *Unión parcial*. En este caso, conforme finalizan las tareas de entrada se comprueban las condiciones con que están etiquetados los flujos de control de entrada a la condición, distinguiéndose dos casos:
  - *Unión parcial 1*. La finalización de una tarea cuya condición asociada sea cierta, hace que se inicie la tarea de salida, sin tener en cuenta lo que ocurra con el resto de tareas de entrada. En este caso el icono correspondiente a la condición se etiqueta con el símbolo 1.
  - *Unión parcial N*. Hasta que no finalicen un número determinado (N; N>1) de tareas de entrada que además cumplan la condición asociada, no se inicia la tarea de salida. En este caso el icono correspondiente a la condición se etiqueta con el símbolo N.

En cuanto a **condiciones de bifurcación** se distingue entre:

- *Bifurcación total* (punto de bifurcación). Cuando finaliza la tarea de entrada, se inician todas las tareas de salida que haya definidas, para que éstas inicien su ejecución en paralelo, sin comprobarse ninguna condición. En este caso el icono se etiqueta con el símbolo B.
- *Bifurcación condicional*. En este caso, la evaluación de las condiciones asociadas a los flujos de salida determina qué tareas de salida se van a iniciar. Se distinguen dos casos:
  - *Sin exclusión mutua*. Las condiciones que etiquetan los flujos de control de salida no son mutuamente excluyentes, por lo que se iniciarán tantas tareas de salida como condiciones asociadas a los flujos de control de salida sean ciertas. En este caso el icono correspondiente a la condición se etiqueta con el símbolo OR.
  - *Con exclusión mutua*. Las condiciones que etiquetan los flujos de control de salida son mutuamente excluyentes, por lo que sólo se inicia una actividad de salida (aquella cuya condición asociada al flujo de control de salida sea cierta). En este caso el icono correspondiente a la condición se etiqueta con el símbolo XOR.
  - *Bifurcación booleana*. Este es un caso particular de la bifurcación condicional con exclusión mutua, en el sentido de que se trata de una bifurcación con dos flujos de control de salida y la misma fórmula asociada para ambos, pero evaluada a cierta en uno de los flujos de control de salida y a falsa en el otro. Puesto que este tipo de condición suele ser frecuente, se ha incluido en el lenguaje gráfico la facilidad de poder representarlas directamente, identificando la condición que determine por cuál de los flujos de control de salida continua la ejecución.

- **Flujo de control**: Utilizado para conectar tareas entre sí, es decir, actividades, condiciones de transición y subprocesos. Un flujo de control siempre tiene una tarea origen y otra destino.

La Figura 6-2 también muestra los símbolos utilizados para representar gráficamente los recursos de un flujo de trabajo. Estos son **aplicaciones**, **datos** y **actores**. Estos recursos se pueden asociar a las ciertas tareas, según las restricciones definidas en el metamodelo, lo cual gráficamente quedará reflejado porque el símbolo en cuestión aparece junto a la tarea a la que está asociado.

La participación humana en un flujo de trabajo (actores) se representa gráficamente mediante dos iconos distintos, para distinguir si al modelar un flujo de trabajo, el actor que inicia o participa en una actividad es un *usuario* concreto, o bien, una *unidad organizacional*. Al igual que para el resto de recursos, el actor va unido a aquellas actividades en las que participa.

Se ha visto como todos los componentes del flujo de trabajo tienen iconos que los representan, y con ellos podemos construir un grafo que lo modele. Ahora bien, existe información importante asociada a dicho flujo de trabajo que no se visualiza en el grafo asociado, sino que el usuario debe introducir. Esta información tiene que ver con los distintos atributos que tiene cada uno de los componentes, y que ya se detallaron al definir el metamodelo. En la herramienta que soporte este lenguaje gráfico, dicha información deberá ser introducida a través de sucesivos diálogos.

6.1.2.2 Ejemplo

Utilizando el lenguaje gráfico definido, en esta sección se muestra parte del flujo de trabajo modelado para representar el proceso de gestión de un PFC. Según la descripción dada en el capítulo 3, desde un punto de vista global, se han identificado las siguientes tareas relacionadas con la realización de un PFC por cada alumno: solicitud del alumno para la realización del proyecto (solicitudPFC), aprobación del tema del proyecto a realizar por el alumno por parte de la comisión de proyectos (aprobacionPFC), realización del proyecto (realizacionPFC), presentación de la memoria (presentacionMemoriaPFC), evaluación del proyecto (evaluacionPFC), asignación definitiva de la nota propuesta al proyecto y generación del acta (asignacionNotaPFC), renovación del proyecto, para continuar durante el siguiente curso académico (renovacionPFC) y baja o renuncia de proyecto asignado (abandonoPFC).

Algunas de estas tareas se han representado como subprocesos, entendiendo que representan un paso complejo dentro del proceso global, y otras como actividades. La Figura 6-3 muestra una visión global de dicho proceso. No se han incluido los recursos para una mayor claridad.

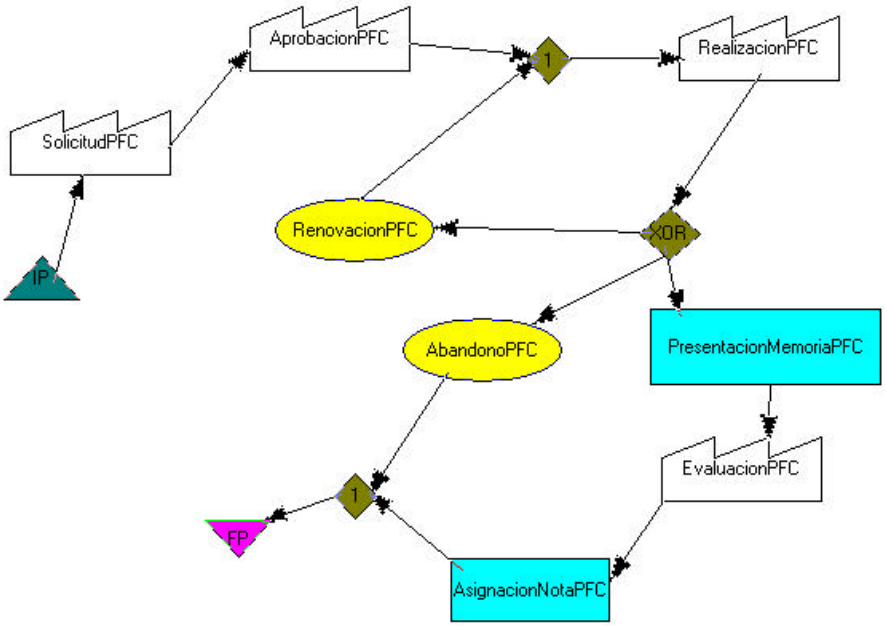


Figura 6-3. Visión global del flujo de trabajo asociado a la realización de un PFC

Para completar la definición del modelo se han de definir cada uno de los subprocesos. La Figura 6-4 muestra el subproceso evaluacionPFC. Inicialmente se debe tomar la decisión de si la evaluación del proyecto realizado es mediante tribunal o no. Esta decisión corresponde tanto al alumno como al director, representándose por las actividades tomaDecisionA y tomaDecisionD, respectivamente. En caso de desacuerdo, se evalúa mediante un tribunal. El resultado de esta actividad condiciona el resto del proceso. Si se decide que el proyecto lo evalúe el director, la única actividad a realizar es la elaboración de un informe por parte del mismo, en el cual se propone una nota al proyecto (actividad evaluacionDirector). Sin embargo, si se decide la evaluación mediante tribunal, hay varias actividades a realizar: solicitar la evaluación mediante tribunal, adjuntándose una propuesta de presidente y secretario del mismo (actividad solEvaluacionTribunal); el nombramiento del tribunal de entre los propuestos y otros profesores del área de conocimiento (actividad nombrarTribunal); y finalmente, la defensa del PFC y generación del acta, con el informe y la propuesta de nota (actividad presentacionPFC). En este caso se han incluido los recursos asociados a las tareas.

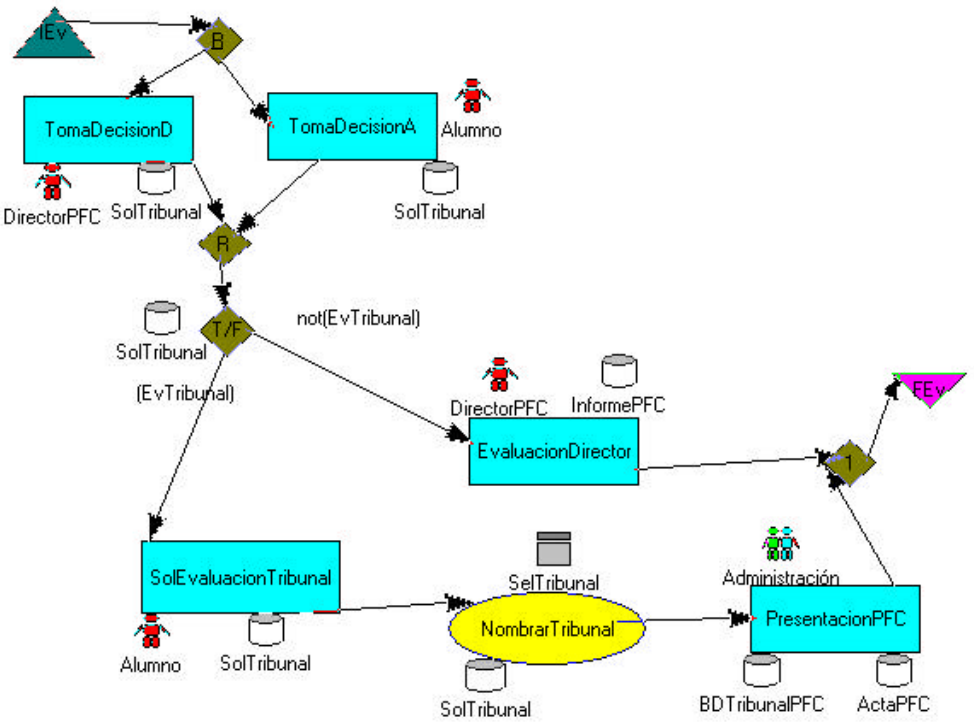


Figura 6-4. Flujo de trabajo correspondiente al proceso de evaluación de un PFC

6.1.3 Un Editor de Modelos de Flujo de Trabajo

La construcción de un modelo de flujo de trabajo y obtención de la especificación correspondiente, puede ser visto como un proceso, en el cual el analista utiliza (en forma textual o gráfica) los servicios proporcionados por un editor, que da soporte al metamodelo de referencia. Desde este punto de vista, y siguiendo con el enfoque orientado a objetos, el editor de modelos de flujo de trabajo es una clase (denominada *wfEditor*) cuyos servicios permiten la construcción de modelos que quedan almacenados como parte de su estado. La Figura 6-5 muestra gráficamente esta clase, con algunos de sus servicios y atributos.

En la Tabla 6-1 se enumeran los principales servicios que proporciona el editor, así como los atributos más importantes que denotan su estado, y que están directamente relacionados con dichos servicios. Según su funcionalidad, los servicios se pueden agrupar en:

→ *Gestión de un flujo de trabajo*: Incluye los servicios de creación, borrado y edición del mismo. Su edición incluye tanto la gestión de las tareas que componen el proceso principal como de los recursos asociados.

→ *Gestión de tareas de un proceso*: Incluye los servicios de inserción, borrado y modificación.

→ *Gestión de los recursos*: Incluye los servicios de definición, inserción, borrado y modificación.

En el apéndice B se da una descripción detallada de todos estos servicios, indicando su perfil, descripción y precondiciones asociadas.

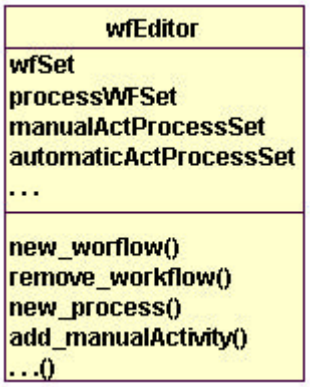


Figura 6-5. El editor de flujos de trabajo visto como una clase

class wfEditor	(Editor de Flujos de Trabajo)
Nombre atributo	***Descripción***
`wfSet`	Flujos de trabajo definidos
`processWFSet`	Procesos de cada flujo de trabajo
`manualActProcessSet`	Actividades manuales de cada proceso
`automaticActProcessSet`	Actividades automáticas de cada proceso
`subprocessProcessSet`	Subprocesos de cada proceso
`totalJoinProcessSet`	Condiciones de unión total de cada proceso

`partialJoinProcessSet`	Condiciones de unión parcial de cada proceso
`totalSplitProcessSet`	Condiciones de bifurcación total de cada proceso
`partialSplitProcessSet`	Condiciones de bifurcación parcial de cada proceso
`resourcesSet`	Conjunto de recursos definidos
`resourceTaskSet`	Recursos asociados a cada tarea
`controlFlowInitialProcessSet`	Actividades finales de cada proceso
`controlFlowEndProcessSet`	Actividades iniciales de cada proceso
`controlFlowProcessSet`	Flujo de control entre tareas de un proceso
Comportamiento	
Nombre del evento	***Descripción***
`new_workflow`	Creación de un flujo de trabajo
`remove_workflow`	Borrado de un flujo de trabajo
`new_process`	Creación del proceso principal asociado al flujo de trabajo
`remove_process`	Borrado de un proceso
`add_manualActivity`	Creación de una actividad manual
`add_automaticActivity`	Creación de una actividad automática
`add_initialActivity`	Creación de una actividad inicial
`add_endActivity`	Creación de una actividad final
`add_totalJoin`	Creación de una condición de unión total
`add_parcialJoin`	Creación de una condición de unión parcial
`add_totalSplit`	Creación de una condición de bifurcación total
`add_parcialSplit`	Creación de una condición de bifurcación parcial
`new_subprocess`	Creación de un subproceso
`add_controlFlow`	Establecer flujo de control
`add_conditionFlow`	Establecer una condición a un flujo de control
`change_activityProp`	Cambiar propiedades de una actividad
`change_subprocessProp`	Cambiar propiedades de un subproceso

change_conditionFlow	Cambiar condición asociada a un flujo de control
transform_subprocess	Convertir subproceso en actividad
transform_activity	Convertir actividad en subproceso
remove_task	Borrado de una tarea
remove_controlFlow	Borrado de un flujo de control
new_data	Definir un recurso dato
new_application	Definir un recurso aplicación
new_user	Definir un recurso usuario
new_organizationalUnit	Definir un recurso unidad organizacional
add_resource	Añadir un recurso
resource_assigment	Asociar un recurso a una tarea
resource_notAssigment	Desasociar un recurso a una tarea
add_propertyResource	Añadir nuevas propiedades a un recusrso
change_propertyResource	Cambiar propiedades de un recurso
add_userUnit	Añadir usuarios a una unidad organizacional
add_subUnitUnit	Composición de unidades organizacionales
remove_resource	Borrado de un recurso
remove_propertyResource	Borrado de una propiedad de un recurso
generateOASIS	Generación de la especificación OASIS

Tabla 6-1. Atributos y servicios del editor de modelos de flujos de trabajo

La especificación de un flujo de trabajo se construye invocando los servicios que proporciona el editor. La ocurrencia de los mismos modifica el valor de sus atributos, que van almacenando la información referente al flujo de trabajo. Este proceso de edición siempre empieza con la invocación del evento new_workflow, con los argumentos correspondientes, que es el evento de creación de instancias de la clase wfEditor.

La Figura 6-6 muestra un fragmento de código correspondiente a la construcción del modelo de flujo de trabajo asociado a la evaluación de un PFC (subproceso evaluacionPFC visto en la página 116). En dicho fragmento se enumeran parte de la invocación de servicios del editor para la definición del modelo.

```
//se crea una instancia del editor
wfEditor::new_workflow(OidGPFC,“gestionPFC”,”Gestion de PFC en la BDFI
                        de la UPV”) &
//se invocan los servicios de construccion del modelo
OidGPFC::new_process(“evaluacionPFC”,“Evaluación de un PFC”,true,true) &
OidGPFC::add_initialActivity(“Iev”) &
OidGPFC::add_manualActivity(“tomaDecisionD”,”El director decide si el
                           PFC se lee ante un tribunal”, true, true) &
OidGPFC::add_manualActivity(“tomaDecisionA”,”El alumno decide si desea
                      Defender su PFC ante un tribunal”, true, true) &
OidGPFC::add_totalSplit(“SJ1”) &
OidGPFC::add_controlFlow(“Iev”,”TJ1”,true) &
OidGPFC::add_controlFlow(“SJ1”,”tomaDecisionD”, true) &
OidGPFC::add_controlFlow(“SJ1”,”tomaDecisionA”, true) &
OidGPFC::new_data(“SolTribunal”,”Solicitud de Petición de Tribunal para
      evaluación”, “//hostfi/administracion/docencia/PFC/Soltribunal”) &
OidGPFC::add_resource(“SolTribunal”) &
OidGPFC::resource_asigment(“tomaDecisionA”,”SolTribunal”) &
OidGPFC::resource_asigment(“tomaDecisionD”,”SolTribunal”) &
OidGPFC::new_user(“Alumno”,”Alumno matriculado en PFC”,”Alumnado”) &
OidGPFC::new_user(“DirectorPFC”,“Profesor que dirige un PFC”,
                 ”Profesorado”) &
OidGPFC::add_resource(“Alumno”) & OidGPFC::add_resource(“DirectorPFC”) &
OidGPFC::resource_asigment(“tomaDecisionA”,”Alumno”) &
OidGPFC::resource_asigment(“tomaDecisionD”,”DirectorPFC”) &
OidGPFC::add_totalJoin(“SJ1”) &
OidGPFC::add_parcialSplit(“PS1”) &
OidGPFC::add_controlFlow(“tomaDecisionD”,”TJ1”, true) &
OidGPFC::add_controlFlow(“tomaDecisionA”,”TJ1”, true) &
OidGPFC::add_manualActivity(“solEvaluacionTribunal”,”Solicitud de
                    tribunal para la evaluacion del PFC”, true, true) &
OidGPFC::add_manualActivity(“evaluacionDirector”,”El director elabora un
 informe de evaluación y propone una calificación al PFC”, true, true) &
OidGPFC::add_controlFlow(“TJ1”,”PS1”, true) &
OidGPFC::add_controlFlow(“PS1”,”solEvaluacionTribunal”, (EvTribunal)) &
OidGPFC::add_controlFlow(“PS1”,”evaluacionDirector”,(not(EvTribunal))) &
OidGPFC::add_automaticActivity(“NombrarTribunal”,”Se nombra el tribunal
                  para la evalución del PFC”, (InfSolTribunal), true) &
. . . .
```

Figura 6-6. Fragmento de una secuencia de invocación de servicios para la construcción de un modelo de flujo de trabajo

De todos los servicios que proporciona el editor, destacamos el servicio de generación de la especificación OASIS (denominado generateOASIS). Este servicio permite, una vez finalizado el proceso de construcción, y siempre que no existan errores, generar la especificación textual OASIS equivalente. Es una transacción compleja, que parte del estado del editor y aplica unas plantillas de generación de código OASIS; la definición de estas plantillas está basada en la formalización del metamodelo de referencia. A continuación se describen dichas plantillas y se muestra la formalización del editor en OASIS.

6.1.3.1 Plantillas para la Generación de Código OASIS

El conjunto de plantillas de generación de código definidas están directamente relacionadas con la formalización OASIS del metamodelo (capítulo 5). En todos ellas, las palabras en negrita se corresponden con palabras clave del lenguaje y las que aparecen en cursiva se reemplazarán en cada proceso de generación por el valor correspondiente. A continuación se enuncian las más representativas, agrupadas en bloques según estén relacionadas con la definición del:

a) Flujo de trabajo y procesos asociados (proceso principal y subprocesos que lo componen).

b) Actividades.

c) Recursos.

d) Asociación de recursos a tareas.

a) Flujo de trabajo y procesos asociados

- ***Plantilla Flujo de Trabajo***. Cada flujo de trabajo que se define equivale a un esquema conceptual OASIS, cuya sintaxis concreta se muestra a continuación. En ella, el literal Definiciones_de_clases se sustituye por la generación de código correspondiente a todas las clases del proyecto.

```
conceptual_schema Nombre_Proyecto
 domains string; bool; nat.
   Definiciones_de_clases
end_conceptual_schema
```

- ***Plantilla Proceso.*** Cada proceso que compone el flujo de trabajo se corresponde con una clase especializada de la clase OASIS denominada process. Esta clase proporciona el patrón genérico de comportamiento y estructura que debe tener cualquier proceso. Para especificar un proceso concreto, utilizamos el operador de especialización y redefinimos en la sección processes, las condiciones de inicio y finalización, así como el subproceso actions.

Si observamos la plantilla de generación, lo único que cambia de unos procesos a otros son los literales Nombre_Proceso, start_condition, end_condition y Secuencia_actividades_ y_condiciones. Éste último se sustituye por la secuencia de creación de las actividades y subprocesos (actividad::start(_Oid,Cod_Act) ó subprocess::start(_Oid,Cod_SubP)), empezando por la actividad inicial y acabando en la actividad final. La secuencia de invocación se hace de acuerdo al flujo de control y condiciones de transición modeladas en el proceso.

```
complex_class Nombre_Proceso specialization_of process
 processes
  Nombre_Proceso <- start & body & stop.
  body <- (check(start_condition) & run & actions & end_run &
         (check(end_condition) & finish or terminate)) or kill.
  actions <- Secuencia_actividades_y_condiciones.
end_complex_class
```

Todo subproceso es un proceso; por lo tanto, la plantilla de generación seguida para un subproceso es esta misma que acabamos de describir.

b) Actividades

Cada una de las actividades de un proceso se corresponde con una clase especializada de la clase OASIS activity. Ésta representa en el metamodelo a una actividad genérica, pero puesto que existe distinción entre manuales o automáticas, la especialización se realizará de la clase correspondiente del metamodelo. A continuación se muestran las distintas plantillas.

- ***Plantilla Actividad Manual***. Cada actividad manual se corresponde con una clase especializada de la clase actividad manual del metamodelo OASIS. En cuanto al código que se genera tenemos: el literal asociación_de_los_recursos_que_utiliza se sustituye por la creación y asociación del recurso correspondiente utilizado por dicha actividad[32]; el literal Campo_acciones_de_la_BD se sustituye por las acciones de la actividad, que se definen al crearla o modificarla con el editor. Al igual que para la plantilla proceso, los términos start_condition y end_condition se sustituyen por las condiciones correspondientes.

[32] El tipo de actividad condiciona el tipo de recursos que pueden tener asociados. El editor tiene en cuenta dicho tipo durante el proceso de definición del flujo de trabajo.

```
complex_class Nombre_Actividad specialization_of manual_activity
processes
      Nombre_Actividad <- start & body & stop.
      body <- (check(start_condition) & run & actions & end_run &
              (check(end_condition) & finish or terminate)) or kill.
      actions <- resources & actions_activity.
       resources <- asociación_de_los_recursos_que_utiliza
       actions_activity <- Campo_acciones_de_la_BD.
end_complex_class
```

- ***Plantilla Actividad Automática***. Cada actividad automática se corresponde con una clase especializada de la clase actividad automática del metamodelo OASIS. La plantilla de generación se muestra a continuación y coincide con el visto para una actividad manual.

```
complex_class Nombre_Actividad specialization_of automatic_activity
processes
      Nombre_Actividad <- start & body & stop.
      body <- (check(start_condition) & run & actions & end_run &
              (check(end_condition) & finish or terminate)) or kill.
      actions <- resources & actions_activity.
       resources <- asociación_de_los_recursos_que_utiliza
       actions_activity <- Campo_acciones_de_la_BD.
end_complex_class
```

c) Recursos

Todos los recursos asociados a algún proceso de un flujo de trabajo son, al igual que en el caso de procesos y actividades, clases especializadas de la clase recurso correspondiente. El editor permite, al definir nuevos recursos, definir nuevas propiedades asociadas a los mismos. Esto se tiene en cuenta en la generación, de forma que para cada nueva propiedad que se añada al recurso, se añadirá una definición de atributo constante (sección constant_attributes) en la clase especializada que se genere. La sintaxis de dichas definiciones es: *Nombre_Nueva_Propiedad1(Tipo_Nueva_Propiedad1)*. A continuación se muestran las plantillas concretas para cada recurso.

- ***Plantilla Dato.*** Cada dato incluido dentro de un flujo de trabajo se corresponde con una clase especializada de la clase data del metamodelo OASIS, de acuerdo con el siguiente esquema. El literal Nombre_Dato se corresponde con el nombre asignado al recurso dato en su definición.

```
complex_class Nombre_Dato specialization_of data
constant_attributes
        Nombre_Nueva_Propiedad1(Tipo_Nueva_Propiedad1);
        ...
        Nombre_Nueva_Propiedadn(Tipo_Nueva_Propiedadn).
end_complex_class
```

- ***Plantilla Aplicación.*** Cada aplicación que forma parte del flujo de trabajo se corresponde con una clase especializada de la clase application del metamodelo OASIS. Su esquema es similar al visto para los datos, tan sólo cambia la superclase y el literal a sustituir por el nombre de la aplicación.

```
complex_class Nombre_Apli specialization_of application
constant_attributes
        Nombre_Nueva_Propiedad1(Tipo_Nueva_Propiedad1);
        ...
        Nombre_Nueva_Propiedadn(Tipo_Nueva_Propiedadn).
end_complex_class
```

- ***Plantilla Usuario.*** Cada usuario participante en el flujo de trabajo se corresponde con una especialización de la clase user del metamodelo OASIS. Se sigue el mismo esquema que para los recursos ya mencionados anteriormente.

```
complex_class Nombre_Usuario specialization_of user
constant_attributes
        Nombre_Nueva_Propiedad1(Tipo_Nueva_Propiedad1);
        ...
        Nombre_Nueva_Propiedadn(Tipo_Nueva_Propiedadn).
end_complex_class
```

- ***Plantilla Unidad Organizacional***. Cada unidad organizacional participante en el flujo de trabajo se corresponde con una especialización de la clase unidad organizacional del metamodelo OASIS. Su esquema es:

```
complex_class Nombre_Unidad specialization_of organisational_unit
constant_attributes
        Nombre_Nueva_Propiedad1(Tipo_Nueva_Propiedad1);
        ...
        Nombre_Nueva_Propiedadn(Tipo_Nueva_Propiedadn).
end_complex_class
```

d) Asociación de recursos a tareas

La asignación de recursos a tareas consta siempre de dos partes:

I. Creación del objeto que representa al recurso, la primera vez que se utilice dentro del flujo de trabajo. Esto se corresponde con la invocación del evento de creación de la clase correspondiente.

II. Asignación de dicho recurso a la tarea correspondiente. En este caso se crea una instancia del objeto agregado que representa dicha asociación, según las clases del metamodelo OASIS.

A continuación se muestran las distintas plantillas de asociación de recursos definidas.

- ***Plantilla de Asociación de Datos a Actividades.*** El código generado refleja las dos partes indicadas anteriormente. El parámetro *Oid* no está instanciado, sino que el sistema lo devolverá al animar la especificación; el resto de parámetros se instancian al valor de los atributos del nuevo dato que se ha definido con el editor. Se distingue entre si los datos son de entrada o de salida.

  Para los datos de entrada a una actividad:

```
Nombre :: new_data(Oid,Cod_Dato,Nombre,Desc,Path) &
usesDataAc :: new_usesDataAc (Oid,Cod_Dato,Nombre,Desc,Path,
                              [Cod_Act,Cod_Dato])
```

  Para los datos de salida de una actividad:

```
Nombre :: new_data(Oid,Cod_Dato,Nombre,Desc,Path) &
returnDataAC :: new_returnDataAc (Oid,Cod_Dato,Nombre,Desc,Path,
                                  [Cod_Act,Cod_Dato])
```

- ***Plantilla de Asociación de Aplicaciones a Actividades.*** Al igual que en la plantilla anterior, el parámetro Oid no está instanciado (lo devolverá el sistema al animar la

especificación) y el resto de parámetros se instancian al valor de los atributos de la aplicación definida.

```
Nombre :: new_app(Oid,Cod_Apli,Nombre,Desc,Path,Params,Perms,[],'')
&
invokesAppAAc :: new_invokesAppAAc(Oid,Cod_Apli,Nombre,Desc,Path,
                                   Params,Perms,[],'',[Cod_Act,Cod_Apli])
```

- ***Plantilla de Asociación de Actores a Actividades.*** En este caso, se distingue si el actor que se asocia a la actividad es un usuario o una unidad organizacional. La plantilla queda como se muestra a continuación. Al igual que en los patrones de asociación de recursos vistos anteriormente, el parámetro *Oid* no está instanciado y el resto de parámetros toman el valor de los atributos definidos para el actor (el usuario o la unidad organizacional, según corresponda).

  Para un usuario:

```
Nombre :: new_user(Oid,Cod_Usuario,Nombre,Desc) &
hasAcActor :: new_hasAcActor (Oid,Cod_Usuario,Nombre,Desc,
                              [Cod_Act,Cod_Usuario])
```

  Para una unidad organizacional:

```
Nombre :: new_orgUnit(Oid,Cod_OrgUnit,Nombre,Desc) &
hasAcActor :: new_hasAcActor (Oid,Cod_OrgUnit,Nombre,Desc,
                              [Cod_Act,Cod_OrgUnit])
```

- ***Plantilla de Asociación de Actores a Procesos.*** Esta plantilla coincide con la anterior, sólo varía que se asocia a un proceso en lugar de a una actividad. Queda por lo tanto como sigue:

  Para un usuario:

```
Nombre :: new_user(Oid,Cod_Usuario,Nombre,Desc) &
hasProcessActor :: new_hasProcessActor (Oid,Cod_Usuario,Nombre,
                                        Desc,[Cod_Act,Cod_Usuario])
```

  Para una unidad organizacional:

```
Nombre :: new_orgUnit(Oid,Cod_OrgUnit,Nombre,Desc) &
hasProcessActor :: new_hasProcessActor (Oid,Cod_OrgUnit,Nombre,
                                        Desc,[Cod_Act,Cod_OrgUnit])
```

6.1.3.2 Formalización del Editor

Desde la perspectiva formal que proporciona OASIS, el editor es una clase que recoge la estructura y funcionalidad descrita anteriormente. En la memoria no se incluye la especificación OASIS del editor ya que no es relevante para el objetivo de la tesis, pero sí es importante destacar que el servicio de generación de OASIS (generateOASIS) establece el paso del marco conceptual de flujos de trabajo, definido por el metamodelo de referencia, al marco conceptual OASIS. Es una transacción compleja, que hace uso de los servicios que proporciona la metaclase OASIS para construir la especificación equivalente al modelo de flujo de trabajo, a partir del estado almacenado en la propia instancia editor. La especificación construida puede ser animada posteriormente.

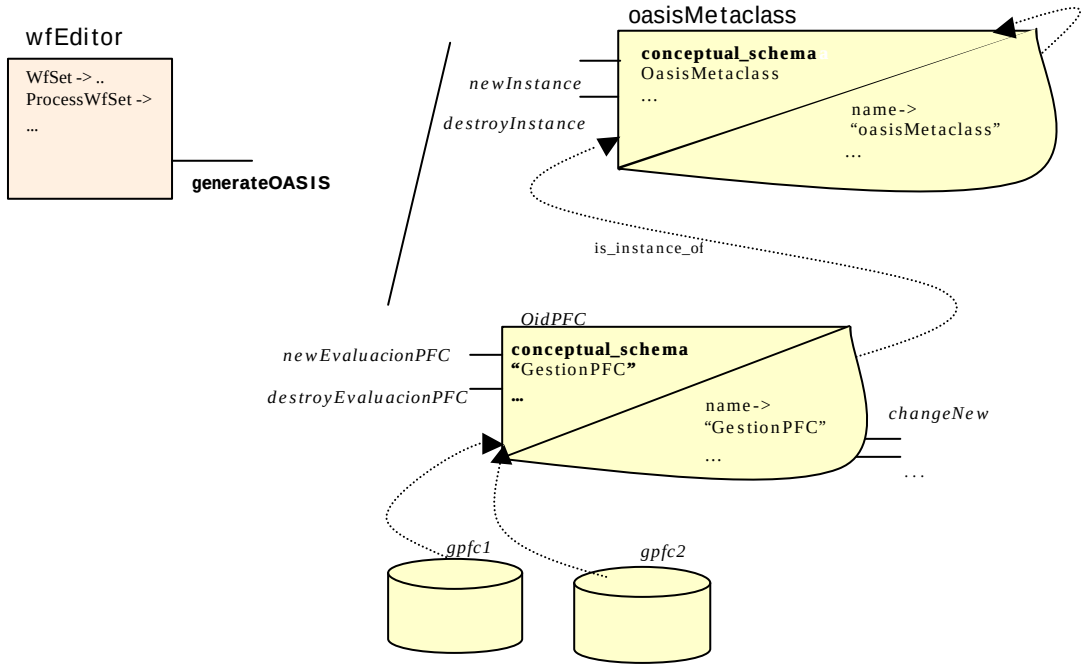


Figura 6-7. Formalización del editor de flujos de trabajo

La Figura 6-7 muestra la relación entre ambos marcos. Cada esquema conceptual que representa a un modelo de flujo de trabajo es una instancia de la metaclase OASIS. Esta instancia se ha construido invocando el servicio newInstance. Inicialmente, el esquema conceptual está vacío pero con la invocación de los servicios de cada *metainstancia*, se construye la especificaciones OASIS correspondientes al modelo de flujo de trabajo.

En la invocación de los servicios se distinguen dos partes:

1. Construcción de las clases correspondientes a la especificación OASIS del metamodelo de referencia.
2. Construcción de las clases correspondientes al flujo de trabajo concreto que se ha modelado.

La primera parte es necesaria porque dado un modelo de flujo de trabajo, su especificación OASIS correspondiente se ha basado en la utilización del operador de especialización, a partir de las clases que formalizan el metamodelo. Por lo tanto, este paso de construcción es el mismo para todos los flujos de trabajo. Una vez definido este marco genérico, la parte 2 es la encargada de invocar los servicios de los metaobjetos para construir clases especializadas de las anteriores, de acuerdo con el modelo construido.

A continuación, y siguiendo con el ejemplo correspondiente al proceso de evaluación de un PFC (evaluacionPFC) que se ha modelado anteriormente, se presenta un fragmento de la traza de eventos que definen dicho esquema conceptual, haciendo uso de los servicios que proporciona la metaclase OASIS (Figura 6-8).

//0. Se crea una nueva instancia de la metaclase OASIS en la que se definirá el esquema conceptual evaluacionPFC.

```
om[ev(newInstance, [“evaluacionPFC”]) -> EVALUACIONPFC[33]

  /El evento newInstance devuelve el oid del esquema creado en la
  /variable EVALUACIONPFC.

/ Se cambia el evento de creación y destrucción de instancias proceso

EVALUACIONPFC[ev(changeNew,[ev(newInstance,nil),
                           ev(newEvaluacionPFC,nil),nil])]
EVALUACIONPFC[ev(changeDestroy,[destroyEvaluacionPFC])]
```

// 1. Se construyen las clases que corresponden a la especificación OASIS del metamodelo de referencia.

```
/Se crea una instancia de la metaclase elemental_class, para definir
/la clase activity

c(EVALUACIONPFC,e_c)[34][ev(newClass,[“Activity”])->ACTIVITY]

/Se crean los atributos constantes y variables de la clase activity

ACTIVITY[ev(addConsAtt,[identifier,string])]
```

[33] Esta sintaxis es la utilizada en [Car99] para la formalización de la metaclaseOASIS (TF-Logic). La herramienta AFTER desarrollada en base a estas ideas, también utiliza esta sintaxis.

[34] También se puede identificar la metaclase por el sustituto: c(id(“oasisMetaclass”, cs,[“evaluacionPFC”]), class(“elemental_class”)).

```
ACTIVITY[ev(addConsAtt,[name,string])]
ACTIVITY[ev(addConsAtt,[description,string])]
ACTIVITY[ev(addVarAtt,[state,string])]
/ La invocación del resto de servicios de la metaclase vista como
/editor de clases, completa la definición.
```

// 2. Se construyen las clases que componen el flujo de trabajo . En este caso se sigue el mismo patrón de invocación de servicios visto anteriormente. Se muestra sólo cómo sería la creación de la clase evaluacionPFC que representa al proceso global.

```
/Se crea una instancia de la metaclase permanentSpecialization_class
c(EVALUACIONPFC,e_c)[ev(newClass,[“evaluacionPFC”])-> EPFC]
EPFC[ev(makeSubClassPS,[“Process”])]
/Se redefine el apartado processes de la clase
EPFC[ev(addProcess,[…])]
. . .
```

Figura 6-8. Fragmento de la secuencia de invocación de los servicios de la metaclase OASIS para la construcción de una especificación de flujo de trabajo

6.2 Prototipación Automática de Modelos de Flujo de Trabajo

La utilización de OASIS como lenguaje de especificación de flujos de trabajo nos garantiza soporte a la estabilización por la semántica operacional del lenguaje. Es decir, una vez obtenida la especificación OASIS de un modelo de flujo de trabajo, ésta es animada en un entorno de ejecución de OASIS con el objetivo de iniciar un proceso de validación con el usuario. Cada vez que se detecte un error, se modifica la especificación del flujo de trabajo y se vuelve a animar, hasta que ésta se considere correcta y estable.

El entorno elegido para realizar la animación de las especificaciones de flujos de trabajo es KAOS [Can95, Can96a]. La elección de este sistema está motivada por la utilización del lenguaje OASIS como lenguaje único de definición, manipulación y programación de aplicaciones. Estas características permiten que puede ser utilizado como entorno de prototipado de procesos [Pen99b].

A continuación se describen, las principales características de KAOS y su arquitectura, pasando después a detallar cómo se aborda la fase de prototipado.

6.2.1 El Sistema KAOS

KAOS[35] es un sistema gestor de bases de datos orientadas a objetos basado en el modelo OASIS, y que utiliza la versión del lenguaje denominada R-OASIS. Está implementado sobre KBMS1 [Man91], un Sistema Gestor de Bases de Conocimiento basado en la programación lógica. Permite almacenar, recuperar y actualizar grandes cantidades de información, almacenada en forma de hechos y reglas. Utiliza el lenguaje lógico *kbProlog* (basado en *Prolog* pero extendido con nuevos predicados y la noción de teoría).

6.2.1.1 Arquitectura del sistema

La arquitectura del sistema se muestra en la Figura 6-9. En ella se pueden ver sus componentes principales y la relación que existe entre ellos.

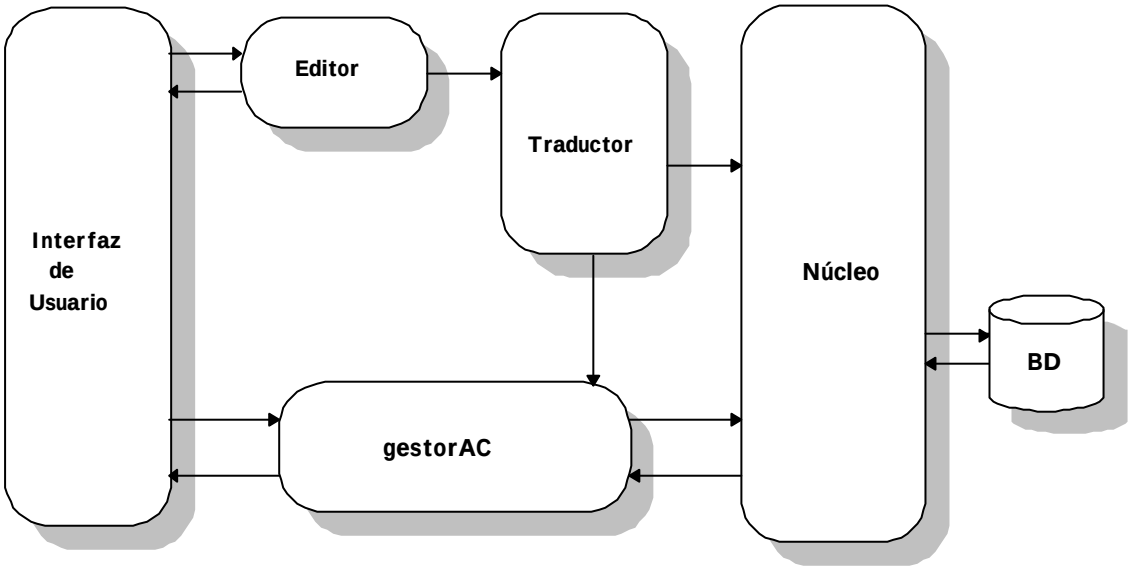


Figura 6-9. Estructura del sistema KAOS

El esquema básico de funcionamiento se describe a continuación. Dada una especificación OASIS, un *traductor* especializado [Meg94] construye un modelo en *kbProlog* del esquema definido, siguiendo las pautas derivadas de la implementación del modelo OASIS. Una vez generado un esquema en forma de (meta)teorías *kbProlog*, el *núcleo* [Car94] construye una versión persistente del mismo.

A partir de ese momento se está en condiciones de manipular la base de datos creada. Interactivamente, es posible observar a los objetos, activar eventos en los mismos, escribir

[35] ***K****nowledge-base* ***A****rchitecture for* ***O****bject* ***S****ocieties.*

programas y ejecutarlos, todo dentro del mismo entorno. Para poder ofrecer todos estos servicios, el sistema consta además de: una i*nterfaz de usuario* que convierte todos los eventos de teclado que éste genere en cadenas de caracteres que se le pasan al *gestor de actualizaciones y consultas* (*gestorAC*) [Pen94]. Éste descompone dichas cadenas en mensajes individuales, que chequea para comprobar su corrección, tanto sintáctica como semántica, y que posteriormente envia al *núcleo.* Además, este módulo es el encargado de ejecutar los programas que se escriban para manipular las bases de datos. El *núcleo* recibe los mensajes que le pasa el *gestorAC* y los resuelve en el caso de que éstos sean preguntas, o los ejecuta en el caso de que éstos sean disparos. Además, anima a los objetos, los hace persistentes, controla las transacciones y el tiempo, gestiona la composición paralela de las clases que compongan el sistema y recupera la base de datos en el caso de que se produzcan errores.

6.2.2 Soporte de KAOS al Prototipado de Flujos de Trabajo

La especificación OASIS correspondiente a un modelo de flujo de trabajo es la entrada al sistema KAOS para iniciarse una sesión de animación y validación de requisitos. Siguiendo con el ejemplo de la evaluación de PFC, a continuación se esboza cómo sería una sesión de trabajo en KAOS, para validar la lógica del proceso.

```
//  EJEMPLO DE EJECUCIÓN[36]
?.- evaluacionPFC::start(Oid,"10","Juan", "Propuesta evaluacion")[37]

Nueva Evaluación PFC
- Actividad-- tomaDecisionA
Datos Alumno: "Juan"
Evaluación Tribunal: "no"
  Proved.. solTribunal::nueva_solicitud(solEval00016, Juan, no)

- Actividad—- tomaDecisionD
Datos Director: "Luis"
Evaluación Tribunal: "no"
  Proved.. solEval000016::act_solicitud(Luis, no)

- Condición de transición—-
  Proved.. solEval00016 ?? decisionA(no)
  Proved.. solEval00016 ?? decisionD(no)

- Actividad—- evaluacionDirector
  Nota: 7
     Proved.. solEval00016::calificación(7)
   Yes
 ?.-[38]
```

[36] En cursiva aparece todo lo que el sistema KAOS va mostrando por pantalla como resultado de la ejecucion del modelo de flujo de trabajo. Lo que está en negrita es lo que introduce el actor.
[37] Con esta orden se lanza la ejecución proceso correspondiente a la evaluación del PFC.
[38] Finaliza la ejecución del flujo de trabajo evaluacionPFC y vuelve a aparecer el prompt del sistema.

6.3 Captura de Requisitos del Proceso de Negocio

La tarea de construir modelos de flujo de trabajo a partir de procesos de negocio de las organizaciones no es en absoluto trivial, sobre todo cuánto mayor sea su complejidad. Los SGFT permiten la definición de flujos de trabajo, pero no existe ningún tipo de soporte metodológico para la realización de esta tarea. Como ya se ha comentado, se proporciona una interfaz gráfica o textual que permite especificar el flujo de trabajo de acuerdo al metamodelo subyacente. El analista o diseñador del flujo de trabajo no cuenta con ningún tipo de metodología para entender el proceso de negocio que se sigue y plasmarlo de acuerdo con la terminología de flujos de trabajo.

En esta tesis se propone una capa de ingeniería de requisitos para la obtención de un modelo de flujo de trabajo [Pen01c]. La aproximación seguida se basa en la captura de requisitos del proceso de negocio mediante casos de usos que se construyen siguiendo una estrategia de modelado ascendente basada en la estructura organizativa. Los casos de uso se van refinando conforme ascendemos en la jerarquía organizacional. Los casos de uso construidos nos permiten obtener un modelo preliminar de flujo de trabajo. Esta transformación se basa en las equivalencias que se han establecido entre ambos metamodelos y un conjunto de patrones de proceso. Finalmente, el modelo de flujo de trabajo se puede refinar y modificar en el propio entorno de definición del SGFT.

En las siguientes secciones se describe el metamodelo de casos de uso utilizado, las equivalencias entre dicho metamodelo y el de flujos de trabajo, y el proceso de obtención del modelo de flujo de trabajo al incluir esta capa de ingeniería de requisitos. Finalmente, se muestra el proceso utilizando el caso de estudio.

6.3.1 Casos de Uso del Negocio

Son numerosos los métodos de desarrollo de software orientado a objetos que utilizan el *modelo de casos de uso* como uno de los pasos iniciales. Entre ellos podemos citar, por ejemplo, *UML* [Jac99] [Lar98], *Fusion* [Col94] y *Octopus* [Awa96]. El término *caso de uso* se define en UML como “una descripción de una secuencia de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable, de valor para un actor” [Boo99]. Aunque existan diferentes definiciones del término, de forma general podemos decir que los casos de uso permiten capturar requisitos funcionales (comportamiento) del sistema de una forma estructurada y orientada al proceso, sin tener que especificar cómo se implementa dicho comportamiento. La simplicidad de los mismos permite que se utilicen como medio de comunicación e interacción entre desarrolladores, usuarios finales y expertos del dominio, para llegar a tener todos ellos una comprensión común del sistema que se está analizando.

Un caso de uso describe un conjunto de pasos (denominados *eventos*) que representan la interacción de los elementos externos al sistema (denominados *actores*) con el propio sistema,

cuando se ejecuta parte de la funcionalidad contenida en el mismo. Un actor, por lo tanto, describe todo aquello que puede intercambiar información con el sistema a desarrollar, ya sean personas, dispositivos u otros sistemas. Una vez identificados los distintos casos de uso de un sistema, estos junto con las posibles relaciones que existan entre ellos, se denomina *modelo o diagrama de casos de uso*. Entre estas relaciones, las más típicas son: relación de uso o inclusión, relación de extensión y relación de especialización. En otras aproximaciones pueden considerarse otras relaciones, aparte de las mencionadas anteriormente; por ejemplo en [Con99] se introduce también una relación de afinidad entre casos de usos.

En [Kru98, Jac99] se introducen los casos de uso del negocio como aquellos que describen el proceso de negocio de una organización en términos de acciones a realizar y actores que participan, pero siempre bajo la visión del negocio.

Puesto que dependiendo de la aproximación utilizada pueden existir distintos matices, en la siguiente sección se detallan cuáles son las características del modelo de casos de uso a considerar en el presente trabajo.

6.3.1.1 Metamodelo de Casos de Uso de Referencia

De acuerdo con el objetivo que se persigue (modelar el proceso de negocio), la construcción de los casos de uso se realiza desde la perspectiva del negocio, representándose los requisitos del negocio y la interacción con sus participantes, denominados actores.

El metamodelo que se toma de referencia en el presente trabajo es el que se muestra en la Figura 6-10. Se considera que un caso de uso del negocio[39] posee un identificador numérico, un nombre, un propósito, una referencia al diagrama que pertenece y una descripción del proceso que representa. La descripción de este proceso se compone de los siguientes elementos:

- *Precondiciones y postcondiciones*: Hacen referencia, respectivamente, al estado en el que debe encontrarse el negocio para que pueda ejecutarse el caso de uso, y al estado al que se llega una vez que se ha ejecutado dicho caso de uso.
- *Actores*: Lista ordenada de los actores que participan en el caso de uso. El primer actor de la lista es considerado el actor principal y el resto actores secundarios.
- *Flujo de eventos:* Describe la comunicación entre el negocio y los actores. Por una parte, se describen los eventos que generan los actores, y por otra, las obligaciones del negocio. Además, se incluyen los puntos de extensión, que dan cuenta de las situaciones excepcionales que pueden producirse.

Las relaciones que se consideran entre los casos de uso, con la finalidad de describir el modelo de casos de uso que represente al negocio son: *utiliza* (relación uses[40]) y *extiende*

[39] En lo sucesivo se denomina caso de uso.
[40] En las últimas versiones de UML, se le denomina relación de inclusión y se representa por includes.

(relación extends*).* La relación uses significa la inserción textual de un caso de uso dentro de otro; la relación extends es similar, un caso se puede insertar en otro cuando se cumplen ciertas condiciones.

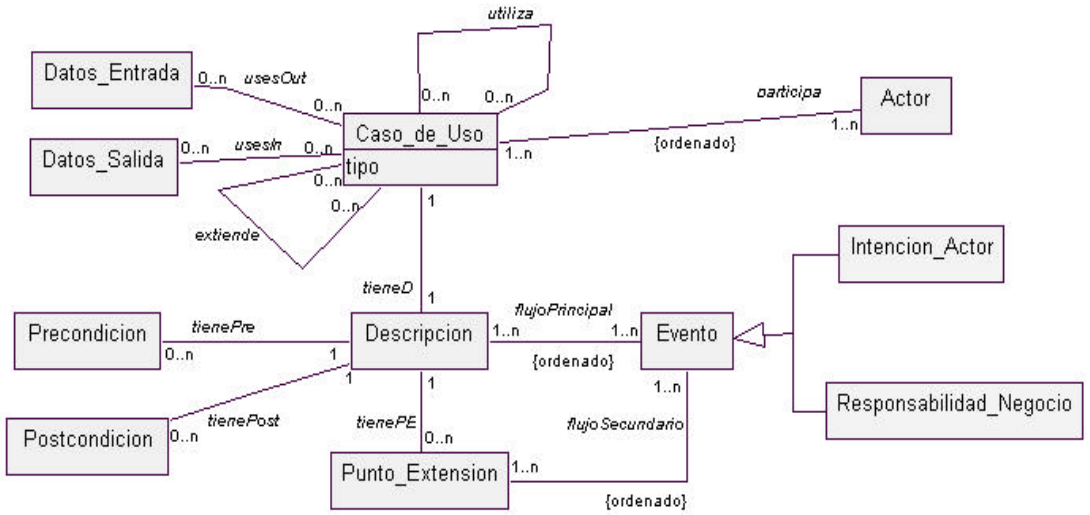


Figura 6-10. Metamodelo de casos de uso del negocio

Dado que nuestro interés es utilizarlos en un dominio de aplicación concreto, se han incorporado nuevas propiedades que dan cuenta de aspectos específicos del dominio, y por ello los denominados *casos de uso extendidos.* Antes de pasar a ver las nuevas propiedades que se han añadido, introducimos dos conceptos, que son necesarios para una mayor claridad :

- Llamaremos ***caso de uso elemental*** a aquel que no está relacionado con ningún otro caso de uso, mediante cualquiera de las relaciones vistas.
- Llamaremos ***caso de uso complejo*** a aquel que no es elemental, es decir, a aquel caso de uso que está relacionado con otros casos de uso mediantes las relaciones de uso o extensión.

Vista las definiciones anteriores, y teniendo en cuenta que nuestro objetivo es capturar requisitos de los procesos de negocio para obtener un modelo de flujo de trabajo, las propiedades que se han incorporado a todo caso de uso dan cuenta de aspectos relacionados directamente con los flujos de trabajo. Estas son:

- *Tipo de proceso*: Para dar valor a esta propiedad se debe tener en cuenta si se trata de un caso de uso elemental o complejo. En caso de ser un caso de uso elemental, el

proceso puede ser *manual* o *automático*, dependiendo de si es necesaria o no la partipación humana para su realización y exista o no alguna aplicación que realice dicho proceso. Si se trata de un caso de uso complejo, es imposible precisar, a este nivel, si se trata de un proceso manual o automático, ya que en la práctica, se podrá descomponer tanto en actividades manuales como automáticas. Por lo tanto, en este caso el tipo de proceso se etiqueta como *complejo*.

- *Datos de entrada*: Los datos que necesita el caso de uso para poder llevarse a cabo. Estos datos pueden tomarse, por ejemplo, de una base de datos o de cualquier otra fuente de datos.
- *Datos de salida*: Los datos que genera la ejecución del caso de uso.

El *modelo de casos de uso extendido* obtenido se describe utilizando dos notaciones: una gráfica y una textual. La notación gráfica es utilizada habitualmente en los modelos de casos de uso. La Figura 6-11 muestra dicha notación, que como se puede apreciar es muy concisa e intuitiva. El caso de uso se representa como una elipse y tiene asociados los actores que están involucrados en el mismo. Las relaciones entre casos de uso se visualizan con líneas que los unen. Estas líneas tienen distinta forma y distinta etiqueta según la relación de que se trate. En este caso, el caso de uso CU1 utiliza el CU2, y el CU3 es una extensión del caso de uso CU1.

La descripción textual del caso de uso viene dada en forma de plantillas que recogen todas las propiedades del caso de uso, normalmente en lenguaje natural. En nuestro caso, y de acuerdo con todas las propiedades vistas anteriormente, la plantilla textual para describir cada uno de los casos de uso extendidos que se vayan detectando es la que se muestra en la Figura 6-12.

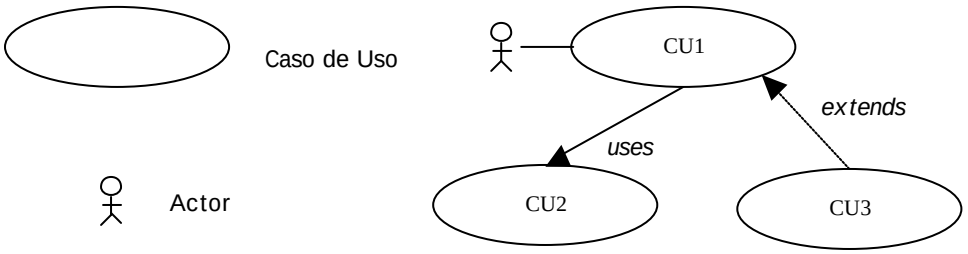


Figura 6-11. Notación gráfica para representar los casos de uso extendidos

Identificación	
ID:	**Nombre**:
Propósito:	
Diagrama:	
Relaciones:	
Uso	
Extensión	
Proceso	
Tipo Proceso	
Datos Entrada	
Datos Salida	
Precondiciones	
Postcondiciones	
Actores	
Flujo de Eventos (Comunicación Actor-Negocio)	
(Intención Actor)	*(Responsabilidad del Negocio)*
Extensiones.	

Figura 6-12. Plantilla textual para los casos de uso extendidos

6.3.2 Equivalencias entre Conceptos del Metamodelo de Casos de Uso y del Metamodelo de Flujo de Trabajo

Vistos el metamodelo de casos de uso extendido y el metamodelo de flujo de trabajo, se pueden establecer una serie de equivalencias entre ellos, puesto que muchos de los conceptos que se manejan en ambos coinciden. La Tabla 6-2 resume las equivalencias entre ambos modelos.

Un modelo de caso de uso construido de acuerdo con las características vistas anteriormente se corresponde con un modelo de flujo de trabajo. Aquellos componentes que sean casos de uso elementales, se corresponden con actividades en el modelo de flujo de trabajo, mientras que los casos de uso complejos se corresponden con subprocesos. En este último caso, la obtención del proceso correspondiente se basa en el flujo de eventos de los casos de uso y las relaciones entre ellos.

A continuación se enuncian las reglas de transformación que permiten obtener un modelo de flujo de trabajo, a partir de un modelo de casos de uso, teniendo en cuenta las equivalencias de la Tabla 6-2.

Conceptos de Casos de Uso	***Conceptos de Flujos de Trabajo***
Modelo de Casos de Uso	Modelo de Flujo de Trabajo
Caso de Uso Elemental	Actividad
Nombre del Caso de Uso Tipo de Proceso Datos de Entrada Datos de Salida Precondición Postcondición Actores Flujo de Eventos	Nombre de la Actividad Tipo de Actividad Datos de Entrada Datos de Salida Condición de Inicio Condición de Finalización Actores Flujo de Acciones
Caso de Uso Complejo	Subproceso
Flujo de eventos + Relaciones	Patrones de Proceso

Tabla 6-2. Equivalencias entre casos de uso del negocio y flujos de trabajo

Se definen el siguiente conjunto de reglas:

R1. Cada caso de uso elemental del modelo de casos de uso es transformado en una actividad del modelo de flujo de trabajo.

R1.a. El tipo de proceso de un caso de uso determina el tipo de actividad.

R1.b. Las precondiciones y las postcondiciones de un caso de uso se convierten, respectivamente, en las condiciones de inicio y las condiciones de finalización de la actividad que lo representa.

R1.c. El flujo de eventos del caso de uso determina las acciones concretas asociadas a una actividad[41].

R1.d. Los datos de entrada y los datos de salida de un caso de uso se convierten, respectivamente, en los datos de entrada y datos de salida de la actividad que lo representa.

R1.e. Los actores de un caso de uso se convierten, respectivamente, en los actores asociados a la actividad que lo representa. Se deben tener en cuenta varios casos:

R1.e.1. En caso de existir más de un actor asociado al caso de uso, en el modelo de flujo de trabajo asociado sólo se considera como actor el actor primario del caso de uso.

[41] Es importante recordar que las actividades son tareas atómicas que tienen asociada unas acciones especificas, en principio, no descomponibles.

R1.f.2. Puesto que en el caso de un modelo de flujo de trabajo, los actores asociados a las actividades pueden ser usuarios concretos o unidades organizacionales, a la hora de asociar a la actividad el actor del caso de uso, se creará una unidad organizacional.

R2. Cada caso de uso complejo es transformado en un subproceso del modelo de flujo de trabajo. La definición de dicho subproceso se hace en función de dos patrones de proceso que se han definido, junto con la aplicación de la regla R1. Estos patrones, que surgen como resultado del análisis de las relaciones entre casos de uso, son dos: el patrón de uso y el patrón de extensión. A continuación se describen más detalladamente cada uno de ellos.

6.3.2.1 El patrón de uso (relación-*uses*)

Si existe una relación de uso (*uses*) entre casos de uso, se define el siguiente patrón de generación de proceso tras analizar las relaciones concretas entre el flujo de eventos y punto de uso o inclusión de otro caso de uso, dentro del caso de uso complejo. Este patrón de proceso se denomina *patrón de uso*.

Dado un caso de uso, uc1, el cual está relacionado por uso con otro caso de uso, uc2; y el flujo de eventos de uc1 es {ev_1, ev_2 , ... , ev_i, UP1, ev_{i+1}, ... , ev_n }, donde UP1 representa el punto de uso al caso de uso uc2. Tenemos que, en el modelo de flujo de trabajo se va a generar un proceso[42] que consta de un conjunto de tareas relacionadas mediante flujo de control. En concreto, los eventos que existen en el caso de uso uc1 justo hasta el punto de uso (desde ev_1 a ev_i), constituyen una actividad, que agrupa dichos eventos como acciones concretas; del mismo modo, los eventos de uc1 que aparecen después del punto de uso (desde ev_{i+1} a ev_n), constituyen otra actividad. Entre ambas actividades, se incluye una tarea que corresponde al caso de uso uc2; esta tarea, en principio, es un subproceso, puesto que dependiendo de la complejidad con que esté definido uc2, se pueden generar nuevos subprocesos al volver a aplicar los patrones si existen nuevas relaciones con otros casos de uso, o bien, convertirse en una actividad, si no existen tales relaciones.

La Figura 6-13 muestra gráficamente el patrón de uso. La notación utilizada es la presentada en la tesis, tanto para el modelo de casos de uso, como para el modelo de flujo de trabajo. En este caso concreto, la actividad ac1 agrupa las acciones correspondiente al flujo de eventos ev1 y ev2; mientras que la actividad ac2 contiene únicamente las acciones correspondientes al flujo de eventos ev3[43].

[42] Como ya se ha dicho, este proceso constituye un subproceso dentro del modelo de flujo de trabajo resultante.

[43] Este es el resultado que se genera en el modelo de flujo de trabajo que se está derivando, al aplicar el patrón de proceso visto. Hemos de tener en cuenta que este modelo de flujo de trabajo es una

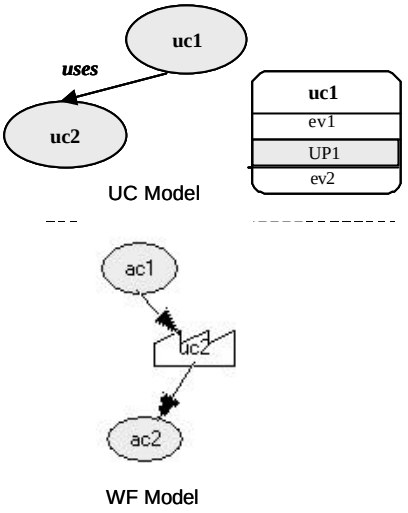


Figura 6-13. El patrón de uso

6.3.2.2 El patrón de extensión (relación-*extends*)

En caso de que la relación entre casos de uso sea de extensión (*extends*), se define el siguiente patrón de generación de proceso, denominado *patrón de extensión*. Dado un caso de uso, uc1, el cual está relacionado por extensión con otro caso de uso, uc2; y el flujo de eventos de uc1 es {ev_1, ev_2 , ... , ev_i, EP1, ev_{i+1}, ... , ev_n }, donde EP1 representa el punto de extensión al caso de uso uc2. Tenemos que se va a generar un proceso[44] en el modelo de flujo de trabajo, formado por dos actividades cuyas acciones van desde (ev_1 a ev_i) y desde (ev_{i+1} a ev_n), respectivamente, y por una bifurcación condicional entre ambas que se corresponde con el punto de extensión EP1. La bifurcación condicional tendrá tantos flujos de salida como condiciones distintas tenga el punto de extensión, siendo ahí donde se invoca al subproceso que corresponde al uc2. Al igual que antes dependiendo de la complejidad con que esté definido uc2, se pueden generar nuevos subprocesos o bien convertirse en una actividad.

La Figura 6-14 muestra gráficamente el patrón de extensión. En este caso, la actividad ac1 agrupa las acciones correspondiente al flujo de eventos ev1 y ev2; mientras que la actividad ac2 contiene únicamente las acciones correspondientes al flujo de eventos ev3. Estas actividades junto el subproceso generado para el uc2 pueden expandirse en subprocesos o convertirse en una actividad, respectivamente, en función de su complejidad.

propuesta y que como tal se puede modificar, a posteriori, por el analista de flujos de trabajo, como se verá en la siguiente sección.

[44] Al igual que en el patrón de uso, este proceso constituye un subproceso dentro del modelo de flujo de trabajo resultante.

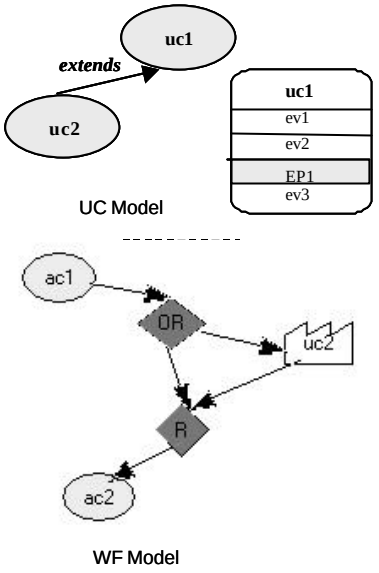


Figura 6-14. El patrón de extensión

6.3.3 Derivación de Modelos de Flujo de Trabajo a partir de Modelos de Casos de Uso

Las equivalencias establecidas entre los casos de uso extendidos y los flujos de trabajo son la base para definir un proceso de generación de modelos de flujos de trabajo a partir de modelos de casos de uso extendidos, que expresen los requisitos del proceso de negocio. Este proceso de generación consta de cinco pasos fundamentales, que podemos agrupar en dos partes: obtención del modelo de casos de uso y obtención del modelo de flujo de trabajo.

- La primera parte consta de tres pasos, dos de los cuales se aplican de forma iterativa hasta obtener el modelo de casos de uso completo.
- La segunda parte, consta de dos pasos, tras los cuales se obtiene el modelo de flujo de trabajo.

La Figura 6-15 muestra una visión general del proceso de generación, mostrando los diferentes pasos a seguir y la documentación que se genera en cada uno de ellos. A continuación se describen estos pasos con mayor nivel de detalle.

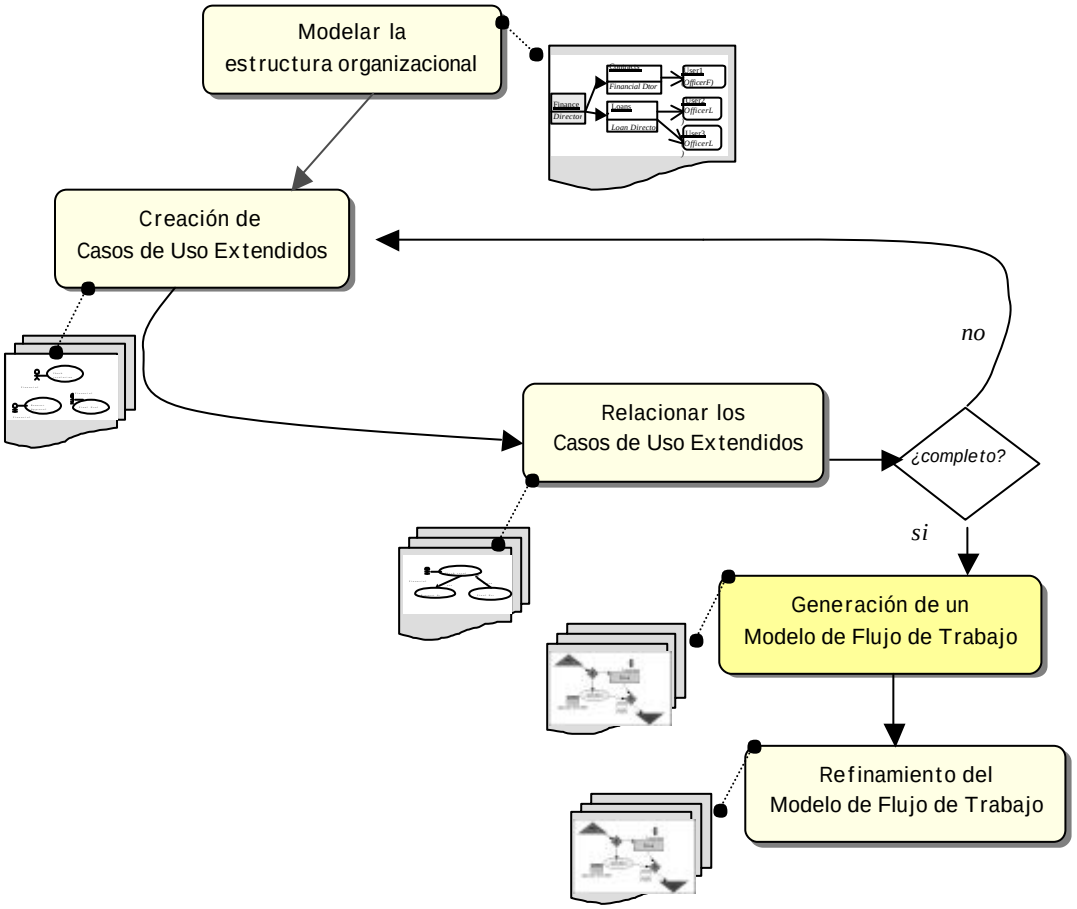


Figura 6-15. Proceso de generación de flujos de trabajo a partir de casos de uso

➔ ***Paso 1.*** **Modelar la estructura organizacional.** El primer paso de este proceso consiste en modelar la organización en términos de su estructura organizativa, es decir, usuarios que la forman y relaciones que existen entre ellos (grupos de usuarios, roles que desempeñan, dependencias, etc.). El modelo organizacional es imprescindible para realizar correctamente los pasos 2 y 3, puesto que como se verá a continuación, va a ser el que guíe el orden de creación y refinamiento de los casos de uso extendidos.

Teniendo como objetivo disponer de un modelo organizacional, la cuestión a resolver es qué modelo organizacional adoptamos, puesto que existen varias propuestas tales

como [Yu96] y [Ley00], que captan las relaciones entre los usuarios de una organización, pero desde enfoques distintos. Por ejemplo, el marco i* [Yu96] propone una visión de los procesos de negocio que existen en una organización como un conjunto de actores que dependen unos de otros para llevar a cabo una cierta meta u objetivo, realizando las tareas oportunas y utilizando los recursos necesarios para ello. Este marco incluye dos modelos: *modelo de dependencias estratégicas*, que describe en una red las relaciones entre actores, y el *modelo de razonamientos estratégicos*, el cual describe la razón por la que cada actor se relaciona con cualquier otro. Trabajos más recientes, como [Cat01], usan el marco i* para modelar requisitos tempranos, que posteriormente son transformados en especificaciones pUML (*preciseUML*) [Eva99] mediante un conjunto de guías. Esta transformación está básicamente orientada a la estructura, obteniéndose un diagrama de clases junto con un conjunto de restricciones en OCL [War99]. Por otra parte, el modelo organizacional que aparece en [Ley00] y que es muy similar al utilizado por ciertos productos de flujos de trabajo comerciales, como MQSeries WF, se basan más en determinar la estructura de la organización, en cuanto a departamentos, equipos, roles, posiciones, relaciones entre ellos y personas que forman la organización. Puesto que nuestro objetivo final es obtener un modelo de flujo de trabajo, el modelo organizacional elegido está más cercano a los modelos utilizados en los sistemas de flujo de trabajo. En concreto, la estructura de la organización se modela en términos del modelo organizacional introducido en el metamodelo de referencia (pág 75). De acuerdo con este modelo se ha definido una notación gráfica que permita de forma sencilla tener una visión de la estructura organizacional (ver Figura 6-16).

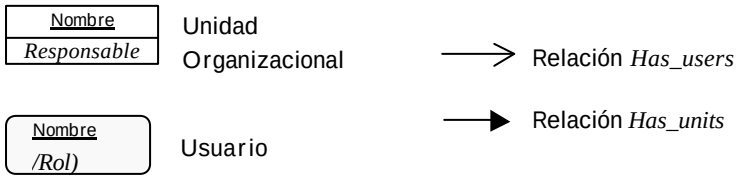


Figura 6-16. Notación gráfica para la estructura organizacional

Una vez obtenido el modelo organizacional, se inicia un proceso iterativo formado por los pasos 2 y 3 para obtener el modelo de casos de uso extendidos. Este proceso iterativo está guiado por el modelo organizacional que se recorre de abajo hacia arriba (se sigue una aproximación *bottom-up*) y finaliza cuando el modelo de casos de uso esté completo, lo cual supone haber recorrido toda la estructura organizativa. Antes de comentar qué tareas se realizan concretamente en cada uno de estos pasos, la idea subyacente en este proceso iterativo de construcción del modelo de casos de uso es la siguiente: cada miembro de la organización tiene su propia vista del proceso de negocio. Estas vistas son distintas pero

todas ellas son valiosas a la hora de elicitar los requisitos del mismo. La vista que cada usuario tiene está condicionada fundamentalmente por la tarea concreta que realiza, la cual conoce o debe conocer perfectamente; sin embargo, puede ocurrir, sobre todo si el proceso es complejo, que no tenga una visión completa y correcta del proceso global que se realiza. Por el contrario, el usuario responsable de una cierta unidad organizacional no tiene por qué conocer con detalle todas y cada una de las tareas que se realizan por parte de los miembros bajo su responsabilidad, pero sí tendrá una visión global del orden de dichas tareas, es decir, del proceso que se sigue. Por lo tanto, podemos aprovechar este conocimiento a distintos niveles de los miembros de una organización, para aplicar de forma iterativa los pasos 2 y 3 y obtener un modelo de casos de uso extendido. A continuación vemos las tareas concretas de cada uno de estos pasos.

- ***Paso 2*. Creación de Casos de Uso Extendidos.** En este paso, se crea un caso de uso extendido por cada una de las tareas individuales que tienen lugar en la organización. Los requisitos del proceso de negocio se capturan mediante entrevistas con los empleados que actualmente realizan una determinada actividad dentro de la organización. El objetivo de estas entrevistas es capturar el conocimiento que el propio usuario, realizador de cierta actividad, tiene de la misma. Como resultado se detectan un conjunto de casos de uso que representan actividades concretas que se realizan en la organización y que son documentados con la plantilla textual presentada.

- ***Paso 3*. Relacionar los Casos de Uso Extendidos.** Los casos de uso extendidos que se han detectado en el paso 2, no existen de forma aislada, sino que suelen relacionarse unos con otros. El objetivo de este paso es refinar el modelo de casos de uso obtenido, aplicando las relaciones de uso y extensión entre ellos. Esta tarea la realiza el responsable de cada unidad organizacional que tiene una visión más amplia del proceso global, frente a cada uno de los usuarios que la forman. Esto permite producir modelos más concisos, al eliminarse redundancias y factorizar comportamientos duplicados.

- ***Paso 4*** **Generación de un Modelo de Flujo de Trabajo.** Este paso consiste en la generación automática de un modelo de flujo de trabajo preliminar, a partir del modelo de casos de uso extendido obtenido tras finalizar las iteraciones del paso 2 y 3. La obtención del modelo de flujo de trabajo se realiza en base a las equivalencias establecidas entre ambos modelos, junto con los patrones de proceso definidos (sección 6.3.2; pág. 136).

➔ ***Paso 5*. Refinamiento del Modelo de Flujo de Trabajo.** El modelo de flujo de trabajo obtenido tras el paso anterior contiene un conjunto de tareas, con sus recursos asociados. Sin embargo, existen aspectos del proceso de negocio que no pueden ser capturados con la expresividad que proporcionan los casos de uso, como es la obtención de todas las relaciones de flujo de control entre tareas. En concreto, si el caso de uso es complejo, con la aplicación de los patrones de proceso sí obtenemos el flujo de control. En cambio, los casos de uso elementales quedan como tareas aisladas. La inclusión del flujo de control no generado, así como cualquier modificación que se desee realizar sobre el modelo de flujo de trabajo, se hará utilizando las funcionalidades de definición del SGFT.

En la siguiente sección se muestra un ejemplo de aplicación de este proceso de generación de un modelo de flujo de trabajo, siguiendo el caso de estudio.

6.3.4 Ejemplo

Dentro de la gestión de PFC, el proceso de solicitud y asignación de temas de proyectos a los alumnos depende del tipo de proyecto a realizar (A, B y C). Dicho proceso presenta variaciones en función del tipo. En este caso, podemos aplicar el proceso de generación visto y utilizar los casos de uso para capturar los requisitos del negocio. Vemos los distintos pasos.

Paso1. Modelar la estructura organizativa. En este caso, se han considerado tres unidades organizacionales: Alumnado, Comisión de Proyectos y Administración (Figura 6-17).

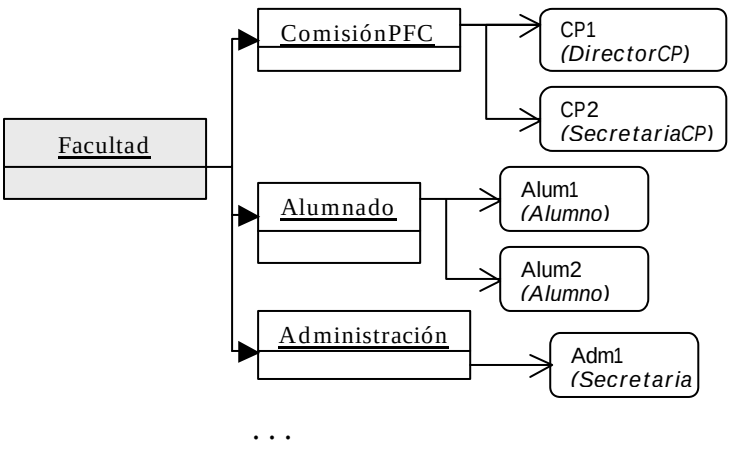


Figura 6-17. Parte del modelo organizacional de la FI

Paso 2. Creación de Casos de Uso Extendidos. El modelo organizacional guía el proceso de obtención de los distintos casos de uso correspondientes a las tareas que tienen lugar en el proceso de solicitud y asignación de temas de PFC. Cada uno de los casos de uso identificados (ver Figura 6-18) se representa gráficamente y se documenta con la plantilla textual mostrada anteriormente.

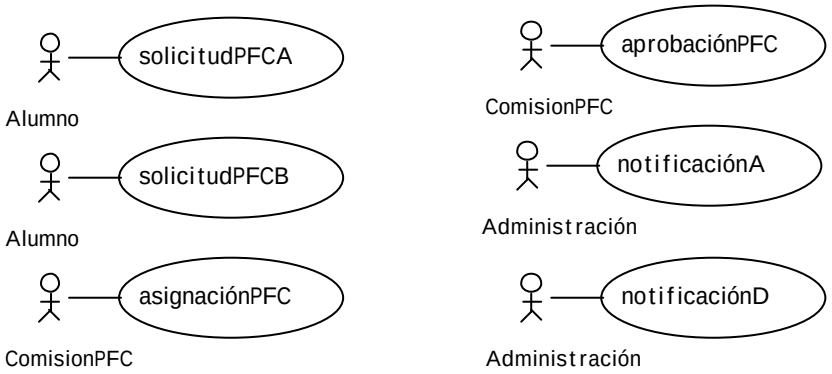


Figura 6-18. Casos de uso extendidos (1)

Paso 3. Relacionar los Casos de Uso Extendidos. La Figura 6-19 muestra el modelo de casos de uso refinado con las relaciones de uso y extensión. Al igual que antes, todos ellos están documentados con la plantilla textual. La Figura 6-20 muestra cómo quedaría dicha plantilla para el caso de uso aprobaciónPFC.

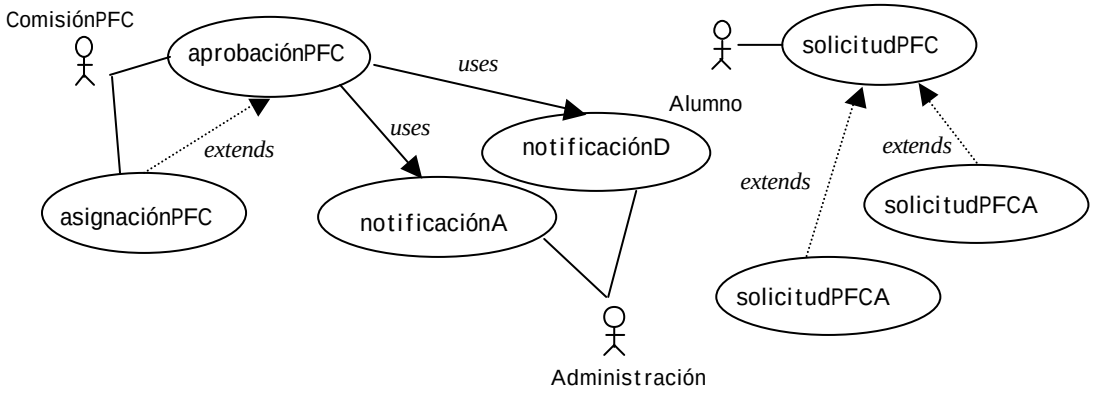


Figura 6-19. Modelo de casos de uso extendido (2)

Paso 4. Generación de un Modelo de Flujo de Trabajo. Este paso está automatizado y aplica las equivalencias establecidas y los patrones de proceso definidos, obteniéndose un modelo de flujo de trabajo (ver Figura 6-21):

Identificación		
ID: 2	**Nombre**: aprobaciónPFC	
Propósito: Asignación de tema de PFC a un alumno		
Diagrama: D1		
Relaciones:		
Uso	notificaciónA, notificaciónD	
Extensión	asignaciónPFC	
Proceso		
Tipo Proceso	complejo	
Datos Entrada	solicitudPFC	
Datos Salida	PFCasignados	
Precondiciones	El alumno debe estar matriculado de la asignatura PFC	
Postcondiciones	--	
Actores	ComisiónPFC, Administración	
Flujo de Eventos (Comunicación Actor-Negocio)		
(Intención Actor)		*(Responsabilidad del Negocio)*
		1. Comprobar datos solicitud PFC
2. Aprobación tema de PFC		
		3. Uso "notificaciónA"
		3. Uso "notificacionD"
Extensiones.		
Si en el punto 2 si "tipoPFC = A" entonces "asignaciónPFC"		

Figura 6-20. Plantilla para el caso de uso aprobaciónPFC

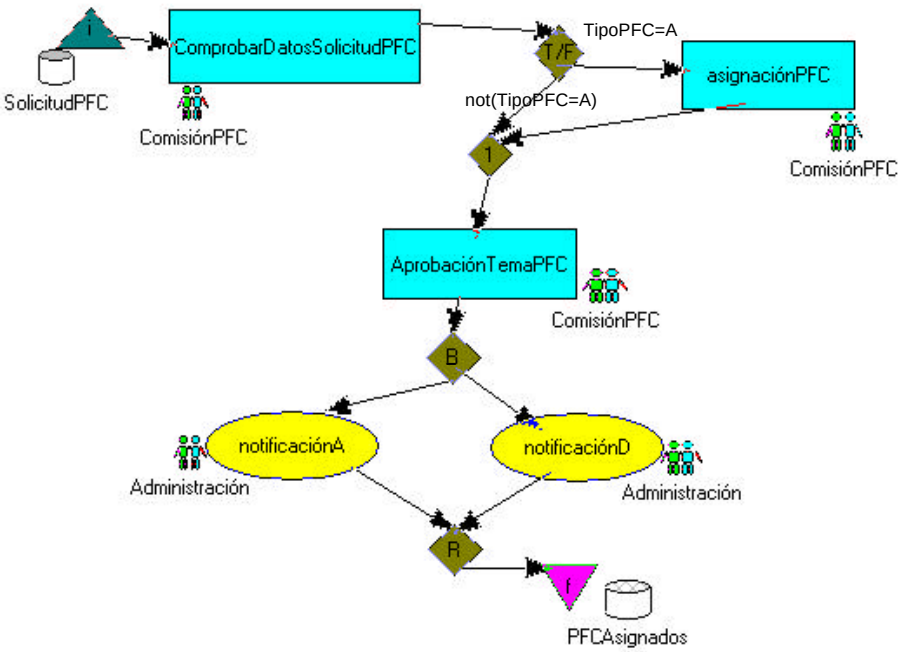


Figura 6-21. Subproceso generado para aprobaciónPFC

Paso 5. Refinamiento del Modelo de Flujo de Trabajo. En el subproceso de aprobación de un PFC, se ha generado automáticamente el flujo de control, pero se puede refinar el modelo incluyendo la aplicación asociada a las actividades automáticas y los datos concretos a cada unas de las actividades.

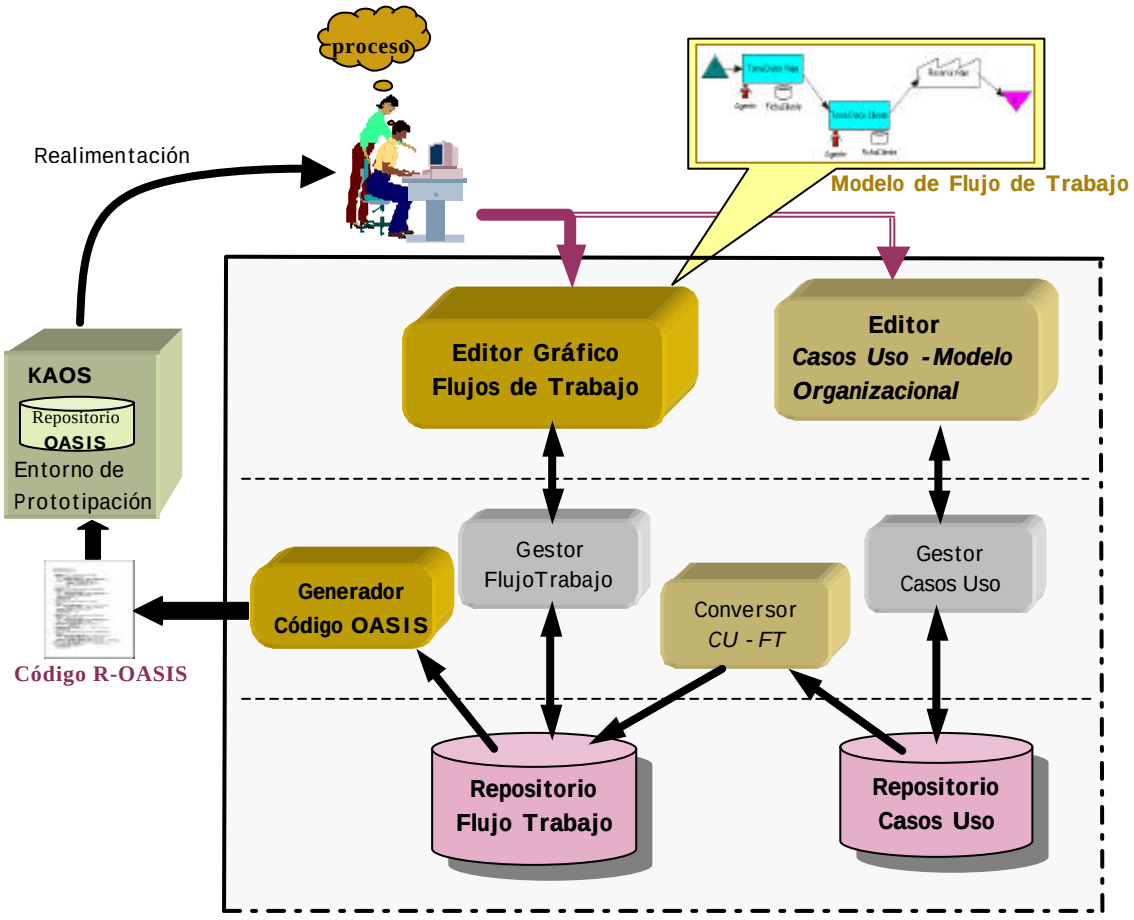


Figura 6-22. Arquitectura del entorno de soporte al desarrollo de flujos de trabajo (*Definición*)

6.4 Conclusiones

En este capítulo se ha presentado el subproceso de definición y estabilización de flujos de trabajo que integra la especificación, prototipación y captura de requisitos como pasos principales dentro del mismo. La Figura 6-22 muestra la arquitectura de la parte de entorno que da soporte a este subproceso. La construcción del modelo puede iniciarse directamente

sobre un editor de flujos de trabajo que proporciona el lenguaje gráfico definido y los servicios de edición presentados, o bien, utilizar la aproximación propuesta para la captura de requisitos del proceso de negocio mediante la construcción de casos de uso del negocio.

En caso de decidir utilizar la segunda opción, el conversor CU-FT implementa las equivalencias y patrones definidos entre los casos de uso y los flujos de trabajo, obteniéndose una versión del modelo de flujo de trabajo. Las facilidades del editor gráfico permiten refinar y acabar el modelo, que finalmente queda almacenado en un repositorio.

En ese momento, si se desea, se puede iniciar un proceso de prototipación automática para la validación de los requisitos del proceso de negocio modelado, con los miembros de la organización. Esta prototipación produce una realimentación al analista que es utilizada para modificar el modelo realizado hasta que represente fielmente al proceso de negocio. La entrada al entorno de prototipación, KAOS en este caso, es la salida del módulo de generación de código OASIS, que implementa las plantillas de generación descritas en el capítulo. Al finalizar el proceso de construcción y validación, el modelo de flujo de trabajo queda almacenado en el repositorio. En [Pen02] aparece el desarrollo completo de la gestión de Proyectos Final de Carrera de la Facultad de Informática siguiendo la aproximación presentada.

Capítulo 7.

Ejecución de Flujos de Trabajo

En este capítulo se aborda la ejecución de modelos de flujos de trabajo en SGFT. La generación de especificaciones en el SGFT destino se incluye como parte del modelo de proceso de desarrollo, haciéndose uso de técnicas de generación de código basadas en modelos [Bell98]. La obtención de la especificación del flujo de trabajo en el lenguaje específico del SGFT destino permite integrar en dichos sistemas las actividades de construcción y estabilización propuestas en el capítulo anterior.

El capítulo se estructura como sigue. En la sección 7.1 se da una descripción general de cómo está planteada la ejecución de modelos de flujo de trabajo, y en la sección 7.2 se aplica a un sistema concreto, OPERA. Finalmente, en la sección 7.3 se muestran las conclusiones.

7.1 Ejecución de un Modelo de Flujo de Trabajo

Los SGFT ofrecen buenas prestaciones a nivel de ejecución de flujos de trabajo, con entornos eficientes que permiten la automatización de procesos definidos de acuerdo al metamodelo subyacente. Precisamente es en la funcionalidad de ejecución donde han centrado gran parte de los esfuerzos, tanto a nivel de investigación como a nivel tecnológico, durante los últimos años. En el modelo de proceso de desarrollo propuesto, la fase de ejecución se plantea como una utilización de los SGFT existentes; en ellos se realiza la ejecución del modelo construido y validado de acuerdo al metamodelo de referencia que hemos propuesto. Por lo tanto, el entorno de soporte a la fase de ejecución debe permitir dicho enlace.

La solución planteada consiste en incluir una etapa denominada *implementación*, cuyo objetivo es la obtención de una versión ejecutable del modelo de flujo de trabajo construido.

Necesitamos un proceso de transformación de conceptos del metamodelo de referencia a conceptos del metamodelo soportado por el SGFT concreto, sobre el que se va a realizar la ejecución de

l modelo. Este proceso es totalmente dependiente del SGFT destino. Un compilador de modelos implementan las equivalencias entre metamodelos.

Desde el punto de vista del entorno de soporte al desarrollo de flujos de trabajo, la Figura 7-1 muestra cómo quedaría su arquitectura. Se muestra en color más ténue la parte de definición y construcción, y aparece resaltada la parte de ejecución. El punto de unión entre ambas partes es el metamodelo de referencia, puesto que en base a él se han establecido las equivalencias del proceso de transformación. Un compilador de modelos toma como entrada la información del modelo de flujo de trabajo que está almacenada en el repositorio y genera la especificación correspondiente en el lenguaje destino del SGFT elegido. Debe existir un compilador de modelos distinto por cada SGFT en el que se desee ejecutar el flujo de trabajo, de acuerdo con lo mencionado anteriormente.

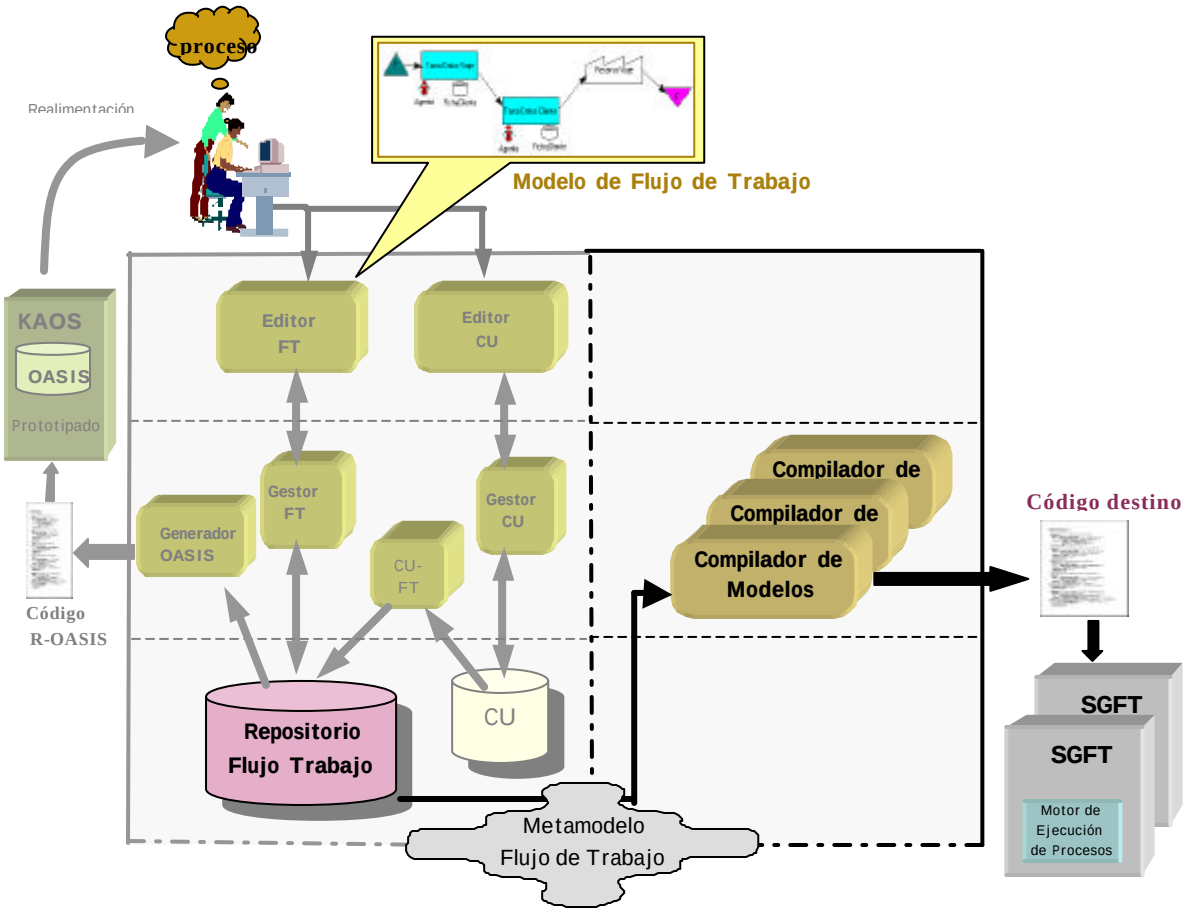


Figura 7-1. Arquitectura del entorno de soporte al desarrollo de flujos de trabajo (*Ejecución*)

7.2 Ejecución en el Sistema OPERA

El sistema OPERA[45] [OPE, Hag99] generaliza conceptos que provienen de los SGFT y de los Entornos de Ingeniería del Software centrados en el Proceso (PSSE). Es un núcleo de soporte al proceso, que interpreta descripciones de procesos especificados en un lenguaje de modelado concreto, denominado OCR (*Opera Canonical Representation*). Estas especificaciones de proceso son cargadas en el núcleo de OPERA antes de su ejecución.

La Figura 7-2 muestra la arquitectura genérica de OPERA. La pieza central es el servidor o núcleo de soporte al proceso que almacena toda la información de una instancia proceso particular. Aunque en la figura sólo se muestra un servidor, pueden existir varios con la construcción de un clúster de servidores. El uso de múltiples servidores puede mejorar el rendimiento puesto que la carga de trabajo se puede distribuir entre ellos.

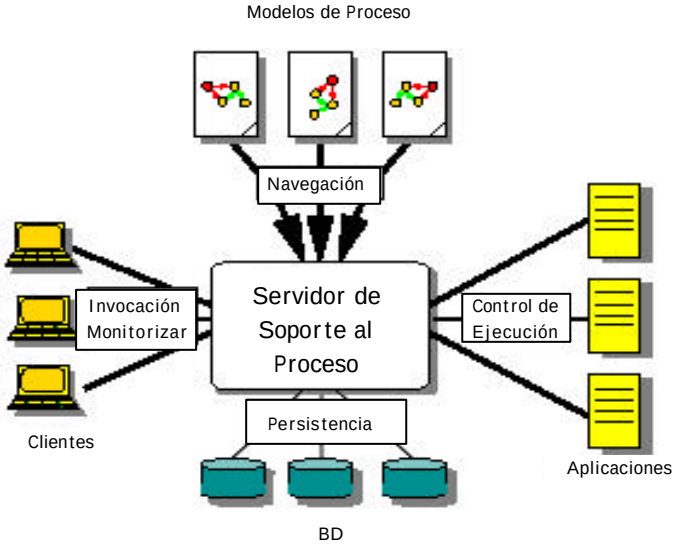


Figura 7-2. Arquitectura del sistema OPERA (fuente [Hag99])

De acuerdo con los principios de la computación basada en procesos, el servidor sólo es responsable de coordinar el control y el flujo de datos entre aplicaciones externas, pero no de realizar parte de la computación en sí misma. Ejecutar un proceso significa usar la descripción del proceso para determinar el orden de invocación de aplicaciones, ejecutarlas de acuerdo a ese orden, monitorizar su progreso y transferir datos entre ellas.

El servidor proporciona interfaces para la invocación de aplicaciones externas y para la comunicación con sus clientes. Un cliente puede ser un usuario o un programa que accede al

[45] ***O**pen **P**rocess **E**ngine for **R**eliable **A**ctivities*

servidor a través de una interfaz especial de API. El interfaz del cliente permite almacenar nuevos modelos de procesos en el servidor para crear instancias que se pueden monitorizar, controlando cómo progresa su ejecución. Por ejemplo, es posible parar la ejecución de un proceso activo o abortarla por completo.

Además, el servidor usa bases de datos como almacenamiento persistente para diversos propósitos. Entre ellos, la función más importante es almacenar los estados de los procesos para poder garantizar que la información de los mismos perdura a posibles caídas del servidor. El almacenamiento persistente también es usado para guardar modelos de proceso que pueden entonces ser instanciados cuando sea necesario.

7.2.1 Modelo Básico de Procesos

En OPERA, los modelos de procesos están especificados en OCR, como ya se ha comentado anteriormente. Para definir el modelo básico de procesos que soporta OPERA se sigue un enfoque orientado a objetos, de forma que los componentes del procesos son clases y el uso de la herencia permite tener mecanismos de extensión en dicho modelo. Las entidades principales de un proceso son: tareas (tasks), programas (programs) y recursos (resources). A continuación se enuncian las características más importantes de cada una de ellas:

- **Tareas.** Entidad que puede ser ejecutada. Engloba los conceptos de proceso, bloque y actividad. El proceso es la tarea de mayor granularidad y representa a aquellos que no forman parte de ningún otro proceso (*top level process)*. Las actividades son los pasos básicos de ejecución y los bloques se utilizan para dar estructura al proceso, permitiéndose la introducción de bucles. Existe definida una jerarquía de generalización (Figura 7-3), según la cual, las clases tarea, tarea compleja y tarea componente son clases abstractas, mientras que la actividad, proceso, subproceso y bloque son clases concretas.

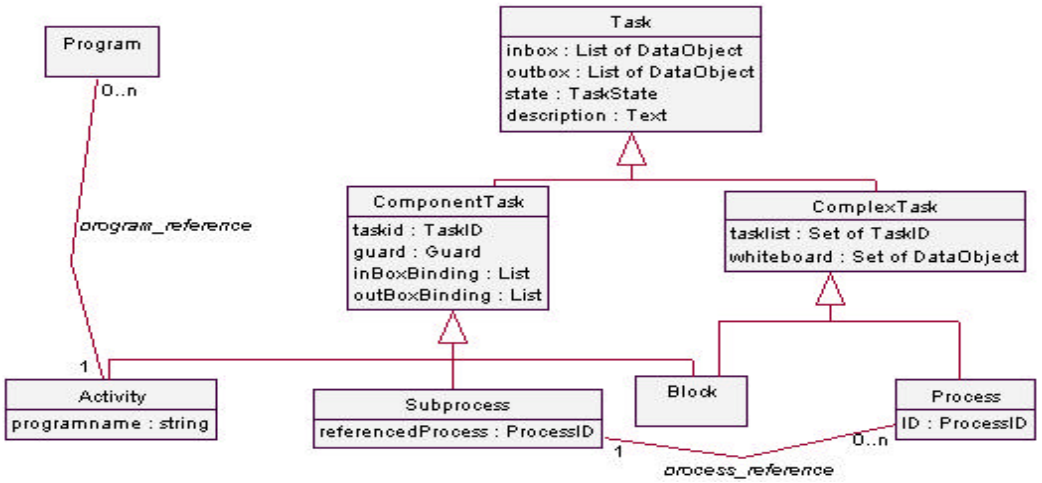


Figura 7-3. Jerarquía de generalización para las tareas (fuente [Hag99])

- **Programas.** Representan el enlace de las actividades que forman el proceso con el mundo real. Las acciones asociadas a estos programas pueden ser tanto el inicio de una actividad humana como el envio de una sentencia SQL a una base de datos o la ejecución de una aplicación. El objeto que representa al programa contiene toda la información necesaria para que éste se inicie, incluyendo los datos que necesita como entrada y los datos que genera como salida. Al igual que con las tareas, se define una jerarquía de generalización para los distintos tipos de programas.

- **Flujo de control.** Se especifica mediante un mecanismo similar a reglas, basado en el concepto de reglas ECA (Evento-Condición-Acción). Las tareas de un proceso tienen una guarda que indica cuándo dicha tarea se puede ejecutar. Esta guarda es de la forma (A, C), donde A es el activador y C es la condición. El actividador es un predicado sobre el estado del proceso, sus componentes o el mundo exterior. La condición es un predicado sobre los datos visibles a la tareas (esto permite modelar flujos condicionales).

- **Datos.** Los tipos de datos que soporta son escalares, existiendo dos mecanismos de extensión. Mediante el operador de especialización, definir nuevos tipos, o bien, mediante la clase denominada ReferenceObject que almacena una referencia a un objeto CORBA, pero que puede generalizar a cualquier otro tipo de información.

- ***Whiteboard.*** Es el equivalente a las variables globales en los lenguajes de programación. Contiene datos que pueden ser leídos/escritos por todas las tareas, tal como información común acerca del estado, implementación de contadores, etc. Además, se permiten realizar ciertas operaciones sobre los datos, tal como incrementar/decrementar enteros, concatenar cadenas, etc.

- **Flujo de Datos.** Los datos de un proceso forman parte de la lista de parámetros de las tareas, o bien, del denominado *whiteboard*. Hay dos formas básicas de transferencia de datos: como datos de entrada o como datos de salida en las tareas. En la sintaxis de OCR, si son datos de entrada se especifican como parámetros asociados a las tareas, y si son datos de salida como parte de la sección *store*. En este último caso, quedan almacenados en el *whiteboard*. En la Figura 7-4 que muestra la plantilla de especificación OCR se identifica este flujo de datos.

De acuerdo con el modelo de procesos presentado, se construye la especificación textual en OCR. Esta especificación es la entrada al sistema OPERA y tiene dos partes diferenciadas: el registro de los programas externos y la especificación de la estructura del proceso, siguiendo la plantilla que se muestra esquemáticamente en la Figura 7-4.

```
PROGRAM Nombre (Parámetros_entrada)
  RETURNS (Parámetros_salida) (
  SYS Tipo_de_programa,
  Atributo_tipo_programa   Valor_del atributo
);

...  // Resto de programas que se registren

PROCESS Nombre (Parámetros_entrada)
  RETURNS (Parámetros_salida)
  WHITEBOARD (Parámetros),
TASKS
  TASK NombreTarea : Programa_invocado(Parámetro_entrada)
      STORE (Parámetros_salida),
      ACT   fórmula
      COND  formula

...   // Resto de tareas del proceso

END TASKS
END PROCESS
```

Figura 7-4. Plantilla de una especificación OCR

7.2.2 Correspondencia entre Metamodelos

La Tabla 7-1 muestra esquemáticamente las equivalencias entre el metamodelo de referencia y el metamodelo soportado por OPERA, denotados por M_r y M_o, respectivamente. A continuación se describen dichas equivalencias, con mayor nivel de detalle:

E.1. El concepto de proceso definido en M_r coincide con el definido en M_o. En ambos casos, se incluye la especificación de las tareas que lo componen, el flujo de datos, el flujo de control y los recursos asociados. Lo que cambia de M_r a M_o es la forma concreta de representar estos componentes, tal como se muestra en el resto de equivalencias establecidas (E.2 a E.7) .

E.2. En M_r se distingue entre subproceso y actividad, además éstas últimas pueden ser manuales o automáticas. Sin embargo, aunque en M_o existe una jerarquía de generalización de tareas en procesos, subprocesos, actividades y bloques, en realidad, en el lenguaje OCR todas ellas quedan englobadas bajo la definición de tarea (task). Esto implica que tanto un subproceso como una actividad en M_r se representan como una tarea en M_o. La distinción entre actividades automáticas y manuales desaparece, puesto que en OCR toda tarea siempre tiene asociado un programa. Por lo que tenemos dos equivalencias E.2.1 y E.2.2.

	Metamodelo de Referencia (***M***)	Metamodelo de OPERA (***M_o***)
E1	Proceso	Proceso (Process)
E2	Subproceso	Tarea (Task)
	Actividad Manual Automática	Tarea (Task)
E3	Condiciones de Transición	Guarda de una Tarea (ACT, COND)
E4	Flujo de Control	Guarda de una Tarea (ACT, --)
E5	Aplicaciones	Programas (Program)
E6	Datos	Datos Parámetros de las tareas *Whiteboard*
E7	Actores	-- --

Tabla 7-1. Equivalencias entre el metamodelo de referencia y el de OPERA

E.2.01 En el caso de una actividad automática en M_r, puesto que siempre tiene asociada una aplicación, la equivalencia a M_o es directa. La actividad automática equivale a una tarea y la aplicación asociada es el programa que invoca dicha tarea.

E.2.02 Para las actividades manuales de M_r, la equivalencia no es directa, puesto que no existe una aplicación asociada. Es necesaria la interacción con el usuario para determinar qué programa se va a asociar a dicha tarea en M_o. Esto implica básicamente dos cosas: tener que desarrollar una aplicación que realice el conjunto de acciones especificadas para la actividad manual, si ésta se quiere incluir como una parte totalmente automatizada dentro del proceso; o bien, que simplemente se le asocie un programa que indique al actor correspondiente las acciones a realizar y que espere la notificación de finalización por parte de dicho actor.

E.3. Las condiciones de transición del M_r quedan incluidas en M_o como parte de las guardas asociadas a las tareas de salida.

E.3.01 En el caso de una condición de bifurcación en M_r, todas las correspondientes tareas de salida en M_o tienen como activador (ACT) una condición que se evalúa sobre el estado de la tarea de entrada (por ejemplo, que dicha tarea haya finalizado) y como condición (COND), las condiciones asociadas a los flujos de salida. Dependiendo del tipo de bifurcación, existen distintas posibilidades. Éstas se detallan en las plantillas de generación que se enumeran en la siguiente sección.

E.3.02 Para las condiciones de unión en M_r, la tarea de salida en M_o tiene como activador (ACT) una condición que se evalúa sobre el estado de las tareas de entrada (puede ser una conjunción o una disyunción según el tipo de condición de unión) y la condicición (COND) se establece a partir de las condiciones asociadas a los flujos de salida, si existen[46]. En la siguiente sección se detallan las plantillas de generación definidas para cada tipo de condición de unión.

E.4. El flujo de control del M_r queda incluido como parte del activador (ACT) de la guarda asociada a las tareas en M_o. En concreto, el ACT de una tarea es una condición sobre el estado de otra tarea. Cuando esta condición es cierta, la tarea inicia su ejecución.

E.5. Todas las aplicaciones definidas en M_r se corresponden con programas en M_o. Sin embargo, en M_o pueden existir programas que no estén definidos en M_r. En el caso de las actividades manuales de M_r, si éstas se desean incluir como parte automatizada del proceso, se deben definir y registrar los programas asociados a las mismas en M_o (comentado en E.2.2).

E.6. Todos los datos de un flujo de trabajo en M_r se corresponden con datos en M_o, que aparecen como parámetros de las tareas o del proceso y además forman parte del *whiteboard*. En concreto tenemos:

E.6.01 Los datos asociados a la actividad inicial del proceso en M_r se corresponden en M_o con los parámetros de entrada del proceso.

E.6.02 Los datos asociados a la actividad final del proceso en M_r se corresponden en M_o con los parámetros de salida del proceso.

E.6.03 Los datos de entrada / salida de cada actividad en M_r se corresponden en M_o con los parámetros de entrada / salida de cada tarea (Flujo de datos).

E.6.04 Todos los datos que forman parte de un proceso en M_r, en M_o forman parte del *whiteboard* asociado a dicho proceso.

[46] Salvo para el caso de la bifurcación parcial N, no existen condiciones asociadas a los flujos de entrada.

E.7. El concepto de actor de M_r no tiene un equivalente en M_o, puesto que no se modela como parte del proceso la participación humana en el mismo.

En base a las equivalencias establecidas entre ambos metamodelos, se definen un conjunto de plantillas para la generación de código OCR. En la siguiente sección se relacionan todas ellas.

7.2.3 Plantillas para la Generación de Código OCR

El compilador de modelos a OPERA [Sal02] aplica un conjunto de plantillas para la obtención de la especificación de un flujo de trabajo en el lenguaje específico, en este caso OCR. La notación empleada en la descripción de las plantillas sigue las siguientes convenciones: las palabras en negrita se corresponden con palabras reservadas del lenguaje OCR, las que aparecen en cursiva se reeemplazarán en el proceso de generación por el valor correspondiente, y finalmente, el símbolo // denota la inclusión de otras plantillas asociadas. Se han agrupado en tres bloques, dependiendo de si están relacionadas con:

1. la definición del proceso global que representa al flujo de trabajo,
2. la definición de los componentes del proceso, o
3. la definición del flujo de control.

7.2.3.1 Definición del Flujo de Trabajo

Puesto que un flujo de trabajo está representado por un proceso, la plantilla proceso establece el código OCR que se debe generar para representarlo. También se ha definido la plantilla whiteboard para especificar globalmente todos los datos que utilizan dicho proceso[47].

- ***Plantilla Proceso***. Dado un proceso definido según el metamodelo de referencia, la especificación OCR correspondiente es la siguiente:

```
//Plantilla Programa
PROCESS NombreProceso (Datos_entrada)
          RETURNS (Datos_salida)
          WHITEBOARD ( //Plantilla Whiteboard )
TASKS
//Plantilla Tarea
END TASKS
END PROCESS
```

[47] Utilizamos el término proceso para denotar el proceso de más alto nivel que representa al flujo de trabajo.

El literal *NombreProceso* se sustitye por el nombre del proceso (equivalencia E1) y los literales *Datos_entrada* y *Datos_salida* por los datos de entrada y salida del proceso, según las equivalencias E6.01 y E6.02 establecidas previamente. La aplicación de las plantillas *Whiteboard*, *Tarea* y *Programa* completan la definición del proceso.

- ***Plantilla Whiteboard***. La especificación OCR correspondiente al whiteboard que se genera para todo proceso sigue la siguiente plantilla:

```
WHITEBOARD ( Datos_proceso )
```

El literal *Datos_proceso* se reemplaza por la declaración de todos los datos asociados al proceso (equivalencia E.6.04). Se declara el nombre del dato y su tipo, siguiendo el formato *NombreDato: TipoDato*.

7.2.3.2 Definición de Componentes del Proceso

Los componentes básicos de un proceso en OPERA son dos. Por una parte las tareas y por otra, los programas asociados a las mismas. A continuación se muestran las dos plantillas para la generación de código OCR correspondiente a estos elementos.

- ***Plantilla Programa***. Para cada aplicación asociada a una actividad automática, según el metamodelo de referencia, se genera una especificación OCR equivalente. En el código generado hay una parte común y otra que es dependiente del tipo de programa que se invoque. A continuación se muestra la parte común:

```
PROGRAM NombreAplicacion (AplicacionDatos_entrada)
     RETURNS (AplicacionDatos_salida) (
     SYS  TipoAplicacion,
     HOST    PathAplicacion,
     COMMAND  CallAplicacion
  );
```

Los literales *NombreAplicacion*, *AplicacionDatos_entrada* y *AplicacionDatos_salida* se sustituye por el nombre de la aplicación a registrar y sus correspondientes datos de entrada y salida. Si se trata una aplicación *Unix*, el literal *TipoAplicacion* toma el valor Unix y *PathAplicacion* y *CallAplicacion* toman como valores la ubicación física y la invocación de la aplicación, respectivamente. Esta información fue almacenada como parte de sus propiedades al definirla según el metamodelo de referencia.

Si la aplicación que se invoca es de otro tipo, como por ejemplo un proceso que se ejecuta en SAP R/3 [SAP] o en FlowMark[48], al código anterior se añade más información, que es dependiente de cada caso. Por ejemplo, para SAP tenemos el siguiente código:

```
PROGRAM NombreAplicacion (AplicacionDatos_entrada)
   RETURNS (AplicacionDatos_salida)  (
   SYS  SAP,
   HOST      PathAplicacion,
   CLIENT    ClientAplicacion,
   USER      UserAplicacion,
   PASSWORD  PasswordAplicacion,
   WORKFLOW  CallAplicacion
);
```

y al igual que antes, la información correspondiente está almacenada como parte de sus propiedades cuando se definió el recurso aplicación.

- ***Plantilla Tarea***. Para cada cada uno de los pasos del proceso (actividades y subprocesos) definido según el metamodelo de referencia, la especificación OCR que se genera es:

```
TASK NombreActividad : NombreAplicacion (AplicacionDatos_entrada)
  STORE (AplicacionDatos_salida  ->  WB.Datos_Proceso)
  ACT   //Plantillas Flujo de Control
  COND  //y Condiciones de Transición
```

El literal *NombreActividad* se sustituye por el nombre de la actividad o subproceso en cuestión, y *NombreAplicacion* por el nombre de la aplicación a invocar (equivalencia E.2). En caso de estar definida como una actividad manual en el metamodelo de referencia, la aplicación a invocar debe ser definida en este punto, según se indica en la equivalencia E.2.2.

El literal *AplicaciónDatos_entrada* se reemplaza por los datos de entrada a la actividad, que son a su vez de entrada a la aplicación asociada; de forma similar, el literal *AplicacionDatos_salida* se reemplaza por los datos de salida de la aplicación, que lo son también de la tarea (equivalencia E.6.03). Además, su valor queda almacenado en alguno de los datos del proceso que contiene el *whiteboard* (el literal *WB.Dato_Proceso*

[48] Estos dos sistemas son los actualmente soportados por OPERA.

representa dicho depósito). La definición de la tarea se completa con las plantillas de definición del flujo de control.

7.2.3.3 Definición de Flujo de Control

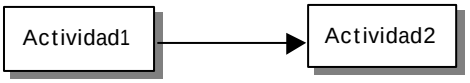
Para la generación del flujo de control asociado al proceso se han definido un conjunto de plantillas dependiendo de si el flujo es secuencial o existen condiciones de unión o de bifurcación entre las actividades. A continuación se detallan todas ellas.

- ***Plantilla Flujo de Control Secuencial***. En OCR, el flujo de control se especifica en el activador de las tareas (equivalencia E.4). El esquema de generación es el que se muestra a continuación, siempre y cuando dicho flujo entre activadades sea secuencial.

```
TASK NombreActividad : ...[49]
     STORE ...
     ACT   EstadoTarea(TareaPred)
     COND  true
```

En la sección ACT, el literal *TareaPred* se sustituye por el nombre de la actividad predecesora y *EstadoTarea* por el estado que debe alcanzar dicha actividad para que se inicie la que estamos especificando. En el caso de la actividad inicial del proceso, puesto que no tiene predecesora, se indica la palabra reservada STARTUP. El siguiente ejemplo muestra la generación obtenida tras la aplicación de la plantilla.

Ejemplo: Dadas dos actividades que forman parte de un proceso más complejo, denominadas Actividad1 y Actividad2, tales que Actividad1 es la actividad inicial del proceso y Actividad2 es la actividad sucesora, tras aplicar la plantilla, el código OCR generado es el siguiente (sólo se muestran las guardas de las tareas).




```
TASK Actividad1 : ...
    STORE ...
    ACT   initial (STARTUP),
    COND  true

TASK Actividad2 : ...
    STORE ...
    ACT   finished(Actividad1),
    COND  true
```

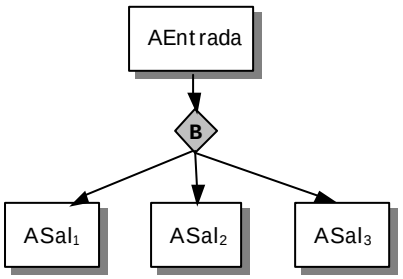
[49] Esta plantilla de generación se centra en la guarda de la tarea. Se asume lo visto en la plantilla tarea, quedando aquí representado por puntos suspensivos.

- ***Plantilla Condición de Bifurcación Total***. Para cada actividad de salida $ASal_i$ ($2 \leq i$), el esquema OCR que se genera es el siguiente:

```
TASK ASal_i : ...
     STORE ...
     ACT   EstadoTarea(TareaEnt)[50]
     COND  true
```

En la sección ACT, el literal *TareaEnt* se sustituye por el nombre de la actividad de entrada en la bifucación y *EstadoTarea* por el estado que debe alcanzar dicha actividad para que se inicien todas las de salida (Las $ASal_i$ que estamos especificando). La sección COND queda true, ya que en la bifurcación total no hay condición asociada a los flujos de salida. El siguiente ejemplo muestra la generación obtenida tras la aplicación de la plantilla.

Ejemplo: Como parte de la definición de un proceso, tras la finalización de la actividad AEntrada, hay una bifurcación total. Se muestra el código OCR generado para las actividades de salida (sólo las guardas asociadas).




```
TASK ASal1 : ...
  STORE ...
  ACT   finished (AEntrada),
  COND  true

TASK ASal2 : ...
  STORE ...
  ACT  finished(AEntrada),
  COND true

TASK ASal3 : ...
  STORE ...
  ACT  finished(AEntrada),
  COND true
```

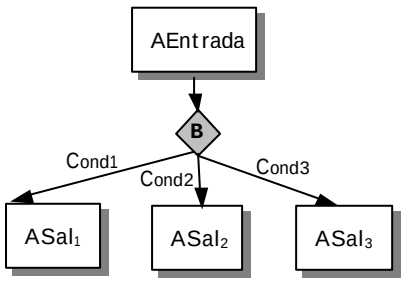
- ***Plantilla Condición de Bifurcación Parcial***. Para cada actividad de salida $ASal_i$ ($2 \leq i$), el esquema OCR que se genera es el siguiente:

```
TASK ASal_i : ...
     STORE ...
     ACT   EstadoTarea(TareaEnt)
     COND  CondFlujoSal
```

[50] Esta condición es la misma para todas las tareas de salida.

Al igual que en la bifurcación total, el literal *TareaEnt* se sustituye por el nombre de la actividad de entrada y *EstadoTarea* por el estado que debe alcanzar dicha actividad para que se inicien todas las de salida. Sin embargo, en este caso la sección COND no es *true*, sino que el literal *CondFlujoSal* se sustituye por la condición asociada al flujo de salida de cada $ASal_i$. El siguiente ejemplo muestra la generación obtenida tras aplicar de la plantilla.

Ejemplo: Como parte de la definición de un proceso, tras la finalización de la actividad AEntrada, hay una bifurcación parcial. Se muestra el código OCR generado para las actividades de salida (sólo las guardas asociadas).



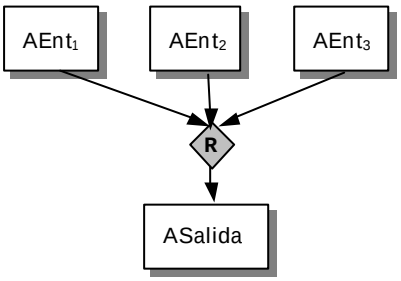

```
TASK ASal1 : ...
  STORE ...
  ACT   finished (AEntrada),
  COND  Cond1

TASK ASal2 : ...
  STORE ...
  ACT  finished(AEntrada),
  COND Cond2

TASK ASal3 : ...
  STORE ...
  ACT  finished(AEntrada),
  COND Cond3
```

- ***Plantilla Condición de Unión Total***. Para la actividad de salida ASalida, el esquema OCR que se genera es el siguiente:

```
TASK ASalida : ...
     STORE ...
     ACT  EstadoTarea(TareaEnt1) and ... and  EstadoTarea(TareaEnti)
     COND true
```

En la sección ACT, la condición asociada es una conjunción de los estados que deben alcanzar las actividades de entrada para que se inicie la de salida. En concreto, el literal $TareaEnt_i$ ($2 \leq i$) se sustituye por el nombre de la actividad de entrada correspondiente y *EstadoTarea* por el estado que debe alcanzar dicha actividad. La sección COND es *true*, puesto que en la unión total no hay condición asociada a los flujos de entrada. El siguiente ejemplo muestra la generación obtenida tras la aplicación de la plantilla.

Ejemplo: En la definición de un proceso, aparece la siguiente unión total. Se muestra el código OCR generado para la actividad de salida (sólo la guarda asociada).



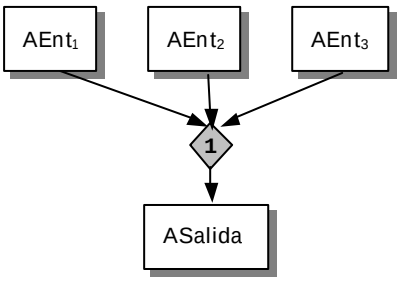

```
TASK ASalida : ...
  STORE ...
  ACT   finished (AEnt1 and AEnt2 and AEnt3),
  COND  true
```

- ***Plantilla Condición de Unión Parcial 1.*** Para la actividad de salida ASalida, el esquema OCR que se genera es el siguiente:

```
TASK ASalida : ...
     STORE ...
     ACT  EstadoTarea(TareaEnt1) or ... or  EstadoTarea(TareaEnti)
     COND true
```

Al igual que para la unión total, el literal *TareaEnt*$_i$ ($2 \leq i$) se sustituye por el nombre de la actividad de entrada correspondiente y *EstadoTarea* por el estado que debe alcanzar dicha actividad. Pero en este caso, la condición que se genera es una disyunción, puesto que sólo se exige que una de las actividades de entrada alcance un cierto estado para iniciarse la de salida. La sección COND es *true* como en la unión total. El siguiente ejemplo muestra la generación obtenida tras la aplicación de la plantilla.

Ejemplo: En la definición de un proceso, aparece la siguiente unión parcial 1. Se muestra el código OCR generado para la actividad de salida (sólo la guarda asociada).




```
TASK ASalida : ...
  STORE ...
  ACT   finished (AEnt1 or AEnt2 or AEnt3),
  COND  true
```

- ***Plantilla Condición de Unión Parcial N***. No es una plantilla totalmente nueva, sino que combina las dos anteriores (unión total y parcial 1), teniendo en cuenta las equivalencias que se pueden establecer entre ellas. Concretamente, una condición de unión parcial N se puede expresar como varias condiciones de unión total y una condición de unión parcial 1, más un conjunto de condiciones asociadas a ciertos los flujos de entrada.

  Sean AEnt$_i$, $2 \leq i$, el conjunto de actividades de entrada; ASalida la actividad de salida y N el valor de la unión parcial; dicha información se puede expresar en base a uniones totales y parciales como:

  - AEnt$_i$, $2 \leq i$ son las actividades de entrada,
  - se introducen k actividades auxiliares, AAux$_k$, tal que k= C^{i}_{N}
  - se introducen k uniones totales; cada una de ellas tiene como actividad de salida, la AAux$_k$ correspondiente y como actividades de entrada cada posible valor de C^{i}_{N}, junto con las condiciones asociadas al flujo de control, y finalmente
  - se introduce una unión parcial 1, que tiene como salida la ASalida y como entrada todas las AAux$_k$.

La Figura 7-5 muestra la equivalencia para el caso de N=2. Una vez establecida la equivalencia, se aplican las plantillas definidas para la unión total y unión parcial 1. El siguiente ejemplo muestra la generación obtenida para un caso concreto.

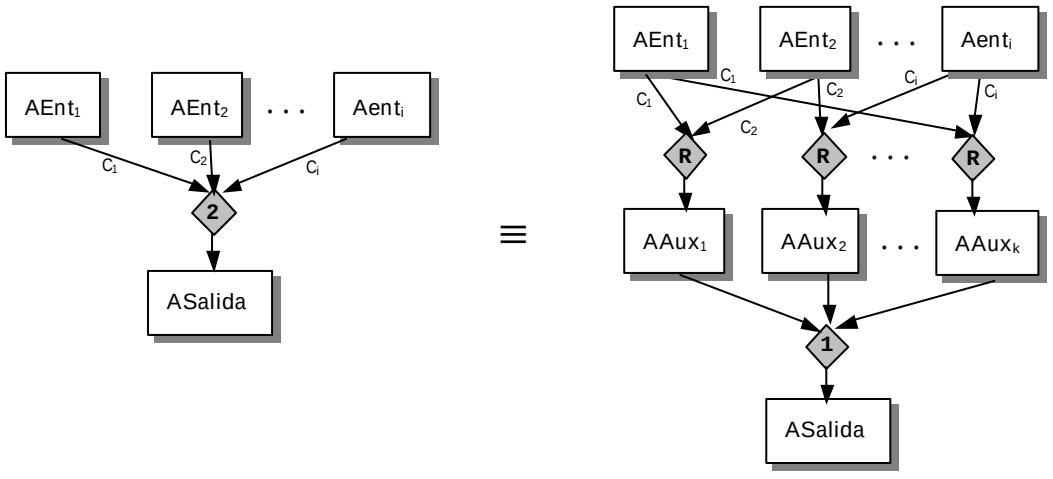
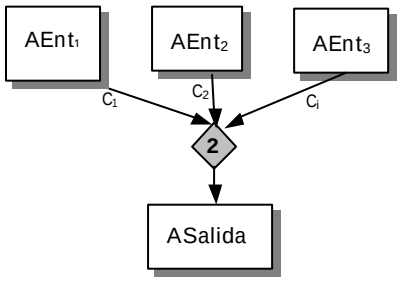


Figura 7-5. Equivalencias entre condiciones de unión

Ejemplo: En la definición de un proceso, aparece la siguiente unión parcial N, con N=2. Se muestra el código OCR generado para la actividad de salida y las actividades auxiliares que se introducen (sólo la guarda asociada).




```
TASK AAux1 : ...
  STORE ...
  ACT   finished(AEnt1 and AEnt2),
  COND  c1 and c2

TASK AAux2 : ...
  STORE ...
  ACT  finished(AEnt1 and Aent3),
  COND c1 and c3

TASK AAux3 : ...
  STORE ...
  ACT  finished(Aent2 and Aent3),
  COND c2 and c3

TASK ASalida : ...
 STORE ...
 ACT  finished(AAux1 or AAux2 or AAux3),
 COND  true
```

En este caso:

N=2, y

$K = C_2^3 = 3$.

7.3 Conclusiones

En este capítulo se ha descrito la inclusión de la funcionalidad de ejecución que proporcionan los SGFT como una parte del entorno de soporte al proceso de desarrollo de flujos de trabajo. Un compilador de modelos establece el enlace entre la especificación construida en base al metamodelo de referencia, y la especificación en el lenguaje del SGFT destino, de acuerdo con su metamodelo. Esto nos permite que la fase de construcción y estabilización pueda ser utilizado por cualquier SGFT. En el resto del capítulo se aborda un caso concreto: la ejecución del modelo construido en el sistema OPERA. Se han establecido las equivalencias y plantillas para la generación de código OCR a partir de la información contenida en el repositorio.

Se está trabajando en la definición de compiladores de modelos a SGFT comerciales, en concreto a MQSeries WF y a SAP R/3. El proceso a seguir para la construcción del compilador de modelos consta de dos pasos, tal como se ha visto para OPERA. En primer lugar estudiar el metamodelo soportado por el SGFT destino, y en segundo lugar, establecer las equivalencias entre metamodelos para poder definir el conjunto de plantillas de generación a aplicar.

Capítulo 8.

Análisis de Ejecuciones de Flujos de Trabajo

En este capítulo se aborda el subproceso de análisis. Su objetivo es poder extraer conocimiento que ayude a optimizar el flujo de trabajo seguido por una organización, a partir de información histórica recogida tras distintas ejecuciones del mismo en un SGFT. Estos datos reales de ejecución, que se encuentran almacenados en algún tipo de repositorio, son el punto de partida para iniciar un proceso de análisis que permita el descubrimiento de conocimiento acerca del flujo de trabajo, tal como la detección de cuellos de botella, la no disponibilidad de recursos que ocasionen bloqueos del proceso o cuestiones de eficiencia. Este tipo de información no se puede obtener en base a simulaciones o animaciones del modelo de flujo de trabajo construido, sino que se precisan datos referentes a ejecuciones reales. Finalmente, en base al resultado de este análisis para la extracción de conocimiento se inicia un proceso de mejora del flujo de trabajo.

El capítulo se estructura como sigue. En la sección 8.1 se introducen los principales conceptos y técnicas relacionados con la extracción de conocimento, desde la construcción de almacenes de datos y su explotación mediante técnicas *OLAP*, hasta la minería de datos; finalmente se presenta *OLAP-Mining* como una propuesta integradora. En la sección 8.2 se aborda el problema a resolver, enumerándose qué items interesa analizar en un flujo de trabajo y planteándose un modelo de almacén de datos. La explotación del mismo se realiza mediante tecnología *OLAP-Mining*. Finalmente, en la sección 8.3 se integra la solución propuesta en el marco de los SGFT, en concreto como parte del entorno de soporte al proceso de desarrollo de flujos de trabajo propuesto en esta tesis.

8.1 Introducción al Análisis de Datos

En los últimos años, la existencia de gran cantidad de datos almacenados, como parte de los sistemas de información de las organizaciones, ha provocado la necesidad de desarrollar técnicas para la búsqueda y extracción de información y conocimiento a partir de los mismos. Esta información puede ser de gran interés para la toma de decisiones estratégicas en la organización, control de la producción, análisis de mercados, exploración en experimentos científicos, etc. Por lo tanto, como resultado natural de la evolución de la tecnología de la información, surge lo que de forma general se denomina *Proceso de Descubrimiento de Conocimiento*[51] en Bases de Datos. La investigación y los resultados, al menos inicialmente, se particularizaban a bases de datos, pero puede verse como un proceso general para cualquier tipo de repositorio o fuente de datos.

En general, un proceso de descubrimiento de conocimiento consiste en una secuencia iterativa pasos (ver Figura 8-1). A continuación se enumeran todos ellos:

1. *Limpieza de datos*. Eliminar datos erróneos, inconsistentes o que provoquen ruido.
2. *Integración de datos*. Combinación de múltiples fuentes de datos.
3. *Selección de datos*. Los datos relevantes para el análisis son recuperados de la base de datos o repositorio.
4. *Transformación de datos*. Los datos son transformados o consolidados de forma adecuada para llevar a cabo el análisis, realizando, por ejemplo, operaciones de agrupación de datos.
5. *Minería de datos*. Proceso en el que se aplican métodos para la extracción de patrones de datos.
6. *Evaluación de patrones*. Se identifican aquellos patrones o reglas que sean interesantes, en base a una serie de medidas predeterminadas.
7. *Presentación del conocimiento o información*. Se presenta el conocimiento al usuario en base a técnicas de visualización y representación de conocimiento.

De todos estos pasos, los cuatro primeros, desde la limpieza a la transformación de datos se realizan, en numerosas ocasiones, como parte de la construcción de un almacén de datos[52] y su explotación mediante técnicas OLAP[53]. Respecto a los tres últimos, muchas veces quedan englobados en un proceso iterativo denominado genéricamente, minería de datos[54]. En el resto de la sección se da una introducción general a estos dos campos,

[51] *Knowledge Discovery in Databases* (KDD), en inglés.
[52] *Data Warehouse*, en inglés.
[53] *On-line Analitical Processing*
[54] *Data Mining*, en inglés.

acabando finalmente con una propuesta de integración de ambos, denominada OLAP-Mining [Han97, Han01].

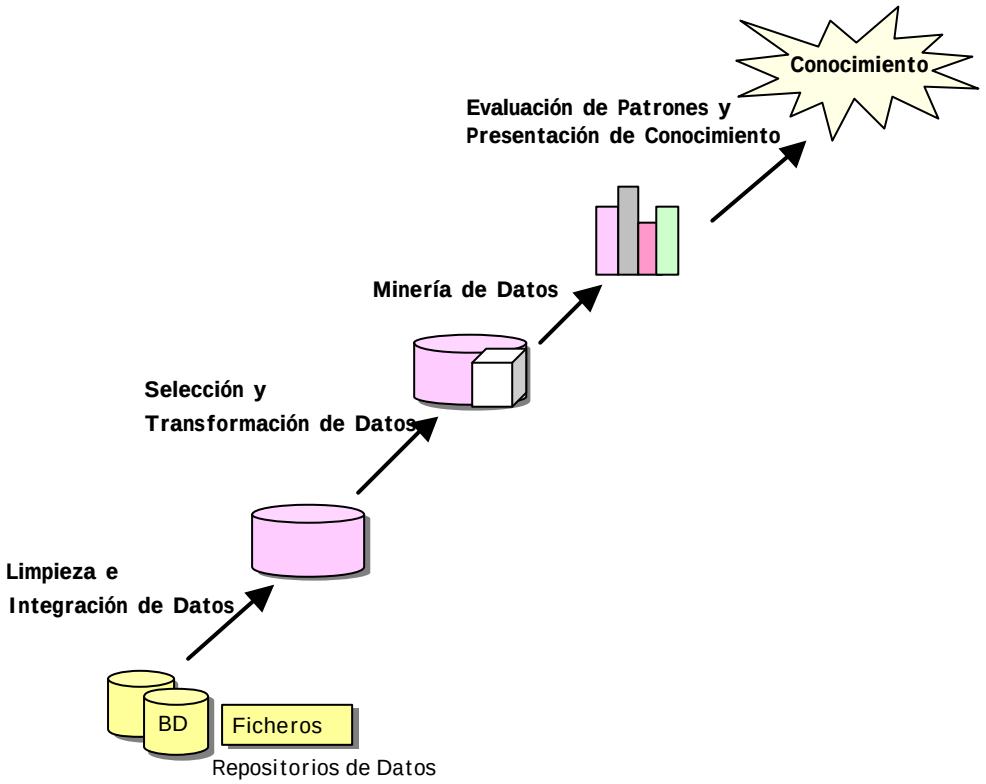


Figura 8-1. Pasos del proceso de descubrimiento de conocimiento

8.1.1 Almacenes de Datos y Tecnología OLAP

Una de las primeras definiciones del término *almacén de datos*, totalmente aceptada hoy en día, la encontramos en [Inm96]. W. H. Inmon define un *almacén de datos* como: "una colección de datos orientados a temas, integrados, historiados y no volátiles que sirven de apoyo al proceso de toma de decisiones". Esta definición presenta las cuatro características básicas que lo distinguen de otros repositorios de datos, como pueden ser los sistemas de bases de datos relacionales, sistemas de ficheros o sistemas de procesamiento de transacciones. Estas características son:

- *Orientado al tema*. Un almacén de datos está organizado alrededor de temas importantes, como pueden ser clientes, ventas y productos, más que orientarse al almacenamiento óptimo de las operaciones que ocurren día a día dentro de una

organización. Proporciona una vista simple y concisa sobre un tema en particular, excluyendo los datos que no son de utilidad para la toma de decisiones.

- *Integrado*. Un almacén de datos se construye integrando datos de múltiples fuentes heterogéneas. En la mayoría de las ocasiones es necesario aplicar técnicas de limpieza e integración de datos, para que éstos esten en un estado consistente.
- *Historiado*. Un almacén de datos contiene datos históricos (representan un valor en un momento concreto), por lo tanto, el tiempo es un elemento que siempre forma parte del almacén, implícita o explícitamente.
- *No volátil*. Un almacén de datos está siempre físicamente separado del repositorio de datos que maneja la organización en el entorno operacional y que sirve de base para construir el almacén de datos. Por ello, sólo se necesita disponer de dos operaciones: carga inicial de los datos y acceso a los mismos. No es necesario que el almacén de datos dé soporte a transacciones, mecanismos de recuperación o control de concurrencia.

En [Han01] encontramos la siguiente definición que resume las principales ideas relacionadas con el término *almacén de datos*: "colección de datos consistente semánticamente que sirve como implementación física de un modelo de datos de apoyo a la toma de decisiones y que almacena la información que la organización necesita para la toma de decisiones estratégicas. Además, en muchas ocasiones, el almacén de datos es visto también como una arquitectura, construida para integrar datos de múltiples fuentes heterogéneas y dar soporte a consultas estructuradas y/o *ad-hoc*, informes analíticos y toma de decisiones". Esta segunda definición, aunque está de acuerdo con la definición de Inmon, incorpora otras características y matices con los que se utiliza el término *almacén de datos* hoy en día.

Al proceso de construcción y uso de un almacén de datos, se le conoce en la literatura por el término *Data Warehousing*, aunque existen autores que denotan *Data Warehousing* sólo al proceso de construcción del almacén y *Warehouse DBMS* a la gestión y utilización del mismo. La construcción del almacén de datos requiere tres tareas básicas, que son integración, limpieza y consolidación de datos. Por su parte, la utilización o explotación del almacén requiere el uso de tecnología de apoyo a la toma de decisiones, conocida como tecnología OLAP, frente a la tecnología OLTP (*On-line Transaction Processing*).

El término OLAP fue acuñado por Codd en [Cod93] para caracterizar los requisitos de resumir, consolidar, proyectar, aplicar fórmulas y sintetizar datos en múltiples dimensiones. Los sistemas OLAP proporcionan una presentación multidimensional de los datos contenidos en un almacén de datos, permitiendo a sus usuarios realizar un análisis de los mismos y tomar decisiones al respecto. Además, en estos sistemas se crean estructuras multidimensionales que organizan y resumen la información contenida en el almacén, con el fin de mejorar la eficiencia de las consultas analíticas. El diseño de la estructura del almacén afecta directamente a la facilidad con que se puedan definir y construir dichas estructuras.

La utilidad de los almacenes de datos puede ser vista desde dos perspectivas diferentes:

- Los almacenes de datos permiten que las organizaciones obtengan información de diversa índole, a partir de los datos almacenados en las transacciones que se realizan día a día. Por ejemplo, se pueden analizar las preferencias de los clientes, las estrategias de producción, las operaciones que se realizan o la gestión de la relación con los clientes.
- Los almacenes de datos permiten integrar bases de datos heterogéneas frente a la aproximación tradicional de construir *wrappers* e integradores.

Esta tesis se centra en la primera utilidad que se cita, es decir, en que el almacén de datos permite obtener información acerca de los ítems o temas que interesen a la organización, y en base a ellos se construye y utiliza dicho almacén.

Finalmente, comentar que los almacenes de datos aportan varias ventajas a las organizaciones. Éstas se pueden resumir en dos: el aumento de la competitividad en el mercado, como resultado de mejorar el proceso que sigue la organización, y el aumento de la productividad de los directivos en la toma de decisiones estratégicas. Por otro lado, también se pueden citar algunos problemas relacionados con la construcción y uso de los almacenes de datos. Los recursos y el esfuerzo necesario para la integración, limpieza y consolidación de datos, así como su diseño y creación pueden llegar a ser demasiado elevados, especialmente si se permite que el almacén pueda responder a cualquier requisito del usuario, puesto que esto haría aumentar su tamaño y complejidad.

8.1.1.1 Un Modelo de Datos Multidimensional: Cubo de Datos

Mientras que el modelo entidad-relación es adecuado para sistemas OLTP, los sistemas OLAP utilizan modelos multidimensionales, puesto que requieren un esquema conciso, orientado al tema y que facilite el análisis de los datos.

Un modelo multidimensional muy popular y utilizado en las aplicaciones de análisis de datos (sistemas OLAP) es el ***cubo de datos*** [Gra95, Gra97]. Éste permite al analista agrupar y visualizar los datos que forman la información histórica, en diferentes niveles de abstracción y desplazarse entre dichos niveles en busca de patrones inusuales, tendencias y anomalías.

Un cubo de datos se define a partir de un conjunto de atributos. Estos atributos pueden pertenecer a una de las siguientes categorias: ***dimensiones*** y ***medidas***. Cada dimensión del cubo representa una perspectiva o entidad, con respecto a la cual se desea almacenar información. Esa información que se almacena, dependiendo del valor de la dimensión, es lo que constituye la medida o atributo medible.

Por ejemplo, supongamos que una empresa desea analizar sus ventas en base a la información histórica de los últimos 5 años y en dicho análisis desea tener en cuenta el tipo de producto vendido, la ciudad y el cuatrimestre del año en que se vendió. En este caso, el

cubo de datos a construir tiene 3 dimensiones, producto, ciudad y cuatrimestre, y una medida, total de ventas. Gráficamente quedaría tal como muestra la Figura 8-2 (los datos son ficticios).

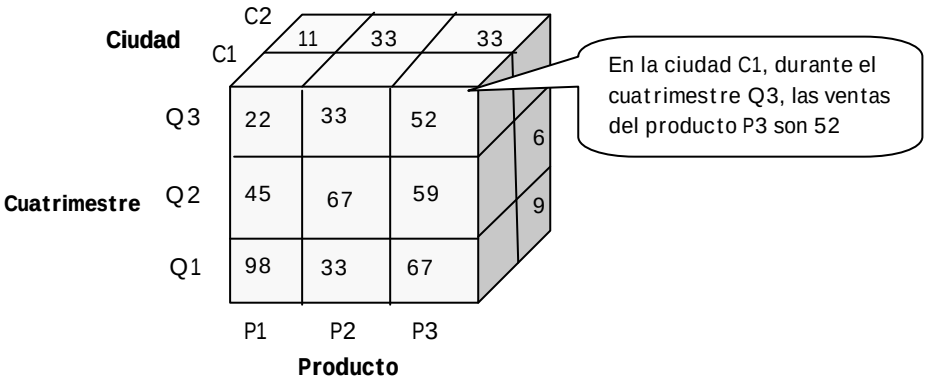


Figura 8-2. Cubo de datos de 3-Dimensiones: producto, ciudad, cuatrimestre

Las características más importantes de los dos tipos de atributos que posee un cubo de datos se resumen a continuación:

A) <u>Medidas</u>: Puesto que la idea es poder agregar[55] datos en distintas dimensiones, para cada medida del cubo se debe definir cuál es la función de agregación a utilizar. Según [Gra97], esta función de agregación puede ser de 3 tipos: distributiva, algebraica y holistica.

- *Distributiva.* Estas funciones se pueden calcular particionando las entradas en conjuntos disjuntos, agrupando cada uno de ellos individualmente y después volviendo a agrupar el resultado (parcial) de cada conjunto para obtener el resultado final. Entre las funciones de agregación distributiva se encuentran en SQL estándar, la función COUNT, SUM, MIN, y MAX.
- *Algebraica*. Estas funciones pueden ser expresadas como una función escalar entre funciones de agregación distributivas. Por ejemplo, la función de agregación que calcule la media es algebraica, ya que se define como SUM/COUNT.
- *Holistica*. Estas funciones no pueden calcularse a partir de resultados parciales obtenidos al agrupar los datos en grupos disjuntos. Como ejemplos de este tipo de funciones están la mediana, la moda y el rango.

En el ejemplo anterior, la función de agregación definida para la medida total de ventas es la suma (SUM). Con lo cual, cuando se agreguen datos, el nuevo valor de total de ventas se calculará como la suma de los datos total de ventas agregados.

[55] Se utiliza el término agregar con el significado de agrupar.

B) <u>Dimensiones</u>: Aunque al hablar de cubo de datos podamos pensar en 3 dimensiones, en realidad es un modelo *n*-dimensional. De forma que:

- Si un cubo tiene definidas ***n*** dimensiones y ***v*** , v_2, ... v_n son, respectivamente, la cardinalidad del dominio de cada una de las dimensiones, entonces tenemos que el número total de celdas del cubo es Õ **(** v_i **+ 1)**. El valor que se suma a cada dominio representa el valor **ALL**. Este valor se añade para considerar el máximo nivel de agregación en un dimensión. Por ejemplo, en el cubo de ventas, si la dimensión producto toma el valor ALL, lo que se mostraría serían las ventas por ciudad y trimestre, independientemente del producto vendido (se incluyen todos ellos).

- Si un cubo tiene definidas ***n*** dimensiones, existen 2^n formas distintas de agrupar dichas dimensiones, es decir, 2^n vistas posibles del cubo. Estas posibles vistas en función de las dimensiones se pueden representar gráficamente como un retículo[56]. La construcción del retículo a partir de las dimensiones del cubo de datos fue introducido en [Har96]. En este retículo hay dos puntos (vistas) importantes: el que considera todas las dimensiones, que se corresponde con el menor agrupamiento de datos, y el que no aparece ninguna dimensión, que se corresponde el mayor agrupamiento de datos. Este último punto sólo contiene los valores totales agregados para las medidas consideradas.

  Siguiendo con el ejemplo de las ventas de una empresa, la Figura 8-3 muestra la estructura reticular correspondiente a las posibles vistas del cubo. Así, mientras la vista (producto, ciudad, trimestre), denotada por (P,C,T), proporciona el total de ventas desglosado en base a estas tres dimensiones; la vista ( ) proporciona un único valor, el total de ventas de la empresa, considerando todos los productos, todas las ciudades y todos los trimestres.

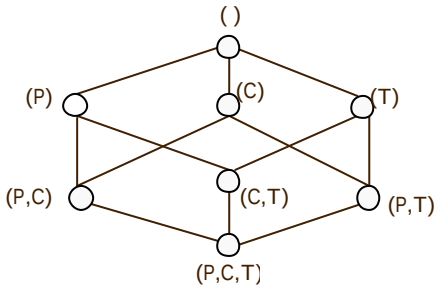


Figura 8-3. Vistas de un cubo de 3-dimensiones

[56] En la literatura, a cada posible vista, en función de las dimensiones consideradas se le denomina .

- Otra cuestión importante respecto a las dimensiones del cubo es que frecuentemente éstas tienen definidas jerarquías que especifican distintos niveles de agregación dentro de la misma dimensión (se especifican conceptos a distintos niveles de abstracción). Puede ocurrir que en esa jerarquía exista una relación de orden total entre los distintos niveles, o bien, una relación de orden parcial. En cualquier caso se pueden representar como un retículo.

  En el ejemplo de las ventas de una empresa, la dimensión ciudad podemos denominarla lugar de la venta y establecer la siguiente jerarquía: calle < ciudad < provincia < pais. En este caso se trata de una relación de orden total.

Estructura reticular

El marco propuesto en [Har96] con el uso de la estructura reticular (*lattice framework*) para representar tanto las distintas vistas del cubo, como las distintas jerarquías dentro de una dimensión, tiene como objetivo establecer relaciones de dependencia entre consultas, esto es, que ciertas consultas al cubo pueden ser resueltas utilizando resultados de otras consultas.

- Define la *relación de dependencia entre consultas* como sigue: Sean *Q1* y *Q2* dos consultas, se dice que *Q1* depende de *Q2* (*Q1* £ *Q2*) si el resultado de *Q1* se puede obtener a partir del resultado de *Q2*.
- De acuerdo con la definición anterior, si *a* y *b* son dos elementos de un retículo, *b* es *antecesor* de *a*, si y sólo si, *a* £ *b*.

Este marco tiene varias ventajas que se resumen a continuación:

- Se puede razonar o realizar consultas que implique más de una dimensión, y que a su vez, dichas dimensiones tengan definidas una jerarquía. Teniendo así, un retículo para las dimensiones, y a su vez, otro retículo para cada dimensión jerárquica. A la combinación de dimensiones jerárquicas se le llama *direct product lattice*. Esto permite:

  a) Encontrar dependencias por interacción de distintas dimensiones.

  b) Encontrar dependencias dentro de una dimensión, debido a la jerarquía existente.

- Podemos modelar las consultas realizadas por el usuario. Estas se mueven entre los vértices del retículo.
- Nos permite decidir en qué orden materializar las vistas (Algoritmo *Greedy*).

Implementación del Cubo de Datos

Finalmente, un aspecto importante del cubo de datos multidimensional es el referente a su implementación. Básicamente se plantean dos cuestiones:

1. Alternativas para implementar el cubo de datos.
2. Materialización física del cubo de datos implementado.

Respecto al primer punto, hay varias alternativas a la hora de implementar el cubo de datos. Estas son:

1.a) *Implementar todo el cubo de datos*. Esta aproximación es la que proporciona mejores tiempos de respuesta. En contrapartida, si las dimensiones del cubo de datos son muy grandes, el cómputo previo y almacenamiento de cada celda del cubo puede no ser la mejor opción, en cuanto a tiempo y espacio en disco consumido. Está opción es la que se asume en [Gra95, Gra97].

1.b) *No implementar nada del cubo de datos*. Esta aproximación es la opuesta a la anterior, por lo que no se necesita tiempo previo para el cómputo del cubo, ni tampoco espacio para su almacenamiento. Sin embargo, el principal inconveniente está en que el tiempo de respuesta sea muy alto, puesto que cada consulta requiere un cómputo completo.

1.c) *Implementar sólo parte del cubo de datos*. Esta aproximación es la propuesta en [Har96]. La idea básica se centra en que los valores de muchas celdas del cubo se pueden calcular a partir de otras celdas del mismo. En el caso de cubos de datos de gran tamaño, podemos materializar sólo una parte del cubo y calcular el valor del resto de celdas aprovechando la relación de dependencia que existe entre ellas. Hay que encontrar un equilibrio adecuado entre, por una parte, el tiempo de cómputo de parte del cubo y el espacio requerido para su almacenamiento, y por otra, el tiempo de respuesta de las consultas realizadas al cubo.

Una cuestión importante es elegir correctamente las celdas del cubo de datos que se van a materializar. Esto es equivalente a decidir qué vistas materializar (En [Har96] se propone la estructura retícular y el algoritmo *Greedy*).

Respecto al segundo punto, también hay varias alternativas al materializar físicamente el cubo de datos implementado, es decir, existen varios modos de almacenamiento:

2.a) *Almacenamiento Multidimensional* (MOLAP). Utiliza tecnología multidimensional, concretamente los datos del cubo se almacenan en estructuras multidimensionales basadas en vectores. Su principal ventaja es el rendimiento. Como ejemplo podemos citar Express de ORACLE [ORA] y Arbor de Essbasse [ARB].

2.b) *Almacenamiento Relacional* (ROLAP). Se utiliza tecnología relacional. El cubo se almacena en tablas de bases de datos relacionales. Se puede mejorar su

rendimiento si se almacenan ciertas tablas resúmenes que contengan información ya calculada sobre ciertas vistas del cubo. Su principal ventaja es la escalabilidad. Microstrategy [MIC] utiliza este tipo de almacenamiento.

2.c) *Almacenamiento Híbrido* (HOLAP). Combina la tecnología relacional con la multidimensional para almacenar el cubo, beneficiándose de las ventajas de ambas. Por ejemplo, Microsoft SQL Server 7.0 OLAP Service [MSQ] soporta un servidor de HOLAP.

Explotación del Cubo de Datos: Operaciones OLAP

Las operaciones OLAP típicas son las que se enumeran a continuación:

- Agregación (*Roll-up*). Elimina un nivel de agregación en el análisis, agregando los grupos actuales. Puede verse como un aumento del nivel de abstracción, presentando los datos más agrupados en las distintas dimensiones.
- Disgregación (*Drill-down*). Introduce un nuevo nivel de agregación en el análisis, disgregando los grupos actuales. Puede verse como una disminución del nivel de abstracción, presentando los datos más divididos en las distintas dimensiones.
- Pivote (*Pivot*). Se reorienta la vista multidimensional de los datos. Se intercambian filas y columnas, cuando se presentan los datos en una tabla cruzada.
- Selección (*Slice*). Se extrae información agregada para un valor dado de una dimensión.
- Proyección (*Dice*). Se extrae un subcubo o intersección de varias vistas.

8.1.1.2 Diseño de un Almacén de Datos

El cubo se construye a partir de la información contenidas en un almacén de datos, por lo tanto, un aspecto importante es el diseño de dicho almacén. Siempre se parte de un modelo multidimensional, de acuerdo con las distintas dimensiones y medidas de interés para el análisis. Este modelo puede adoptar diversos esquemas: estrella, copo de nieve o constelación. A continuación se describen sus principales características.

- **Esquema Estrella**. Es el modelo multidimensional básico y el más utilizado. Se caracteriza por contener:
    - Una tabla central, llamada *tabla de hechos*, que contiene la información que es objeto de análisis, sin redundancia.
    - Un conjunto de tablas dependientes de la anterior, que hacen referencia a las dimensiones. Existe una tabla por cada dimensión, denominada *tabla de dimensión*.

La Figura 8-4 muestra gráficamente este esquema. En la tabla de hechos, cada tupla contiene una parte que son referencias a las dimensiones y otra que constituyen los datos medibles, asociados a los valores concretos de las dimensiones. Estos datos suelen ser de tipo numérico para permitir agregaciones de datos y operaciones estadísticas. Las tablas de dimensiones describen aspectos relevantes que pueden tener una incidencia sobre el dato medido. Existe una relación 1:M entre la tabla de hechos y cada una de las tablas de dimensiones.

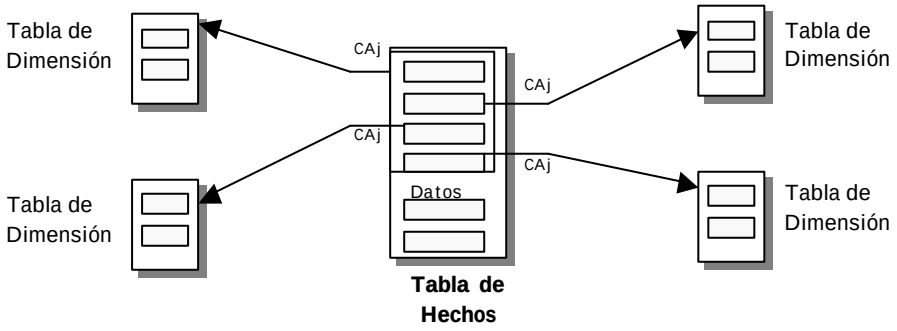


Figura 8-4. Esquema multidimensional en estrella

- **Esquema Copo de Nieve**. Es una variante del modelo anterior. En él, las tablas de dimensiones están normalizadas, con lo cual éstas se pueden dividir y dar lugar a tablas adicionales.
- **Esquema Constelación**. Este modelo también está basado en el esquema estrella, puesto que puede ser visto como una colección de estrellas. Aparece cuando se requiere que exista más de una tabla de hechos y además las diferentes tablas de hechos comparten una o más tablas de dimensiones. Es decir, se desean integrar en un mismo esquema varios esquemas estrella, asumiendo que existen dimensiones que son compartidas.

8.1.1.3 Arquitectura de un Almacén de Datos

Los almacenes de datos suelen adoptar una arquitectura de 3-niveles (ver Figura 8-5). El primer nivel es el propio almacén de datos que frecuentemente se encuentra en un sistema de base de datos relacional, junto con herramientas de monitorización y administración de dicho almacén; además existen metadatos que tienen que ver con la propia definición del almacén. El segundo nivel, o nivel intermedio es un servidor OLAP, que puede ser

implementado en un modelo ROLAP, en un modelo MOLAP o una aproximación híbrida. Este servidor se encarga de la construcción del cubo o cubos de datos. El tercer nivel, es un cliente que contiene herramientas de consulta e informes, de análisis y/o data mining, para la explotación y visualización de datos del almacén.

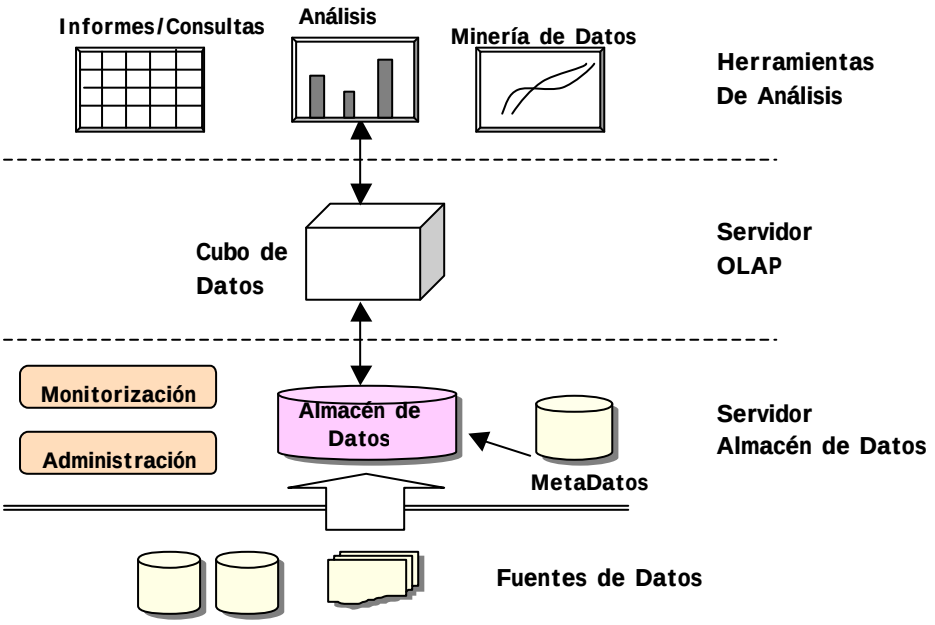


Figura 8-5. Arquitectura de 3-niveles de un almacén de datos

8.1.2 Minería de Datos

El término minería de datos se define en [Fay96] como "la extracción no trivial de conocimiento implícito, previamente desconocido, y potencialmente útil, a partir de un conjunto de datos". En esta disciplina confluyen conocimientos de otros campos, tales como aprendizaje, estadística, bases de datos y técnicas de visualización para descubrir y presentar conocimiento de forma comprensible a los usuarios.

En principio, las técnicas de minería de datos son aplicables a cualquier tipo de repositorio de información; esto incluye almacenes de datos, bases de datos relacionales, bases de datos transaccionales, sistemas de bases de datos avanzados, ficheros planos y el www[57].

[57] Estas siglas corresponden a *World Wide Web*.

Temas típicos que aborda la minería de datos son la búsqueda de asociaciones, clasificaciones y secuencias. A continuación se describe brevemente en qué consiste cada uno de ellos:

- **Asociaciones**. Búsqueda de relaciones entre un cierto conjunto de ítems, a las que se denominan reglas de asociación.
- **Clasificaciones**. Búsqueda de reglas que particionen los datos en grupos disjuntos.
- **Secuencias**. Ordenación de datos teniendo en cuenta la componente temporal.
- Otros temas típicos dentro de minería de datos son: análisis de clusters, análisis de series temporales, predicción, caracterización/comparación de datos, etc. Una descripción más detallada de todos ellos se puede encontrar en [Agr93b, Fay96, Han01].

Nos centramos en la búsqueda de asociaciones, por ser de interés para uno de los objetivo que se persigue en la tesis (el análisis de ejecuciones de flujos de trabajo).

8.1.2.1 Asociaciones

Dado un conjunto de datos, la búsqueda de asociaciones consiste en el descubrimiento de ciertas reglas o patrones, que indiquen que la ocurrencia de un cierto dato o conjunto de datos (llamados ítems) implica la presencia de otros ítems, a partir del conjunto de transacciones o acciones almacenadas. Se representan como reglas de la forma : ABC $\Rightarrow$ FG, donde A,B,C, F,G son ítems y la regla lo que indica es que siempre que ocurre A,B y C también ocurre F y G.

De todas las reglas o patrones encontrados, nos planteamos una pregunta: ¿son todas ellas interesantes?. La respuesta es no; sólo una pequeña parte de estas reglas son interesantes para el usuario, y ello significa básicamente que aportan conocimiento nuevo. Existen algunas medidas objetivas para determinar si una regla es interesante o no. Se basan en conceptos estadísticos y en la estructura de la regla que se descubre. Entre ellas destacan especialmente dos medidas: la confianza y el soporte de la regla.

Dada una regla de asociación de la forma, P $\Rightarrow$ Q. Tenemos.

a) El soporte de la regla (*Rule* ***Support***), se define como el porcentaje de transacciones sobre el total de las analizadas (conjunto de datos analizados), en los que se satisface la regla. Se puede definir como:

$$Support\ (P \Rightarrow Q) = \text{Probabilidad}\ (P \cup Q)$$

Es decir, como la probabilidad de que una transacción, del conjunto de transacciones realizadas, contenga tanto P como Q.

b) La confianza de una regla (*Rule* ***Confidence***), se define como el porcentaje de transacciones en las que ocurre Q sobre el total de transacciones en las que ocurre P. Se puede ver como la probabilidad condicional:

$$Confidence\ (P \Rightarrow Q) = \text{Probabilidad}\ (P\ /\ Q)$$

Es decir, como la probabilidad de que la transacción que contenga a P también contenga a Q.

El problema, por lo tanto, se centra en encontrar reglas de asociación que satisfagan las restricciones especificadas por el usuario; que estén por encima de un valor mínimo de confianza y soporte dado.

La búsqueda de asociaciones, de modo general, se descompone en dos subproblemas:

1. Generar todas las posibles combinaciones de ítems. De éstas elegir las que están por encima de un cierto umbral prefijado, en cuanto a número de apariciones en el total de ejecuciones (valor soporte mínimo). Estas combinaciones son las que se denominan *large itemsets*. El resto de combinaciones que no superan el umbral se denominan *small itemsets*.
2. Para cada *large itemset*, generar todas las posibles reglas y calcular el valor de confianza. Finalmente, sólo se muestran aquellas que superan el valor de confianza mínimo prefijado.

Según [Agr93a], la principal dificultad en la búsqueda de asociaciones está en el paso 1. En [Agr93b] se propone un algoritmo eficiente para la construcción de los conjuntos de ítems que estén por encima de un cierto umbral, denominado algoritmo *Apriori*.

8.1.3 OLAP-Mining

Las funcionalidades propias de OLAP y de minería de datos se pueden ver como totalmente disjuntas, de hecho, en un principio son campos de investigación separados. En [Agr93a] se propone el algoritmo *Apriori*, para la búsqueda de reglas de asociación en grandes conjuntos de datos, mientras que el concepto de cubo de datos y explotación del mismo es posterior [Gra95]. Sin embargo, actualmente ambos campos pueden verse, desde una perspectiva más global, como partes de un proceso general de descubrimiento de conocimiento (visto en la Figura 8-1). Por ello, aparecen trabajos que presentan una integración de ambos campos. Este es el caso de *OLAP-Mining* (también denotado por OLAM, *On-Line Analytical Mining*).

En [Han97] se propone *OLAP-Mining* como un mecanismo que integra la técnicas propias de la tecnología OLAP con las de minería de datos. Esta integración facilita la búsqueda de conocimiento interesante en cubos de datos. La aplicación de técnicas propias

de minería de datos sobre cubos de datos, facilita la búsqueda de patrones o conocimiento de forma multidimensional y a varios niveles de abstracción.

La funciones típicas de minería de datos y de exploración del cubo se pueden entremezclar e integrar, haciendo que el proceso de búsqueda de información sea interactivo y con resultados interesantes. Las funciones más relevantes citadas en [Han97] son:

- Realizar la búsqueda a diferentes niveles y con diferentes porciones de datos. Es decir, antes de empezar el proceso de búsqueda, realizar una operación sobre el cubo y seleccionar los datos. Se denomina *Cubing then Mining*.
- Aplicar operaciones del cubo sobre los resultados de un proceso de búsqueda. Por ejemplo, clasificar y luego para cada clase obtenida aplicar operaciones del cubo (*Mining then Cubin*).
- Aplicar operaciones propias del cubo mientras se está realizando un proceso de búsqueda. Por ejemplo, búsqueda de reglas de asociación a diferentes niveles de abstracción (*Cubing while Mining*).
- En un proceso de búsqueda poder retroceder uno o varios pasos para poder explorar otros caminos alternativos en dicho proceso (*Backtracking*).
- Comparar los resultados obtenidos con diferentes procesos de búsqueda. Por ejemplo, búsquedas con diferentes algoritmos de clustering, o de búsqueda de reglas de asociación (*Comparative Mining*).

Esta integración permite revisitar los problemas típicos que se abordan en minería de datos, desde una nueva perspectiva, la utilización de tecnología OLAP. Por ser de interés para el trabajo, a continuación se detalla cómo se aborda la búsqueda de asociaciones, utilizando el concepto de cubo de datos y explotación del mismo con operaciones OLAP. En [Han97, Han98, Han01] se detallan el resto de posiblidades que ofrece *OLAP-Mining*.

8.1.3.1 Asociación basada en OLAP

Las reglas de asociación pueden ser de tres tipos:

- ✓ Inter-atributo. Asociaciones entre conjuntos de atributos.
- ✓ Intra-atributo. Asociaciones entre valores de un mismo atributo.
- ✓ Híbrida. Combina las dos anteriores.

Cada tipo de asociación requiere algoritmos distintos. En el caso de asociaciones inter-atributo, la estructura mutidimensional del cubo de datos facilita la búsqueda de reglas de asociación entre atributos a múltiples niveles. En el cubo de datos ya se encuentra calculado el número de ocurrencias de una determinada relación, por lo que a partir de estos datos es fácil calcular los valores de confianza y soporte a dicha regla, quedándonos sólo con las que estén por encima del mínimo establecido.

Para la búsqueda de asociaciones intra-atributos se puede aplicar el algoritmo *Apriori*, mencionado anteriormente, incluyendo restricciones en el proceso de búsqueda de dichas reglas [Kam97].

8.2 Análisis de Ejecuciones de Flujos de Trabajo

La extracción de conocimiento interesante a partir de grandes volumenes de datos es de gran utilidad en diversos dominios, ya mencionados en la sección anterior. Las técnicas relacionadas con OLAP y minería de datos son aplicables a cualquier tipo de repositorio de datos. Por lo tanto, se plantea la utilización de toda esta tecnología en un dominio de aplicación concreto, los SGFT.

El problema a resolver es el que se enuncia a continuación. Se parte de un proceso definido según el metamodelo de referencia; dicho proceso representa el flujo de trabajo seguido por una organización y se ha ejecutado un cierto número de veces por un SGFT. En cada ejecución, cierta información relevante sobre el mismo ha sido almacenada en lo que se denomina registro de ejecución o información de auditoría[58]. La información que almacena el SGFT es de diferente tipo, a saber, instante en que se ha iniciado o ha finalizado la ejecución de un proceso, y las actividades que lo componen, estado final alcanzado (terminación con éxito o sin éxito), instante en que un cierto recurso es utilizado por el proceso, o por una actividad concreta dentro del proceso, etc.

Esta información de auditoría registrada por el SGFT se puede utilizar para analizar el proceso y obtener nueva información acerca del mismo. Desde información puramente estadística como tiempos y frecuencias de ejecución, hasta el descubrimiento de conocimiento interesante acerca del proceso, aplicando técnicas de minería de datos, como puede ser el encontrar asociaciones entre actividades y/o recursos. Ahora bien, es importante tener en cuenta que no todos los SGFT almacenan la misma información, ni lo hacen en el mismo formato, a pesar de los esfuerzos de estandarización [WFM98b].

La finalidad última de este análisis es optimizar el flujo de trabajo. Dicha optimización conlleva poder eliminar situaciones anómalas como posibles bloqueos o cuellos de botella que existan actualmente, la mejora de ciertos parámetros como son el tiempo de ejecución y garantizar un rendimiento más óptimo de los recursos existentes.

Una vez planteado el problema, la primera tarea a realizar es determinar qué items interesan analizar en un flujo de trabajo. En base a ellos, se propone un modelo de datos multidimensional que facilite la realización de dicho análisis y su implementación en un almacén de datos concreto. Finalmente, se propone el uso de tecnología existente que

[58] En la literatura relacionada con los SGFT se le denomina *audit trail*.

permita la explotación del almacén de datos construido. Las siguientes secciones abordan estos puntos.

8.2.1 Elementos de Interés a Analizar

El metamodelo de referencia definido en el capítulo 4 se toma como base para determinar qué items interesa analizar en un flujo de trabajo. Éstos son: el proceso que lo representa, los componentes de dicho proceso (actividades y subprocesos) y los recursos que utiliza (datos, aplicaciones y actores). Para cada uno de estos elementos se identifican las medidas y las dimensiones a considerar en el análisis.

8.2.1.1 Proceso

Desde el punto de vista del proceso global, las medidas de interés a obtener son básicamente dos:

M.1 Frecuencia de ejecución. Número de veces de que se ha ejecutado el proceso en cuestión.

M.2 Cálculo de tiempos. Interesa calcular no sólo la duración total del proceso, sino también cómo se reparte el tiempo en función de los distintos estados por los que pasa dicho proceso. En concreto, se definen los siguientes tiempos de interés:

M.2.1. Tiempo que el proceso esta inactivo (*Inactive_time*). Es el tiempo que transcurre desde que se crea la instancia proceso hasta que se cumple la condición de inicio y el proceso pasa al estado activo.

M.2.2. Tiempo que el proceso está en espera de ejecución (*Waiting_time*). Es el tiempo que transcurre desde que el proceso ya está preparado para iniciar su ejecución (ya se ha cumplido la condición de inicio) hasta que realmente empieza su ejecución (concretamente, hasta que se crea una instancia de la primera actividad del proceso).

M.2.3. Tiempo que el proceso está en ejecución (*Execution_time*). Es el tiempo que transcurre desde que se crea una instancia de la primera actividad del proceso, hasta que finaliza la ejecución de su última actividad.

M.2.4. Duración del proceso (*Duration*). Es el tiempo que transcurre desde que se crea la instancia proceso hasta que se destruye dicha instancia, independientemente de cual sea el estado final alcanzado por el proceso (puede finalizar con éxito, sin éxito o detener su ejecución por cualquier motivo). Esta medida del proceso, se puede obtener como suma de los tiempos anteriores, o bien, calculando el tiempo transcurrido desde su creación hasta su destrucción.

Para la definición de las medidas anteriores se parte del diagrama de transición de estados de un proceso (según el metamodelo de referencia). La Figura 8-6 muestra los distintos tiempos considerados como intervalos entre estados del proceso. Para todos ellos interesa calcular el valor medio, el valor mínimo y el valor máximo, en función del número total de ejecuciones que hayan tenido lugar.

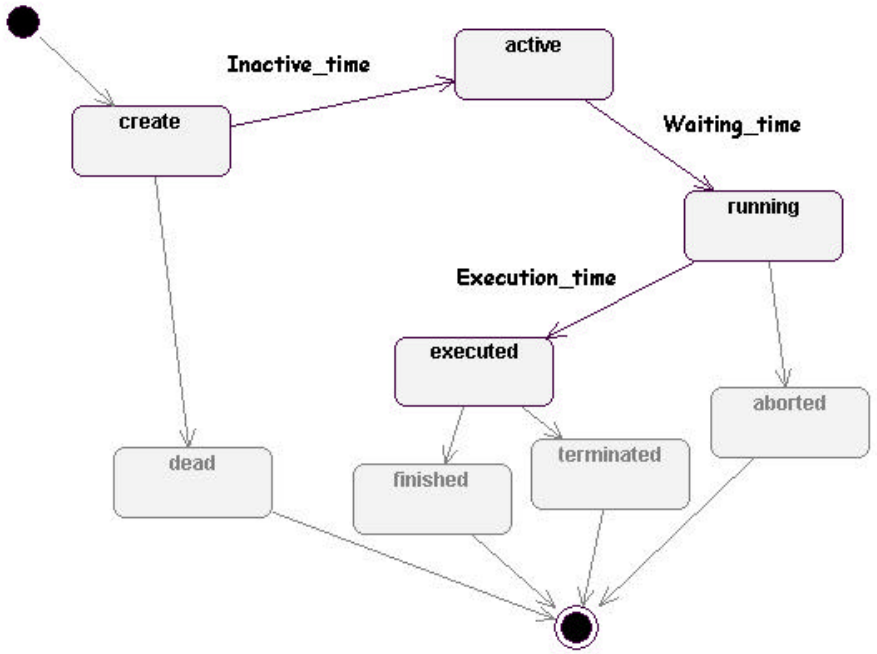


Figura 8-6. Distribución de la duración total de un proceso, en función de sus estados.

Para la obtención de estas medidas, interesa tener en cuenta diferentes aspectos o puntos de vista (dimensiones). En concreto, consideramos importantes las siguientes dimensiones:

D.1 Dimensión tarea (*Task*). Indica qué se está realizando. Para el caso de un proceso, identifica al proceso que se ha ejecutado y al camino (dentro del proceso) seguido en la ejecución.

D.2 Dimensión estado (*State*). Indica el estado final alcanzado, en este caso, por el proceso que se ha ejecutado.

D.3 Dimensión actor (*Actor*). Indica quién realiza la tarea. En el caso del proceso, indica el actor responsable del proceso (o el que lo inicia).

De acuerdo con lo anterior, dado un cierto proceso P, se obtiene la siguiente información de interés:

1. Número de ejecuciones de P analizadas.
2. Duración media / mínima / máxima de P.
3. Tiempo medio / mínimo / máximo que P está inactivo.
4. Tiempo medio / mínimo / máximo que P está en espera de ejecución.
5. Tiempo medio / mínimo / máximo que P está en ejecución.
6. La información etiquetada de 1 a 5, en función del estado final alcanzado por P.
7. La información etiquetada de 1 a 5, en función del camino seguido en la ejecución de P.
8. La información etiquetada de 1 a 5, en función del actor responsable de la ejecución de P.
9. La información etiquetada de 1 a 5, en función del camino seguido en la ejecución de P y del estado final alcanzado.
10. La información etiquetada de 1 a 5, en función del estado final alcanzado por P y del actor responsable del mismo.
11. La información etiquetada de 1 a 5, en función del camino seguido en la ejecución de P y del actor responsable del mismo.
12. La información etiquetada de 1 a 5, en función del estado final alcanzado, del camino seguido y del actor responsable de P.

8.2.1.2 Actividad

Para una actividad, las medidas de interés coinciden con las del proceso, en cuanto a que interesa obtener frecuencias de ejecución y cálculo de tiempos. El diagrama de estados para la actividad es el mismo que para el proceso (Figura 8-6), por lo que las medidas referentes a tiempos transcurridos coinciden. Sólo que en el caso de actividades, aparte de las dimensiones incluidas para el proceso, también es importante considerar como dimensión el resto de recursos. Por lo tanto, además de las dimensiones tarea, estado y actor, se añaden:

D.4 Dimensión aplicación (*Application*). Esta dimensión indica qué aplicación se invoca como parte de la ejecución de la actividad.

D.5 Dimensión dato (*Data*). Esta dimensión indica qué dato se invoca como parte de la ejecución de la actividad. Pudiéndose distinguir entre datos de entrada y datos de salida, si así interesa.

D.6 Dimensión host-dato (*Host_data*). Esta dimensión indica en qué host se encuentran los datos utilizados por una actividad.

D.7 Dimensión host-aplicación (*Host_app*). Esta dimensión indica en qué host se encuentra la aplicación invocada por una cierta actividad, como parte de su ejecución.

En este caso, no se detalla la información que se obtiene, como se ha hecho en el caso del proceso, ya que ésta sigue exactamente el mismo patrón, sólo que la inclusión de nuevas dimensiones hace que el número de vistas aumente. En la sección 8.2.3.1 (pág. 195) se detallan aquellas vistas que proporcionan información de interés. Por ejemplo, cómo varía la duración de una actividad, dependiendo de varios factores como: quién sea el actor que la ejecute, si los datos que utiliza dicha actividad para su ejecución se encuentran en el mismo host o no que la aplicación que las utiliza, etc.

8.2.1.3 Recursos

Para el caso de los recursos, las información que interesa obtener es, básicamente la misma que para actividades y procesos, es decir, la referente a:

a) Frecuencia de utilización del recurso.

b) Tiempos de utilización del mismo.

A continuación se detalla para cada recurso concreto, la información que consideramos de interés.

- **Actores**. Para el caso de los actores, que representan la participación humana en el proceso, la información que interesa obtener tras el análisis es:

  M.1. Número de veces que ha participado el actor en las diferentes tareas que forman parte del conjunto de procesos analizados.

  M.2. Tiempo medio / mínimo / máximo de participación de cada actor.

  Las dimensiones a considerar son: dimensión tarea y dimensión estado.

- **Aplicaciones**. Para el caso de las aplicaciones que se invocan en el proceso, la información a obtener es:

  M.1. Número de veces que se ha invocado una aplicación en el conjunto de procesos analizados.

  M.2. Tiempo medio / mínimo / máximo de ejecución de cada aplicación.

Las dimensiones a considerar son: dimensión tarea, dimensión estado, dimensión host-aplicación, dimensión dato, dimensión dato-host.

- **Datos**. Finalmente, la información que interesa obtener para el caso de los datos es:

  M.1. Número de veces que un cierto dato ha sido utilizado por el conjunto de procesos analizados.

  Las dimensiones a considerar son: dimensión tarea, dimensión estado, dimensión host-dato.

8.2.2 Un Modelo de Datos Multidimensional para Procesos

Una vez hemos determinado qué interesa analizar en un flujo de trabajo, estamos en condiciones de proponer un modelo de datos multidimensional que dé soporte a dicho análisis. Este modelo tiene las dimensiones y medidas que a continuación se detallan, recopilando lo visto en la sección anterior.

➢ **Dimensiones**.

Se han definido un total de 7 dimensiones. Además, en cada una de ellas se establece una jerarquía de agregación que da cuenta de los distintos niveles en los que se pueden agregar / disgregar los datos. Éstas son:

- Dimensión TAREA (*Task*). Indica qué se esta realizando (actividad / proceso). La jerarquía de agregación definida se muestra en la Figura 8-7. Conforme vamos subiendo en dicha jerarquía, los datos se van agrupando más. El nivel más bajo es el que nos permite ver los datos en función de cada una de las actividades. Podemos ir agregando (operación *roll-up*) y presentarlos agrupados por actividades manuales, por subprocesos, por distintos caminos seguidos dentro de un proceso, por procesos, y finalmente no tener en cuenta esta dimensión (el valor none representa el valor ALL definido en el cubo de datos). Esto se puede representar por la siguiente relación de orden [Har96]: activity<manual_activity<subprocess<path<process<none. La jerarquía completa que se ha definido es una relación de orden parcial, ya que las actividades se pueden agrupar bien por actividades manuales o bien por actividades automáticas. De igual modo se puede considerar los distintos caminos dentro de un proceso o no.

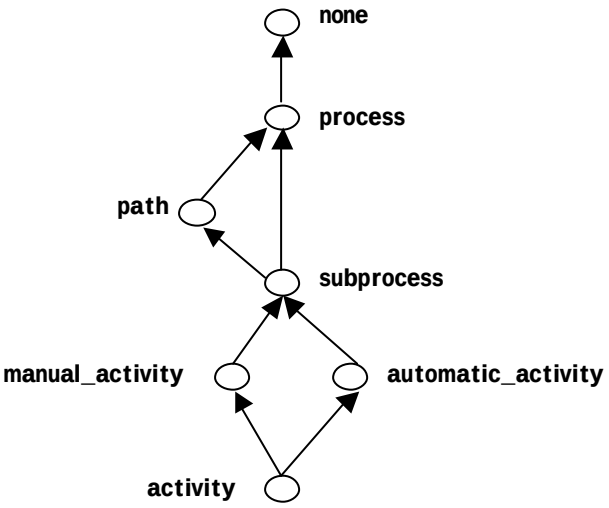


Figura 8-7. Jerarquía de agregación para la dimensión tarea

- Dimensión ESTADO (*State*). Indica en qué estado está una tarea. En esta dimensión no existe ninguna jerarquía de agregación definida. Sólo existen dos posibilidades, que los datos se muestran disgregados en función del estado en que finaliza cada tarea, o bien que se agregen y no se tenga en cuenta dicho estado (valor none). En realidad, la jerarquía definida es: state < none.

- Dimensión ACTOR (*Actor*). Indica quién realiza una tarea. En este caso, la Figura 8-8 muestra la jerarquía de agregación definida para esta dimensión. En este caso, también existe una relación de orden parcial, en cuanto que los usuarios que realizan las tareas se pueden agrupar por roles desempeñados o por unidades organizacionales, dentro de una compañía u organización.

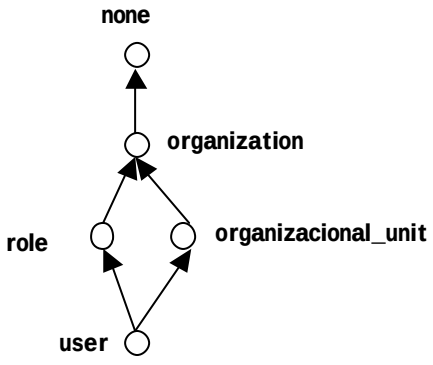


Figura 8-8. Jerarquía de agregación para la dimensión actor

- Dimensión DATO (*Data*). Indica qué datos utiliza una tarea. Para esta dimensión se define una jerarquía de agregación que posibilita agrupar los datos en función de su tipo, por ejemplo, según sean de entrada o salida para las tareas (ver Figura 8-9).

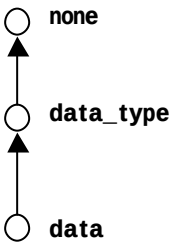


Figura 8-9. Jerarquía de agregación para la dimensión dato

- Dimensión APLICACIÓN (*Application*). Indica qué aplicación utiliza una tarea. En este caso dato, se establece una jerarquía de agregación que permite agregar los datos, dependiendo del tipo de aplicación (ver Figura 8-10).

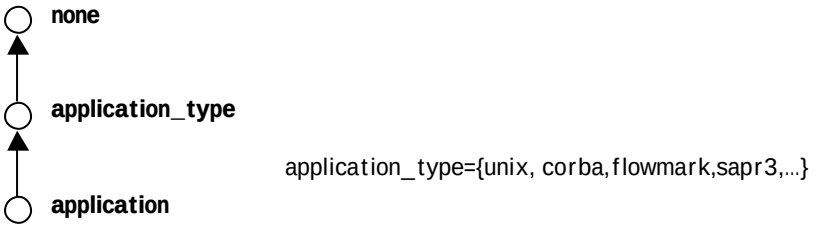


Figura 8-10. Jerarquía de agregación para la dimensión aplicación

- Dimensión HOST_APLICACIÓN (*Host_app*). Indica en qué máquina se está ejecutando la aplicación invocada desde una tarea. La Figura 8-11 muestra la jerarquía de agregación definida. Se trata de una relación de orden total, que permite agrupar / disgregar los datos atendiendo a la máquina en la que está ubicada la aplicación a invocar por una actividad manual, el tipo de máquina, la subred a la que pertenece dicha máquina y finalmente, la red o dominio principal.

- Dimensión HOST_DATO (*Host_data*). Indica en qué máquina están los datos utilizados por una tarea. La jerarquía definida para esta dimensión coincide con la definida para la dimensión host_aplicación (Figura 8-11).

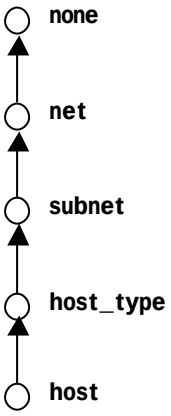


Figura 8-11. Jerarquía de agregación para la dimensión host_aplicación.

- **Medidas**.

En cuanto a las medidas, básicamente interesa obtener dos:

- Numero de ejecuciones.
- Tiempo consumido (medio / máximo / mínimo). Con el fin de obtener mayor información en el análisis, se ha desglosado en:
  - Duración total (*Duration*).
  - Tiempo inactivo (*Inactive_time*).
  - Tiempo en espera de ejecución (*Waiting_time*).
  - Tiempo en ejecución (*Execution_time*).

  Estos tiempos se definen en función del diagrama de estados definido para las tareas (procesos y actividades), en el metamodelo de referencia. De forma que, el tiempo inactivo, es el que transcurre desde que la tarea se crea hasta que pasa a estar activa, es decir, se cumple la condición de inicio. El tiempo en espera de ejecución, es el que transcurre desde que se activa la tarea hasta que ésta inicia su ejecución. Durante este tiempo, la tarea no se inicia puesto no tiene los recursos necesarios. Finalmente, está el tiempo de ejecución real de la tarea.

8.2.2.1 Diseño del Almacén de Datos

A partir de las dimensiones y medidas que se consideran de interés en el análisis de ejecuciones de flujos de trabajo, se construye un almacén de datos que permite obtener la información deseada acerca del proceso o procesos analizados [Pen01b]. Dicho almacén de datos sigue un esquema en forma de constelación, esto es, existen varias tablas de hechos que comparten las tablas de dimensiones.

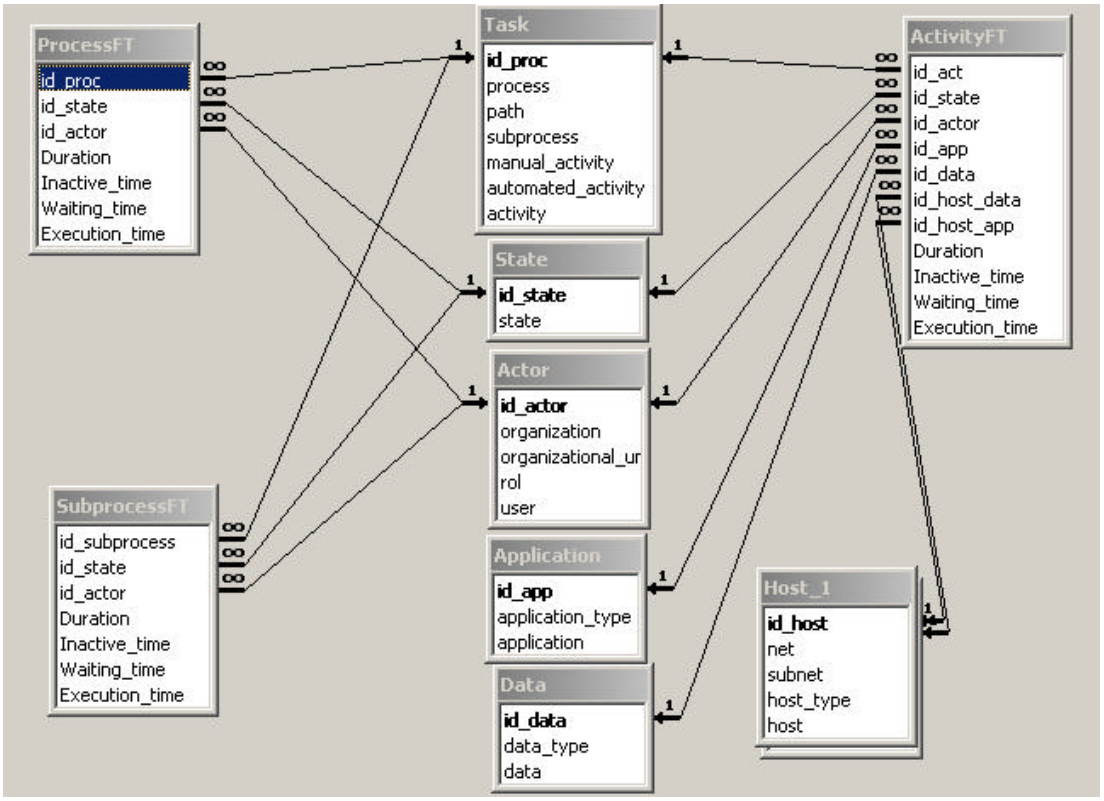


Figura 8-12. Esquema del almacén de datos propuesto

La Figura 8-12 muestra el diseño relacional del almacén de datos propuesto para el análisis de ejecuciones de flujos de trabajo. Las tablas de hechos son ProcessFT, SubprocessFT y ActivityFT. El resto de tablas corresponden a las dimensiones del almacén. A continuación se describen las principales características de todas ellas.

- **Tabla ProcessFT**. Contiene información asociada a cada ejecución de un cierto proceso o conjunto de procesos. En concreto, los datos que se almacenan en dicha tabla hacen referencia, por una parte a las propias medidas del proceso y por otra al valor de las distintas dimensiones.

  - Las medidas que se almacenan para cada instancia proceso, de acuerdo con las definiciones dadas anteriormente, son: tiempo que el proceso esta inactivo (*Inactive_time*), que está en espera de ejecución (*Waiting_time*), que está en ejecución (*Execution_time*) y duración total (*Duration*).

    Para que se pueda construir el almacén de datos con todas estas medidas, es necesario que en el registro de ejecución del SGFT quede almacenada información

sobre cómo avanza la ejecución de un proceso. Más concretamente, debe quedar registrado el instante de creación de la instancia proceso, así como todos los instantes en los que el proceso cambia de estado, hasta que finaliza su ejecución y se destruye dicha instancia.

- Las dimensiones relativas a cada instancia proceso son:

  1. Dimensión tarea (*id_proc*). Se almacena la información acerca del proceso que se ha ejecutado, en concreto el nombre y/o identificador del mismo, y el camino seguido en la ejecución del mismo (secuencia de actividades). Esta dimensión queda almacenada en la tabla *Task*.
  2. Dimensión estado (*id_state*). Se almacena el estado final alcanzado en la ejecución del proceso. Esta dimensión se almacena en la tabla *State*.
  3. Dimensión actor (*id_actor*). Se almacena el actor responsable del proceso. En concreto, se almacena la compañía a la que pertenece, la unidad organizacional o departamento dentro de la misma, el rol que desempeña y el nombre del usuario concreto. Toda esta información asociada con la dimensión actor se almacena en la tabla *Actor*.

- **Tabla SubprocessFT**. Contiene información asociada a las ejecuciones de subprocesos. La información que contiene es exactamente la misma que la tabla ProcessFT, ya que un subproceso es simplemente un proceso que forma parte de otro de mayor granularidad. Por lo tanto, las dimensiones y las medidas que se consideran son las mismas. Lo único que habría que añadir es, en la dimensión tarea, el nombre del subproceso cuya información se está almacenando, aparte del proceso al que pertenece y el camino en el que se encuentra dicho subproceso.

- **Tabla ActivityFT**. Contiene información acerca de la ejecución de las distintas actividades que forman parte del proceso o procesos que se ejecuten. Al igual que en las tablas de hechos anteriores, los datos almacenados hacen referencia a las medidas de las actividades y a las dimensiones de las mismas. Estas son:

  - En cuanto a las medidas tenemos: tiempo que la actividad está inactiva (*Inactive_time*), que está en espera de ejecución (*Waiting_time*), que está en ejecución (*Execution_time*) y duración total (*Duration*).
  - En cuanto a las dimensiones relativas a cada instancia actividad:

    1. Dimensión actividad (*id_act*). En la tabla *Task* se almacena la información acerca de la actividad que se ha ejecutado,. En concreto el nombre y/o identificador del proceso al que pertenece la actividad, el camino dentro del proceso al que pertenece la actividad que se ejecuta, si forma parte de un

subproceso, el nombre y/o identificador del mismo, y finalmente, el nombre y/o identificador de la actividad, y si esta es manual o automática.

2. Dimensión estado (*id_state*). Se almacena el estado final alcanzado en la ejecución de la actividad (tabla *State*).

3. Dimensión actor (*id_actor*). Contiene el actor asociado a la actividad. Al igual que para el caso del proceso, se almacena la compañía y la unidad organizacional o departamento al que pertenece, el rol que desempeña y su nombre. Toda esta información asociada con la dimensión actor está en la tabla *Actor*.

4. Dimensión aplicación (*id_app*). Contiene la aplicación asociada a la actividad. Esta será la que se ejecute cuando la actividad pase al estado *running*. La tabla *Application* almacena la información de esta dimensión, en concreto, el tipo de aplicación y el nombre y/o identificador de la misma.

5. Dimensión datos (*id_data*). Contiene los datos asociados a la actividad, en concreto, se almacena el tipo de dato y el nombre o path del dato en la tabla *Data* que recoge la información de esta dimensión.

6. Dimensión host (*id_host_data* e *id_host_app*). Esta dimensión almacena en qué host o máquina están tanto los datos que utiliza una actividad (*id_host_data*) como la aplicación que se ejecuta al invocarse desde una cierta actividad (*id_host_app*). La tabla *Host* almacena la información de esta dimensión, en concreto, el host que se invoca, el tipo del mismo, la red en la que se encuentra y la subred, si existe.

Las dimensiones almacenadas en las tablas *Task* y *State* son compartidas por todas las tablas de hechos.

El almacén de datos diseñado es una solución general y suficientemente expresiva como para poder realizar el análisis de ejecuciones de procesos, aplicando técnicas de *OLAP-Mining*. De forma similar se pueden incluir tablas de hechos para los recursos, de acuerdo con los elementos de interés definidos en este caso.

8.2.3 Aplicación de Tecnología OLAP-Mining

La Figura 8-13 muestra la arquitectura genérica de un almacén de datos de 3 niveles, aplicada al análisis de ejecuciones de flujos de trabajo. En este caso, las fuentes de datos son los registros de ejecución generados por el SGFT. Pueden existir registros correspondientes a distintos SGFT, dependiendo de si el flujo de trabajo se ejecuta en uno o en varios sistemas.

En el nivel 1 se encuentra el almacén de datos que se ha definido de acuerdo con el diseño presentado en la sección anterior. Un proceso de recolección y transformación de la

información contenida en los registros ejecución, extrae los datos correspondientes a las medidas y dimensiones de interés, procediéndose a la carga del almacén. El segundo nivel es el servidor OLAP que se encarga de la creación y mantenimiento de los cubos de datos, y finalmente, el tercer nivel lo constituyen las herramientas de análisis.

Para el caso de análisis de procesos nos interesan dos tipos de resultados, que se describen más ampliamente en las siguientes secciones:

a) Obtención de estadísticas sobre el flujo de trabajo. (Todas las medidas ya mencionadas al determinar los ítems de interés en la sección 8.2.1).

b) Obtención de reglas de asociación, que permitan la extracción de nuevo conocimiento a partir de datos reales de ejecución.

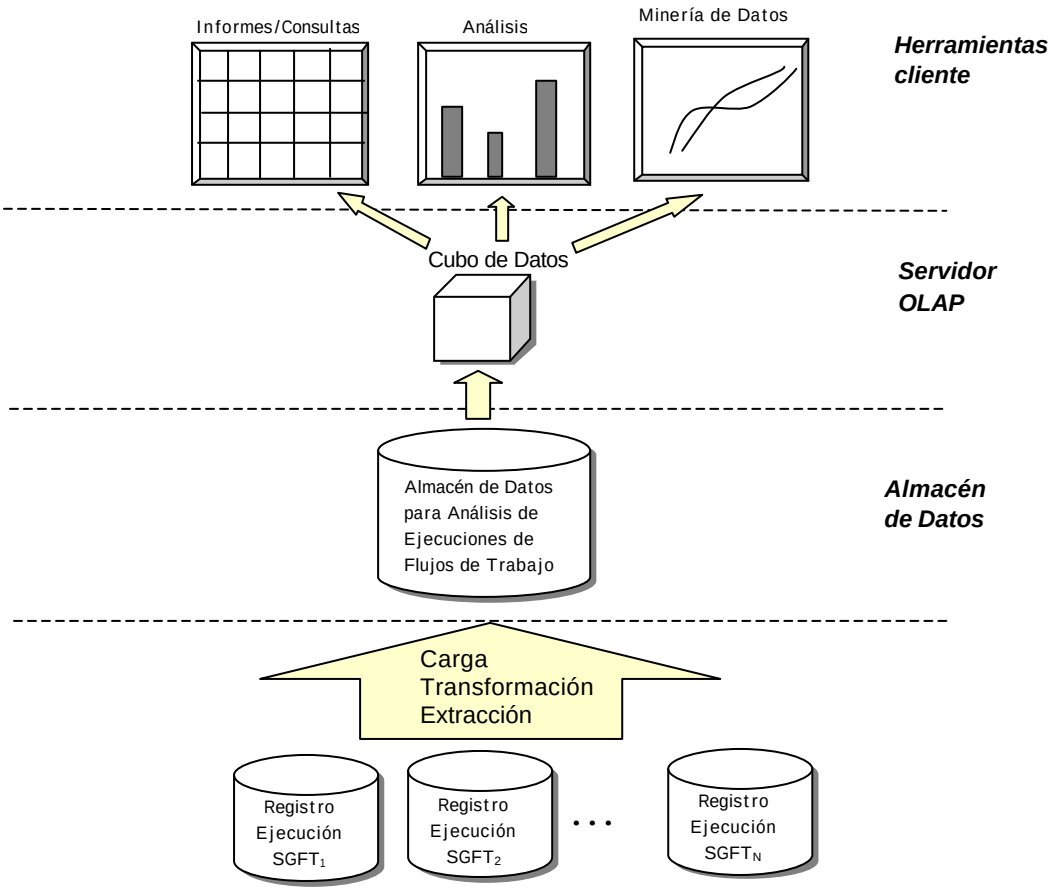


Figura 8-13. Arquitectura de 3-niveles para el análisis de ejecuciones de procesos

8.2.3.1 Explotación del Cubo de Datos

El cubo de datos construido tiene un total de 7 dimensiones y 5 medidas. Cada celda del mismo se puede representar como un conjunto de valores de la forma (d_i, m_j), tal que $1 \leq i \leq 7$ y $1 \leq j \leq 5$. Cada d_i representa una dimensión del cubo, en concreto, la dimensión tarea, estado, actor, dato, aplicación, host_dato y host_aplicación, respectivamente. Cada m_i representa una medida, en concreto, número total de ejecuciones, tiempo inactivo, tiempo en espera, tiempo en ejecución y duración, respectivamente. Los tiempos se almacenan en milisegundos. Por ejemplo, una celda concreta del cubo de dato tiene la siguiente información: (a1, finished, user1, dato1, aplicacion1,host1,host2, 7,25,75, 120, 220)

Una vez construido el cubo de datos, la explotación del mismo se hace en función de las distintas vistas que éste proporciona. En este caso, puesto que hay definidas 7 dimensiones, el total de vistas es de 128 (2^7). La Figura 8-14 muestra, esquemáticamente y de forma incompleta, el retículo con las distintas vistas, y cómo las operaciones básicas de agregación y disgregación permiten mostrar los datos de forma más o menos agrupada.

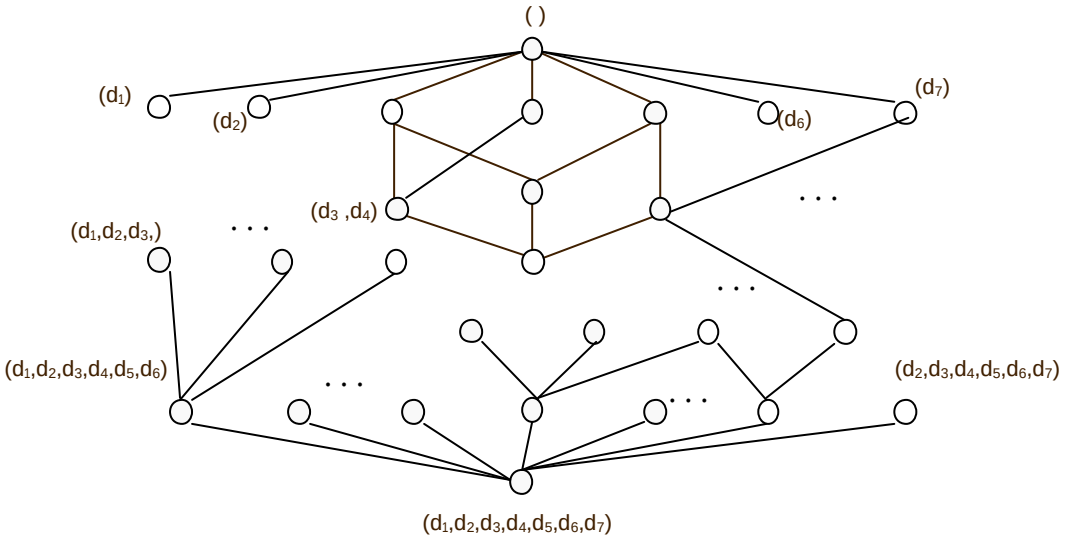


Figura 8-14. Vista parcial del retículo construido para el análisis de ejecuciones de procesos.

De todas las posibles vistas, no todas ellas son interesantes. Hay vistas que aportan información relevante al proceso y otras que no tienen ningún sentido. A continuación se citan algunas de ellas. Además, puesto que existen jerarquías de agregación definidas en la dimensiones, es posible aplicar las operaciones de agregación/disgregación por la combinación de todas ellas

- **(Task).** Esta vista proporciona información de las tareas, independientemente del resto de dimensiones. Permite averiguar qué tareas son cuellos de botella, en el sentido de que consumen mayor tiempo en su ejecución. Puesto que existe una jerarquía de agregación, es posible visualizar los datos a distintos niveles.

  Ejemplo: Para la vista **(Task)** se visualiza la medida m_1[59]. Si elegimos como nivel de la jerarquía las actividades, tenemos el resultado que de forma gráfica visualiza la Figura 8-15. Si elegimos por procesos, es decir, subimos el nivel de granularidad, el resultado obtenido es el que muestra la Figura 8-16.

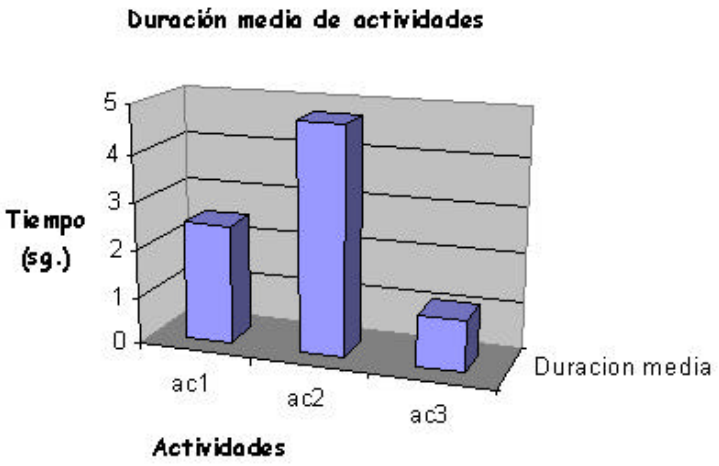


Figura 8-15. Vista (Task) con el valor activity dentro de la jerarquía.

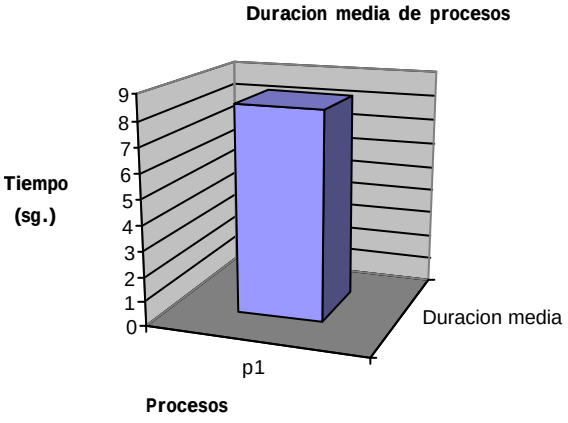


Figura 8-16. Vista (Task) con el valor process dentro de la jerarquía.

[59] Estos datos están basados en los resultados de los test llevados a cabo dentro del proyecto MariFlow.

- **(State).** Esta vista no tiene sentido de forma aislada, sino agregada con la dimensión *task*, puesto que el concepto de estado está asociado a una tarea. Por ejemplo, es interesante disgregar la información de la vista **(Task)** en función del estado (vista **(Task,State)**).

  Ejemplo: La Figura 8-17 y Figura 8-18 muestran los resultados del ejemplo anterior, pero con la inclusión de la dimensión estado[60].

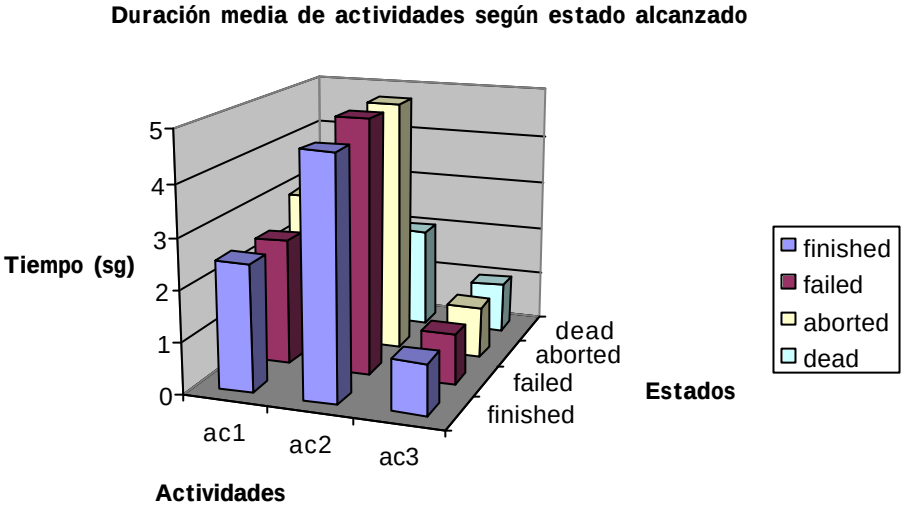


Figura 8-17. Vista (Task,State) con el valor activity dentro de la jerarquía Task.

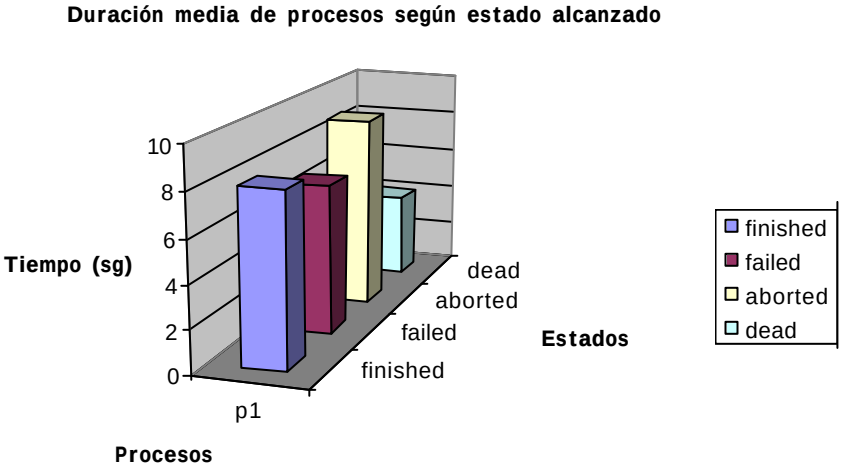


Figura 8-18. Vista (Task, State) con el valor process dentro de la jerarquía Task.

[60] El estado failed se corresponde con el estado terminated del metamodelo de referencia.

- **(Actor).** Esta vista proporciona información sobre los actores, indicandonos el número de veces que un actor a participado en el conjunto de procesos analizados, y el tiempo medio de participación de cada actor. Es posible agregar/disgregar dentro de la jerarquía y obtener datos agrupados por roles, por unidades_organizacionales y por organizaciones.
- **(Data).** Esta vista proporciona información sobre el número de veces que un dato ha sido utilizado en el conjunto de procesos analizados y el tiempo de utilización del mismo.
- **(Application).** Esta vista proporciona información sobre el número de veces que se ha invocado una aplicación en el conjunto de procesos analizados y tiempo consumido en la ejecución. Se puede utilizar la jerarquía definida en esta dimensión para visualizar estos tiempos en función del tipo de aplicación, o por cada aplicación concreta.
- **(HostData).** Esta vista tampoco tiene demasiado interés por sí sola, sino agregada a la dimensión Dato. Así por ejemplo, la vista **(Data, HostData)**, disgrega la información de los datos en función del host en el que se ubican, pudiéndose determinar si ésta afecta al tiempo de utilización del mismo.
- **(HostApp).** Al igual que ocurre con la vista anterior, el interés de esta dimensión no es por sí sola, sino agregada con la dimensión Aplicación. De esta forma, podemos ver cómo varían los tiempos de ejecución de una aplicación en función de la máquina en la que se encuentre (vista **(Application, HostApp)**).
- Las vistas **(Task,Actor)**, **(Task,Data)** y **(Task,Application)** proporcionan información sobre la utilización de los recursos por parte de las tareas. Podemos utilizar las jerarquías de agregación para desglosar estos datos en función de actividades, procesos, etc. Además, si incluimos la dimensión estado en las vistas anteriores, disgregamos los datos en función del estado de la tareas. Por ejemplo, en la vista **(Task, State, Actor),** obtenemos el número de veces y tiempo consumido por cada tarea y el estado en el que finaliza, en función del actor que la realiza o es responsable.

8.2.3.2 Búsqueda de Asociaciones

En el caso de análisis de ejecuciones de flujos de trabajo, nos interesa encontrar reglas de la forma $P \Rightarrow Q$, siendo P y Q una conjunción de elementos que componen el flujo de trabajo. Cada regla tiene asociado un valor de:

- *Confianza*. Porcentaje de ejecuciones del proceso, tal que ejecutándose el antededente de la regla (P), se ejecuta el consecuente de la misma (Q), y
- *Soporte*. Porcentaje del total de ejecuciones del proceso, en las que se ejecuta tanto el antededente (P) como el consecuente (Q) de la misma,

siendo sólo relevantes para el análisis aquellas reglas que estén por encima de un valor mínimo de confianza y soporte establecidos.

En el caso de los flujos de trabajo, interesan encontrar los siguientes tipos:

a) *Asociaciones entre elementos de un flujo de trabajo.* Por ejemplo, asociaciones entre los elementos actividad y usuario, del tipo "el 75 % de las veces que se ejecuta la actividad a1, el usuario que la lleva a cabo es user1". (Asociación inter-atributos).

b) *Asociaciones entre valores de un mismo elemento del flujo de trabajo.* Por ejemplo, asociaciones en el elemento actividad, del tipo "el 80% de las veces que se ejecuta la actividad a1, tambien se ejecuta la actividad a10[61]". (Asociación intra-atributos).

c) *Asociaciones entre elementos de un flujo de trabajo y sus valores.* Por ejemplo, "el 80% de las veces que se ejecuta la actividad a1 por el usuario user1, también se ejecuta la actividad a10 por el user1"; o " El 70% de las veces que se ejecuta la actividad a1 y se envia el dato dato1, la ejecucion de la actividad a10 acaba en fallo". (Asociación híbrida).

En el caso de búsqueda de asociaciones del tipo a), podemos hacer uso del cubo de datos ya construido, puesto que ciertos datos ya han sido calculados. Concretamente, las distintas vistas del cubo nos aportan el número de ocurrencias de conjuntos de elementos que denotan los antecedentes y consecuentes de las posibles de reglas. A partir de estos datos, es fácil calcular los valores de confianza y soporte de dichas reglas y quedarnos sólo con aquellas que estén por encima de los mínimos establecidos. Podemos ver un ejemplo sencillo.

Ejemplo. La vista (actividad, actor) del cubo, nos da el número de veces que cada actividad ha sido ejecutada por un cierto usuario. La vista (actividad) el número de ejecuciones de una cierta actividad y la vista (process) el número de ejecuciones de un cierto proceso.

Supongamos que los datos contenidos en el cubo para cada una de las vistas son los que se muestran a continuación :

(actividad, actor)

	u1	u2
a1	7	3
a2	2	8
a3	6	4

(actividad)

a1	10
a2	10
a3	10

(process)

p1	10

[61] Lógicamente, siempre y cuando no exista una relación de precedencia / sucesión directa en el flujo de control.

Las reglas generadas para la vista (actividad, actor) son las que se muestran a continuación. En este ejemplo, el valor de confianza y soporte de la regla coincide. Esto es debido a que el número de ejecuciones totales del proceso, coindice con el número de ejecuciones totales de cada actividad.

Reglas	Confidence	Support	
a1 -> u1	"7/10	"7/10	70%
a1 -> u2	"3/10	"3/10	30%
a2 -> u1	"2/10	"2/10	20%
a2 -> u2	"8/10	"8/10	80%
a3 -> u1	"6/10	"6/10	60%
a3 -> u2	"4/10	"4/10	40%

En el caso de asociaciones de tipo b), la aplicación del algoritmo *Apriori* [Agr94], junto con la inclusión de restricciones en el proceso de búsqueda, permite la obtención de la reglas de asociación. Finalmente, para el caso c), se combinan las dos aproximaciones anteriores.

Una vez definido el cubo y el tipo de asociaciones que interesa obtener para el análisis de ejecuciones de flujos de trabajo, hacemos uso de la tecnología existente en el campo de OLAP-Mining. Ésta nos proporciona herramientas que implementan tanto la construcción y explotación del cubo de datos a partir de un modelo multidimensional, como los algoritmos para la búsqueda de asociaciones. La herramienta DBMiner es un ejemplo.

8.2.3.3 DBMiner

Es un sistema de OLAP-Mining. Ha sido desarrollado en base a los resultados de investigación del grupo de *Minería de Datos* del *Intelligent Database Systems Research Laboratory*[62] de la *Simon Frauser University* en Canada. Actualmente es propiedad de *DBMiner Technology Inc*[63]. Existen varias versiones del producto, de las cuales destacamos la versión *2.0 Enterprise*, puesto que es de acceso libre y puede descargarse para ser utilizada. También existe una versión educacional (*DBMiner 1.1 y 2.0*). Las características más relevantes de *DBMiner 2.0 Enterprise* son:

- Es capaz de trabajar con cubos de datos de grandes dimensiones. Estos cubos han sido previamente construidos utilizando los servicios OLAP de MS SQL Server 7.0.
- Integra las típicas consultas OLAP, utilizando MS Excel 2000 [MSE] como herramienta para la visualización gráfica y en forma de rejilla de los datos del cubo; así como un visualizador de cubos de 3 dimensiones.

[62] http://db.cs.sfu.ca
[63] http://www.dbminer.com

- Integra características de seguridad.
- Posee una intuitiva interfaz de usuario para la visualización y búsqueda de datos.
- Proporciona un poderoso lenguaje de consulta para minería de datos (DMQL[64]).
- Consta de un conjunto de módulos para la búsqueda de asociaciones, clasificaciones y clusters.
- Rápidez y eficiencia.

El sistema consta de los siguientes módulos:

- *3D- Cube Explorer*. Este módulo permite la visualización, en forma de cubos de 3 dimensiones, de parte del cubo de datos construido. Permite la realización de las típicas consultas OLAP, visualizando sus resultados de forma gráfica.

- *OLAP Browser*. Muestra los datos del cubo en forma de rejilla o en forma gráfica, permitiéndo también la realización de consultas OLAP. Su funcionalidad es similar al módulo anterior, aunque más potente en el sentido de que no está limitada a 3 dimensiones, permitiendo ver los datos desde distintas perspectivas. Este módulo es el que utiliza MS Excel 2000.

- *Association*. Descubre asociaciones o relaciones de dependencia entre los datos.

- *Clasification*. Analiza los datos para construir un modelo que puede utilizarse para categorizar o clasificar los datos en base a las tendencias detectadas.

- *Clustering*. Este módulo agrupa los datos en base a su similaridad en la dimensión elegida.

Todas estas características, junto con la posibilidad de disponer de la misma sin coste alguno, han motivado la elección de *DBMiner 2.0 Enterprise* como herramienta para la realización del análisis de ejecuciones de flujo de trabajo. En cualquier caso, la utilización de esta herramienta concreta no limita la solución propuesta.

8.3 Integración con Sistemas de Gestión de Flujos de Trabajo

La solución propuesta para abordar el análisis de ejecuciones de flujos de trabajo, o procesos en general, se integra como una parte más del entorno propuesto para dar soporte al

[64] *Data Mining Query Language.*

desarrollo de flujos de trabajo. En la Figura 8-19 se muestra el entorno, en el que aparece resaltado el subproceso de análisis.

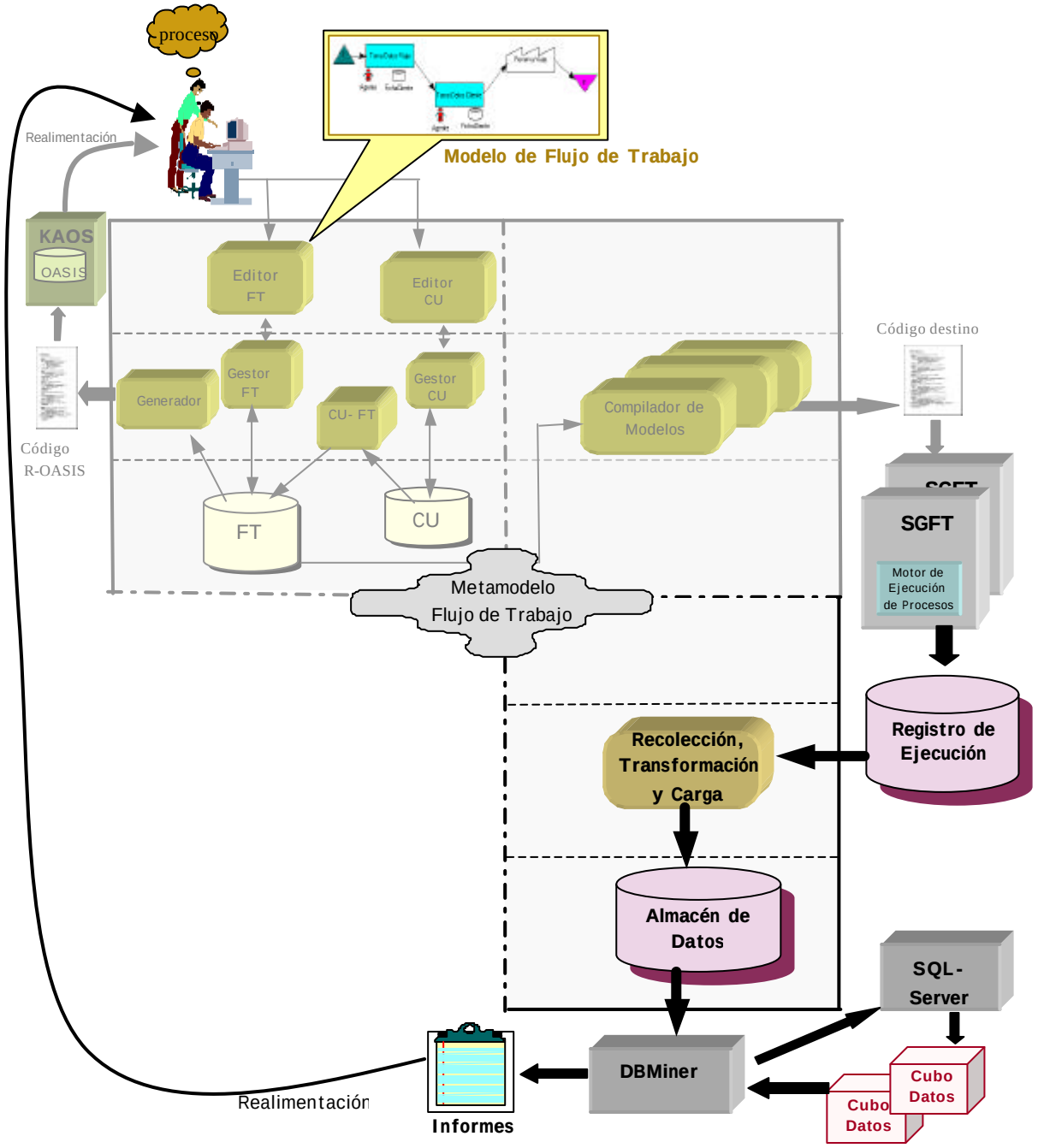


Figura 8-19. Arquitectura del entorno de soporte al desarrollo de flujos de trabajo (*Análisis de Ejecuciones*)

De acuerdo con el esquema propuesto, el registro de ejecución es la entrada al módulo de recolección, tranformación y carga que deja el almacén de datos listo para iniciar su explotación. A partir de este momento se hace uso de la tecnología existente para construir los cubos y proceder a la obtención de informes sobre tiempos y utilización de los componentes del flujo de trabajo, así como búsqueda de asociaciones. La herramienta DBMiner nos aporta está funcionalidad. Los resultados obtenidos sirven de realimentación para mejorar el modelo de flujo de trabajo previo. A continuación se explica con mayor detalle el proceso que extrae la información de ejecución del SGFT y la deposita en el almacén.

8.3.1.1 Proceso de Recolección, Transformación y Carga

Este proceso se ha divido en dos. Por una parte la recolección y transformación de la información contenida en el registro de ejecución del SGFT, y por otra, la propia carga del almacén de datos (ver Figura 8-20). Esto permite introducir un formato intermedio para almacenar los datos resultantes del proceso de recolección y transformación, que a su vez son entrada del proceso de carga. El formato elegido ha sido XML[65] [W3C98].

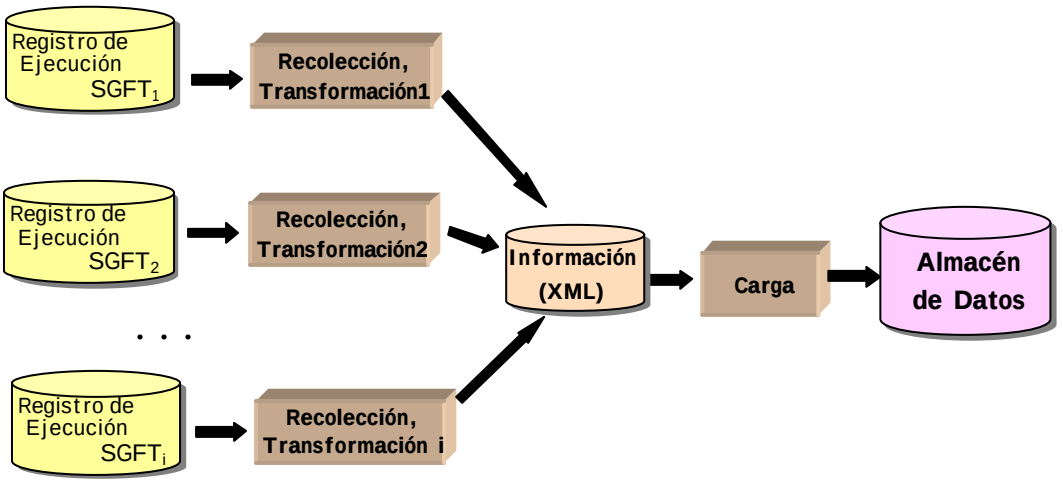


Figura 8-20. Esquema del proceso de recolección, transformación y carga.

Las principales características de ambos módulos, así como del formato intermedio elegidos son las siguientes:

- **Módulo de Recolección y Transformación.** Este módulo es distinto para cada SGFT, ya que es dependiente del registro de ejecución. Depende tanto del formato concreto de

[65] *Extensible Markup Language*

almacenamiento, como de la información contenida, lo cual varía de unos SGFT a otros. Su objetivo es extraer toda la información posible del registro de ejecución, en base a las medidas de interés definidas y el diseño del almacén de datos propuesto.

Esta información, dependiendo de cada caso, podrá obtenerse directamente o será necesario aplicar un conjunto de transformaciones. Estas tranformaciones podrán ser sencillas, como por ejemplo, cálculo de la duración de un proceso, a partir del instante de inicio y finalización del mismo que han quedado registrados por el SGFT; o más complejas, como por ejemplo, la determinación del camino seguido en la ejecución de un proceso, visto éste como la secuencia de actividades que se han ejecutado. Toda la información calculada queda almacenada en un documento XML.

- **Formato Intermedio.** Se ha definido un DTD[66] para validar la creación de documentos XML. Este DTD recoge la definición de tablas del esquema de almacén de datos propuesto para el llevar a cabo el análisis. Por ejemplo, la tabla de hechos ActivityFT, queda como sigue:

```
<!ELEMENT TACTIVITY (ID_ACT, ID_STATE, ID_ACTOR?, ID_APP?,
                     ID_DATA?, ID_HOSTAPP?, ID_HOSTDATA?, DURATION,
                     INACTIVE_TIME, WAITING_TIME, EXECUTION_TIME)>
<!ELEMENT ID_ACT (#PCDATA)>
<!ELEMENT ID_STATE (#PCDATA)>
<!ELEMENT ID_ACTOR (#PCDATA)>
<!ELEMENT ID_APP (#PCDATA)>
<!ELEMENT ID_DATA (#PCDATA)>
<!ELEMENT ID_HOSTAPP (#PCDATA)>
<!ELEMENT ID_HOSTDATA (#PCDATA)>
<!ELEMENT DURATION (#PCDATA)>
<!ELEMENT INACTIVE_TIME (#PCDATA)>
<!ELEMENT WAITING_TIME (#PCDATA)>
<!ELEMENT EXECUTION_TIME (#PCDATA)>
```

  De forma similar se definen el resto de tablas. En el [Fra00] se puede consultar el DTD completo.

- **Módulo de Carga.** El objetivo de este módulo es leer los datos contenidos en un documento XML y cargarlos en el almacén de datos. Este proceso es relativamente sencillo, puesto que el DTD definido y el esquema del almacén son equivalentes, y se utilizan traductores de XML para llevar a cabo gran parte del proceso. Además, este módulo de carga no depende de los distintos SGFT, ya que siempre lee un documento XML, de acuerdo con el diseño del almacén de datos propuesto.

[66] *Document Type Definition*

Las ventajas de esta aproximación, frente a tener un único módulo que realice todas las funciones, son básicamente dos:

a) Posibilidad de reutilizar gran parte del entorno en caso de que en un futuro próximo, los SGFT registren sus ejecuciones como documentos XML. En este caso, se puede utilizar XSLT para transformar el documento que contiene los registros de ejecución en el documento que espera como entrada el módulo de carga.

b) El formato XML es más portable que la propia base de datos que contiene los registros de ejecución. Se va a trabajar con grandes volúmenes de datos y no tiene por qué realizarse todo el proceso de análisis en la misma máquina, sino que el registro de ejecución y el almacén de datos pueden estar en máquinas distintas y este formato facilita su portabilidad.

8.3.2 Aplicación al Sistema OPERA

La solución propuesta para el análisis de ejecuciones de flujos de trabajo se ha aplicado a un sistema concreto, OPERA. Para ello, se han realizado dos tareas:

1. Estudiar el registro de ejecución de OPERA para determinar qué información queda almacenada, y en base a ella, determinar si es posible extraer todas las medidas de interés definidas.
2. Definir las transformaciones del registro de ejecución de OPERA a la información contenida en el almacén en cuanto a dimensiones y medidas. Este proceso de transformación obtiene el documento XML que es la entrada para la carga del almacén.

Respecto a la primera tarea, el registro de ejecución de OPERA queda almacenado en tablas ORACLE (según el esquema que muestra la Figura 8-21). A partir de esta información, no es posible construir el almacén de datos definido con todas sus dimensiones y medidas, sino que éste queda simplificado, tal como muestra la Figura 8-22. Podemos observar que las dimensiones recursos (datos, aplicaciones y actores), así como la dimensión host, no aparecen, puesto que OPERA no registra está información tras la ejecución. Se podría plantear incluir la dimensión aplicación, ya que sí queda registrado qué aplicación es invocada por una cierta actividad, pero puesto que no se almacena la máquina en la que se ejecuta, ni los datos que utiliza, la información que se puede extraer pensamos que no es relevante para el análisis. Además, en OPERA tampoco quedan registrados los instantes de tiempo en que las instancias proceso o las instancias actividad van cambiando de estado. Sólo se registra el instante de creación y de destrucción de dichas instancias junto con el estado final alcanzado. Por lo tanto, sólo se considera como medida el atributo duración, puesto que a partir de la información almacenada, no se puede obtener cómo se reparte dicho tiempo entre ejecución propiamente dicha, espera e inactividad.

Respecto a la segunda tarea, en [Fra01] se detallan todas las transformaciones, que permiten obtener el documento XML correspondiente.

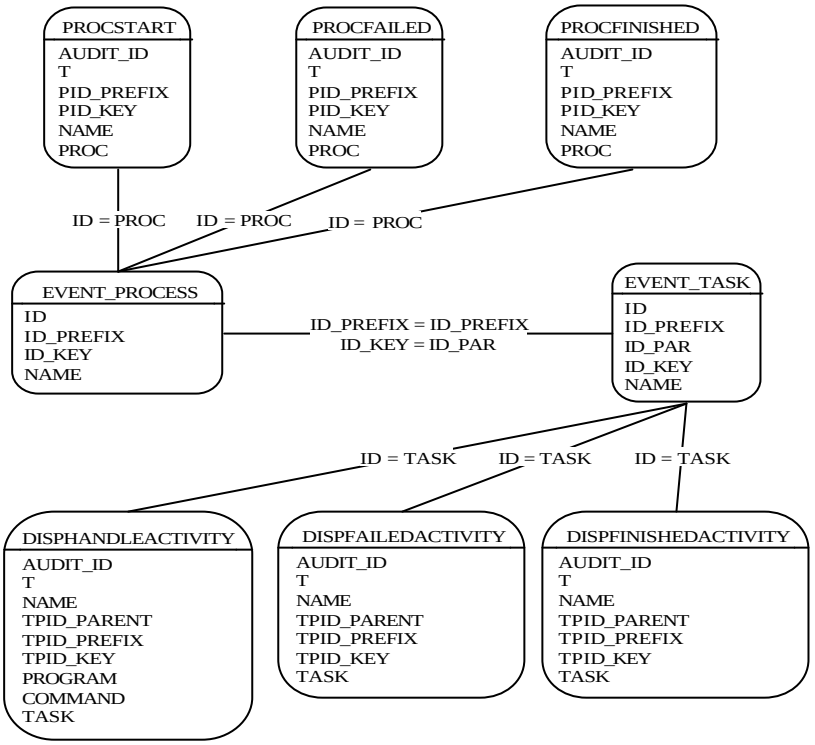


Figura 8-21. Esquema del registro de ejecución de OPERA

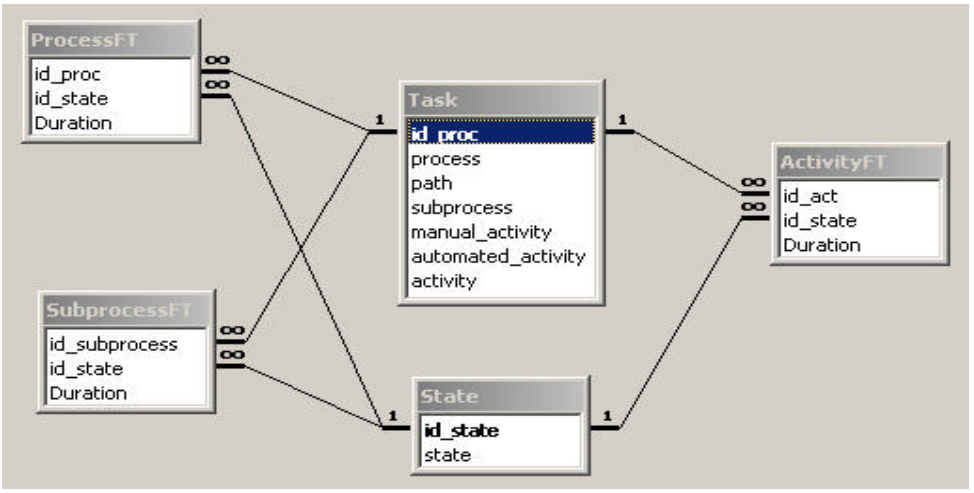


Figura 8-22. Esquema del almacén de datos para el sistema OPERA

8.4 Conclusiones

En este capítulo se ha presentado una solución general para abordar el análisis de ejecuciones de flujos de trabajo. Esta solución se basa en la utilización de tecnología OLAP-Mining, siendo el punto central de la propuesta la definición de un modelo de datos multidimensional, que establece las dimensiones y medidas de interés en dicho análisis. El metamodelo de referencia juega también un papel importante en este subproceso, puesto que establece los componentes de un flujo de trabajo que interesa analizar.

La aplicación de la solución propuesta a SGFT siempre se basa en conocer el registro de ejecución concreto de estos sistemas y determinar si es posible obtener toda la información que debe almacenar el modelo multidimensional propuesto. Una vez realizada esta tarea, se han de definir las tranformaciones pertinentes para llevar a cabo la recolección y transformación de la información en el formato XML propuesto como entrada al cargador del almacén de datos.

Finalmente, una vez construido el almacén con dicha información, las herramientas de OLAP-Mining, como es el caso de DBMiner, permiten explotar dicho almacén en busca de datos estadísticos sobre la ejecución del flujo de trabajo y asociaciones entre los componentes del mismo. Todo ello permite detectar posibles cuellos de botella, mal aprovechamiento de recursos, etc.

Capítulo 9.

Implementación

Este capítulo se presenta un prototipo de entorno CASE que da soporte al desarrollo y mejora de flujos de trabajo desde una perspectiva metodológica. Esta formado por un conjunto de herramientas desarrolladas para cada uno de los subprocesos definidos en los capítulos anteriores. Su objetivo es dar soporte a las distintas actividades a realizar, a la vez que servir como una forma de validar las ideas presentadas.

El capítulo se estructura como sigue. En la sección 9.1 se introducen las distintas herramientas desarrolladas en las cuales se ha dividido la funcionalidad global del entorno, dándose algunas características de la plataforma de implementación utilizada. Las secciones 9.2, 9.3 y 9.4 están dedicadas a cada una de las herramientas desarrolladas. Finalmente, en la sección 9.5 aparecen las conclusiones.

9.1 Funcionalidad

En la tercera parte de esta tesis se ha ido presentado una aproximación concreta para la realización de cada uno de las actividades que componen el proceso de desarrollo de flujos de trabajo. Un entorno de soporte a dicho proceso debe proporcionar toda la funcionalidad descrita en los capítulos anteriores. Se ha optado por el desarrollo de varias herramientas, cada una de las cuales proporciona parte de dicha funcionalidad.

- Una herramienta de desarrollo que incorpora las funcionalidades de definición de flujos de trabajo y obtención de la especificación para su validación en un entorno de prototipación o para su ejecución en un SGFT.

- Una herramienta de captura de requisitos del proceso de negocio, siguiendo una aproximación basada en casos de uso del negocio, que permite la obtención de un modelo preliminar de flujo de trabajo.
- Una herramienta de mejora que permite iniciar un proceso de extracción de conocimiento a partir de los registros de ejecución, tras la construcción y carga de un almacén de datos.

La implementación de todas ellas se ha realizado en *Borland Delphi Client/Server* [DEL], un entorno de desarrollo rápido de aplicaciones (RAD) cliente/servidor. El lenguaje que utiliza es orientado a objetos, proporcionando una biblioteca de componentes visuales que facilita la programación basada en componentes. Incorpora un motor de base de datos propio (*Borland Database Engine* (BDE)), permitiéndo el acceso a distintos sistemas de bases de datos como dBase, Paradox, a fuentes de datos ODBC y diversos sistemas SQL: Oracle, InterBase, Informix, Sybase.

9.2 Herramienta de Desarrollo de Flujos de Trabajo

Esta herramienta incorpora las facilidades de modelado de flujos de trabajo y generación de la especificación equivalente en OASIS y en OPERA. Las principales características de esta herramienta, a nivel de modelado de flujos de trabajo se pueden resumir en:

- Se ha introducido el concepto de *Proyecto* para agrupar todos aquellos procesos que están relacionados entre sí. Así, un proceso se define una sola vez, pero se puede reutilizar tantas veces como se desee, como subproceso de otro proceso más general. Un proyecto siempre tiene asociado al menos un proceso. Por otro lado, para facilitar el refinamiento del modelo del flujo de trabajo, conforme se van perfilando los requisitos del proceso, se permiten dos acciones: convertir en subproceso una actividad y una actividad en un subproceso.

- Los recursos (actores, datos y aplicaciones) que se definen son globales, es decir, no pertenecen a ningún proyecto en concreto. Se definen una sola vez, pero se pueden asociar a varias entidades (subprocesos, actividades y condiciones de transición) dentro del mismo proyecto o de proyectos distintos. De esta forma también se facilita la reutilización de los mismos.
  Cuando se define un recurso, dependiendo de si se trata de un actor, un dato o una aplicación, podemos dar un valor a sus propiedades al igual que cuando se define una entidad. Pero además, se pueden añadir nuevas propiedades a dichos recursos. Para su definición se indica el nombre de la nueva propiedad, el tipo de la misma y el valor por defecto.

- Se puede realizar, en cualquier momento, una comprobación de la corrección del flujo de trabajo que se está definiendo. Esta comprobación se puede realizar de todo el proyecto o de un proceso concreto. Se comprueba si el modelo cumple las restricciones del metamodelo de referencia. Algunos de los errores que se detectan son: recursos dados de alta en el proyecto como parte de algún proceso, pero no asociados a ninguna entidad; errores en el número de flujos de entrada o salida de las condiciones de transición, dependiendo del tipo de las mismas, y la no existencia de la actividad inicial o final de un proyecto.

A nivel de generación de código, la especificación OASIS se genera de forma totalmente automática, puesto que se basa en la formalización del metamodelo de referencia utilizado. La especificación en OPERA es un proceso semiautomático, puesto que puede requerir la intervención del usuario o analista para tomar una decisión ante posibles alternativas (según lo visto en el capítulo 7). A continuación se describe la arquitectura de la herramienta y se muestra un ejemplo de utilización basado en el caso de estudio.

9.2.1 Arquitectura

El desarrollo se ha realizado siguiendo una arquitectura típica de tres niveles: nivel de presentación, nivel de lógica de la aplicación y nivel de persistencia. El nivel de presentación implementa una interfaz gráfica de usuario (IGU), con menús y barras de herramientas, siguiendo las guías de estilo de una aplicación Windows de interfaz de múltiple documento (MDI). La funcionalidad básica del nivel intermedio es la gestión en memoria de los objetos correspondientes a los diferentes componentes del flujo de trabajo (gráficos y no gráficos).

Finalmente, el nivel de persistencia, se encarga del acceso al repositorio, independizando al resto de la aplicación de dicha tarea. Se trata de un repositorio relacional, cuyas tablas contienen toda la información del flujo de trabajo diseñado. Dicha información hace referencia tanto al valor asignado a cada una de las propiedades de cada componente, como a información gráfica del mismo (se almacenan las coordenadas de su posición gráfica). La estructura completa del repositorio se puede consultar en [Seg00]. El sistema gestor de bases de datos de bases de datos utilizado es *Paradox* [DEL].

9.2.1.1 Interfaz gráfica de Usuario

El aspecto general de la herramienta se muestra en la Figura 9-1. En el área de dibujo se definen los diferentes procesos que pueden formar un proyecto, proporcionándose a través de la barra de herramientas y los menús todos los elementos gráficos para modelar el flujo de trabajo; la introducción del valor de las propiedades asociadas a los mismos se realiza mediante sucesivos diálogos. Además, en el árbol de accesos se muestran todos los procesos abiertos del proyecto, así como sus entidades colocadas jerárquicamente. Finalmente, la

barra de estado muestra aparte de la hora del sistema, información contextual de la aplicación y el valor de las coordenadas (x,y) del cursor en relación con la zona de dibujo.

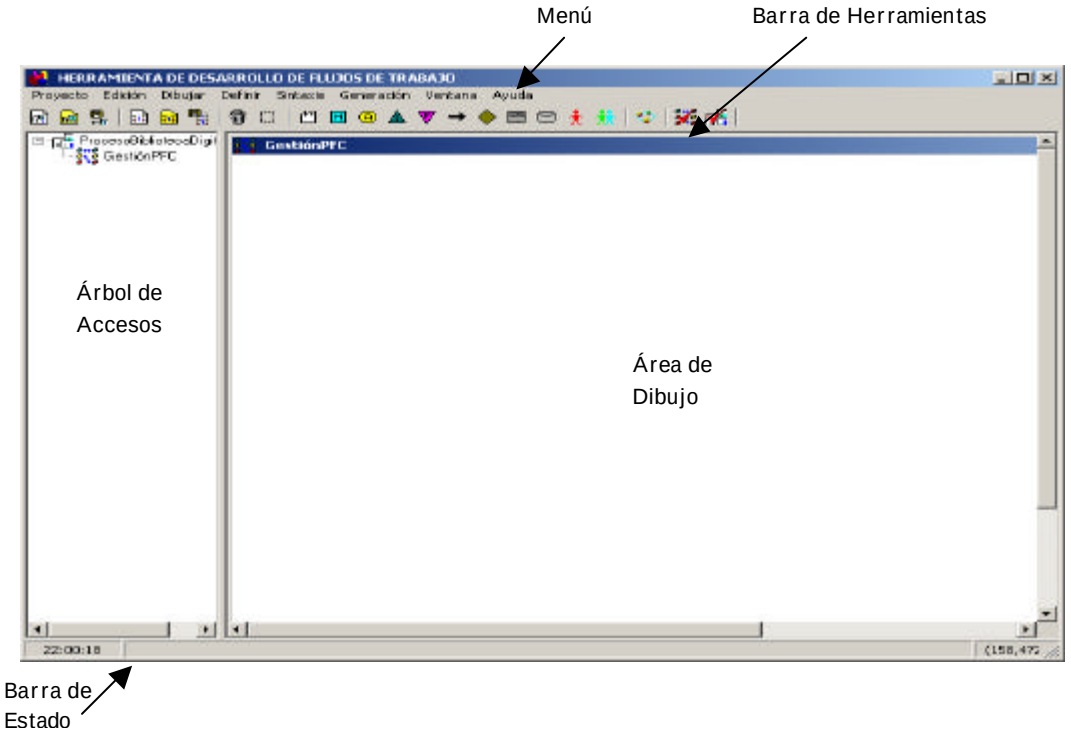


Figura 9-1. Secciones de la interfaz gráfica de usuario

A continuación se ilustra el proceso de definición de un modelo de flujo de trabajo, siguiendo el caso de estudio, y la posterior generación de la especificación correspondiente.

1. **Creación de un nuevo proyecto que representa al flujo de trabajo a modelar**. La herramienta al seleccionar la opción del menú Proyecto/Nuevo Proyecto muestra el formulario de la Figura 9-2 para crear un nuevo proyecto, al que hay que darle un nombre y una descripción. A continuación, y de forma automática, se muestra el formulario de la Figura 9-3 para crear un proceso, que será el proceso principal (el que representa al flujo de trabajo). En este caso, además del nombre y descripción se pueden indicar las condiciones de inicio y finalización del mismo.

2. **Definición del proceso**. La herramienta proporciona los iconos correspondientes al lenguaje gráfico definido para la creación de todos los componentes del proceso, hasta completar la definición del mismo. La introducción de propiedades se realiza mediante diálogos. La Figura 9-4 muestra una instántanea de la definición del proceso

de gestión de PFC, en concreto el correspondiente a la introducción de propiedades de la actividad denominada PresentacionMemoriaPFC.

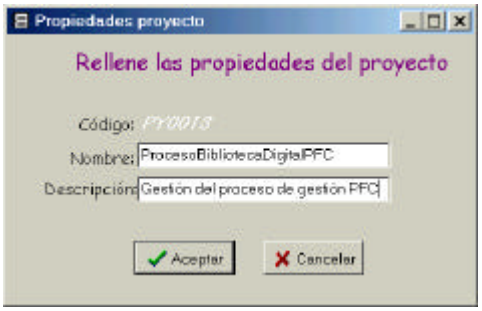


Figura 9-2. Formulario asociado a la creación de un proyecto

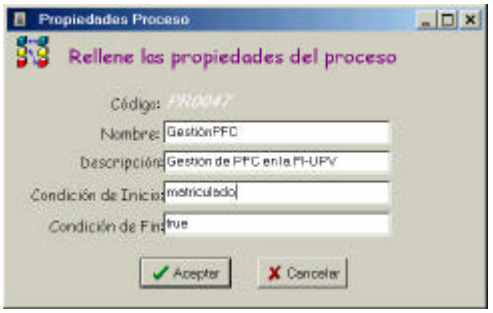


Figura 9-3. Formulario asociado a la creación de un proceso

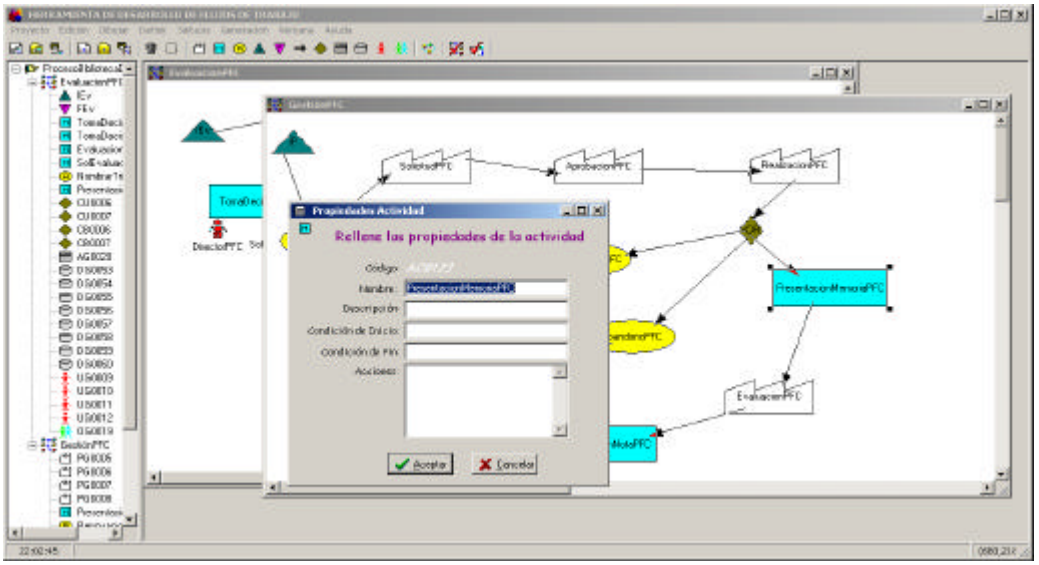


Figura 9-4. Definición del proceso gestiónPFC (1)

Un aspecto a resaltar es la definición y asignación de recursos a tareas como parte de la definición de un proceso. Para asociar un recurso a una entidad, éste debe estar definido. La definición de recursos permite no sólo dar valor a sus propiedades de acuerdo al metamodelo de referencia, sino también permite añadir nuevas propiedades a esos recursos. Las nuevas propiedades se especifican dándoles un nombre, un tipo, un valor por defecto y una descripción. La Figura 9-5 muestra el formulario de definición de datos, en el que se puede apreciar como se han definido dos nuevas propiedades, nota y convocatoria, para el dato InformePFC.

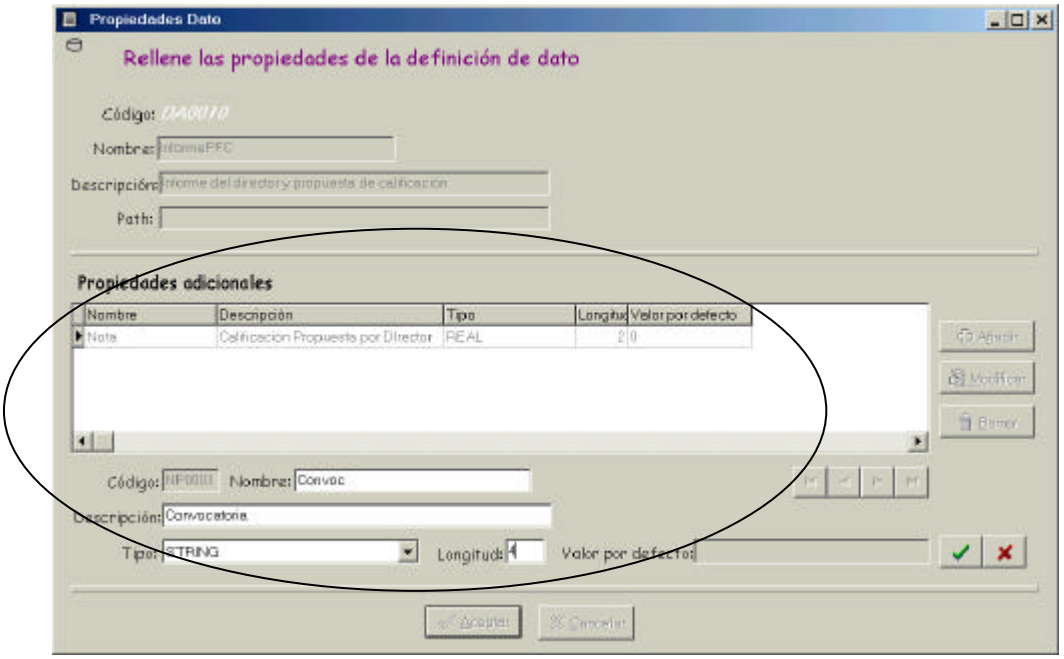


Figura 9-5. Definición de un recurso dato

Una vez definido el dato podemos, si se desea, añadirlo al proceso que se está modelando y asociarlo a una tarea. Los iconos correspondientes a los recursos muestran un aspecto diferente en función de si están asociados o no a una tarea. En la Figura 9-6 se muestra un recurso usuario (Alumno) que está incluido en el proceso pero no está asociado a ninguna tarea, y otro recurso usuario (DirectorPFC) está incluido y asociado a una tarea. También se indica en dicha figura, el botón de la barra de herramientas que realiza la acción de asociar / desasociar recursos.

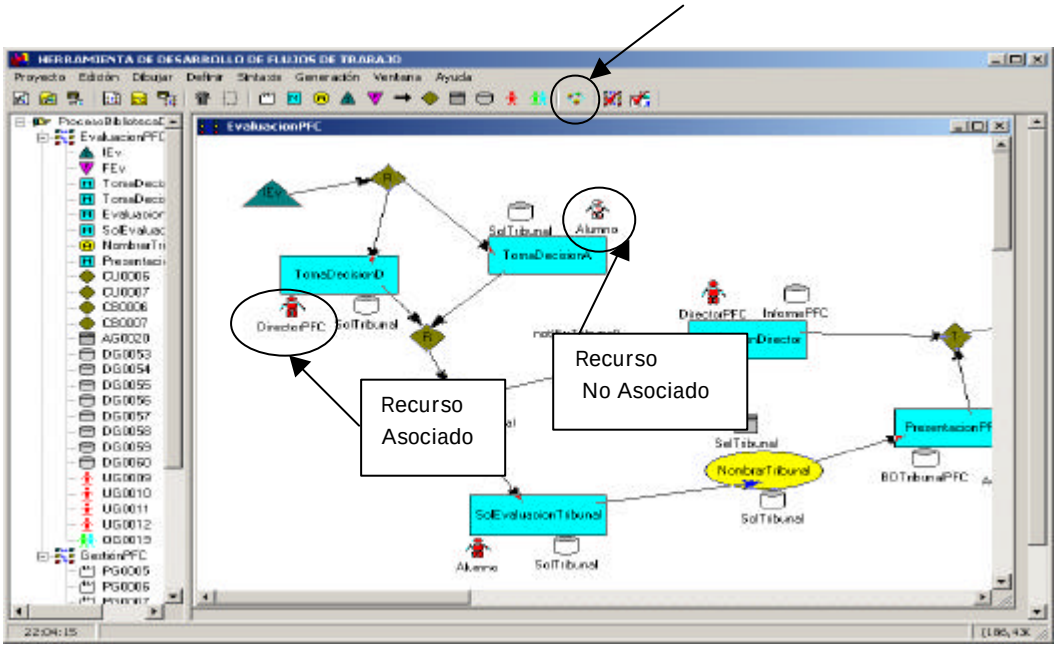


Figura 9-6. Definición del proceso gestionPFC (2)

3. **Comprobación de la corrección**. La herramienta permite en cualquier momento chequear el modelo para detectar errores. La comprobación se puede hacer de un proceso o de todo el proyecto. La Figura 9-7 muestra el resultado de comprobar el proceso actual (evaluacionPFC). En este caso vemos que se ha detectado un error. Si la comprobación se hace de todo el proyecto, se muestra el resultado de comprobar todos sus procesos y como información final se indica el número total de errores en el proyecto (Figura 9-8). En ambas figuras aparece indicado el botón de la barra de herramientas que realiza la acción correspondiente.

4. **Generación de código.** La herramienta permite la generación de código OASIS con la opción *Generación/Generar OASIS*. Esta acción sólo se ejecuta si no existen errores. Una caja de diálogo solicita el nombre del archivo de texto en el que queremos que se almacene el código generado. Como ya se ha comentado anteriormente, este proceso de generación es automático, y cuando finaliza se muestra un mensaje (Figura 9-9).

   La opción *Generación/Generar OPERA* inicia un proceso semiautomático de generación de código. En concreto, la intervención del usuario se requiere para las actividades manuales, a las que necesitamos asociarle un programa en OPERA. Finalmente, se muestra una caja de diálogo para introducir el nombre con el que se almacena el archivo OCR generado y se informa al usuario que el proceso ha finalizado.

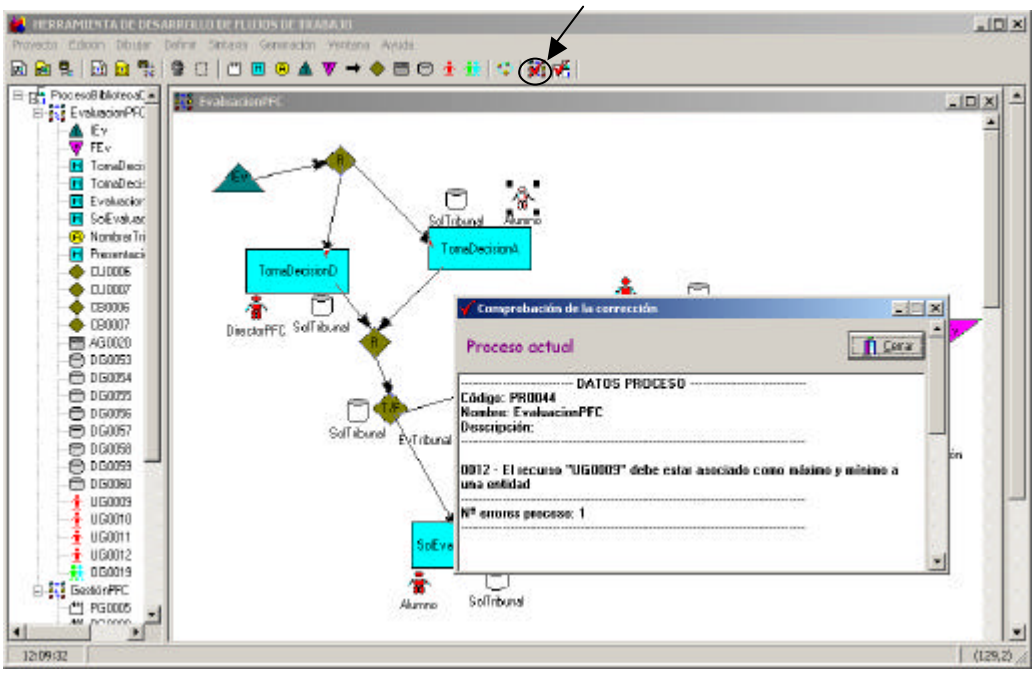


Figura 9-7. Comprobación de la corrección de un proceso

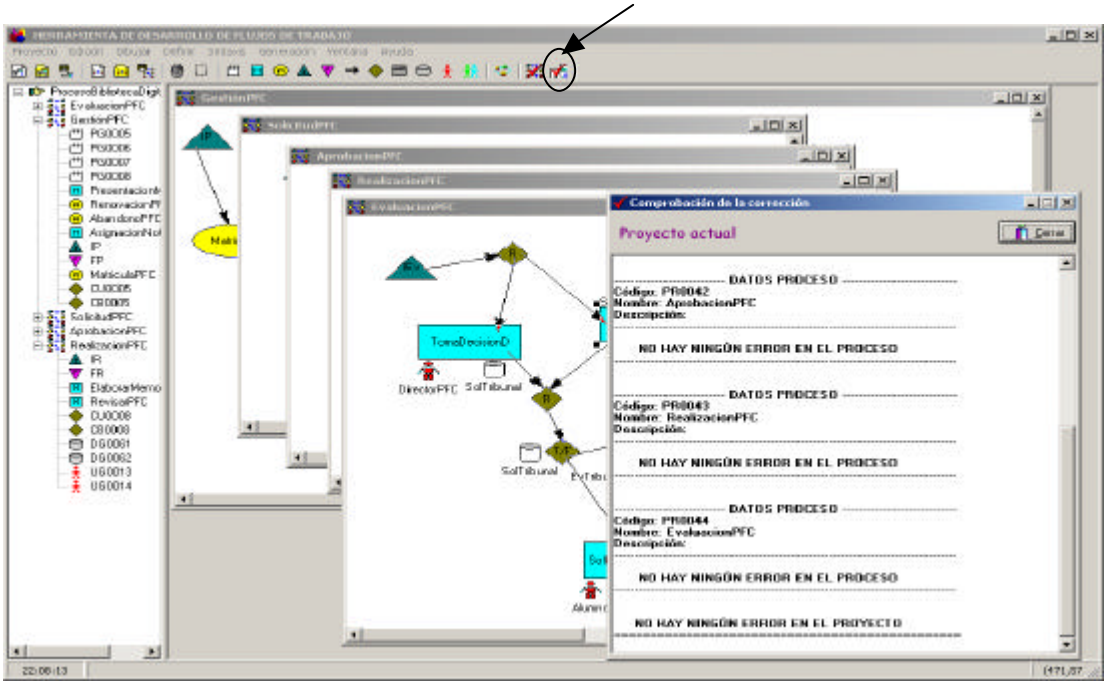


Figura 9-8. Comprobación de la corrección de un proyecto

Figura 9-9. Mensaje de finalización del proceso de generación

9.3 Herramienta de Captura de Requisitos

Esta herramienta proporciona una interfaz gráfica para la elaboración de los casos de uso del negocio. Dicha interfaz es sencilla e intuitiva puesto que su único objetivo es recopilar la información asociada a los casos de uso del negocio y almacenarla en un repositorio relacional. Posteriormente, un módulo implementa las transformaciones definidas para la obtención de un modelo preliminar de flujo de trabajo.

9.3.1 Arquitectura

En esta herramienta se sigue también una arquitectura de tres niveles. El nivel de presentación es una interfaz gráfica al estilo Windows. El nivel intermedio contiene la lógica de la aplicación para la construcción de los casos de uso y el manejo de la información asociada en las correspondientes plantillas. El nivel de persistencia lo constituye el repositorio relacional que contiene la información de los casos de uso construidos. El diseño completo del mismo se puede consultar en [Ceb00]. En este caso también se ha elegido *Paradox* como sistema gestor de bases de datos. El módulo conversor se sitúa en el nivel intermedio y se encarga de acceder al repositorio de casos de uso para recuperar la información, aplicar las transformaciones y generar la nueva información en términos de flujos de trabajo. Esta información es almacenada en otro repositorio cuyo esquema es el correspondiente a los modelos de flujos de trabajo.

A continuación se ilustra el proceso de construcción de los casos de uso y generación de una versión preliminar de flujo de trabajo.

1. **Construcción de los casos de uso del negocio** que se documentan de acuerdo con la plantilla propuesta (Figura 9-10).

2. **Derivación del modelo de flujo de trabajo**. La opción del menú *Proyecto/Generación Flujo Trabajo* inicia el proceso de conversión. Una vez finalizado, se muestra un

diálogo para la introducción del nombre del flujo de trabajo y se informa del resultado de la operación.

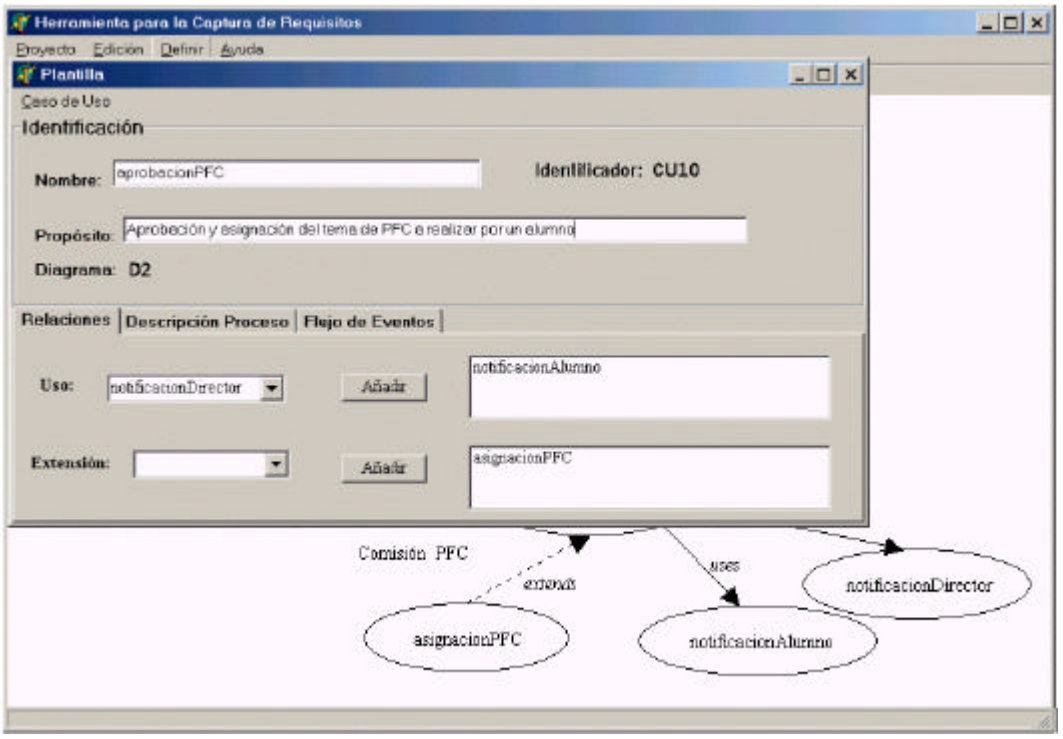


Figura 9-10. Interfaz gráfica para la construcción de los casos de uso del negocio

9.4 Herramienta de Mejora de Flujos de Trabajo

De acuerdo con la solución propuesta al análisis de ejecuciones de flujos de trabajo (pág. 203), esta herramienta está formada por:

- El cargador del almacén de datos cuyo objetivo es leer los datos contenidos en un fichero XML y cargarlos en el almacén.
- Un conjunto de módulos de recolección, transformación y carga cuyo objetivo es extraer toda la información posible del registro de ejecución de los SGFT. En nuestro caso sólo se ha implementado completamente para el sistema OPERA.

A continuación se describe cada una de ellas.

9.4.1 Cargador del almacén de datos

El esquema en XML y el del almacén de datos son equivalentes por lo que este proceso de carga es relativamente sencillo. Consta de dos partes (Figura 9-11):

- Lectura del fichero XML. Se emplea un traductor de XML, en concreto *Extended Document Object Model* (XDOM) [Koh00], que contiene componentes para el procesamiento de documentos XML en *Borland Delphi*. Esto hace que el proceso de lectura se simplifique considerablemente.

- Escritura en el almacén de datos. Hay que tener en cuenta que al cargar las tablas de dimensiones puede suceder que haya valores que ya existan. Esto es debido a que el proceso de recolección es totalmente independiente del almacén de datos. La solución adoptada consiste buscar los elementos existentes y reasignar los identificadores. Durante el proceso de escritura también se deben transformar las cadenas de texto al tipo correspondiente de cada campo.

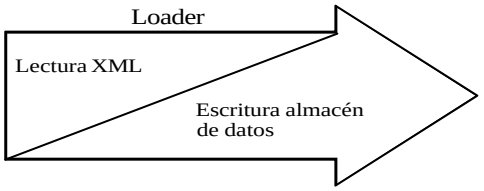


Figura 9-11. Esquema del cargador del almacén de datos

A continuación se describe el proceso de carga del almacén.

1. **Invocación del cargador del almacén de datos** (*Loader*). La Figura 9-12 muestra el formulario para introducir el nombre del fichero XML a cargar. Se puede seleccionar mediante un diálogo de búsqueda de ficheros al que se accede desde el botón de la derecha. El almacén de datos se puede seleccionar de la lista de bases de datos conocidas por el sistema. Por defecto se selecciona aquella base de datos con nombre *workflow*, si no existe, el usuario debe elegir una. Al pulsar el botón de la parte inferior derecha se inicia el proceso de almcenamiento.

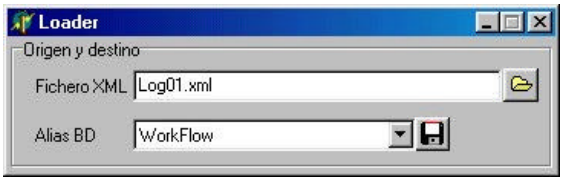


Figura 9-12. Invocación del cargador

2. **Resumen de la carga**. Una vez finalizada la carga, se muestra un formulario con información sobre la cantidad de procesos, actividades, subprocesos, etc. que se han introducido en el sistema (Figura 9-13). Los resultados mostrados en cuadro de resumen son los siguientes:

   - *Origen*: nombre del fichero XML y origen de los datos que contiene.
   - *Destino*: almacén de datos donde se ha guardado el contenido del fichero XML.
   - Cantidad de *elementos* que se han insertado en el almacén de datos y que no existían, así como la cantidad de los insertados que sí existían. Estos elementos son: procesos, actividades, subprocesos, aplicaciones, datos y actores.

   Destacar que en las tablas de dimensiones los elementos repetidos no se almacenan, pero es interesante saber la cantidad de ellos que ya existían en el almacén de datos

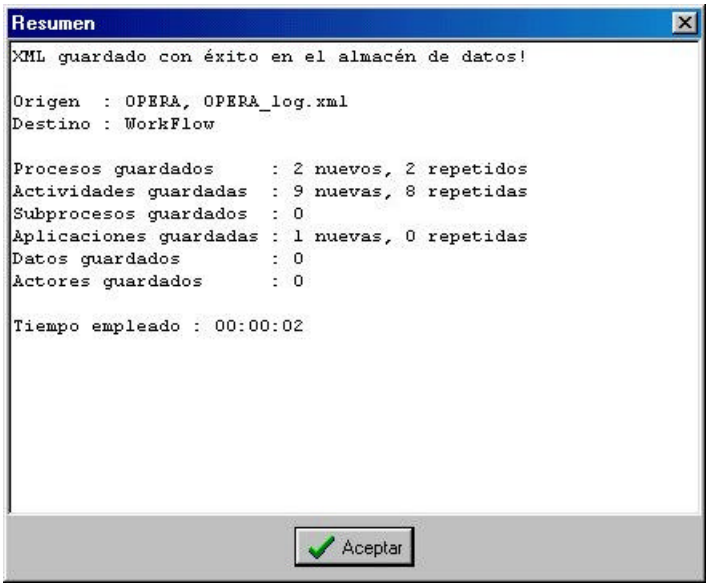


Figura 9-13. Resumen de la carga del almacén

9.4.2 Cargador del registro de ejecución de OPERA

De acuerdo con el registro de ejecución de OPERA y la estructura del almacén de datos, este módulo se encarga de extraer la información significativa y generar el documento XML. El proceso que se sigue es:

1. **Invocación del cargador de OPERA** (*Loader OPERA*). Se muestra un formulario que permite seleccionar el registro de ejecución de OPERA de entre todas las bases de

datos reconocidas por el sistema (Figura 9-14). Cuando pulsamos en el botón con una carpeta se abre la base de datos seleccionada. En ese momento se comprueba si la base de datos tiene todas las tablas de OPERA. Una vez comprobado se muestra el número de tuplas que contiene cada tabla. Este dato es informativo y puede servir para hacernos una idea del tiempo que invertirá la aplicación en crear el documento XML. El siguiente paso puede ser la introducción de caminos equivalentes o guardar el archivo que se genera, dándole un nombre.

El formulario incluye unas barras de progreso para ver como evoluciona la creación del documento XML. La barra superior indica la tuplas procesadas dentro de una tabla. La inferior las tablas que se han recorrido.

Figura 9-14. Cargador del registro de ejecución de OPERA

2. **Resumen del proceso de recolección y transformación**. Una vez finalizada la generación del documento XML con la información de interés, se muestra al igual que antes, un formulario resumen con (Figura 9-15):
   - *Origen*: nombre del fichero XML y origen de los datos que contiene.
   - *Destino*: almacén de datos donde se ha guardado el contenido del fichero XML.
   - Cantidad de *elementos* que se han insertado en el documento XML, así como la cantidad de veces que se repite. Los elementos son: procesos, actividades y aplicaciones.

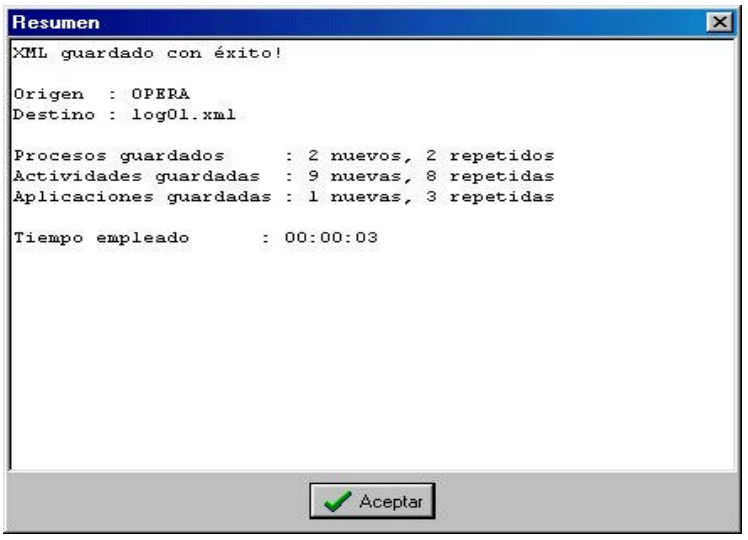


Figura 9-15. Resumen del cargador de registro de ejecución de OPERA

9.5 Conclusiones

Se han desarrollado un conjunto de herramientas para dar soporte al proceso de desarrollo de flujos de trabajo, de acuerdo con la aproximación presentada para cada uno de los subprocesos que lo componen. La unión de todas ellas hace que dispongamos de un entorno integrado de definición y mejora de flujos de trabajo.

La herramienta para la captura de requisitos del proceso de negocio permite la definición y almacenamiento de los casos de uso del negocio e incluye el proceso de generación automática de un modelo preliminar de flujos de trabajo. La herramienta de desarrollo de flujos de trabajo permite la especificación de modelos de flujo de trabajo y la generación de la especificación en distintos lenguajes destino. En concreto, en OASIS y en OPERA. En el primer caso, para poder validar dicha especificación en KAOS; y en el segundo caso para poder ejecutarla en un SGFT eficiente. La herramienta de análisis permite la construcción del almacén de datos definido y su carga a partir de los registros de ejecución de OPERA, quedando preparado para su explotación con DBMiner.

PARTE IV

Conclusiones

Capítulo 10.

Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones de la tesis. En la sección 10.1 se resumen las principales contribuciones al desarrollo de flujos de trabajo, de acuerdo con los objetivos iniciales. Se recopilan y resaltan los aspectos de mayor interés, puesto que en cada capítulo ya se han presentado los resultados más importantes como conclusiones parciales. En la sección 10.2 se relacionan futuras líneas de investigación que quedan abiertas y en las que se puede seguir trabajando. Finalmente, en la sección 10.3 se indican las publicaciones relacionadas con esta tesis.

10.1 Contribuciones

La principal contribución del trabajo realizado es la inclusión de aspectos metodológicos y de formalización en el desarrollo de flujos de trabajo, con la definición de un modelo de proceso que propone el uso de métodos, técnicas y herramientas para aumentar la calidad del flujo de trabajo construido. Se ha incidido principalmente en dos actividades: la **definición** del modelo de flujo de trabajo y la **mejora** del mismo. A continuación se resaltan las principales aportaciones en cada una de estas fases.

En la **definición del flujo de trabajo** se ha introducido una capa de Ingeniería de Requisitos que permite:

- Enriquecer el proceso de definición de flujos de trabajo con una etapa previa, que facilita la captura de requisitos de los procesos de negocio a implementar y obtiene automáticamente un modelo de flujo de trabajo. Se utilizan casos de uso del negocio para expresar los requisitos, los cuales se obtienen tras un proceso ascendente e iterativo, guiado por la estructura organizacional. Posteriormente se aplican un conjunto de patrones y transformaciones, basadas en las equivalencias establecidas entre los metamodelos de casos de uso y flujos de trabajo, para obtener una implementación del proceso de negocio en términos de tareas, recursos y flujo de control.
- Añadir una fase de prototipado automático basado en el uso de lenguajes formales de especificación, que permite la validación de modelos de flujo de trabajo antes de ser ejecutados en un SGFT. Concretamente, se ha utilizado OASIS como marco expresivo, el cual permite, por una parte, la formalización del metamodelo de flujo de trabajo definido, y por otra, la obtención automática de especificaciones equivalentes a los modelos construidos de acuerdo con dicho metamodelo. La semántica operacional de OASIS permite animar las especificaciones e iniciar un proceso de validación.

En la **mejora del flujo de trabajo**, se ha propuesto un marco general para abordar el análisis de ejecuciones de flujo de trabajo, basado en la utilización de técnicas de Extracción de Conocimiento, que han permitido:

- Plantear una solución concreta al análisis de ejecuciones, basada en *OLAP-Mining*, que permite obtener información de interés para mejorar los modelos de flujo de trabajo que se están ejecutando en un SGFT. Se parte de información real de ejecución que es almacenada por el propio SGFT y analizada posteriormente. Por una parte, se obtiene un conjunto de informes que dan cuenta de tiempos de ejecución de las tareas, de utilización de recursos, etc. , y por otra, se permite la búsqueda de asociaciones entre componentes del flujo de trabajo. Todo ello permite la detección de cuellos de botella y mejora del rendimiento de los modelos de flujo de trabajo que se están ejecutando.

- Definir un modelo de datos multidimensional para abordar el análisis de ejecuciones de flujos de trabajo, determinando las dimensiones y medidas de interés en dicho análisis, así como las jerarquías de agregación definidas en cada dimensión, en base al metamodelo de referencia.

Para completar el modelo de proceso de desarrollo propuesto, de modo que abarque no sólo la definición y mejora de flujos de trabajo, sino también su **ejecución**, se han utilizado técnicas de generación de código basadas en modelos. Esto ha permitido incluir una fase de implementación que enlaza la definición de modelos de flujo de trabajo realizada en base a

las actividades anteriores, con su ejecución en un SGFT. En esta fase, un compilador de modelos transforman el modelo de flujo de trabajo definido para que pueda ser ejecutado en el SGFT destino.

Todas las ideas anteriores se han plasmado en la implementación de un prototipo que consta de un conjunto de herramientas de soporte a las distintas actividades del proceso de desarrollo propuesto.

Finalmente, hay que destacar la definición de un metamodelo de flujo de trabajo de referencia que establece el punto de unión entre las fases de definición, ejecución y mejora. Este metamodelo está de acuerdo con los estándares de la WfMC y es lo suficientemente expresivo para permitir generalizar y aplicar el trabajo realizado en esta tesis a la mayoría de los SGFT existentes en el mercado, los cuales no disponen de un soporte metodológico en estas fases.

10.2 Trabajos Futuros

El modelo de proceso de desarrollo de flujos de trabajo definido en esta tesis abre nuevas líneas de trabajo, tanto a nivel de continuación y consolidación del trabajo realizado, como a nivel de aplicación de las soluciones aportadas a otros ámbitos.

a) Extensión del Trabajo Desarrollado.

Existen varias líneas de trabajo en las que se puede continuar. Se pueden agrupar en extensiones del modelo de proceso de desarrollo propuesto desde un punto de vista global, y en extensiones de cada uno de los suprocesos de desarrollo. A continuación se detallan todas ellas.

El **modelo de proceso de desarrollo** se puede extender, desde un punto de vista global, de forma que proporcione un soporte explícito a:

- La fase de evaluación del proceso de negocio completándose así el modelo de proceso de desarrollo. Una aproximación a seguir es la generalización de las ideas presentadas para el análisis de ejecuciones de flujos de trabajo. Pero en este caso, en lugar de utilizar los registros de ejecución como fuentes de datos para iniciar el proceso de extracción de conocimiento, se utilicen los propios datos operacionales de la empresa. Los elementos de interés a analizar vendrán marcados por los propios objetivos de la organización. Para esta fase, dar un conjunto de informes preestablecidos es más complejo y está relacionado con la toma de decisiones.

- La evolución y reutilización de procesos en el marco del modelo propuesto. Este aspecto se puede abordar a partir de la formalización del metamodelo, y más concretamente, a partir del metanivel que proporciona OASIS.

También se pueden extender cada uno de los **subprocesos de desarrollo**, en concreto respecto a la captura de requisitos del proceso de negocio y derivación automática de un modelo de flujo de trabajo se puede incorporar:

- La utilización de herramientas cooperativas entre los miembros de la organización y los analistas que facilite el proceso iterativo y ascedente de captura de requisitos del proceso de negocio. En [Mac99] hay una propuesta concreta de construcción de escenarios basados en este tipo de herramientas. Otros trabajos directamente relacionados con las herramientas cooperativas son [Bor99, Bor01].
- La definición de algún tipo de relación de orden temporal entre los casos de uso del negocio para permitir la derivación automática de todo el flujo de control. Esta ordenación se puede establecer de forma similar a la definida en [Lei00].

Respecto a la definición del flujo de trabajo, se pueden incluir nuevas funcionalidades, entre las cuales destacamos:

- Inclusión de una mayor expresividad a la definición de recursos permitiendo no sólo la inclusión de propiedades constantes, sino también de propiedades variables. Esto supone permitir especificar su comportamiento y ejecutarlo posteriormente en el SGFT.
- Definición de métricas para medir la calidad de los modelos construidos, que permita comparar distintas representaciones de un mismo proceso de negocio.

Finalmente, en la fase de mejora del flujo de trabajo se pueden también incluir nuevas funcionalidades, entre las cuales destacamos:

- La obtención totalmente automatizada de un conjunto de informes preestablecidos, a partir del almacén de datos construidos.
- La posibilidad de modificar el diseño del almacén de datos, con la inclusión de nuevas dimensiones o medidas y generar automáticamente el resto de información.

b) Aplicación del Trabajo Desarrollado.

El trabajo desarrollado en esta tesis se ha centrado en un dominio de aplicación concreto, los SGFT, pero muchos de los resultados obtenidos son igualmente aplicables a otros campos relacionados con la tecnología de procesos. Esto permite incidir en distintas líneas, entre las podemos citar:

- Incorporación de todas las técnicas propuestas en SGFT comerciales, tales como SAP R/3 o MQSeries WF, de forma que aumenten sus prestaciones a nivel de definición y mejora. Además, la aplicación del trabajo realizado nos permitiría varias cosas. En relación con la fase de definición, una mayor comprobación de la expresividad del metamodelo propuesto para representar los distintos componentes del flujo de trabajo. Y respecto a la fase de mejora, la obtención de más datos reales de ejecución que permitan comprobar la capacidad de generación de informes a partir del modelo de datos multidimensional propuesto.

- Utilización del trabajo realizado en dominios concretos de aplicación centrado en el proceso. Un caso especialmente interesante es la definición de procesos interoganizacionales, en los que ciertas partes del mismo constituyen trozos del proceso de negocio seguido por un conjunto de organizaciones. La aproximación seguida en esta tesis es válida siempre que el proceso se pueda representar globalmente. Ejemplos concretos serían los entornos de comercio electrónico, en los en los que es importante definir correctamente el proceso que se sigue, independientemente de la plataforma en la que posteriormente se soporte. En [Laz01] se muestra un ejemplo de aplicación.

- Aplicación a Entornos de Ingeniería del Software Centrados en el Proceso (PSEE[67]) cuyo objetivo es integrar en un mismo entorno todo el subproceso de gestión y de producción de software, conocido como Proceso del Software [Der99]. Aplicar los resultados a un caso concreto de Proceso Software como es el *Proceso Unificado de Rational* (RUP), que incluye tanto actividades básicas relacionadas con la producción del software como con la gestión del mismo [Kru98, Jac99]. Algunas aproximaciones a este trabajo ya han sido realizadas en [Pen99a Pen99c], y propuestas similares las podemos consultar en [Ara99, Ara01]. De igual modo, se podría estudiar su aplicabilidad en otras metodologías ágiles como Programación Extrema (XP) [Bec99, Jef01] y propuestas similares [Fow01].

10.3 Publicaciones Relacionadas con la Tesis

En esta sección se indican los publicaciones directamente relacionadas con el trabajo realizado en esta tesis. Aparecen clasificadas en capítulos de libros, congresos internacionales, nacionales e informes técnicos. También se han incluido los proyectos final de carrera dirigidos en el marco de esta tesis.

[67] *Process-sensitive Software Engineering Environments*

- *Capítulos de Libro*

Canós, J.H., Penadés, M.C. *Sistemas de Flujo de Trabajo*. Ingeniería del Software y Bases de Datos. Tendencias Actuales. X Escuela de Verano de Informática. Colección Ciencia y Técnica nº 28 (ISBN 84-8427-077-7). Págs. 218-240. Ediciones de la Universidad de Castilla-La Mancha. Julio 2000.

- *Congresos Internacionales*

Canós, J.H., Penadés, M.C., Carsí, J.A. *From Software Process to Workflow Process: the Workflow Life Cycle*. In Proc. of International Process Technology Workshop (ITPW'99). Sponsored by ISPA (International Software Process Association), in cooperation with SIGSOFT. Grenoble (France), September 1999.

Penadés, M. C., Canós, J.H., Sánchez, J. *Automatic Derivation of Workflow Specifications from Organizational Structures and Use Cases*. In Proc. of Workshop on Requirements Engineering (WER2001). Pages 166-180. Buenos Aires (Argentina). November 2001.

- *Congresos Nacionales*

Penadés, M.C., Canós, J.H., Carsí, *Workflow Support to the Software Process*, In Proc. of IV Workshop MENHIR, pp. 22-26 , Sedano (Burgos), Mayo 1999.

Penadés, M.C., Canós, J.H., Carsí, J.A. *Hacia una Herramienta de Soporte al Proceso Software Basada en la Tecnología de Workflow*, En Actas de las IV Jornadas de Ingeniería del Software (JISBD'99), ISBN. 84-699-0956-8, pp. 381-392, Cáceres, Noviembre 1999.

Canós, J.H., Sánchez, J., Penadés, M.C., *Una Aproximación Ascendente al Modelado de Flujos de Trabajo*. En Actas de las V Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2000), ISBN. 84-8448-065-8, pp. 79-87, Valladolid, Noviembre 2000.

Penadés, M.C., Canós, J.H., *Un Entorno para el Desarrollo de Modelos de Flujo de Trabajo*, En Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2001), ISBN. 84-699-6275-2, pp. 223-236, Universidad de Castilla-La Mancha, Almagro (Ciudad Real), Noviembre 2001.

Penadés, M.C., Alonso, G., Canós, J.H., *Un Modelo de Almacén de Datos para Análisis de Ejecuciones de Flujos de Trabajo*. En Actas del I Taller de Almacenes de Datos y Tecnología (ADTO'2001), pp. 19-24, celebrado en el marco de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2001), Almagro (Ciudad Real), España, Noviembre, 2001.

- *Informes Técnicos*

Penadés, M.C., Canós, J.H., Ramos, I., *Modelado Conceptual de Flujos de Trabajo*, Informe técnico del Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, DSIC-II/8/00, Abril 2000.

Penadés, M.C., Canós, J.H., Sánchez, J., *Automatic Derivation of Workflow Specifications from Use Case Models,* Informe técnico del Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, DSIC-II / 4 / 01, Marzo 2001.

Penadés, M.C., Canós, J.H., Alonso, G., *Desarrollo de Modelos de Flujos de Trabajo. Un Caso Práctico*, Informe técnico del Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, DSIC-II / 2 / 02, Enero 2002.

- *Proyectos Final de Carrera*

Moreno, C., Cantó, J., Carmona, J.M.. *Construcción de una Interfaz Gráfica para el Modelado de Flujos de Trabajo,* Proyecto Final de Carrera, Escuela Universitaria de Informática, Universidad Politécnica de Valencia, Septiembre 1999.

Ceballos, D., Olmeda, F., *Diseño e Implementación de un Generador de Modelos de Workflow a partir de Casos de Uso*, Proyecto Final de Carrera, Escuela Universitaria de Informática, Universidad Politécnica de Valencia, Junio, 2000.

Segura, M.P., *Representación Gráfica de Flujos de Trabajo y Generación Automática de Código en R-OASIS*. Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Septiembre 2000.

Franco, P., *Diseño y Construcción de una Herramienta para el Análisis de Ejecuciones de Flujos de Trabajo*, Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Junio, 2001.

Salesa, I., *Generación Automática de Modelos de Flujo de Trabajo,* Proyecto Final de Carrera. Facultad de Informática. Universidad Politécnica de Valencia. (en realización)

Bibliografía

[Agr93a] Agrawal, R., Imielinski, T., Swami, A., *Mining Association Rules between Sets of Items in Large Databases*, ACM SIGMOD Conference on Management Data, pp. 207-216, Washington D.C., May 1993.

[Agr93b] Agrawal, R., Imielinski, T., Swami, A., *Database Mining: A Performance Perspective*, IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowlegde-Based Databases, Vol. 5, No. 6, pp. 914-925, December 1993.

[Agr86] Agresti, W.W. *'Tutorial: New Paradigms for Software Development'*. LOS ANGELES.CA. IEEE Computer Society Press, 1986.

[Alo97] Alonso, G., Agrawal, D., El Abbadi, A., Mohan, C., *Funcionality and Limitations of current workflow Management Systems*, IEEE-Expert, 1997.

[Alo98] Alonso, G., Mohan, C., *Workflow Management Systems: The next generation of distributed processing tools*, In Advanced Transaction Models and Architectures, S. Jajodia and L. Kerschberg (Eds.), Kluwer Academic Publishers, pp. 35--62, 1997.

[Ara99] Araujo, R., Borges, M., *Sobre a Aplicabilidade de Sistemas de Workflow no Suporte a Processos de Software*. In Proc. of Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software (IDEAS 1999), Costa Rica, 1999.

[Ara01] Araujo, R., Borges, M., *Extending the Software Process Culture – An Approach Based on Groupware and Workflow*, In Proc. of 3rd International Conference on Product Focused Software Process Improvement (PROFES 2001), pp. 297-311, Lecture Notes in Computer Science, Kaiserslautern, Germany, September 2001.

[ARB] Arbor. Essbase OLAP Server. Hyperion Solutions Corporation http://www.hyperion.com

[Awa96] Awad, M., Kuusela J., Ziegler, J., *Object Oriented Technology for Real-Time Systems: A practical approach using OMT and Fusion*, Prentice Hall, 1996.

[Bal83] Balzer, R., Cheatham, T.E., Green, C., *Software Technology in the 1990's : using a new paradigm*, IEEE Computer, 16(11): 39-45, 1983.

[Bar99] Baresi, L., Castano, F., Ceri, S. et al., *Wide Workflow Development Methodology*. In Proc. of International Joint Conference on Work Activities Coordination and Collaboration (WACC'99), pp. 19-28, San Francisco, CA, USA, February 1999.

[Bec99] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[Bell98] Bell, R., *Code Generation from Object Models*, Embedded Systems Programming, March 1998. http://www.embedded.com/98/9803fe3.html

[BIZ] Bizflow 2000. Handysoft. http://www.handysoft.com/products/products.asp

[Boo99] Booch, G., Rumbaugh, J., Jacobson, I., *El Lenguaje de Modelado Unificado. Guía de usuario*, Ed. Addison-Wesley, 1999.

[Bon95] Bonner, A.J., Kifer, M., *Transaction Logic Programming*, Technical Report CSRI-323, Noviembre 1995. ftp:/csri.toronto.edu/csri-technical-reports/323/report.ps.z

[Bor99] Borges, M., Pino, J., *PAWS: Towards a Participatory Approach to Business Process Reengineering*, In Proc. of the International Workshop on Groupware, pp. 262-268, IEEE Computer Society, Cancun, Mexico, September 1999.

[Bor01] Borges, M., Mendes, R., Cerqueira, B., *Bridging the gap between organizations and their software processes – An approach based on patterns and workflow systems* Technical Report, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, 2001.

[Box00] Box, D., Skonnard, A., Lam, J., *Essential XML*, Addison-Wesley, 2000.

[Can95] Canós, J. H., Penadés, M.C., Ramos, I., Pastor, O., *A knowledge-base architecture for object societies*, In Proc. of Database and Expert Systems Applications Workshop (DEXA'95), pp. 19-24 , OMNIPRESS, England, 1995.

[Can96a] Canós, J.H., *OASIS como lenguaje único para Bases de Datos Orientadas a Objetos*, Tesis Doctoral, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1996.

[Can96b] Canós, J.H., Penadés, M.C., Carsí J.A., Bonet, B. *The R-OASIS Language.* Informe Técnico del Departamento de Sistemas Informáticos y Computación, II-DSIC-26/96, 1996.

[Can98] Canós, J.H., Jaén, Fco. J., Ramos, I., *Towards Requirements Engineering of Active Database Systems.* In Proc. of Workshop on Requirements Engineering (WER'98), Brasil, 1998.

[Can99] Canós, J.H., Penadés, M.C., Carsí, J.A., *From software processes to workflow processes: the workflow lifecycle*, In Proc. of International Software Technology Workshop (IPTW'99), Grenoble, Francia, 1999.

[Can00a] Canós, J.H., Penadés, M. C., *Sistemas de Gestión de Flujos de Trabajo*. Ingeniería del Software y Bases de Datos. Tendencias Actuales, SP-UCLM Colección Ciencia y Técnica nº 28, pág. 218-240 (ISBN: 84-8427-077-7) , Julio 2000

[Can00b] Canós, J.H., Sánchez, J., Penadés, M.C., *Una Aproximación Ascendente al Modelado de Flujos de Trabajo*. En Actas de las V Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2000), pp. 79-87, Valladolid, España, Noviembre 2000.

[Car94] Carsí, J.A., *Diseño e Implementación del Núcleo de un SGBDDAOO Basado en OASIS*, Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Julio, 1994.

[Car96] Carsí, J.A., Penadés, M.C., Canós, J.H., Bonet, B., *KAOS User Manual*. Informe Técnico del Departamento de Sistemas Informáticos y Computación, II-DSIC-27/96, 1996.

[Car99] Carsí, J.A., *OASIS como Marco Conceptual para la Evolución del Software*, Tesis Doctoral, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1999.

[Cas95] Casati, F., Ceri, S., Pernici, B., Pozzi, G., *Conceptual Modeling of Workflows*. In Proc. of the International Conference of Conceptual Modeling (ER), Gold Coast, Australia, Lecture Notes in Computer Science, Springer Verlag, December 1995.

[Cat01] Castro, J., Kolp, M., Mylopoulos, J., *A Requirements-Driven Development Methodology*, In Proceedings of 13th Conference on Advanced Information Systems Engineering (CAISE´01), Interlaken, Switzerland, June 2001.

[Ceb00] Ceballos, D., Olmeda, F., *Diseño e Implementación de un Generador de Modelos de Workflow a partir de Casos de Uso*, Proyecto Final de Carrera, Escuela Universitaria de Informática, Universidad Politécnica de Valencia, Junio, 2000.

[CHA] CHAIMS: Compiling High-level Access Interfaces for Multi-site Software. http://www-db.stanford.edu/chaims/

[Cod93] Codd, E.F., Codd, S.B., Salley, C.T. *Beyond Decision Support*, Computer World, 27, July, 1993.

[Col94] Coleman D., et al, *Object Oriented Development: The Fusion Method*, Prentice Hall, 1994.

[Con99] Constantine, L., Lockwood, L., *Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design*. Addison-Wesley, 1999.

[COS] COSA Workflow. COSA Solutions. http://www.cosa.nl

[Cur92] Curtis, B., Kellner, M.I., Over, J., *Process Modelling*. Communications of the ACM, 35(9):75-90, September, 1992.

[DBM] DBMiner. http://www.dbminer.com

[DEL] Borland Delphi. http://www.inprise.com

[Der99] Derniame, J.C. et al. (ed.), *Software Process: Principles, Methodology, and Technology*, Lecture Notes in Computer Science 1500, Springer-Verlag, 1999.

[DeW91] De Witt, D. *et al.*, *El Manifiesto del Sistema de Base de Datos Orientado al Objeto*, Novática Vol. XVII, núm 91, 1991.

[DOL] DOLPHIN. Fujitsu. http://www.fnc.fujitsu.com

[Dub93] Dubois, E., Du Bois, P., Petit, M., *O-O Requirements Analysis : An Agent Perspective*. In Proc. of the 7th European Conference on Object Oriented Programming (ECOOP'93), pp. 458-481, 1993.

[Dub94] Dubois, E., Du Bois, P., Dubru, F., *Animating Formal Requirements Specifications of Cooperative Information Systems.* In Proc. Of the Second International Conference on Cooperative Information Systems, pages 101-112, Toronto, May 1994.

[EAS] eiStream. Eastman Software Enterprise. http://www.eastmansoftware.com/products/

[Eva99] Evans, A., Kent, S., *Core Meta-Modelling Semantics of UML: The pUML Approach*, UML'99- The Unified Modeling Language, In Proc. of "UML'99" The Unifield Modeling Language: Beyond the Standard - The Second Conference, Eds. Robert France and Bernhard Rumpe, Fort Collins, CO, USA. pp. 140-150. October, 1999.

[EXO] Proyecto EXOTICA. http://www.almaden.ibm.com/cs/exotica

[Fay96] Fayyand, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. , *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996.

[Fee93] Feenstra, R.B., Wieringa, R.J., *LCM 3.0: A Language for Describing Conceptual Models*, Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1993.

[Fer00] Fernández, J.L., Toval, A., Hoyos, J.R., *Rigorously Transforming UML Class Diagrams*, En Actas de las V Jornadas de Ingeniería del Software y Base de Datos (JISBD 2000), pp. 265-277, Valladolid, España, Noviembre, 2000.

[FIL] Visual and Panagon Workflow. FileNet Corporation USA. http://www.filenet.com/English/Products/index.asp

[Fow01] Fowler, M., *The New Methodology*, November 2001. http://www.martinfowler.com/articles/newMethodology.html

[Fra01] Franco, P., *Diseño y Construcción de una Herramienta para el Análisis de Ejecuciones de Flujos de Trabajo*, Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Junio, 2001.

[Geo95] Georgakopoulos, D., Hornick, M., Sheth, A., *An Overview of Workflow Managenement: from Process Modeling to Workflow Automation Infraestructure*, Distributed and Parallel Databases. Vol.3, n.2, April 1995.

[Gep97] Geppert, A. and Tombros, D., *Logging and Post-Mortem Analysis of Workflow Executions based on Event Histories*, In Proc. of 3rd International Conference on Rules in Database Systems (RIDS'97), pp 67-82, LNCS 1312, Springer Verlag, Heidelberg, Germany, 1997.

[Gep98] Geppert, A., Kradolfer, M., Tombros, D., *Workflow Specification and Event-Driven Workflow Execution*, Workflow Management Systems, Informatik.Informatique 1, pp. 1-6, 1998.

[Gra95] Gray, J., et al., *Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab and Sub-totals*, Technical Report MSR-TR-95-22, Microsoft Research, November, 1995.

[Gra97] Gray, J., et al., *Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab and Sub-totals*, Data Mining and Knowledge Discovery, 1:29-54, 1997.

[Hag99] Hagen, C. J., *A Generic Kernel for Reliable Process Support*, Dissertation. ETH Nr. 13114, ETH Zurich, Switzerland, 1999. http://www.inf.ethz.ch/department/IS/iks/research/opera

[Han97] Han, J., *OLAP-Mining: An Integration of OLAP with Data Mining*, In Proc. IFIP Conference on Data Semantics, pp. 1-11, Leysin, Switzerland, October, 1997.

[Han98] Han, J., *Towards On-Line Analytical Mining in Large Databases*, ACM SIGMOD Record, 27:97-107, 1998.

[Han01] Han, J., Kamber, M., *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.

[Har84] Harel, D., *Dynamic Logic*, In Handbook of Philosophical Logic II, D.M. Gabbay, F. Guenter Editors, pp. 497-694, Reidel, 1984.

[Har93] Hartmann, T., Jungclaus, R., Saake, G., *Animation Support for a Conceptual Modelling Language*, In Proc. of the 4th International Conference on Database and Expert Systems Applications (DEXA'93), Lectures Notes in Computer Science 720, pp. 56-67, Springer-Verlag, Prague, September 1993.

[Hol95] Hollingsworth, D., *The Workflow Reference Model*, Technical report TC00-1003, WfMC, January, 1995. http://www.wfmc.org/

[Inm96] Inmon, W.H, *Building the Data Warehouse*, Jonh Wiley & Sons, 1996.

[Jac92] Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G., *Object Oriented Software Engineering. A Use Case Driven Approach*, Reading, Massachusetts, Addison –Wesley, 1992.

[Jac99] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*. Addison-Wesley, 1999.

[Jef01] Jeffries, R., *What is Extreme Programming?*, XP Magazine, November 2001. http://www.xprogramming.com/xpmag/whatisXP.htm

[Jun95] Jungclaus, R., Saake, G., Hartmann, T., Sernadas, C., *TROLL – A Language for Object-Oriented Specification of Information Systems*, ACM Transactions on Information Systems, Vol. 14, No. 2, pp. 175-211, April 1995.

[Kam97] Kamber, M., Han, J., Chiang, J.Y., *Metarule-guided Mining of Multidimensional Association Rules using Data Cubes*, In Proc. of 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97), pp. 207-210, Newport Beach, California, 1997.

[Koh00] Köhler, D., *Extended Document Object Model.* http://www.philo.de/xml/

[Kru98] Krutchen, P., *The Rational Unified Process- An Introduction*. Addison –Wesley, 1998.

[Laz01] Lazcano, A., Schuldt, H., Alonso, G., Schek, H.J., *WISE: Process based E-Commerce*, In IEEE Data Engineering Bulletin, Special Issue on Infrastructure for Advanced E-Services, Vol. 24, No. 1, March 2001.

[Lar98] Larman C., *UML and Patterns*. Addison Wesley, 1998.

[Lei00] Leite, J.C, Hadad, G., Doorn, J.H., Kaplan, G. *A Scenario Construction Process*. Requirements Engineering Vol. 5 (2000) pp. 38-61, Springer-Verlag, 2000.

[Let98] Letelier, P., Ramos, I., Sánchez, P., Pastor, O., *OASIS Versión 3.0. Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*, Servicio de Publicaciones de la Universidad Politécnica de Valencia, SP-UPV 98.4011, 1998.

[Let99] Letelier, P., *Animación Automática de Especificaciones OASIS utilizando Programación Lógica Concurrente*, Tesis Doctoral. Departamento de Sistemas Informáticos y Computación. Universitat Politécnica de Valencia, 1999.

[Ley98] Leymam, F., *Production Workflow Systems*, Tutorial in 6th International Conference on Extending Database Technology (EDBT'98), Valencia, España, March 1998.

[Ley00] Leymam, F., Roller, D., *Production Workflow. Concepts and Techniques*, Prentice Hall, 2000.

[LOT] Lotus Notes. http://www.lotus.com/home.nsf/tabs/lotusnotes

[Mac99] Machado, M., Santos, F., Werner, C., Borges, M., *Uma infra-estrutura de Apoio à Aquisiçao Cooperativa de Conhecimento em Engenharia de Domínio,* In Proc. of XIII Simposio Brasileiro de Engenharia de Software (SBES'99), Florianopolis (Brasil), Outubro, 1999.

[Man91] Manley, J., et al., *KBMS1 a User Manual (Version 3)*, HP-Labs. Internal Report HPL-91-178, Bristol, 1991.

[Meg94] Megino, E., *Implementación de un modelo orientado a objetos en una arquitectura de sistemas de gestión de bases de conocimiento*, Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Octubre 1994.

[MEN] Proyecto MENTOR: Middleware for Enterprise-wide Workflow Management. http://www-dbs.cs.uni-sb.de/~mentor/

[MET] Proyecto METEOR. http//lsdis.cs.uga.edu/proj/meteor/meteor.html

[MIC] Microstrategy. http//www.microstrategy.com

[MOB] Proyecto MOBILE. http://www6.informatik.uni-erlangen.de/research/wfm.html

[MQS] IBM MQSeries Workflow. http://www.redbooks.ibm.com

[MSE] Microsoft Excel 2000. http://www.microsoft.com/office/excel/default.asp

[MSQ] Microsoft SQL Server 7.0 OLAP Service. http://www.microsoft.com/sql/default.asp

[Obl99] OBLOG Software S.A. The OBLOG Software Development Approach (White Paper), 1999. http://www.oblog.pt

[OMG97] Object Management Group. *Workflow Management Facility: Request for Proposal*. Technical Report cf/97-05-06. July 1997. http//www.omg.org/library/

[OMG98] Object Management Group. *Workflow Management Facility Specification*. Revised Submission bon/98-06-07, July 1998. http//www.omg.org/library/

[OMG00] Object Management Group. *Workflow Management Facility Specification, v1.2*. Technical Report April, 2000 http//www.omg.org/library/

[OPE] Proyecto OPERA. http://www.inf.ethz.ch/department/IS/iks/research/opera.html

[ORA] Oracle Express. Oracle Corporation. http://technet.oracle.com/docs/products/express/

[Pas92] Pastor O., *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el Modelo Orientado a Objetos*, Tesis Doctoral, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1992.

[Pas95a] Pastor, O., Ramos, I. *OASIS version 2 (2.2): A Class-Definition Language to Model Information Systems using an Object-Oriented Approach*, Servicio de Publicaciones de la Universidad Politécnica de Valencia, SPUPV-95.788, 1995.

[Pas95b] Pastor, O., Ramos, I., Canós, J.H. *Oasis v2: A Class Definition Languag*, In Proc. of Database and Expert Systems Applications (DEXA'95), pp. 79-91, Lecture Notes in Computer Science 978, Springer-Verlag, 1995.

[Pen94] Penadés, M. C., *Actualizaciones, Consultas y Programas en un sistema Gestor de Bases de Datos Orientadas a Objetos*, Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Diciembre 1994.

[Pen99a] Penadés, M.C., Canós, J.H., Carsí, J.A., *Workflow Support to the Software Process*, In Proc. of IV Workshop MENHIR, pp22-26, Sedano (Burgos), Mayo, 1999.

[Pen99b] Penadés, M.C., *Modelado de Flujos de Trabajo utilizando OASIS*. Trabajo de investigación (programa 3er ciclo), Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Septiembre 1999.

[Pen99c] Penadés, M.C., Canós, J.H., Carsí, J.A. *Hacia una Herramienta de Soporte al Proceso Software Basada en la Tecnología de Workflow*, En Actas de las IV Jornadas de Ingeniería del Software (JISBD'99), pp. 381-392, Universidad de Extremadura, Noviembre 1999.

[Pen00] Penadés, M.C., Canós, J.H. *Modelado Conceptual de Flujos de Trabajo*, Informe técnico del Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, DSIC-II/8/00, Abril 2000.

[Pen01a] Penadés, M.C., Canós, J.H., *Un Entorno para el Desarrollo de Modelos de Flujo de Trabajo*, En Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2001), pp. 223-236, Universidad de Castilla-La Mancha, Almagro (Ciudad Real), España, Noviembre 2001.

[Pen01b] Penadés, M.C., Alonso, G., Canós, J.H., *Un Modelo de Almacén de Datos para Análisis de Ejecuciones de Flujos de Trabajo*. En Actas del I Taller de Almacenes de Datos y Tecnología (ADTO'2001), pp. 19-24, celebrado en el marco de las VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2001), Almagro (Ciudad Real), España, Noviembre, 2001.

[Pen01c] Penadés, M.C., Canós, J.H., Sánchez, J., *Automatic Derivation of Workflow Specifications from Organizational Structures and Use Cases*. In Proc. of IV Workshop on Requirements Engineering (WER2001), pp. 166-180, Buenos Aires (Argentina), Noviembre, 2001.

[Pen02] Penadés, M.C., Canós, J.H., Alonso, G., *Desarrollo de Modelos de Flujos de Trabajo. Un Caso Práctico*, Informe técnico del Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, DSIC-II/2/02, Enero 2002.

[Pre01] Pressman, R.S., *Ingeniería del Software: un enfoque práctico 5ª ed.*, McGraw-Hill, 2001.

[PUM] The Precise UML Group. http://www.cs.york.ac.uk/puml/

[Ram93] Ramos, I. et. al., *Objects as Observable Processes*, In Proc. of the 4th International Workshop on the Deductive Approach to Information Systems and Databases, Tech. Report, DLSI, Universitat Politécnica de Catalunya, 1993.

[Ram95a] Ramos I., Pelechano V., Penadés M.C., Bonet B., Canós J.H., Pastor O., *Análisis y Diseño Orientado a Objetos de un Entorno de Prototipación Automática*, Informe Técnico del Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, II-DSIC-09/95, Julio 1995.

[Ram95b] Ramos I., Pelechano V., Penadés M.C., Bonet B., Canós J.H., Pastor O., *Especificación en R-OASIS de un Entorno de Prototipación Automática*, Informe Técnico del Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, II-DSIC-10/95, Julio 1995.

[Rol96] Rolland, C., Ben Achour, C., Cauvet, C., Ralyé, J., Sutcliffe, A., Maiden, N., Jarke, M., Haumer, P., Polh, K., Dubois, E., Heymans, P., *A Proposal for a Scenario Classification Framework*, Technical Report CREWS 96-01, Cooperative Requirements Engineering with Scenarios. http://sunsite.informatik.rwth-aachen.de/CREWS/

[SAP] SAP AG. http://www.sap.com

[Sad99] Sadiq, W., Orlowska, M., *On Capturing Process Requirements of Worflow Based Business Information Systems*, In Proc. of 3d International Conference on Business Information Systems (BIS'99), pp. 195-209, Springer-Verlag, Poznan (Poland), April 1999. http://staff.dstc.edu.au/wasim

[Sal02] Salesa, I., *Generación Automática de Modelos de Flujo de Trabajo,* Proyecto Final de Carrera. Facultad de Informática. Universidad Politécnica de Valencia. (en realización)

[San00] Sánchez, P., *Animación de Especificaciones OASIS mediante Redes de Petri Orientadas a Objetos*, Tesis Doctoral, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2000.

[Seg00] Segura, M.P., *Representación Gráfica de Flujos de Trabajo y Generación Automática de Código en R-OASIS*. Proyecto Final de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Septiembre 2000.

[SER] SERFloware. SER. http://www.ser.de/en/products/products-main.html

[She96] Sheth, A. et al., *Report from the NSF Workshop on workflow and Process Automation in Information Systems*. Computer Science Department Technical Report, UGA-CS-TR-96-003, University of Georgia, October 1996. http://lsdis.cs.uga.edu/activities/NSF-workflow/

[Sid97] Siddiqi, J., Morrey, I.C., Roast, C.R., Ozcan, M.B., *Towards Quality Requirements via Animated Formal Specifications,* Annals of Sotware Engineering, n.3, 1997.

[STA] Staffware Workflow. Staffware Corporation. http://www.staffware.com

[TEM] TeamWare Flow. Fujitsu. http://www.fnc.fujitsu.com

[TIB] TIB/InConcert. TIBCO. http.//www.tibco.com/products/in_concert/

[War99] Warmer, J.B., Kleppe, G.A., *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley Object Technology Series, March 1999.

[Wie93] Wieringa, R.J. et al., *Actors, Actions and Initiate in Normative System Specification*, Annals of Mathematics and Artificial Intelligence, vol. 7, pp. 289-346, 1993.

[W3C98] W3C, *Extensible Markup Language 1.0 (XML)*, REC-xml-19980210, version 1.0, February, 1998. http://www.w3.org/.

[WFM95] Workflow Management Coalition Members, *WAPI-Naming Conventions.* Technical report WfMC-TC-1013, WfMC, November, 1995. http://www.wfmc.org/.

[WFM98a] Workflow Management Coalition Members, *Workflow Client API Specifications (WAPI).* Technical report WfMC-TC-1002, WfMC, July, 1998. http://www.wfmc.org/.

[WFM98b] Workflow Management Coalition Members, *Workflow Audit Data Specifications.* Technical report WfMC-TC-1015, WfMC, September, 1998. http://www.wfmc.org/.

[WFM99a] Workflow Management Coalition Members, *Terminology & Glosary*. Technical report WfMC-TC-1011, WfMC, February, 1999. http://www.wfmc.org/.

[WFM99b] Workflow Management Coalition Members, *Workflow Standard-Process Definition MetaModel & WPDL*. Technical report WFMC-TC-1016, WfMC, October, 1999. http://www.wfmc.org/.

[WFM99c] Workflow Management Coalition Members, *Workflow Standard-Interoperability. Abstract Specification*. Technical report WFMC-TC-1012, WfMC, December, 1999. http://www.wfmc.org/.

[WFM00a] Workflow Management Coalition Members, *Workflow Standard-Interoperability. Internet e-mail MIME Binding.* Technical report WFMC-TC-1018, WfMC, January, 2000. http://www.wfmc.org/.

[WFM00b] Workflow Management Coalition Members, *Workflow Standard-Interoperability. Wf-XML Binding.* Technical report WFMC-TC-1023, Draft, WfMC, January, 2000. http://www.wfmc.org/.

[WFM01] Workflow Management Coalition Members, *Workflow Process Definition Interface- XML Process Definition Language.* Technical report WfMC-TC-1025, Draft, WfMC, May, 2001. http://www.wfmc.org/.

[WID] Proyecto WIDE. . http://dis.sema.es/projects/WIDE/

[Yu96] Yu, E., Mylopoulos, J., Lespérance, Y., *AI Models for Business Process Reengineering*, IEEE Expert, Vol. 11, No. 4, pp. 16-23, August, 1996.

Apéndices

Apéndice A.

Especificación R-OASIS del Metamodelo de Flujos de Trabajo

En este apéndice se incluye la especificación R-OASIS completa del metamodelo de flujo de trabajo de referencia, correspondiente a la formalización vista en el capítulo 5. Las palabras en negrita se corresponden con palabras reservadas del lenguaje y se utiliza el símbolo / / para los comentarios.

// Especificación R-OASIS

```
conceptual schema  WF_model

//CLASE PROCESS
complex_class process aggregation_of
   activity(inclusiva,static,multivalued,disjoint,strict,not null),
   process(relational,static,multivalued,nodisjoint,flexible,null).

constant_attributes
      key    identifier(string);
       name(string);
       description(string).

variable_attributes
     state(string).
       valuation
         [start]     state(create).
         [run]       state(running).
         [end_run]   state(executed).
```

```
        [finish]    state(finished).
        [terminate] state(terminated).
        [kill]     state(dead).
      end_valuation

 private_events
     new     start;
   destroy   stop;
             run;
             end_run;
             finish;
             terminate;
             kill.

processes
      process <- start & body & stop.
      body <-  (check(start_condition) &  run  &  actions  &  end_run  &
               (check(end_condition) & finish  or terminate)) or kill.
      actions.
end_class


//CLASE ACTIVITY
class activity
constant_attributes
 key   identifier(string);
       name(string);
       description(string).

variable_attributes
   state(string).
      valuation
        [start]    state(create).
        [run]      state(running).
        [end_run] state(executed).
        [finish]  state(finished).
        [terminate] state(terminated).
        [kill]     state(dead).
      end_valuation

 private_events
     new      start;
   destroy    stop;
              run;
              end_run;
              finish;
              terminate;
              kill.

 processes
     activity <- start & body & stop.
     body <-  (check(start_condition) &  run  &  actions  &  end_run  &
              (check(end_condition) & finish   or  terminate)) or kill.
     actions.
end_complex_class
```

```
//CLASE MANUAL_ACTIVITY
complex_class manual_activity specialization_of activity

end_complex_class



//CLASE AUTOMATED_ACTIVITY
complex_class automatic_activity specialization_of activity

end_complex_class



//CLASE APPLICATION
class application
  constant_attributes
     key   identifier(string);
           app_name(string);
           execution_params(string);
           path(string);
           data(list);
           call(string).

  variable_attributes
    state(string)
       valuation    R,Call: string.
              [new_app] state(active).
              [invoke_app(Call,Data,Result)] state(running).
              [result_app(Result)] state(executed).
         end_valuation
    return_value(string)
      valuation    R,Call: string.
              [new_app] return_value(‘’).
              [result_app(Result)] return_value(Result).
       end_valuation

private_events   Call, Result:string. Data:list;
        new       new_app;
       destroy   destroy_app;
                  invoke_app(Call,Data,Result);
                  result_app(Result);
                  end_app;

processes
      application <- new_app & application1.
       application1<- call(Call) & data(Data) &
                      invoke_app(Call,Data,Result) & result_app(Result) &
                        end_app & application2.
       application2<- application1 or destroy_app.

end_class
```

```
//CLASE DATA
class data
constant_attributes
    key  identifier(string).

private_events
   new     new_data;
  destroy  destroy_data.

processes
       data <- new_data & destroy_data.
end_class



//CLASE USER
class user

private_events
   new     new_user;
  destroy  destroy_user;

processes
       user <- new_user & destroy_user.
end_class



//CLASE ORGANISATIONAL_UNIT
complex_class organisational_unit aggregation_of
    user(relacional,dynamic,multivalued,nodisjoint,strict,not null),
    organisational_unit(relacional,static,multivalued,disjoint,
                        flexible, null).

private_events
      new     new_orgUnit;
   destroy    destroy_orgUnit;

processes
       organisational_unit <- new_orgUnit & destroy_orgUnit.
end_complex_class



//CLASE ACTOR
complex_class actor  generalization_of  user, organisational_unit.
constant_attributes
    key  identifier(string);
         name(string);
         description(string).
end_complex_class
```

```
//RELACIÓN INVOKES
complex_class invokesAppAAc aggregation_of
    automated_activity(relacional,static,univalued,disjoint,flexible,not
    null),
   application (relacional,static,univalued,nodisjoint,strict,not null).

constant attributes
  key   identifier(string);
        name(string);
        description(string).

private_events
     new      new_invokesAppAAc;
    destroy   destroy_ invokesAppAAc;

processes
       invokesAppAAc <- new_ invokesAppAAc & destroy_ invokesAppAAc.
end_complex_class


// RELACIÓN USES
complex_class usesDataAc aggregation_of
    activity (relacional,static,univalued,nodisjoint,flexible,not null),
    data (relacional,static,multivalued,nodisjoint,flexible,not null).

constant attributes
  key   identifier(string);
        name(string);
        description(string).

private_events
    new       new_usesDataAc;
   destroy    destroy_usesDataAc;

processes
       usesDataAc <- new_usesDataAc & destroy_usesDataAc.
end_complex_class


// RELACIÓN RETURN
complex_class returnDataAc aggregation_of
    activity (relacional,static,univalued,nodisjoint,flexible,not null),
    data (relacional,static,multivalued,nodisjoint,flexible,not null).

constant attributes
  key   identifier(string);
        name(string);
        description(string).

private_events
    new       new_returnDataAc;
   destroy    destroy_ returnDataAc;
processes
       returnDataAc <- new_returnDataAc & destroy_returnDataAc.
end_complex_class
```

```
//RELACIÓN MAY_HAVE
complex_class hasAcActor aggregation_of
    activity (relacional,static,univalued,disjoint,flexible,not null),
    actor (relacional,static,multivalued,nodisjoint,flexible,null).

constant attributes
  key identifier(string);
      name(string);
      description(string).

private_events
    new       new_hasAcActor;
  destroy     destroy_hasAcActor;

processes
      hasAcActor <- new_hasAcActor & destroy_hasAcActor.
end_complex_class


//RELACIÓN HAS
complex_class hasManualAcActor aggregation_of
   manual_activity (relacional,static,univalued,disjoint,flexible,not null),
   actor (relacional,static,multivalued,nodisjoint,flexible,not null).

constant attributes
  key identifier(string);
      name(string);
      description(string).

private_events
    new       new_hasAcActor;
  destroy     destroy_hasAcActor;

processes
      hasAcActor <- new_hasAcActor & destroy_hasAcActor.
end_complex_class


// RELACIÓN RESPONSIBLE
complex_class hasProcessActor aggregation_of
   process (relacional, static, univalued, disjoint, flexible, null),
   actor (relacional, static, univalued, nodisjoint, flexible, null).

constant attributes
  key   identifier(string);
        name(string);
        description(string).

private_events
   new new_hasProcessActor;
  estroy      destroy_hasProcessActor;

processes
      hasProcessActor <- new_hasProcessActor & destroy_hasProcessActor.
end_complex_class
```

```
//RELACIÓN MANAGED_BY
complex_class managedOrgUnitUser aggregation_of
   organisational_unit (relacional,static,multivalued,nodisjoint,flexible,
                        not null),
   user(relacional,static,univalued,disjoint,flexible,null).

constant attributes
  key identifier(string);
      name(string);
      description(string).

private_events
    new       new_managedOrgUnitUser;
  destroy     destroy_managedOrgUnitUser;

processes
   managedOrgUnitUser <-new_managedOrgUnitUser & destroy_managedOrgUnitUser.
end_complex_class


//RELACIÓN SUBSTITUTE
complex_class substituteUser aggregation_of
   user(relacional,static,multivalued,nodisjoint,flexible,null),
   user(relacional,static,munivalued,nodisjoint,flexible,null).

constant attributes
  key identifier(string);
      name(string);
      description(string).

private_events
    new       new_substituteUser;
  destroy     destroy_substituteUser;

processes
   substituteUser <-new_substituteUser & destroy_substituteUser.
end_complex_class

end_conceptual_schema
```

Apéndice B.

Servicios del Editor de Modelos de Flujos de Trabajo

En este apéndice se describen todos los servicios proporcionados por el editor de modelos de flujo de trabajo, de acuerdo con el metamodelo de referencia definido en el capítulo 4. Los servicios están agrupados en tres bloques según su funcionalidad. Se distingue entre gestión del flujo de trabajo, gestión de las tareas de un proceso y gestión de los recursos. La descripción de cada servicio contiene la siguiente información: nombre del servicio, perfil, descripción, argumentos, precondiciones y observaciones.

- **Flujo de Trabajo.** En primer lugar tenemos los servicios de creación y destrución de un flujo de trabajo. Una vez creado el flujo de trabajo, podremos editar tanto las tareas que forman el proceso principal asociado al mismo, como los recursos asociados a dichas tareas.

SERVICIO	**Creación de un flujo de trabajo**
PERFIL	`new_workflow(name,description)`
DESCRIPCIÓN	Se crea un nuevo flujo de trabajo y se devuelve el identificador asociado.
ARGUMENTOS	A1. Nombre asignado al flujo de trabajo. A2. Descripción del flujo de trabajo.
PRECONDICIONES	- -
OBSERVACIONES	- -

SERVICIO	**Borrado de un flujo de trabajo**
PERFIL	`remove_worflow(identifier)`
DESCRIPCIÓN	Se elimina el flujo de trabajo y todos las tareas que lo componen.
ARGUMENTOS	A1. Identificador del flujo de trabajo.
PRECONDICIONES	P1. El flujo de trabajo debe existir.
OBSERVACIONES	OB1. Los recursos asociados al flujo de trabajo borrado no son eliminados sino que quedan como parte de los recursos ya registrados en el editor.

SERVICIO	**Creación del proceso principal asociado al flujo de trabajo**
PERFIL	`new_process(name,description,startCondition,endCondition)`
DESCRIPCIÓN	Se crea un nuevo proceso y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado al proceso. A2. Descripción del proceso. A3. Condición de inicio del proceso. A4. Condición de finalización del proceso.
PRECONDICIONES	P1. Para crear un proceso es necesario tener abierto un flujo de trabajo.
OBSERVACIONES	OB2. El proceso creado queda incluido como componente del flujo de trabajo actual

SERVICIO	**Borrado de un proceso**
PERFIL	`remove_process(identifier)`
DESCRIPCIÓN	Se elimina un proceso y todas las tareas que lo componen.
ARGUMENTOS	A1. Identificador del proceso.
PRECONDICIONES	P1. El proceso debe pertenecer al flujo de trabajo actual.
OBSERVACIONES	OB1. Los recursos asociados al proceso borrado no son eliminados sino que quedan como parte de los recursos ya registrados en el editor.

- **Tareas.** Para las tareas que forman parte del proceso que representa a un flujo de trabajo, los servicios proporcionados por el editor se agrupan en tres categorías: servicios de inserción, de modificación y de borrado de las mismas. A continuación se detallan todos ellos.

A) Inserción de tareas.

SERVICIO	**Creación de una actividad manual**
PERFIL	`add_manualActivity(name,description,startCondition, endCondition, actions)`
DESCRIPCIÓN	Se crea una actividad manual como parte del proceso actual.
ARGUMENTOS	A1. Nombre asignado a la actividad. A2. Descripción de la actividad. A3. Condición de inicio de la actividad. A4. Condición de finalización de la actividad. A5. Acciones asociadas a la actividad.
PRECONDICIONES	P1. Para crear una actividad es necesario tener abierto al menos un proceso (proceso actual).
OBSERVACIONES	- -

SERVICIO	**Creación de una actividad automática**
PERFIL	`add_automaticActivity(name,description,startCondition, endCondition, actions)`
DESCRIPCIÓN	Se crea una actividad automática como parte del proceso actual.
ARGUMENTOS	A1. Nombre asignado a la actividad. A2. Descripción de la actividad. A3. Condición de inicio de la actividad. A4. Condición de finalización de la actividad. A5. Acciones asociadas a la actividad.
PRECONDICIONES	P1. Para crear una actividad es necesario tener abierto al menos un proceso (proceso actual).
OBSERVACIONES	- -

SERVICIO	**Creación de un condición de unión total**
PERFIL	`add_totalJoin`
DESCRIPCIÓN	Se crea una condición de unión total como parte del proceso actual.
ARGUMENTOS	- -
PRECONDICIONES	P1. Para crear una condición es necesario tener abierto al menos un proceso.
OBSERVACIONES	- -

SERVICIO	**Creación de una condición de unión parcial**
PERFIL	`add_partialJoin(number)`
DESCRIPCIÓN	Se crea una condición de unión parcial como parte del proceso actual.
ARGUMENTOS	A1. Número de tareas de entrada que deben finalizar para iniciarse la tarea de salida.
PRECONDICIONES	P1. Para crear una condición es necesario tener abierto al menos un proceso.
OBSERVACIONES	- -

SERVICIO	**Creación de una condición de bifurcación total**
PERFIL	`add_totalSplit`
DESCRIPCIÓN	Se crea una condición de bifurcación total como parte del proceso actual.
ARGUMENTOS	- -
PRECONDICIONES	P1. Para crear una condición es necesario tener abierto al menos un proceso.
OBSERVACIONES	- -

SERVICIO	**Creación de una condición de bifurcación parcial**
PERFIL	`add_partialSplit`
DESCRIPCIÓN	Se crea una condición de bifuración parcial como parte del proceso actual.
ARGUMENTOS	- -
PRECONDICIONES	P1. Para crear una condición es necesario tener abierto al menos un proceso.
OBSERVACIONES	- -

SERVICIO	**Creación de un subproceso**
PERFIL	`new_subprocess(name,description,startCondition,endCondition)`
DESCRIPCIÓN	Se crea un nuevo subproceso y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado al subproceso. A2. Descripción del subproceso. A3. Condición de inicio del subproceso. A4. Condición de finalización del subproceso.
PRECONDICIONES	P1. Para crear un subproceso es necesario tener abierto un proceso.
OBSERVACIONES	OB1. El proceso creado queda incluido como componente del proceso actual.

SERVICIO	**Establecer flujo de control**
PERFIL	`add_controlFlow(entityO,entityTarget)`
DESCRIPCIÓN	Establece el flujo de control entre dos tareas del proceso, estableciendo una relación de precedencia entre ellas.
ARGUMENTOS	A1. Tarea origen del flujo de control. A2. Tarea destino del flujo de control.
PRECONDICIONES	P1. Para añadir un flujo de control es necesario tener abierto un proceso. P2. La tarea origen no puede ser origen de ningún otro flujo de control, a excepción de las condiciones de bifurcación. P3. La tarea destino no puede ser destino de ningún otro flujo de control, a excepción de las condiciones de unión. P4. La tarea destino no puede ser una tarea inicial. P5. La tarea origen no puede ser una tarea final.
OBSERVACIONES	- -

SERVICIO	**Establecer una condición a un flujo de control**
PERFIL	`add_conditionFlow(controlFlow,condition)`
DESCRIPCIÓN	Se añade una condición a un flujo de control que conecta dos tareas.
ARGUMENTOS	A1. Flujo de control. A2. Condición.
PRECONDICIONES	P1. El flujo de control debe pertenecer al proceso actual. P2. Sólo se puede añadir en el caso de que la tarea origen sea un condición de bifurcación parcial o que la tarea destino sea una condición de unión parcial.
OBSERVACIONES	- -

B) Modificación de tareas.

SERVICIO	**Cambiar propiedades de una actividad**
PERFIL	`change_activityProp(name,nameN,desN,startCN,endCN,actN)`
DESCRIPCIÓN	Se cambia el valor de las propiedades de la actividad seleccionada.
ARGUMENTOS	A1. Nombre de la actividad seleccionada. A2. Nuevo nombre asignado a la actividad. A3. Nueva descripción asignada a la actividad.

A4. Nueva condición de inicio de la actividad.
A5. Nueva condición de finalización de la actividad.
A6. Nuevas acciones asignadas a la actividad.

PRECONDICIONES P1. La actividad seleccionada debe formar parte del proceso actual.

OBSERVACIONES OB1. No es necesario cambiar el valor de todas sus propiedades. La palabra reservada *no-change* como argumento indica que la propiedad asociada no cambia su valor.

SERVICIO	**Cambiar propiedades de una subproceso**

PERFIL `change_subprocessProp(name,nameN,desN,startCN,endCN)`

DESCRIPCIÓN Se cambia el valor de las propiedades del subproceso seleccionado.

ARGUMENTOS A1. Nombre del subproceso seleccionado.
A2. Nuevo nombre asignado al subproceso.
A3. Nueva descripción asignada al subproceso.
A4. Nueva condición de inicio al subproceso.
A5. Nueva condición de finalización al subproceso.

PRECONDICIONES P1. El subproceso seleccionado debe formar parte del proceso actual.

OBSERVACIONES OB1. No es necesario cambiar el valor de todas sus propiedades. La palabra reservada *no-change* como argumento indica que la propiedad asociada no cambia su valor.

SERVICIO	**Cambiar condición asociada a un flujo de control**

PERFIL `change_conditionFlow(controlFlow,condN)`

DESCRIPCIÓN Se cambia la condición asociada al flujo de control seleccionado.

ARGUMENTOS A1. Flujo de control seleccionado.
A2. Nueva condición asociada al flujo de control.

PRECONDICIONES P1. El flujo de control seleccionado debe pertenecer al proceso actual.

P2. La tarea origen no puede ser origen de ningún otro flujo de control, a excepción de las condiciones de bifurcación.

P3. La tarea destino no puede ser destino de ningún otro flujo de control, a excepción de las condiciones de unión.

P4. La tarea destino no puede ser una tarea inicial.

P5. La tarea origen no puede ser una tarea final.

OBSERVACIONES - -

SERVICIO	**Convertir subproceso en actividad**
PERFIL	`transform_subprocess(subp,nameA,descA,startCA,endCA,actA)`
DESCRIPCIÓN	Se transforma un subproceso en una actividad. Cambia la granularidad del componente del proceso y se pueden inicializar sus propiedades.
ARGUMENTOS	A1. Nombre del subproceso seleccionado. A2. Nombre asignado a la actividad. A2. Descripción asignada a la actividad. A3. Condición de inicio asignada a la actividad. A4. Condición de finalización asignada a la actividad. A5. Acciones asignadas a la actividad.
PRECONDICIONES	P1. El subproceso seleccionado debe pertenecer al proceso actual.
OBSERVACIONES	OB1. Al realizar esta operación, todos los componentes del subproceso seleccionado, si existen, son eliminados.

SERVICIO	**Convertir actividad en subproceso**
PERFIL	`transform_activity(act,nameP,descP,startCP,endCP)`
DESCRIPCIÓN	Se transforma una actividad en un subproceso. Cambia la granularidad del componente del proceso.
ARGUMENTOS	A1. Nombre de la actividad seleccionada. A2. Nombre asignado al subproceso. A2. Descripción asignada al subproceso. A3. Condición de inicio asignada al subproceso. A4. Condición de finalización asignada al subproceso.
PRECONDICIONES	P1. La actividad seleccionada debe pertenecer al proceso actual.
OBSERVACIONES	OB1. Al realizar esta operación, las acciones de la actividad son eliminadas y todos recursos asociados liberados.

C) Borrado de tareas.

SERVICIO	**Borrado de una tarea**
PERFIL	`remove_task(nombre)`
DESCRIPCIÓN	Se elimina del proceso actual la tarea seleccionada.
ARGUMENTOS	A1. Nombre de la tarea seleccionada.
PRECONDICIONES	P1. La tarea seleccionada debe pertenecer al proceso actual.
OBSERVACIONES	OB1. Todos sus recursos son desasociados y los flujos de control que tengan dicha tarea como origen o destino son borrados.

SERVICIO	**Borrado de un flujo de control**
PERFIL	`remove_controlflow(nombre)`
DESCRIPCIÓN	Se elimina del proceso actual el flujo de control seleccionado.
ARGUMENTOS	A1. Nombre del flujo de control seleccionado.
PRECONDICIONES	P1. El flujo de control seleccionado debe pertenecer al proceso actual.
OBSERVACIONES	OB1. Si existe una condición asociada al mismo, también es eliminada.

- **Recursos.** Para los recursos asociados a un flujo de trabajo, los servicios proporcionados por el editor se agrupan en cuatro categorías: servicios de definición, de inserción, de modificación y de borrado de los mismos. A continuación se detallan todos ellos.

A) Definición de recursos.

SERVICIO	**Definir un recurso dato**
PERFIL	`new_data(name,description,path)`
DESCRIPCIÓN	Se define un nuevo dato y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado al dato. A2. Descripción del dato. A3. Ubicación física del dato.
PRECONDICIONES	P1. El dato definido no debe existir.
OBSERVACIONES	--

SERVICIO	**Definir un recurso aplicación**
PERFIL	`new_application(name,descrip,path,call,permissions)`
DESCRIPCIÓN	Se define una nueva aplicación y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado a la aplicación. A2. Descripción de la aplicación. A3. Ubicación física de la aplicación. A4. Invocación de la aplicación. A5. Permisos asociados a la aplicación.
PRECONDICIONES	P1. La aplicación definida no debe existir.
OBSERVACIONES	--

SERVICIO	**Definir un recurso usuario**
PERFIL	`new_user(name,description)`
DESCRIPCIÓN	Se define un nuevo usuario y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado al usuario. A2. Descripción del usuario.
PRECONDICIONES	P1. El usuario definido no debe existir.
OBSERVACIONES	- -

SERVICIO	**Definir un recurso unidad organizacional**
PERFIL	`new_organizationaUnit(name,description)`
DESCRIPCIÓN	Se define una nueva unidad organizational y se inicializan sus propiedades.
ARGUMENTOS	A1. Nombre asignado a la unidad organizacional. A2. Descripción de la unidad organizacional.
PRECONDICIONES	P1. La unidad organizacional definida no debe existir.
OBSERVACIONES	- -

B) Inserción de recursos.

SERVICIO	**Añadir un recurso**
PERFIL	`add_resource(name)`
DESCRIPCIÓN	Se añade un recurso al proceso actual.
ARGUMENTOS	A1. Nombre asignado al proceso.
PRECONDICIONES	P1. El recurso debe estar definido. P2. Es necesario tener abierto un proceso (proceso actual).
OBSERVACIONES	OB1. El recurso está incluido en el proceso, peo no está asociado a ninguna tarea.

SERVICIO	**Asociar un recurso a una tarea**
PERFIL	`resource_assigment(nameT,nameR)`
DESCRIPCIÓN	Se asocia un recurso a una tarea del proceso actual.
ARGUMENTOS	A1. Nombre de la tarea. A2. Nombre del recurso.
PRECONDICIONES	P1. La tarea debe pertenecer al proceso actual. P2. El recurso debe estar definido. P3. Si la tarea es una actividad manual, no se puede asociar un recurso aplicación.

P4. Si la tarea es una condición de transición, no se puede asociar un recurso aplicación ni tampoco un recurso actor.

OBSERVACIONES - -

C) Modificación de recursos.

SERVICIO	**Añadir nuevas propiedades a un recurso**
PERFIL	`add_propertyResource(name,namP,descP,typeP,valueP)`
DESCRIPCIÓN	Se añade una nueva propiedad a un recurso y se define su tipo.
ARGUMENTOS	A1. Nombre del recurso. A2. Nombre de la nueva propiedad. A3. Descripción de la nueva propiedad. A4. Tipo de la nueva propiedad. A5. Valor por defecto de la nueva propiedad.
PRECONDICIONES	P1. La propiedad que se define no debe existir.
OBSERVACIONES	- -

SERVICIO	**Cambiar propiedades de un recurso**
PERFIL	`change_propertyResource(name,prop,value)`
DESCRIPCIÓN	Se cambia el valor de la propiedad especificada de un recurso.
ARGUMENTOS	A1. Nombre del recurso. A2. Nombre de la propiedad del recurso. A3. Valor asignado a la propiedad.
PRECONDICIONES	P1. El recurso debe estar definido. P2. La propiedad debe estar definida en el recurso. P3. El tipo del valor asignado debe coincidir con el definido para dicha propiedad
OBSERVACIONES	- -

SERVICIO	**Añadir usuarios a una unidad organizacional**
PERFIL	`add_userUnit(nameU,nameOU)`
DESCRIPCIÓN	Se añade un usuario como parte de una unidad organizacional.
ARGUMENTOS	A1. Nombre del usuario. A2. Nombre de la unidad organizacional.
PRECONDICIONES	P1. El usuario debe estar definido. P2. La unidad organizacional debe estar definida. P3. El usuario no debe pertenecer ya a dicha unidad organizacional.
OBSERVACIONES	- -

SERVICIO	**Composición de unidades organizacionales**
PERFIL	`add_subUnitUnit(nameSU,nameU)`
DESCRIPCIÓN	Se añade una unidad organizacional como parte de otra unidad organizacional.
ARGUMENTOS	A1. Nombre de la unidad organizacional componente. A2. Nombre de la unidad organizacional compuesta.
PRECONDICIONES	P1. La unidad organizacional componente debe estar definida. P2. La unidad organizacional compuesta debe estar definida. P3. La unidad organizacional componente no debe pertenecer ya a la unidad organizacional compuesta.
OBSERVACIONES	- -

SERVICIO	**Desasociar un recurso de una tarea**
PERFIL	`resource_notAssigment(nameT,nameR)`
DESCRIPCIÓN	Se libera el recurso asociado a una tarea del proceso actual.
ARGUMENTOS	A1. Nombre de la tarea. A2. Nombre del recurso.
PRECONDICIONES	P1. La tarea debe pertenecer al proceso actual. P2. El recurso debe estar definido. P3. El recurso debe estar asociado a la tarea.
OBSERVACIONES	OB1. El recurso no es borrado del proceso actual.

D) Borrado de recursos.

SERVICIO	**Borrado de una propiedad de un recurso**
PERFIL	`remove_propertyResource(nameR,prop)`
DESCRIPCIÓN	Se elimina una propiedad definida para un recurso.
ARGUMENTOS	A1. Nombre del recurso. A2. Nombre de la propiedad.
PRECONDICIONES	P1. El recurso debe estar definido. P2. La propiedad debe pertenecer al recurso.
OBSERVACIONES	- -

SERVICIO	**Borrado de un recurso**
PERFIL	`remove_resource(name)`
DESCRIPCIÓN	Se elimina un recurso.
ARGUMENTOS	A1. Nombre del recurso.
PRECONDICIONES	P1. El recurso debe estar definido.
OBSERVACIONES	OB1. Todas las asociaciones de dicho recurso a tareas son eliminadas también, en todos los flujos de trabajo.