Un Marco Flexible para la Especificación de Procesos Flexibles

Aplicación a la Resolución de Emergencias

Memoria de Tesis
Master en Ingeniería del Software,
Métodos Formales y Sistemas de Información

Dirigido por: **Dr. D. José H. Canós Cerdá**

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Manuel Llavador Campos

Septiembre 2007

RESUMEN

Los sistemas de gestión de flujos de trabajo, también conocidos en la literatura por el término inglés *workflow management systems*, **permiten modelar, ejecutar y monitorizar** procesos, vistos como **conjuntos de** tareas o **actividades, cuyas realizaciones permiten alcanzar unos ciertos objetivos** o metas en el contexto de una o varias organizaciones. Este tipo de sistemas han sido implantados, desde hace varias décadas, en compañías, organizaciones y entidades de todo tipo para gestión documental, gestión de correo electrónico, gestión de pedidos o gestión de sincronización en el acceso a recursos compartidos, entre otros.

Se utilizan modelos, especificaciones de los procesos, para representar los flujos de trabajo, que son extraídos siguiendo alguna metodología a partir de los procesos de negocio. **El objetivo principal de cualquier metodología para la especificación de flujos de trabajo es ser** lo suficientemente **flexible** como **para soportar la especificación de cualquier proceso**. Cada metodología adopta una notación que la caracteriza, entre las que destacan dos: la basada en diagramas de flujo y la basada en reglas. Aunque en el fondo son muy similares, afrontan el problema desde distintos puntos de vista lo que hace que, dependiendo de las características concretas del proceso, sean más o menos adecuadas.

Por otra parte, **una de las características más buscadas** en los sistemas de gestión de flujos de trabajo es la capacidad de adaptarse a los distintos tipos de cambios que pueden sufrir los procesos durante su ejecución, o lo que es lo mismo, **ser suficientemente flexibles como para soportar cambios en tiempo de ejecución**.

Se han utilizado distintos formalismos para definir la semántica concreta de cada uno de los elementos de los flujos de trabajo de las distintas metodologías (p.e. Redes de Petri, π-calculo o lógica de predicados de primer orden) ya que, en gran medida, **la flexibilidad**, tanto a nivel de metodología como a nivel de ejecución, **viene determinada por el formalismo subyacente.**

En este trabajo **proponemos**

> **Una metodología flexible** para soportar la especificación de cualquier proceso complejo
>
> **Un marco conceptual**, basado en metamodelado, **que proporciona la flexibilidad** necesaria para soportar cualquier cambio **en tiempo de ejecución**

Además, como soporte formal utilizamos **lógica deóntica,** definida como una variante de la lógica dinámica, lo que nos **permite formalizar de forma precisa** ambos tipos de **flexibilidad**.

Este trabajo está motivado en los procesos que deben soportar los Sistemas para Gestión y Resolución de Emergencias. Este campo de aplicación exige una aproximación muy flexible tanto en tiempo de diseño, ya que son procesos muy complejos, como en tiempo de ejecución, ya que son procesos en los que las condiciones reales de ejecución son muy difícilmente predecibles y que, por tanto, es muy probable requieran cambios en tiempo de ejecución.

Contenido

CAPÍTULO 1 – INTRODUCCIÓN

Un proceso de negocio, en general un proceso, es un conjunto de procedimientos o tareas cuya realización permite alcanzar ciertos objetivos o metas en el contexto de una o varias organizaciones. Un flujo de trabajo es la automatización de un proceso por medio de un sistema informático. A los sistemas que ejecutan los flujos de trabajo se les conoce de forma general como Sistemas de Gestión de Flujos de Trabajo (SGFT, en inglés *workflow management systems*).

Este trabajo de investigación se enmarca en el contexto de este tipo de sistemas y, en concreto, en la flexibilidad que ofrecen para la especificación de los flujos de trabajo y el soporte a modificaciones de dichas especificaciones en tiempo de ejecución (flexibilidad en tiempo de ejecución).

En este contexto, identificamos dos problemas principales. El primero de ellos es la dificultad de especificar los procesos, entendiendo que esto implica representarlos en algún formato entendible por el SGFT que lo ha de ejecutar. En otras palabras, el proceso por el cual se obtienen especificaciones de flujos de trabajo a partir de los procesos no es trivial. Para solventar este problema se hace necesario el uso de metodologías que, de forma general, incluyen una etapa inicial de descubrimiento de los procesos, una segunda etapa de diseño de los flujos de trabajo y una última etapa de especificación en el lenguaje soportado por el SGFT. Para la primera fase se emplean técnicas de ingeniería de requisitos, para la segunda técnicas de modelado, y para la tercera lenguajes textuales y/o gráficos que vienen determinados por el sistema de gestión de flujos de trabajo objetivo. En la mayoría de los casos las implementaciones se generan de forma automática a partir de los modelos obtenidos en la etapa de análisis y diseño utilizando compiladores.

El segundo problema tiene que ver con la paradoja del *cumplimiento estricto de procedimientos*. El nivel de automatización y soporte a los usuarios que los SGFTs pueden conseguir es proporcional al nivel de fidelidad del flujo de trabajo respecto al proceso que representa. Esto induciría a pensar que para obtener el máximo de automatización, y por tanto el mayor beneficio del uso de este tipo de sistemas, sería necesario especificar flujos de trabajo lo más fieles a la realidad que sea posible. Sin embargo, está demostrado que para flujos de trabajo en los que se lleva esta fidelidad al extremo la probabilidad de que en casos reales estos sean válidos es muy baja, en gran medida porque cualquier cambio del proceso, por pequeño e insignificante que sea, hará que el sistema deje de funcionar. Este hecho es tan probado que incluso

existe un mecanismo de huelga, llamado *trabajo a reglamento,* que consiste en que los empleados cumplen estrictamente ciertas exigencias que establece su reglamento de trabajo provocando una reducción drástica del rendimiento de la empresa y de esa manera fuerzan a que se cumplan sus reivindicaciones.

Estos dos problemas pasan por la misma solución: flexibilidad, en tiempo de diseño y en tiempo de ejecución. La flexibilidad en tiempo de diseño se refiere a que la metodología, que nos permite obtener los flujos de trabajo correspondientes al proceso a automatizar, sea suficientemente completa como para soportar el proceso de descubrimiento, diseño e implementación para cualquier proceso. La flexibilidad en tiempo de ejecución se refiere a que el SGFT, que nos permite ejecutar los flujos de trabajo, sea capaz de soportar los distintos cambios que puede sufrir un proceso durante su ejecución.

Se han utilizado distintos formalismos para definir la semántica concreta de cada uno de los elementos de los flujos de trabajo de las distintas metodologías (Redes de Petri, π-calculo, lógica de primer orden, etc.). En gran medida, la flexibilidad viene determinada por la expresividad del formalismo subyacente. La principal contribución de este trabajo es la definición de un marco flexible para la especificación de procesos de negocio y la definición de un marco conceptual para la especificación de flujos de trabajo flexibles. La aportación principal es que ambos tipos de flexibilidad se ha conseguido por medio de un formalismo basado en una variante de la lógica dinámica.

En este capítulo presentamos la motivación y objetivos de este trabajo. Al final del capítulo comentamos la estructura general de este documento.

Motivación

La motivación de este trabajo está asociada al estudio de los requisitos para los sistemas que dan soporte a la resolución de emergencias. La resolución de una emergencia es una actividad eminentemente colaborativa, en la que participan un número indeterminado de personas, bajo condiciones en ocasiones extremas, y con el objetivo claro de salvaguardar vidas humanas y, si es posible, bienes materiales. Durante su actuación, los implicados deben tomar decisiones críticas en cortos espacios de tiempo, por lo que es crucial disponer de toda la información necesaria para evitar el mayor número posible de errores. Además, la coordinación de los diferentes actores es clave para garantizar que las acciones se realicen en el orden apropiado, ya que de otro modo el resultado puede incluso empeorar la situación existente.

Las organizaciones tienen la obligación legal de definir en sus planes de emergencias todas las tareas a realizar en los diferentes sucesos en las que por su naturaleza pueden verse implicadas. Cuando se detecta una emergencia se desencadena un proceso que determina una serie de tareas a ejecutar por los actores implicados Así pues, en dicho plan se recoge, para cada posible situación de crisis, los procesos a aplicar. La particularidad de la gestión de emergencias es que, además de la complejidad de la resolución de una emergencia, muchas de las decisiones a tomar no pueden ser predichas de antemano, pues el contexto en el que una cierta emergencia se desarrolla influye notablemente en la respuesta a aplicar.

En la literatura existen dos aproximaciones para la especificación de procesos: las basadas en diagramas de flujo, a las que haremos referencia como flujos de trabajo clásicos, y las basadas en reglas. La aproximación clásica está pensada para facilitar la descripción de procesos de negocio individuales o de grupo en los que existe un orden predeterminado en la ejecución de las distintas tareas. Así pues, los SGFT clásicos utilizan lenguajes de especificación de procesos basados en diagramas de flujo. Cuanto más rico es el lenguaje, obviamente más cerca a la realidad se encuentran las descripciones que con él se pueden construir (Jablonski & Bussler, 1996). Sin embargo, como veremos más adelante, presentan serios problemas para especificar flujos de trabajo en los que el orden de las actividades no está preestablecido sino que depende del valor de las propiedades del proceso (en adelante estado del proceso).

Como alternativa, recientemente se han propuesto varias aproximaciones para la especificación de procesos basadas en reglas (Ellis & Keddara, 2000) (Wainer, 2000) (Kappel, Rausch-Schott, & Retschitzegger, 2000). Según estas aproximaciones, un proceso está definido por un conjunto de reglas que codifican las condiciones de inicio y los cambios producidos tras la ejecución de cada tarea. La novedad es que tanto las condiciones de inicio como los cambios de la actividad están basados exclusivamente en su estado. Evidentemente, en este caso resulta difícil expresar un orden predeterminado de tareas y la visualización del proceso resulta mucho más compleja, sobre todo cuando el conjunto de reglas es elevado.

Así pues, se distinguen dos tipos de flujos de trabajo, aquellos en los que el orden de ejecución de las tareas está predeterminado y aquellos en los que depende de su estado. Las aproximaciones basadas en diagramas de flujo son más adecuadas para el primer tipo, y las basadas en reglas para el segundo.

Sin embargo, en la mayoría de ocasiones, los procesos de negocio no son ni del primer tipo ni del segundo sino una mezcla, más aún cuando los procesos son muy complejos con en el caso de la resolución de emergencia. Para empeorar las cosas, existe cierta expresividad presente en este tipo de sistemas, y en general en los procesos complejos, que no es cubierta por ninguna de las dos aproximaciones, como es la ejecución opcional o prohibida de ciertas tareas.

Así pues, se hace necesaria una nueva aproximación para la especificación de procesos de resolución de emergencias o, en general, la especificación de procesos, que sea más flexible en tiempo de diseño dando la posibilidad de mezclar ambos tipos de requisitos expresivos respecto al orden de ejecución de las tareas.

Por otra parte, mientras que en la aproximación basada en reglas la modificación del proceso en tiempo de ejecución está bien estudiada y soportada, en el caso de las aproximaciones clásicas no, por lo que es necesario que las nuevas aproximaciones sean capaces de soportar procesos flexibles en tiempo de ejecución.

En resumen, el dominio de los sistemas de resolución de emergencias nos ha motivado a desarrollar un marco flexible (en tiempo de diseño) para la especificación de procesos flexibles (en tiempo de ejecución).

Objetivos del trabajo

El objetivo principal de este trabajo es definir pues dicho marco para la especificación flexible de procesos de negocio flexibles, es decir, que proporcione flexibilidad para la especificación de procesos complejos, en tiempo de diseño, y cambiantes o dinámicos, en tiempo de ejecución. Este modelo conceptual estará basado en un formalismo que garantice la validez y corrección de la solución.

Estructura del documento

La realización de los objetivos presentados en la sección anterior se describe a lo largo de este documento en los 6 capítulos que describimos a continuación:

En este capítulo, **Capítulo 1**, hemos realizado una introducción al trabajo de investigación presentando el problema, la motivación, los objetivos y la estructura del documento.

En el **Capítulo 2**, presentamos las distintas aproximaciones para la especificación de procesos, sus ventajas e inconvenientes, así como la expresividad no cubierta.

En el **Capítulo 3**, presentamos una parte del metamodelo para la especificación de procesos propuesto en este trabajo y su formalización en una variante de la lógica dinámica. Tanto el metamodelo como su formalización están basados en los resultados de la tesis doctoral de Mari Carmen Penadés (Penadés Gramaje) y OASIS 3.0 (Letelier Torres, Sánchez Palma, Ramos Salavert, & Pastor López).

En el **Capítulo 4**, presentamos la parte del metamodelo relativa a la especificación del orden de ejecución de las tareas del proceso, lo que constituye el marco flexible para la especificación de procesos desarrollado en este trabajo, y su formalización.

En el **Capítulo 5**, presentamos el mecanismo basado en metanivel que da soporte a la flexibilidad en tiempo de ejecución de los flujos de trabajo especificados con el marco propuesto en este trabajo. Esta parte está totalmente basada en los resultados de la tesis doctoral de Jose Ángel Carsí (Carsí Cubel, 1999).

Por último, en el **Capítulo 6**, se presentan conclusiones acerca de los resultados obtenidos en este trabajo y trabajos futuros, así como los resultados y publicaciones relacionadas.

Resumen

En este capítulo hemos introducido el problema de la flexibilidad en tiempo de diseño y en tiempo de ejecución para la automatización de procesos y hemos motivado el problema en el contexto de los sistemas para la resolución de emergencias en el que los procesos son muy complejos y, en la mayoría de las ocasiones, muy difíciles de predecir, por lo que es necesario que sean fácilmente modificables en tiempo de ejecución.

También hemos presentado el objetivo de este trabajo de investigación: la descripción de un marco flexible para la especificación de procesos flexibles.

Por último hemos presentado la estructura general de este documento.

CAPÍTULO 2 – TRABAJOS RELACIONADOS

La automatización de procesos ha sido objeto de numerosos grupos de investigación y compañías de desarrollo de software. Así pues, existen numerosas aproximaciones al problema. Sin embargo, todas las aproximaciones se pueden agrupar en dos tendencias en función de los lenguajes o notaciones para la especificación de los procesos: las clásicas basadas en diagramas de flujo, y las basadas en reglas (o centradas en el estado).

Este capítulo define los requisitos expresivos de nuestra propuesta. En primer lugar introducimos los trabajos relacionados con cada una de las aproximaciones anteriores, destacando sus ventajas e inconvenientes, y al final presentamos cierta expresividad no cubierta por ninguna de las alternativas. .

Para ilustrar su uso utilizaremos un ejemplo basado en el dominio de los sistemas para la gestión y resolución de emergencias que será introducido en primer lugar.

Un caso de estudio

Este trabajo de investigación ha sido motivado y se aplica en el contexto de los sistemas para la gestión y resolución de emergencias, es decir, sistemas que se encargan, entre otras cosas, de dar soporte a los distintos actores durante el proceso de resolución de una emergencia.

Entre las distintas facetas responsabilidad de un sistema para resolución de emergencias podemos citar la de gestión de la información manejada durante la emergencia, presentación y acceso a dicha información a través de interfaces gráficas, comunicación y colaboración entre los distintos actores, extracción automática de conocimiento o fusión de información, y la gestión del contexto. Sin embargo, una de las más importantes, en la que se centra este trabajo, es la de coordinación de las tareas a realizar por los distintos actores, ya que la resolución de una emergencia es una situación crítica en la que una acción equivocada puede provocar la pérdida de bienes o, peor aún, vidas humanas.

Así pues, podemos decir que la resolución de una emergencia es una actividad eminentemente colaborativa, en la que participan un número indeterminado de personas, bajo condiciones en ocasiones extremas, y con el objetivo claro de salvaguardar vidas humanas y, si es posible, bienes materiales. Durante su actuación, los implicados deben tomar decisiones críticas en cortos espacios de tiempo, por lo que es crucial disponer de toda

la información necesaria para evitar el mayor número posible de errores. Además, la coordinación de los diferentes actores es clave para garantizar que las acciones se realicen en el orden apropiado, ya que de otro modo el resultado puede incluso empeorar la situación existente. La automatización es vital.

Las organizaciones tienen la obligación legal de definir en sus Planes de Emergencias todas las acciones a realizar en los diferentes sucesos en las que por su naturaleza pueden verse implicadas. En dicho plan se recoge, para cada posible situación de crisis, los procesos a aplicar en cada caso. La particularidad de la gestión de emergencias es que muchas de las decisiones a tomar no pueden ser predichas de antemano, pues el contexto en el que una cierta emergencia se desarrolla influye notablemente en la respuesta a

1. Tan pronto como el PM tenga noticia de una emergencia, información contrastada, ordenará que se impida la circulación de trenes en el tramo afectado, dejando liberado el mismo, y ordenará EMERGENCIA NIVEL 1 ó 2. Comunicará a las estaciones Contiguas la situación de ALERTA NIVEL 1 ó 2.
2. Inmediatamente recabará información que le permita valorar la magnitud de la emergencia a los siguientes agentes:
    a. Jefes de las Dependencias afectadas.
    b. Maquinistas que circulan por la zona afectada.
3. En caso de que con la información recibida no se pueda evaluar el alcance de la misma, se realizará un reconocimiento del lugar, por medio de las personas que el PM o el responsable correspondiente designe, dotadas de los equipos de emergencia (linterna, equipo autónomo de respiración, etc…)
4. Detectada la emergencia y en función de su alcance, el PM decidirá la situación de EMERGENCIA NIVEL 2, a los sectores que por su implicación en la misma se encontraban en NIVEL 1, o bien, ordenará la normalización a los sectores que se encontraban en situación de ALERTA o EMERGENCIA.
5. En la situación de EMERGENCIA NIVEL 2, el PM, entre otras actuaciones, requerirá la intervención de los Servicios de Extinción de Incendios, a través del C.C.E o CECOM, coordinando las actuaciones que sean precisas para la resolución de la misma.

Figura 1 Especificación en lenguaje natural del proceso para resolución de emergencia provocada por incendio en túnel de metro

aplicar. Además, los procesos deben ser fácilmente adaptables a las situaciones a las que evoluciones la situación de emergencia.

La Figura 1 muestra la descripción del proceso de resolución de una emergencia provocada por un incendio en un túnel, tal como aparece en el Plan de Autoprotección de la compañía que gestiona el metro en Valencia (Oficina de Emergencias de Metro–Valencia).

Como se puede comprobar, uno de los mayores inconvenientes de los planes de emergencia actuales es que la descripción de los mismos se hace en lenguaje natural, con los consabidos problemas de ambigüedad que limitan considerablemente su utilización. Sólo por citar un ejemplo, se hace uso de abreviaciones y siglas que pueden no ser evidentes para un actor no experto, como por ejemplo un ciudadano involucrado en la emergencia.

En las siguientes secciones mostramos dos alternativas para la especificación de procedimientos de resolución de emergencias o, en general, para la especificación de procesos que permiten evitar estas ambigüedades a la vez que automatizar el proceso haciendo uso de un SGFT.

Flujos de trabajo clásicos basados en diagramas de flujo

Los trabajos relacionados con esta primera aproximación, que hemos denominado flujos de trabajo clásicos, porque son los que se han utilizado tradicionalmente en el contexto de la automatización de procesos de negocio, o basados en diagramas de flujo, porque utilizan dicha notación para la representación explícita de los procesos, son representados por el Workflow Reference Model (Workflow Management Coallition) propuesto por la Workflow Management Coalition (WfMC[1]).

A continuación presentamos las principales características de este modelo de referencia así como un ejemplo de notación gráfica asociada basada en el diagrama de flujo de UML. Cerraremos este apartado con las principales ventajas e inconvenientes detectados en este tipo de aproximaciones.

Workflow Reference Model

Según el modelo de referencia de la WfMC, un flujo de trabajo o *workflow* es la automatización de una parte de un proceso de negocio donde documentos, información o tareas son intercambiados entre los distintos actores o participantes del proceso para conseguir o contribuir a la consecución de un objetivo global de la organización.

[1] www.wfmc.org

Dado que la automatización es un primer paso para la mejora de un proceso de negocio, habitualmente la tecnología de flujos de trabajo, según la WfMC, está asociada a la re-ingeniería de procesos de negocio, la cual se encarga de modelar los procesos de una organización para posteriormente analizar sus deficiencias y, en consecuencia, mejorarlos.

Los sistemas encargados de ejecutar flujos de trabajo son conocidos como sistemas para la gestión de flujos de trabajo (SGFT) o, en inglés, *Workflow Management Systems*. Así pues, un sistema para la gestión de flujos de trabajo permite definir, gestionar y ejecutar especificaciones de proceso de negocio. A pesar de que existe un gran conjunto de sistemas de este tipo, todos ellos comparten las siguientes funcionalidades: Soporte al proceso de definición de los procesos de negocio e implementación como un conjunto de flujos de trabajo, gestión de la ejecución de los flujos de trabajo y gestión de la interacción con los actores y otros sistemas.

Centrándonos en la primera perspectiva, la perspectiva funcional, los SGFT se encargan de dar soporte al usuario durante la definición de los procesos de negocio y su implementación en forma de flujos de trabajo. Esto es, procesos de negocio o modelos de las tareas a realizar para la consecución de un objetivo de la organización son especificados en un lenguaje procesable por una máquina. A las especificaciones derivadas de un proceso de negocio se les conoce como especificaciones de proceso o definiciones de flujo de trabajo.

Existe un extenso y variado grupo de lenguajes para la definición de flujos de trabajo. Sin embargo, todos ellos tienen como idea subyacente la representación del flujo de control y el flujo de datos que determinan el orden de ejecución y la información intercambiada entra las actividades del proceso, por lo que son conocidos como modelos o diagramas de flujo (de control y datos). Entre los distintos lenguajes textuales destaca XPDL y entre las notaciones gráficas las basadas en el diagrama de flujo (o actividad) de UML y el lenguaje BPMN, que ejemplificaremos más adelante.

Respecto a la segunda perspectiva, la ejecución de un flujo de trabajo implica la sincronización o coordinación de las distintas tareas, para que se ejecuten en el orden definido por el flujo de control, así como la gestión de la información intercambiada entre ellas, para que cumplan con la definición del flujo de datos. Además, es deseable que el sistema sea capaz de adaptarse a los cambios en el proceso de negocio evolucionando los flujos de trabajo en ejecución, aunque, como veremos, esta funcionalidad no suele estar soportada.

La tercera perspectiva tiene que ver con aspectos relacionados con la interoperabilidad entre el SGFT y los actores involucrados en la realización de las actividades así como la interacción entre varios SGFT permitiendo su distribución. En este trabajo de investigación no entramos en esta perspectiva aunque sí lo hacemos en otro trabajo de investigación en el que se está desarrollando un entorno para dar soporte a interoperabilidad basado en transformación de documentos (Llavador, 2007).

Así pues, una especificación de proceso de negocio estará formada por un conjunto de definiciones de flujo de trabajo que capturarán los procesos que permiten a la organización conseguir sus objetivos. La siguiente figura muestra el metamodelo de proceso soportado por este tipo de aproximaciones (extraído del estándar XPDL):

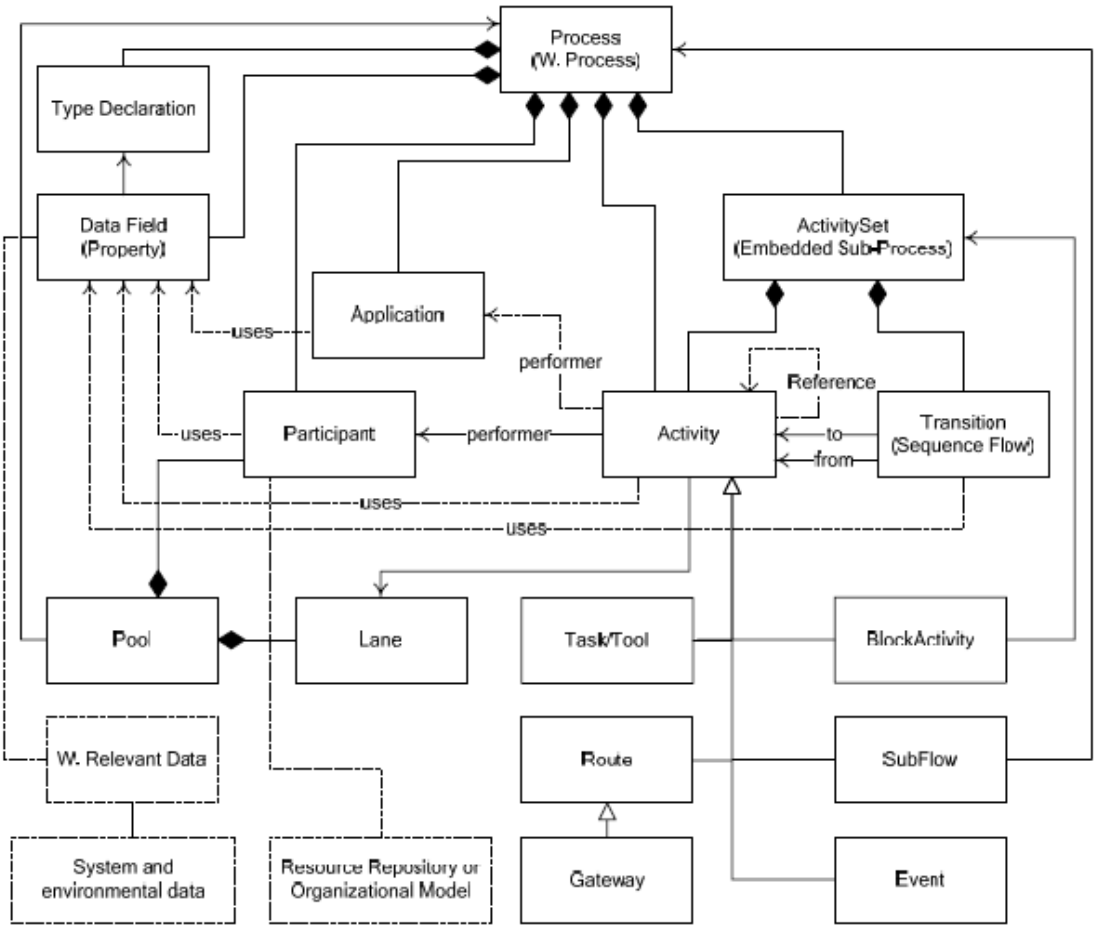

Figura 2 Metamodelo de proceso del lenguaje XPDL

Como se puede observar, un proceso está compuesto por un conjunto de actividades, un conjunto de transiciones o definición de flujo de control, un conjunto de actores, que podrán ser participantes humanos o aplicaciones, y un conjunto de datos que conformarán el flujo de datos y que tendrán unos tipos definidos.

Un flujo de trabajo puede tener, adicionalmente, un conjunto de propiedades o atributos que lo describen, como el autor o la versión, propiedades relevantes en tiempo de ejecución, como la prioridad, y datos relevantes para simulación o análisis, como tiempos estimados de ejecución.

A continuación describimos en más detalle las actividades y transiciones.

Actividad

Una actividad describe una tarea o conjunto de tareas que constituyen una unidad atómica de trabajo. Esta unidad de trabajo será realizada por un actor o una aplicación, por lo que una tarea tendrá asociado un actor o aplicación responsable de la realización del trabajo. Además, una actividad puede tener un conjunto de datos requeridos para la realización de la tarea así como un conjunto de datos generados tras la finalización de la misma, los cuales definirán el flujo de datos. Otra información adicional puede asociarse a una actividad, como por ejemplo precondiciones y postcondiciones que deben cumplirse para iniciar y finalizar la actividad respectivamente, o prioridades que permiten solventar casos en los que los recursos consumidos por la actividad están ocupados.

Las actividades de un flujo de trabajo siguen un orden de ejecución determinado por el flujo de control. Las propiedades *Join* y *Split* nos permitirán definir en detalle el comportamiento del flujo de control cuando hay más de una *transición* que llega a la actividad y cuando hay más de una *transición* que parte de la actividad respectivamente. Hablaremos de ello al definir las transiciones.

Una actividad puede definirse como subproceso en cuyo caso será un contenedor que representará la ejecución separada de un flujo de trabajo y que, por tanto, contendrá sus propias definiciones de actividades, transiciones, datos, etc y las asociaciones con los actores o aplicaciones. En este caso, la actividad definirá parámetros de entrada y salida para permitir el intercambio de información entre el proceso o flujo de trabajo principal y el subproceso.

Otro tipo de actividad es la actividad bucle en cuyo caso actuará como controlador de repetición de un conjunto de actividades que se ejecutarán repetidamente mientras se cumpla una condición.

Existe un tipo especial de actividades, denominadas vacías, que nos permiten realizar transiciones complejas entre actividades haciendo uso de las propiedades *Join* y *Split* pero que no implican la realización

de ninguna tarea por parte de los actores del proceso. Estas actividades también son conocidas como *gateways* en la literatura.

Por último, una actividad puede tener asociado uno o más eventos. Un evento tiene un disparador o condición que los provoca, que determina una cosa relevante que ocurre durante la ejecución de la actividad, y causan un cambio del flujo de control, pudiendo ser el inicio o finalización de la actividad que tiene asociado el evento u otra actividad del proceso. Se distinguen tres tipos de eventos: inicio, intermedio y fin. El evento de inicio indica en qué condiciones se iniciará un proceso (la primera actividad del proceso) pudiendo ser la recepción de un mensaje, un reloj o alarma, una regla de negocio, una invocación desde un proceso contenedor o una combinación de ellos. Los eventos intermedios se producen cuando el proceso ya ha comenzado y todavía no ha terminado y, por tanto, no pueden estar asociados a la tarea inicial o final del proceso, y suelen utilizarse para manejar situaciones excepcionales. Los eventos finales representan el final de la ejecución del proceso (la última actividad del proceso).

Transición

El orden de ejecución de las actividades de un flujo de trabajo viene determinado por las transiciones entre actividades. Una transición tiene una actividad origen, una actividad destino y, opcionalmente, una condición de transición que debe cumplirse para que la transición sea efectiva. Cuando se termina una actividad el SGFT busca qué transición tiene como origen dicha actividad y, si se cumple la condición de transición, configura el flujo de control para que active la actividad destino.

Cuando hay más de una transición posible el comportamiento del flujo de control depende de las propiedades Join y Split de las actividades origen y destino. La propiedad *Join* de una actividad destino de más de una transición determina el comportamiento del flujo de control al alcanzar la actividad. Así, para un conjunto de transiciones que tienen la misma actividad como destino, el valor de la propiedad *Join* de esa actividad se interpreta como:

- AND → el flujo de control no inicia la actividad destino mientras no se cumplan todas las condiciones de transición.
- OR → el flujo de control inicia la actividad destino de las transiciones que cumplen la condición de transición.

- XOR → el flujo de control inicia la actividad destino de la primera transición que cumple la condición descartando el resto.

La propiedad *Split* de las actividades origen de más de una transición determina el comportamiento del flujo de control al salir de la actividad. Así, para un conjunto de transiciones que tienen la misma actividad como origen, el valor de la propiedad *Split* de esa actividad se interpreta como:

- AND → se ejecutan tantas transiciones del flujo de control, de forma concurrente y en hilos separados, como transiciones que cumplan la condición de transición.
- OR → igual que un AND pero con la posibilidad de decidir una transición *otherwise* que se ejecutará cuando ninguna de las otras transiciones cumplan con la condición de transición.
- XOR → el flujo de control sólo ejecutará la primera transición que cumpla la condición de transición y el resto serán descartadas.

Además, tanto la propiedad *Join* como *Split* pueden contender condiciones complejas como, por ejemplo, que se cumplan más de tres condiciones de transición al mismo tiempo. Esto junto con las actividades *gateway* permite especificar cualquier comportamiento para el flujo de control.

La Figura 3 muestra una posible representación gráfica basada en el modelo de flujo de UML del proceso para la resolución de emergencia por incendio en túnel de la Figura 1. Ciertos elementos se han simplificado u omitido, como es el caso del flujo de datos, para mejor la claridad del modelo. El proceso comienza cuando un responsable de la seguridad en el puesto de mando *(Safety Manager*) recibe una notificación de emergencia. De acuerdo con la primera regla del plan de emergencia, una vez se ha recibido la señal el puesto de mando debe parar todos los trenes circulando por la zona afectada y evaluar lo más rápido posible la gravedad del incidente para establecer un nivel de emergencia inicial. Para hacer estas tareas, el puesto de mando hace uso del panel de control (*Control Panel*) y de un sistema experto secuencialmente (*Expert System*).

Llegado a este punto las actividades a ejecutar dependen del nivel de emergencia establecido por el sistema experto. Si se trata de un nivel de emergencia inicial 1 es porque se supone que se trata de una emergencia que puede resolverse con los propios medios del sistema de metro. Así pues, en ese caso debe sucederse un conjunto de actividades para verificar que

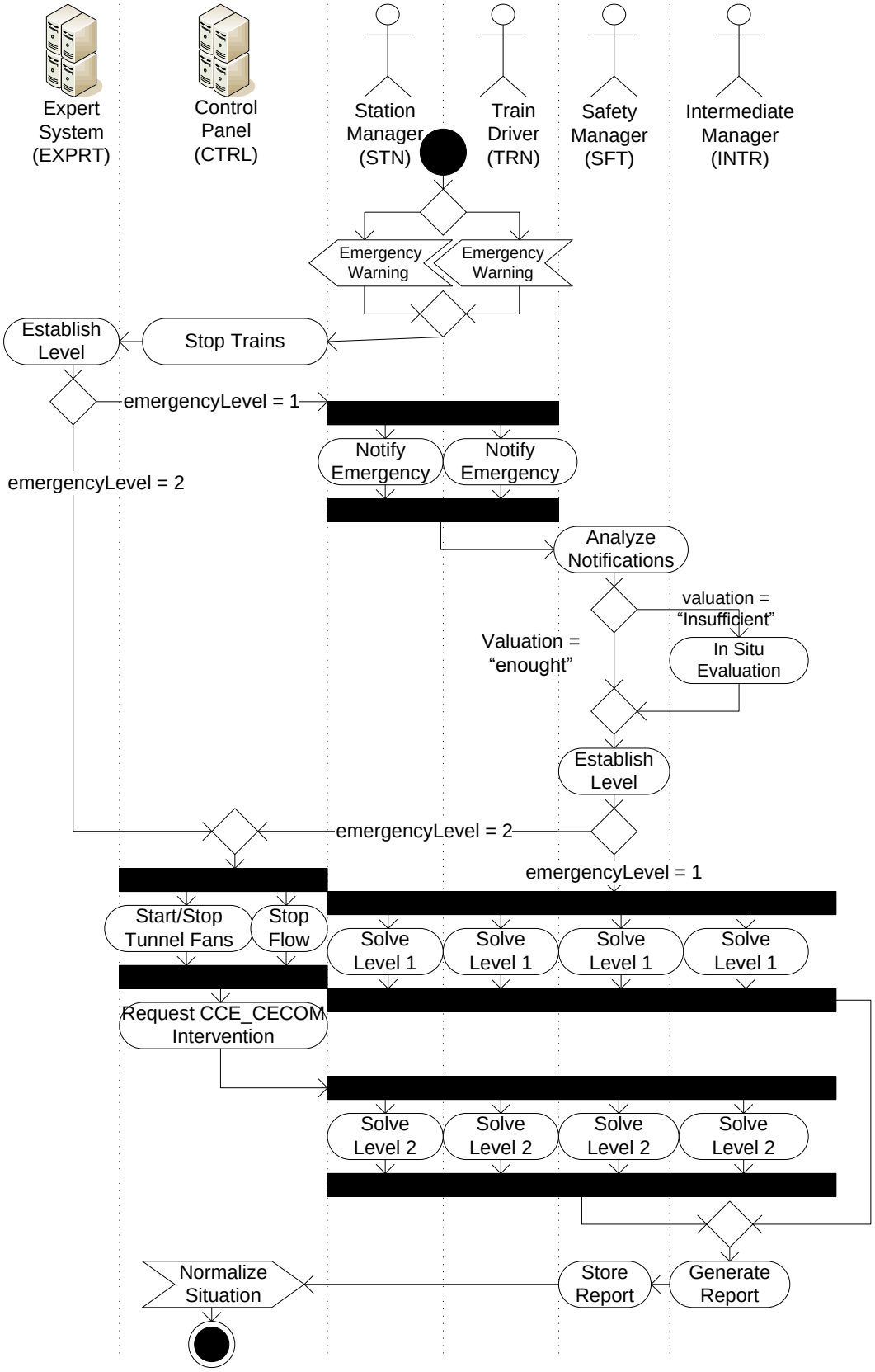

Figura 3 Representación gráfica basada en diagramas de flujo de UML del proceso de resolución de emergencia provocado por un incendio en túnel de metro

efectivamente se trata de una emergencia leve. Como indican las sentencias 2 y 3 del plan en lenguaje natural de la Figura 1, el puesto de mando pedirá a los jefes de estación (*Station Manager*) y los conductores de trenes afectados por la emergencia (*Train Drivers*) que evalúen la magnitud del incidente. Si la información obtenida por parte de los jefes de estación y conductores de tren no es suficiente para que el puesto de mando evalúe definitivamente la magnitud del accidente entonces se enviará a un mando intermedio (Intermediate Manager) para que realice un informe o evaluación "in situ" de la gravedad del accidente. Finalmente, se establecerá un nivel de emergencia definitivo por parte del puesto de mando.

En caso de que el incidente sea definitivamente catalogado como una emergencia de nivel 1, todos los actores involucrados en el proceso de resolución de la emergencia procederán a resolverla por sus propios medios (extintores, kits de primeros auxilios, etc.) siguiendo sus propios subprocesos. Por ejemplo, los conductores de tren deben hacerse cargo de los pasajeros bloqueados en el túnel y evacuarlos a la estación más próxima a la que se tenga acceso donde el jefe de estación se encargará de guiarlos a las salidas de emergencia más convenientes, y el resto de acciones que se han omitido por cuestiones de espacio y por no aportar nada nuevo.

En caso de que el incidente sea catalogado inicialmente como una emergencia de nivel 2 o bien, tras la evaluación preliminar, sea catalogado definitivamente como una emergencia de nivel 2, el puesto de mando, haciendo uso del panel de control, deberá parar o activar los sistemas de ventilación (dependiendo de la situación) y parar el fluido eléctrico del túnel. Estas tareas pueden realizarse simultáneamente para ahorrar tiempo. Una vez cortada la luz y configurados los sistemas de ventilación el puesto de mando deberá requerir la intervención de agentes externos que se encarguen de resolver la situación de emergencia como son bomberos, policía, servicios médicos, etc. Finalmente, los actores involucrados en el proceso deberán ponerse a las órdenes de los agentes externos. Esta información está representada en la sentencia 5 de la especificación en lenguaje natural de la Figura 1.

Una vez resuelta la situación de emergencia, tal como indica la sentencia 4, un mando intermedio generará un informe que será entregado al puesto de mando y se normalizará la situación a través del panel de control.

Ventajas

Como se puede observar, la especificación textual o representación gráfica de procesos en los que existe un orden claro y preestablecido de ejecución de las tareas resulta extremadamente sencilla. Esto hace que una

especificación de proceso definida por medio de un diagrama de actividad UML u otra notación equivalente, como por ejemplo un diagrama en BPMN, sea muy fácil de interpretar incluso por personas no expertas en el área aumentando notablemente su uso en etapas tempranas de captación de requisitos, lo cual les ha otorgando gran popularidad.

Por otra parte, existen gran cantidad de consorcios y empresas que respaldan este modelo de referencia para la especificación de procesos. La siguiente figura muestra los estándares más representativos en cada uno de los aspectos funcionales:

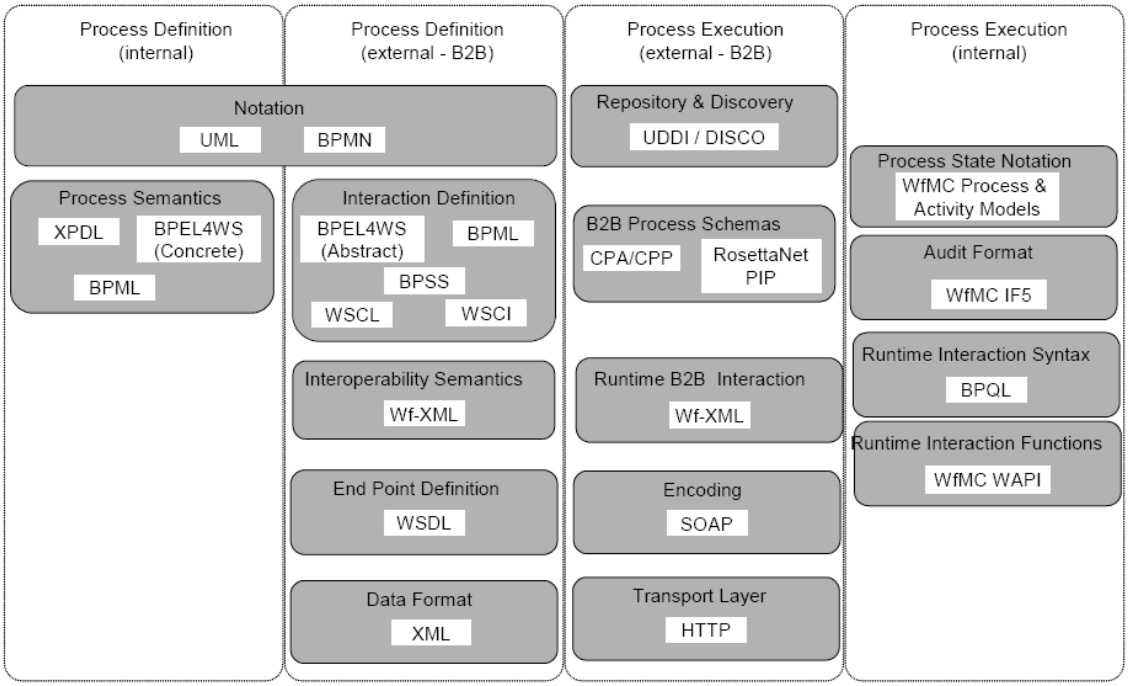

Figura 4 Estándares relacionados con la aproximación basada en flujos para la especificación de procesos

Así pues, existen numerosas herramientas que soportan la definición, ejecución y monitorización de flujos de trabajo de este tipo, entre las que destaca Microsoft BizTalk Server con su versión para desarrollo de software Microsoft Windows Workflow Foundation.

Inconvenientes

Sin embargo, la principal desventaja de este tipo de aproximaciones para la especificación de procesos es que no soportan o, más bien, en ellas resulta complejo definir tareas cuyo orden de ejecución viene determinado exclusivamente por el estado del proceso o, lo que es lo mismo, actividades cuyo orden de ejecución no está preestablecido. Sirva como ejemplo la siguiente situación: supongamos que, tras asignar un nivel inicial de emergencia 1, durante la verificación de que se trata de una emergencia de

poca envergadura o durante la resolución de la emergencia, si ocurre algún incidente que provoca un cambio repentino del nivel de emergencia 1 a nivel de emergencia 2 el proceso debe saltar inmediatamente a las actividades de resolución de una emergencia de nivel 2 empezando por la configuración de ventiladores y corte de suministro eléctrico en el túnel. Si quisiéramos representar dicha situación con este tipo de lenguajes sería necesario añadir un evento intermedio y una transición que partiera de cada una de las actividades en las que puede ocurrir un cambio del valor del nivel de emergencia de 1 a 2, cosa que no es fácil de saber y que requeriría un análisis de todas las actividades para saber si es posible que durante su ejecución ocurra un cambio del valor del nivel de emergencia de 1 a 2.

Ciertos lenguajes, como el BPML o BPEL4WS BPEL (Owen & Raj, 2003) (Andrews, y otros, 2003) permiten simular comportamientos parecidos haciendo uso de los eventos. Sin embargo, este tipo de especificaciones tienen numerosas restricciones semánticas y en cualquier caso complican en exceso las especificaciones perdiendo las representaciones gráficas atractivas de este tipo de modelos.

Por otra parte, el modelo de referencia no da ningún tipo de soporte al cambio o evolución de los procesos en tiempo de ejecución para dar soporte a situaciones no previstas durante la especificación de los procesos, a excepción de manejadores de excepciones que, en cualquier caso, deben ser definidos de antemano mediante eventos intermedios. Seguramente, la argumentación es que este tipo de requisitos expresivos no suelen ser necesarios en el dominio de los procesos de negocio empresariales y supondrían una complejidad adicional al modelo. Sin embargo, existen numerosos dominios complejos y dinámicos, como puede ser el de las emergencias, en los que esta funcionalidad es vital ya que de otra manera la especificación del proceso nunca se corresponderá con las situaciones reales y por tanto los sistemas resultantes nunca serán útiles (Ellis, Keddara, & Rozenberg, 1995) (Casati, Ceri, Pernici, & Pozzi, 1996).

Basados en estado o basados en reglas

Los trabajos relacionados con este segundo conjunto de aproximaciones, se basan en el uso de reglas, expresadas en algún lenguaje de representación de cambios de estado, para la especificación de los procesos de negocio.

A continuación presentamos las principales características de esta aproximación a partir de los trabajos de Clarence Ellis y G. Kappel así como algún ejemplo típico. Cerraremos este apartado con las principales ventajas e inconvenientes detectados en este tipo de aproximaciones.

Proceso como conjunto de reglas

La lógica proposicional ha sido una de las primeras aproximaciones para la representación de conocimiento. Todavía hoy en día sigue siendo relevante debido a que formas más avanzadas de razonamiento en sistemas para gestión de conocimiento están basadas en esta lógica.

En esencia, un sistema basado en reglas está formado por un motor de inferencia, un conjunto de hechos o datos y un conjunto de reglas *if* → *then*. El motor de inferencia encapsula la forma más básica de deducción basada en lógica proposicional, *modus ponens*, que establece que si existe una regla *if hecho1 then hecho2* y se cumple *hecho1* entonces también se cumple *hecho2*. El motor de inferencia ejecuta indefinidamente aquellas reglas para las que los hechos almacenados en la base de conocimiento hacen verdadera la condición de la parte *if*, lo cual implica modificar la base de conocimiento como indique la parte *then*, hasta que ninguna regla puede ser ejecutada.

Numerosos trabajos de investigación han utilizado los sistemas basados en reglas como una forma de representación de software y, como caso particular, para la especificación de procesos. Así pues, en los últimos años se han desarrollado numerosos lenguajes para la especificación de procesos basados en reglas entre los que destacan los trabajos de Ellis, Wainer y Kappel (Ellis & Keddara, 2000) (Wainer, 2000) (Kappel, Rausch-Schott, & Retschitzegger, 2000) que tienen como característica principal la integración del modelo orientado a objetos con el principio de evento/condición/acción de las bases de datos activas propuesto por Dayal (Dayal, Hanson, & Widom, 1994).

Así pues, según esta aproximación, podríamos decir que un proceso está definido por un conjunto de reglas de la forma *estado1 [actividad1] estado2* donde *estado1* representa el estado en el que debe encontrarse el proceso para que se inicie la actividad *actividad1* y *estado2* el estado de los datos alcanzado tras su ejecución.

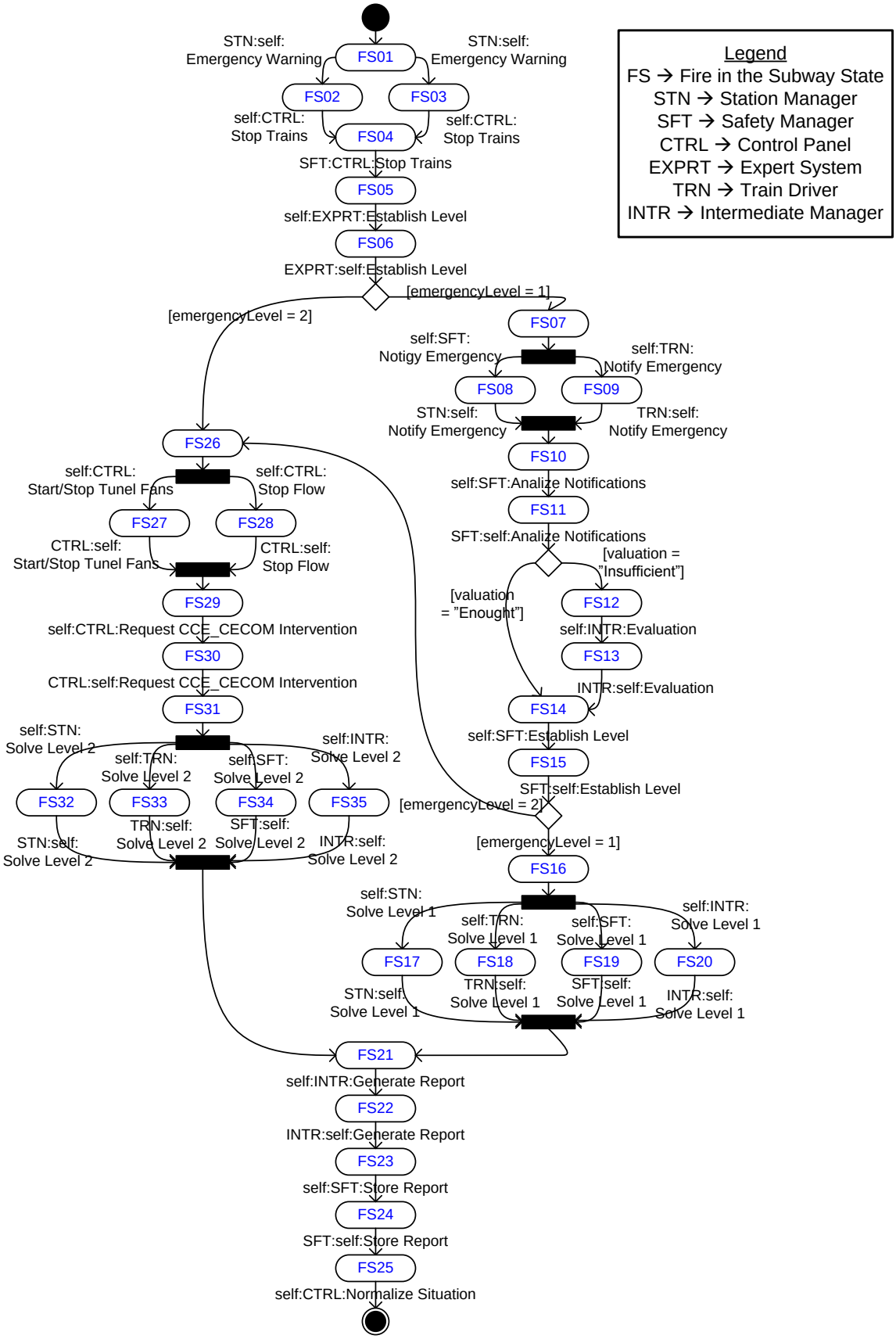

Figura 5 Diagrama de transición entre estados equivalente a la especificación basada en reglas de la Figura 1

Como se puede deducir del párrafo anterior, las especificaciones basadas en reglas pueden expresarse como un diagrama de transición entre estados. El diagrama de transición entre estados que representa el proceso de la Figura 1 se presenta en la Figura 5.

Un ejemplo de regla que daría soporte al comportamiento ya comentado anteriormente de cambio repentino a nivel de emergencia 2 sería el siguiente:

```
emergencyLevel = 1 and stablishLevel(2)
     [start/stopTunelFans && stopFlow]
          tunnelFans = ON/OFF and tunnelFlow = OFF
```

donde la primera parte de la regla representa el estado en el que se cambia de nivel de emergencia 1 a 2 y la segunda parte el estado alcanzado tras la ejecución de las actividades de configuración de los ventiladores del túnel y el corte de suministro eléctrico.

Ventajas

Como se puede observar, la especificación de procesos en los que la ejecución de las actividades está basada en el estado de los procesos resulta extremadamente sencilla siguiendo esta aproximación. Por otra parte, el hecho de poder añadir, modificar o eliminar reglas durante la ejecución del proceso dota de gran flexibilidad en tiempo de ejecución a este tipo de aproximaciones, como se puede ver en los trabajos de Ellis (Ellis & Keddara, 2000).

Los estados suelen estar expresados por medio de lógicas, por ejemplo lógica de primer orden, lo cual otorga gran potencia para el razonamiento, análisis, pruebas, etc. a este tipo de aproximaciones. Por ejemplo, permite generar prototipos o mejorar las especificaciones de forma automática (Roche, Letelier, Navarro, & Llavador, 2006).

Inconvenientes

Sin embargo, estas aproximaciones presentan varios inconvenientes serios. El primero y principal es que a medida que aumenta el número de reglas es complicado visualizar cual será el orden real de ejecución de las tareas o actividades y por tanto resulta complicado especificar actividades cuyo orden de ejecución está predeterminado.

Por otra parte, y relacionado con el anterior, la cantidad de reglas generadas y, por tanto, la complejidad del modelo resultante es mucho mayor que para la aproximación clásica. Para darse cuenta de este hecho es suficiente con comparar el diagrama de flujo de la Figura 3 y el de la Figura 5 que representan el mismo proceso de negocio.

Por otra parte, como se ha comentado, estas aproximaciones siguen un paradigma *if* → *then* que, en realidad, los humanos no utilizamos en nuestros razonamientos, por lo que suele ser más complicado llegar a una especificación correcta de este tipo en comparación con las basadas en flujos de trabajo que en general siguen una aproximación más orientada a objetos.

Por último, no existen herramientas industriales que den soporte a este tipo de especificaciones. La mayoría de herramientas son prototipos desarrollados por grupos de investigación universitarios que se limitan a realizar simulaciones sin llegar a versiones ejecutables de los procesos.

Todos estos inconvenientes limitan seriamente su uso en contextos reales.

Expresividad no cubierta

Llegados a este punto se puede pensar que las aproximaciones basadas en diagramas de flujo y las basadas en reglas son complementarias o que, por lo menos, enfocan el problema desde puntos de vista complementarios y que por tanto con ambas aproximaciones quedaría cubierta cualquier expresividad necesaria para representar un proceso. Sin embargo, esto no es así.

Tanto en la aproximación basada en diagramas de flujo como en la basada en reglas, se asume que cuando una transición entre actividades o una transición entre estados, respectivamente, cumplen la condición de transición necesariamente ésta debe ejecutarse. Por tanto, un diagrama de flujo o un conjunto de reglas es un conjunto de actividades cuya ejecución es obligatoria. Sin embargo, esto no es siempre así, o dicho de otra manera, en numerosas ocasiones es necesario especificar que ciertas actividades pueden ejecutarse o, de forma complementaria, no pueden ejecutarse dependiendo de una precondición basada en el flujo de control o el estado del proceso.

Dos ejemplos de reglas representativas:

- *No se realizará ninguna actividad que implique entrar en el túnel sin que el suministro eléctrico esté cortado previamente.*

- *Después de solicitar informes a los jefes de estación y conductores de tren, y antes de la recepción de los informes, el puesto de mando podrá en cualquier momento, y de forma opcional, consultar cualquier información de la base de datos.*

Así pues, llegamos a la conclusión de que existe un conjunto de expresividad relacionada con la ejecución opcional o prohibida de las actividades que no

es cubierta ni por la aproximación clásica basada en diagramas de flujo ni la basada en reglas.

Resumen

En esta sección hemos presentado los trabajos relacionados con la especificación y ejecución de procesos. Hemos comprobado como las dos aproximaciones para la especificación de procesos, la basada en diagramas de flujo y la basada en reglas, tienen ventajas pero también inconvenientes y, al mismo tiempo, hemos comprobado que hay una cierta expresividad, referida a la ejecución opcional o prohibida de las tareas que no es cubierta por ninguna de ellas.

Así pues, llegamos a la conclusión de que ninguna de las soluciones anteriormente comentadas cubre con toda la expresividad posiblemente requerida para la especificación de un proceso por lo que es necesaria una nueva aproximación.

En los siguientes capítulos presentamos nuestra aproximación basada en la mezcla de las aproximaciones basadas en procesos de negocio y reglas, para obtener las ventajas de ambas aproximaciones resolviendo sus problemas, utilizando como base un formalismo basado en la lógica deóntica que también de soporte a la especificación de actividades cuya ejecución es opcional y/o, complementariamente, prohibida. Además, la aproximación está basada en OASIS por lo que su metanivel proporciona un soporte a la evolución de los procesos en tiempo de ejecución.

CAPÍTULO 3 – METAMODELO DE PROCESO Y SU FORMALIZACIÓN EN UNA VARIANTE DE LA LÓGICA DINÁMICA

Una especificación de proceso es, independientemente de la aproximación adoptada, un conjunto de definiciones de flujo de trabajo donde cada uno de ellos está formado por una o más tareas que se ejecutarán en un orden.

En tiempo de ejecución, las instancias de proceso (ejecuciones individuales de un proceso) interaccionan entre sí para conseguir el objetivo o meta del proceso. Por otra parte, cada actividad o tarea individual dentro de un proceso puede comunicarse con entidades externas (participantes) para conseguir su objetivo. Estas comunicaciones están basadas en canales de comunicación entre clientes y servidores, peticiones de servicios y paso de parámetros.

En este capítulo presentamos una parte del metamodelo de proceso referencia de este trabajo así como su formalización como un conjunto de fórmulas en una variante de la lógica dinámica, cuya semántica declarativa está definida en el marco de una estructura de Kripke[2].

Tanto el metamodelo como la formalización están basados en las propuestas de los Capítulos 4 y 5 de la tesis doctoral de Mari Carmen Penadés (Penadés Gramaje) que, a su vez, están basados en OASIS (Letelier Torres, Sánchez Palma, Ramos Salavert, & Pastor López), de la que se han adaptado todos los aspectos relativos a la especificación del orden de ejecución de las tareas que será considerado en el siguiente capítulo.

Metamodelo de proceso

El metamodelo que se presenta en esta sección recoge todos aquellos aspectos o componentes básicos y esenciales de un proceso. Así pues, estos componentes están de acuerdo con las especificaciones de la WfMC y la del resto de trabajos relevantes en este campo presentados en el capítulo anterior. Como se ha comentado en la introducción a este capítulo, no se presentarán las cuestiones relativas a la definición del orden de ejecución de las actividades, punto de discrepancia de los distintos trabajos relacionados,

[2] En 1965, Saul Kripke publicó una interpretación semántica de una lógica basada en ciertas álgebras como corolario de su semántica para lógicas modales. La idea central de la semántica de Kripke es caracterizar el valor de verdad de las sentencias mediante estados temporales o estados de conocimiento. Así, una sentencia no es sólo cierta, sino que lo es en cierto estado o en un estado de conocimiento. Estos estados de conocimiento son lo que denominaremos mudos.

ya que este aspecto será tratado en detalle en el capítulo siguiente, en el que se presenta una metodología flexible para la especificación del orden de ejecución de las actividades de un flujo de trabajo así como su formalización.

Dimensiones de un Flujo de Trabajo

De acuerdo con la definición clásica de flujo de trabajo como automatización de una parte de un proceso, se considera que un flujo de trabajo tiene tres dimensiones básicas, que son ortogonales entre sí (Leyman & Roller, 2000). Estas son: lógica del proceso, estructura organizacional, e infraestructura de tecnologías de la información.

En primer lugar, la lógica del proceso describe qué tareas se realizan y en qué orden. Las tareas pueden ser una o varias actividades concretas o un subproceso, que se pueden invocar de forma local o remota dentro de la misma compañía o en otra distinta. 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, un *flujo de trabajo* está formado por un *conjunto de tareas* que tienen asociados un conjunto de *recursos*[3], que dan cuenta de: qué se ejecuta, quién lo ejecuta y con qué se ejecuta. A continuación se detalla cada uno de los componentes del flujo de trabajo, con la finalidad de ir refinando el metamodelo propuesto.

Proceso como conjunto de tareas

Un proceso está formado por un conjunto de *tareas* ordenadas. El orden determina la secuencia de ejecución de las distintas tareas que componen el proceso (descrito en el siguiente capítulo). Las tareas pueden ser de dos tipos: *actividades y subprocesos*. Las actividades y subprocesos especifican cada uno de las tareas que componen el proceso a distinto nivel de granularidad.

Además de las tareas ordenadas, todo proceso tiene asociada la siguiente información: un identificador, un nombre, una descripción, una condición de inicio, una condición de finalización y un estado.

[3] 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.

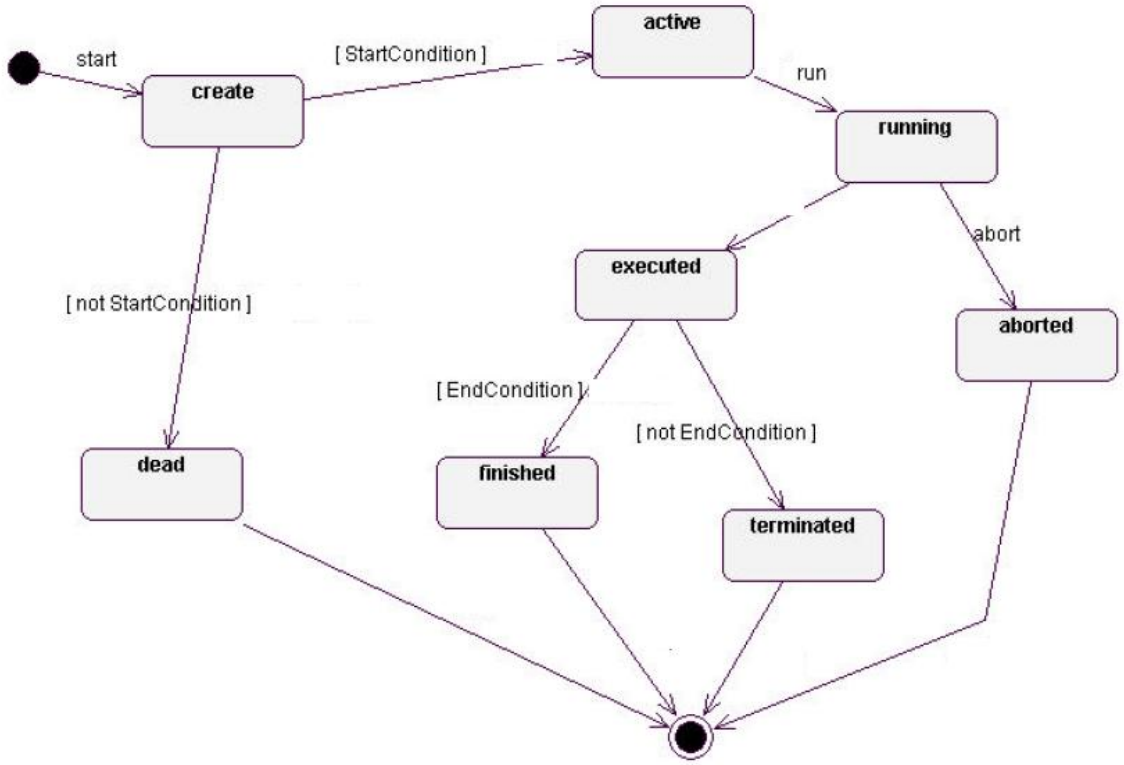

Figura 6 Diagrama de transición de estados de un proceso

La condición de inicio del proceso es una condición que sólo cuando se evalúa a cierto[4] 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 cierto, el proceso finaliza con éxito y si se evalúa a falso[5], el proceso finaliza sin éxito. Finalmente, todo proceso tiene información acerca del estado en que se encuentra[6]. En concreto, interesa conocer cuándo un proceso ha sido creado, cuándo se cumple la condición de inicio y el proceso para 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 finalización del mismo se ha producido con éxito, sin éxito, o bien si ha sido abortado por el usuario. Los posibles estados en los que se puede encontrar un proceso, junto con las transiciones válidas entre ellos, se describe en el diagrama de estados de la Figura 6. Como veremos más adelante, todo proceso tiene asociado un conjunto de datos del dominio que serán utilizados en la realización de las tareas por los participantes. Por ejemplo, en el proceso en el dominio de emergencias presentado en el Capítulo 2, los datos manejados serán el nivel de emergencia, si el flujo

[4] Valor *true* del tipo bool.
[5] Valor *false* del tipo bool.
[6] Esta información sólo es relevante una vez el proceso ha iniciado su ejecuciónl.

eléctrico está habilitado o cortado, etc. El valor de los datos manejados por el proceso también formarán parte del estado del proceso.

Así pues, las propiedades del proceso contienen la información indicada anteriormente, es decir, todo proceso tiene un identificador, un nombre, una descripción, una condición de inicio y una de finalización, y un estado.

Por otra parte, como se ha comentado, 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, se considera un subproceso. 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, realizar cálculos estadísticos, u obtener los trenes afectados por una emergencia en función de características de localización.

Así pues, del mismo modo que en el caso de los procesos, una actividad tiene como propiedades un identificador, un nombre, una descripción, una condición de inicio y de finalización y un estado. Además de las anteriores, toda actividad tiene una propiedad que contiene las acciones concretas asociadas a la misma.

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

anteriormente con la particularidad de que siempre existirá un proceso de nivel más alto al que pertenezca que podrá pasar y recibir datos de entrada o salida, respectivamente. La noción de subproceso introduce modularidad en la definición de un flujo de trabajo, facilitando el modelado de procesos complejos.

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.

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 siguiente figura muestra el diagrama de estados para las aplicaciones:

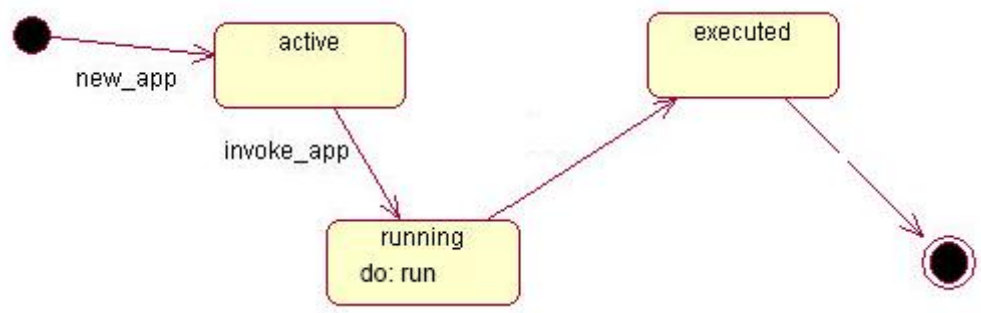

Figura 7 Diagrama de transición de estados de las aplicaciones

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 especialización/generalización que recoja dichas particularidades.

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, o al proceso en general, si son compartidas por varias tareas. Más concretamente, los datos son utilizados por actividades concretas o como parte de la evaluación de las condiciones de transición pero pueden ser compartidas por ellas. 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 o en un almacén de datos). 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.

De forma similar a como ocurre en las aplicaciones, utilizando el operador de especialización creamos una jerarquía, podemos definir nuevos tipos de datos. En esta especialización se añaden nuevos atributos y métodos, que permiten modelar la estructura y comportamiento concreto del dato utilizado.

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 llamaremos de forma genérica *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 quiénes 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. Así pues, el modelo organizacional aquí propuesto es similar al de otros modelos de flujos de trabajo (Hollingsworth, 1995) (Leyman & Roller, 2000).

El Modelo de Objetos de OASIS

Una vez definido el metamodelo de referencia, para poder especificar de forma precisa el comportamiento del proceso en tiempo de ejecución se hace necesario el uso de una aproximación formal. En nuestra propuesta utilizaremos OASIS para tal efecto.

OASIS[7] es una aproximación formal, declarativa y orientada a objetos a la especificación de sistemas de información. Por tanto, en OASIS, los bloques básicos de construcción de los sistemas de información son los *objetos*. Un objeto OASIS es un proceso observable (Ramos I. e., 1993) que encapsula estado y comportamiento. Cada objeto tiene un identificador (oid)[8] 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 *servicio*, pudiendo ser un *evento* o una *operación*. En el primer caso se trata de la abstracción de un cambio de estado atómico e instantáneo. Una operación es un servicio no atómico y que, en general, tiene duración.

La ocurrencia de un servicio en la vida de un objeto sólo tendrá éxito si el objeto se encuentra en un estado que verifica la *precondición* asociada a dicho servicio. 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.

En el modelo OASIS, podemos definir *transacciones*, de manera que un conjunto de servicios 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 servicios y transacciones como acciones atómicas, permitiéndonos especificar las posibles vidas correctas de los objetos.

---

[7] **O**pen and **A**ctive **S**pecification of **I**nformation **S**ystems
[8] En inglés, *object identifier*

Los objetos no están aislados. La interacción entre objetos se modela en OASIS mediante el concepto de *acción*. Una acción es una tupla <Cliente, Servidor, Servicio>, que representa tanto la acción asociada a requerir el servicio en el objeto cliente como la acción asociada a proveer el servicio en el objeto servidor. Las relaciones de disparo, representadas en el modelo OASIS por *triggers*, introducen actividad en la sociedad de objetos, permitiendo que un objeto actúe como cliente de una acción de otro objeto cuando se satisfagan en él determinadas condiciones.

Por otra parte, 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:

- 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 servicios, las restricciones de integridad, las precondiciones, las relaciones de disparo, las transacciones y el proceso que caracteriza las vidas de 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 clase. 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. Los operadores de clase son descritos en detalle en el capítulo 3 de (Letelier Torres, Sánchez Palma, Ramos Salavert, & Pastor López).

El modelo objetual descrito está soportado por un lenguaje denominado también OASIS. El lenguaje permite la definición de esquemas conceptuales según el modelo orientado a objetos presentado. Existen diferentes versiones del lenguaje dependiendo de la expresividad 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 (Pastor, 1992). La última versión es la 3.0 cuya definición completa y detallada puede encontrarse en (Letelier Torres, Sánchez Palma, Ramos Salavert, & Pastor López) y que está basada en una variante de la lógica dinámica que incorpora operadores de la lógica deóntica, o lo que es lo mismo, la lógica deóntica expresada como una variante de la lógica dinámica. En este trabajo, no utilizaremos ninguna notación específica sino que simplemente indicaremos cual es la relación entre los conceptos del metamodelo de flujo de trabajo propuesto y los conceptos del modelo de OASIS así como su relación con la lógica deóntica con la que se formaliza.

Así pues, a continuación mostramos la lógica utilizada y, a continuación, cómo representar el metamodelo de procesos con OASIS.

Marco Formal: Lógica Deóntica como variante de la Lógica Dinámica

De forma global, puede verse que OASIS, en particular la versión 3.0, está basado en la Lógica Deóntica (Aqvist, 1984). La Lógica Deóntica es la lógica de obligaciones, prohibiciones y permisos. Siendo A el conjunto de *acciones* con $a \in A$, se utilizan los siguientes operadores de la Lógica Deóntica:

$O(a)$	obligación de ocurrencia de a.
$F(a)$	prohibición de ocurrencia de a.
$P(a)$	permiso o habilitación de ocurrencia de a.

En (Meyer, 1988) la Lógica Deóntica es descrita como una variante de Lógica Dinámica (Harel, 1984). La Lógica Dinámica es un formalismo natural para estudiar de forma simple y directa ciertas aserciones sobre programas. En Lógica Dinámica, la fórmula $[a]\phi$ se interpreta intuitivamente como "después de la ocurrencia de la *acción* a, ϕ debe satisfacerse", siendo a una acción y ϕ una *Fórmula Bien Formada* de la Lógica de Primer Orden utilizada. La definición de los operadores deónticos en Lógica Dinámica es la siguiente:

$O(a) \Leftrightarrow [\neg a]false$	"la ocurrencia de a es obligatoria".
$F(a) \Leftrightarrow [a]false$	"la ocurrencia de a está prohibida".
$P(a) \Leftrightarrow \neg F(a)$	"a está permitida si y sólo si a no está prohibida".

Donde $\neg a$ representa la no-ocurrencia de la *acción* a (es decir, que no ocurre nada o que ocurre otra acción distinta de a). En estas fórmulas, $false$ es un átomo tal que no hay un estado que pueda hacerlo verdadero. El significado intuitivo de estas fórmulas es: una acción está prohibida si su ocurrencia conduce a un estado de violación. Una acción es obligatoria si su no-ocurrencia conduce a un estado de violación.

Siendo ψ una *Fórmula Bien Formada* en una lógica de primer orden que caracteriza el estado del objeto, utilizaremos los siguientes tipos de fórmulas en Lógica Dinámica:

$\psi \longrightarrow [a]false$	"la ocurrencia de a está prohibida en estados que satisfacen ψ.
$\psi \longrightarrow [\neg a]false$	"la ocurrencia de a es obligatoria en estados que satisfacen ψ".
$\psi \longrightarrow [a]\phi$	"en estados que satisfacen ψ, justo después de la ocurrencia de la acción a, ϕ debe satisfacerse".

Estas fórmulas constituyen un sublenguaje del lenguaje de Lógica Dinámica propuesto y formalizado en (Wieringa & Meyer, 1993). Las fórmulas en la lógica 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, w_0, ρ), donde: Ω es un conjunto de mundos posibles, siendo cada uno de ellos una estructura de interpretación de primer orden, w_0 es un mundo inicial, y ρ es la relación de accesibilidad entre mundos (ver Figura 8). 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, servicios, transacciones, etc., dependiendo del modelo utilizado).

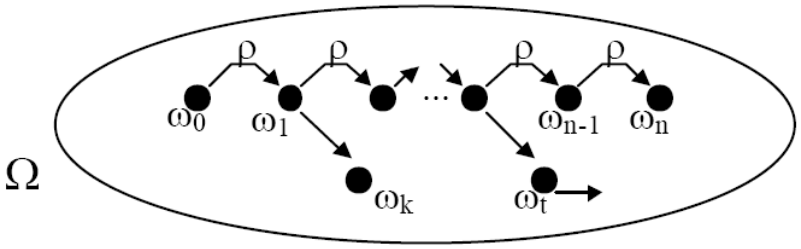

Figura 8 Estructura de Kripke simple

A continuación se presentan las fórmulas y especificaciones de proceso utilizadas en cada una de las secciones de un *tipo* de una plantilla de clase elemental en OASIS. Posteriormente se describe cómo el *tipo* completo se corresponde con fórmulas en la variante de Lógica Dinámica que se ha esbozado. Como veremos, la correspondencia es directa, salvo para especificaciones de *proceso*.

Fórmulas de especificación

Para describir las fórmulas usadas en cada sección de una plantilla de clase, consideraremos ϕ, ϕ' y ψ como *Fórmulas Bien Formadas* en la Lógica de Predicados de Primer Orden usada, con $a \in A$ (conjunto de acciones).

Evaluaciones

Estas fórmulas definen los cambios de estado ante la ocurrencia de acciones. Una *evaluación* se corresponde con la siguiente fórmula de la Lógica Dinámica utilizada:

$$\psi \longrightarrow [a]\phi$$

En este caso, ϕ es una *Fórmula Bien Formada* construida usando sólo átomos con el operador relacional de igualdad (=) y conectivas lógicas de conjunción (*and*). Con esta sintaxis es posible caracterizar explícitamente parte del estado de una instancia antes y después de la ocurrencia de una determinada *acción*.

Derivaciones

Las fórmulas usadas en derivaciones (información derivada) tienen la forma: $\phi \longrightarrow \phi'$. Una fórmula de derivación se debe satisfacer en cada estado de la instancia. ϕ' es una fórmula que expresa mediante una igualdad cómo se obtiene el valor del *atributo derivado* o *atributo* cuyo valor depende del valor de otros atributos. En Lógica Dinámica, las fórmulas para derivaciones pueden ser representadas como:

$$[a](\phi \longrightarrow \phi'), \forall a \in A.$$

Precondiciones

Son fórmulas en la variante de la Lógica Dinámica utilizada. Representan prohibiciones para la ocurrencia de *acciones*. Así, una precondición es definida por la fórmula:

$\psi \longrightarrow [a]false$ "si ψ se satisface entonces la ocurrencia de a está prohibida".

Los permisos (o habilitación de ocurrencia de acciones) constituyen una subespecificación, es decir, si una acción no está prohibida, entonces está permitida.

Por comodidad en la especificación, ψ será una fórmula del tipo $\neg\phi$, pues resulta más natural expresar "si $\neg\phi$ se satisface entonces a es prohibido". Así, la fórmula en Lógica Dinámica para precondiciones se escribe finalmente como:

$\neg\phi \longrightarrow [a]false$ "si $\neg\phi$ se satisface entonces la ocurrencia de a está prohibida".

Disparos

Los disparos permiten especificar actividad basada en el estado del objeto. Un disparo es una obligación y se define con la fórmula:

$\psi \longrightarrow [\neg a]false$ "si ψ se satisface entonces la ocurrencia de a es obligatoria".

Restricciones de integridad

Son *Fórmulas Bien Formadas* en Lógica Temporal usadas para representar restricciones de integridad. Obedecen a las siguientes reglas:

- Toda *Fórmula Bien Formada* en Lógica Temporal interpretada como $always\ \phi$. Además, $always\ \phi$, $sometimes\ \phi$, $next\ \phi$, $\phi\ since\ \phi'$ y $\phi\ until\ \phi'$ son también *Fórmulas Bien Formadas* en Lógica Temporal.
- Sea T un periodo de tiempo expresado usando las unidades de tiempo usuales: $seconds, minutes, hours, days, weeks, months$ o $years$, entonces:

$$sometimes_{op\_rel\ T}\ \phi\ since\ \phi'$$

$$always_{op\_rel\ T}\ \phi\ since\ \phi'$$

Ambas son *Fórmulas Bien Formadas*, siendo op_rel el operador realacional $<$ ó $\leq$ en la primera fórmula (para expresar un $timeout$) y los operadores $>$ ó $\geq$ en la última (para expresar un $delay$).

- Sólo son *Fórmulas Bien Formadas* en Lógica Temporal válidas para esta sección las construidas siguiendo las reglas anteriores.

Si ϕ es un *Fórmula Bien Formada* en la Lógica Temporal presentada, entonces desde la Lógica Dinámica, dicha fórmula es vista como un conjunto de fórmulas del tipo: $[a]\phi$, $(a \in A)$

Especificación de procesos

Existen tres semánticas posibles que pueden ser asociadas en la especificación de un proceso en OASIS:

a) La semántica del lenguaje utilizado para su especificación. En OASIS, el lenguaje utilizado para especificar procesos es un subconjunto de CCS (Milner, 1989), compartiendo así una semántica basada en la noción de sistema de transición etiquetado y todas las propiedades formales establecidas en CCS.
b) La semántica de permisos, es decir, una especificación de proceso establece secuencias de acciones que *pueden* ocurrir.
c) La semántica de obligaciones, es decir, una especificación de proceso establece secuencias de acciones que *deben* ocurrir.

Un proceso tendrá semántica (a) junto con la semántica (b) ó (c) dependiendo de si se trata de un *protocolo* o una *operación*, respectivamente. Así, la sección *operations* se utiliza para especificar secuencias obligatorias de acciones y la sección *protocols* para especificar secuencias permitidas de *acciones*. Un caso particular de *proceso de obligación* es el que actúa como una transacción, es decir, con la política de "todo o nada". En este case se puede declarar con el calificativo de *transaction* para el proceso. Asumiremos siempre por defecto que el proceso no actúa como transacción.

Siguiendo el planteamiento y notación propuestos en CCS, la especificación de un proceso R puede ser vista como un sistema de transiciones etiquetado representado por $\left(\kappa_R, A_R, \left\{\overset{a}{\rightarrow} : a \in A_R\right\}\right)$, donde:

- κ_R es el conjunto de constantes agente[9] utilizados para definir el proceso R.

[9] Un proceso de OASIS corresponde al concepto de agente de CCS. Un agente está representado por una constante agente (su nombre). En la definición de una constante agente se pueden utilizar otras constantes agente.

- A_R es el conjunto de *acciones* usadas en la especificación del proceso R, con $A_R \subseteq A$, y siendo A el conjunto de *acciones*[10] en la signatura del flujo de trabajo.
- $\xrightarrow{a} \subseteq \kappa_R \times \kappa_R$ es la relación de transición.

Para cada constante agente $E \in \kappa_R$ se tiene una ecuación de definición $E \stackrel{\text{def}}{=} Q$, siendo Q una expresión constuida usando los siguiente operadores (con $S \in \kappa_R$ y a, a_1 y a_2 acciones pertenecientes al conjunto A_R):

1. $a.S$, es un prefijo
2. $if\ C\ then\ a.S$, es un condicional, siendo C una expresión $boolean$.

   Extensiones del tipo $if\ C\ then\ a_1.S_1\ else\ a_2.S_2$ pueden ser definidas reescribiéndolas como $(if\ C\ then\ a_1.S_1) + (if\ \neg C\ then\ a_2.S_2)$.

3. $\sum_{i\in I} Q_i'$, donde Q_i' puede obtenerse por alguno de los operadores anteriores y $\sum_{i\in I}$ es la suma con I como conjunto de indexación.

Existe una constante adicional e implícita denotada por 0 y definida como $0 = \sum_{i\in I} a_i.S_i$, con $I = \emptyset$. Por lo tanto, desde 0 no existen transiciones posibles.

Ejemplo 1 Sea un proceso P especificado como:

$$P \stackrel{\text{def}}{=} a.P_1$$
$$P_1 \stackrel{\text{def}}{=} b.P_2 + e.P_3$$
$$P_2 \stackrel{\text{def}}{=} c.P_2 + b.P_2 + d.P_1$$
$$P_3 \stackrel{\text{def}}{=} 0$$

El sistema de transiciones para el proceso P es:

$$\kappa_P = \{P, P_1, P_2, P_3\}$$

$$A_P = \{a, b, c, d, e\}$$

$$\xrightarrow{a} = \{(P, P_1)\}$$
$$\xrightarrow{b} = \{(P_1, P_2), (P_2, P_2)\}$$
$$\xrightarrow{c} = \{(P_2, P_2)\}$$
$$\xrightarrow{d} = \{(P_2, P_1)\}$$
$$\xrightarrow{e} = \{(P_1, P_3)\}$$

[10] Incluye tanto las *acciones* generadas desde otros objetos como las *acciones* generadas por el propio objeto. Es decir, la instancia vista como cliente y servidor.

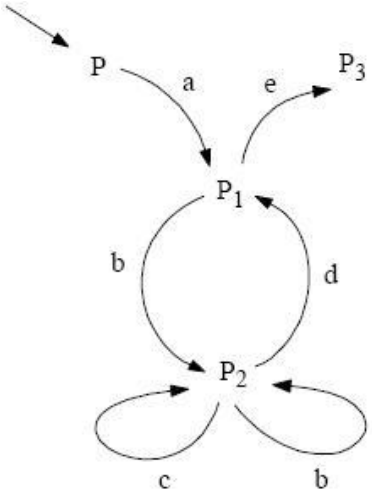

Figura 9 Grafo de transiciones del proceso P.

Un *proceso* puede ser representado por un grafo de transiciones. La Figura 9 muestra el grafo de transiciones para el proceso P del Ejemplo 1.

Llamaremos *estado del proceso* a la constante agente que en un determinado instante representa al proceso y que se corresponde con un nodo del grafo de transiciones del proceso.

En el tipo de un flujo de trabajo, para cada *proceso* R existe implícitamente un atributo variable de $sort\ \kappa_R$ con el mismo nombre del *proceso*. Dicho atributo representa el *estado del proceso* (la constante agente o nodo del grafo de transiciones asociado) en el cual se encuentra el *proceso*. Como veremos, cuando la instancia está en el estado 0 en un determinado *proceso*, el comportamiento de la instancia no está siendo afectado por el *proceso* (el *proceso* no determina ni permisos ni obligaciones para la instancia).

Es importante destacar que un mismo *estado del proceso* puede estar asociado a más de un estado de la instancia (un σ distinto).

Procesos como fórmulas de Lógica Dinámica

Como se ha expuesto, los operadores de la Lógica Deóntica que expresan obligaciones, prohibiciones y permisos pueden ser vistos como una variante de Lógica Dinámica. Esto sugiere que los procesos de obligación y prohibición se corresponden del mismo modo con fórmulas en Lógica Dinámica.

Una especificación de *proceso* determina una conducta particular de la instancia de flujo de trabajo. Diremos que *un proceso afecta a una instancia* si la instancia debe comportarse según la conducta establecida por dicho *proceso*. Comúnmente, la instancia seguirá la conducta del *proceso* en partes de su *traza de ejecución*. Un caso particular es un *proceso* que afecta a la instancia durante toda su existencia.

Un *proceso* que establece permisos se puede expresar como un conjunto de fórmulas en Lógica Dinámica de la forma $\neg\phi \longrightarrow [a]false$, del mismo modo que se indicó para las fórmulas de *precondiciones*.

De forma análoga, un *proceso* que se establece por obligaciones se puede expresar como un conjunto de fórmulas en Lógica Dinámica de la forma $\psi \longrightarrow [\neg a]false$, tal como se indicó para las fórmulas de *disparos*.

Como veremos, en ambos casos se obtendrán adicionalmente fórmulas por transición de estado del proceso. Estas fórmulas son del mismo tipo usado para *evaluaciones*, es decir, $\psi \longrightarrow [a]\phi$.

Sea R un proceso definido por el sistema de transiciones $\left(\kappa_R, A_R, \left\{\overset{a}{\rightarrow} : a \in A_R\right\}\right), E,\ E' \in \kappa_R$. Si $(E, E') \in \overset{a}{\rightarrow}$, esto suele escribirse como $E \overset{a}{\rightarrow} E'$ y se dice que a es una *acción* desde E.

Ejemplo 2 Utilizaremos el proceso P presentado en el ejemplo 3. La relación de transición de P es:

$$\overset{a}{\rightarrow} = \{(P, P_1)\}$$
$$\overset{b}{\rightarrow} = \{(P_1, P_2), (P_2, P_2)\}$$
$$\overset{c}{\rightarrow} = \{(P_2, P_2)\}$$
$$\overset{d}{\rightarrow} = \{(P_2, P_1)\}$$
$$\overset{e}{\rightarrow} = \{(P_1, P_3)\}$$

Por lo tando, en P:

a es una acción desde P.
b es una acción desde P_1 y desde P_2.
c es una acción desde P_2.
d es una acción desde P_2.
e es una acción desde P_1.

Sea r el campo variable asociado al estado del proceso R. Si a es una acción desde un estado E del proceso, llamaremos *predicado de ocurrencia de a en E* al predicado $\left((r = E) \wedge C\right)$ y lo denotaremos por Ω_{Ea}, donde C es una

expresión $boolean$ si para (E, E') está definida. De lo contrario, el predicado de ocurrencia de a en E es sólo $(r = E)$. Por defecto, si a no es una acción desde E, entonces $\Omega_{Ea} = false$.

Ejemplo 3 Los predicados de ocurrencia de cada acción en cada estado del proceso P son:

$$\Omega_{Pa} = (p = P)$$
$$\Omega_{P_1 b} = (p = P_1)$$
$$\Omega_{P_2 b} = (p = P_2)$$
$$\Omega_{P_2 c} = (p = P_2)$$
$$\Omega_{P_2 d} = (p = P_2)$$
$$\Omega_{P_1 e} = (p = P_1)$$

Para el resto de combinaciones entre estado del proceso y acciones, el predicado de ocurrencia es $false$.

Ampliando el concepto anterior al proceso como un todo, llamaremos *predicado de ocurrencia de a en R*, denotado por Ω_{R*a}, a la disyunción de predicados de ocurrencia de a desde E, para cada constante agente E que define al proceso R.

Ejemplo 4 Los predicados de ocurrencia en P para cada una de sus acciones, son:

$$\Omega_{P*a} = (p = P)$$
$$\Omega_{P*b} = \big((p = P_1) \vee (p = P_2)\big)$$
$$\Omega_{P*c} = (p = P_2)$$
$$\Omega_{P*d} = (p = P_2)$$
$$\Omega_{P*e} = (p = P_1)$$

El significado de Ω_{R*a} es el siguiente: "Ω_{R*a} se satisface si y sólo si el proceso R está en algún estado de proceso E y algún Ω_{Ea} se satisface, es decir, a es una *acción* desde E y además se cumple la condición C asociada (si ésta existe)".

Así un proceso R definido por el sistema de transiciones $\left(\kappa_R, A_R, \left\{\overset{a}{\rightarrow} : a \in A_R\right\}\right)$, se corresponde con las fórmulas en Lógica Dinámica presentadas a continuación.

Definición 1 Fórmulas por transiciones. Efectúan la evolución del proceso de acuerdo con las transiciones definidas y con la ocurrencia de acciones. Para cada $a \in A_R$, y para cada $(E, E') \in \overset{a}{\rightarrow}$ se obtiene la siguiente fórmula:

$$\Omega_{Ea} \longrightarrow [a](r = E')$$

Ejemplo 5 Las fórmulas por transiciones para el proceso P son:

$(p = P) \longrightarrow [a](p = P_1)$
$(p = P_1) \longrightarrow [b](p = P_2)$
$(p = P_2) \longrightarrow [b](p = P_2)$
$(p = P_2) \longrightarrow [c](p = P_2)$
$(p = P_2) \longrightarrow [d](p = P_1)$
$(p = P_1) \longrightarrow [e](p = P_3)$

El grafo de alcanzabilidad que determinan estas transiciones corresponde al grafo de transiciones mostrado en la Figura 9.

Además, siendo a_{new} la *acción* de creación para instancias de la clase, entonces:

a) Si R es un *proceso* que establece obligaciones entonces, adicionalmente, existirá la fórmula:

$$[a_{new}](r = 0)$$

Es decir, asumimos que un *proceso* de obligación no afecta a la instancia desde el comienzo de su existencia.

b) Si R es un *proceso* que establece permisos, la fórmula adicional es:

$$[a_{new}](r = E_1)$$

Donde $E_1 \in \kappa_R$ es la constante agente de inicio del proceso.

Es decir, asumimos que un *proceso* de permiso afecta a la instancia desde el comienzo de su existencia.

De acuerdo con el estado del proceso, una *fórmula de permiso* (sólo si R es un proceso que establece permisos) establece la situación en la cual la ocurrencia de una acción se permite. Para cada $a \in A_R$ se obtiene la siguiente fórmula:

$$\neg\Omega_{R*a} \longrightarrow [a]false.$$

Ejemplo 6 Si en el ejemplo tratado el proceso P fuera un proceso que establece secuencias de acciones permitidas, las fórmulas de permiso asociadas serían:

$\neg(p = P) \longrightarrow [a]false$
$\neg\big((p = P_1) \vee (p = P_2)\big) \longrightarrow [b]false$
$\neg(p = P_2) \longrightarrow [c]false$
$\neg(p = P_2) \longrightarrow [d]false$
$\neg(p = P_1) \longrightarrow [e]false$

De acuerdo con el estado del proceso, una fórmula de obligación (sólo si R es un proceso que establece obligaciones) establece la situación en la cual la ocurrencia de una acción es obligatoria. Para cada $a \in A_R$ se obtiene la siguiente fórmula:

$$\Omega_{R*a} \longrightarrow [\neg a] false.$$

Ejemplo 7 Si en el ejemplo tratado el proceso P fuera un proceso que establece secuencias de acciones obligatorias, las fórmulas de obligación asociadas serían:

$$(p = P) \longrightarrow [\neg a] false$$
$$\big((p = P_1) \vee (p = P_2)\big) \longrightarrow [\neg b] false$$
$$(p = P_2) \longrightarrow [\neg c] false$$
$$(p = P_2) \longrightarrow [\neg d] false$$
$$(p = P_1) \longrightarrow [\neg e] false$$

Aspectos adicionales

Manteniendo el marco formal establecido, en este apartado se presentan aspectos adicionales que extienden la capacidad expresiva que puede ser usada en especificaciones de *proceso*.

Tratamiento de pasos y operaciones

Un *paso* es un conjunto consistente de *acciones* que acontecen en un instante dado en la traza de ejecución de una instancia. Una *acción* está asociada a un *servicio*. El *servicio* puede ser un *evento* o una *operación*. El *evento* no tiene duración, es instantáneo. En cambio, una *operación* establece una secuencia obligatoria de *acciones* y, en general, tiene una duración.

El cliente no tiene porqué conocer si un *servicio* corresponde a un *evento* o una *operación* en el servidor. Por esto, en la *acción* acontecida en el cliente, y que refleja la solicitud del *servicio*, no se hace referencia alguna que distinga entre solicitar un *evento* o solicitar una *operación*.

En el servidor, el tratamiento para servir una *operación* es distinto al dado cuando el *servicio* es un *evento*. Al proveer *operaciones* consideraremos lo siguiente:

- Se asume que las *precondiciones* o *evaluaciones* (si existen) asociadas a una *operación* son *precondiciones* o *evaluaciones* para el *evento* de inicio de la *operación*.
- Para asegurar la correcta realización de una *operación*, mientras una operación esté en ejecución, no deben servirse otros *eventos* ni *operaciones* que hagan incompatible el concepto de *paso* (conjunto

consistente de acciones) con la obligatoriedad de ejecución de las *acciones* de la *operación*[11].

Una *operación* puede ser vista como una secuencia de *acciones* (las que determina su especificación) más dos *acciones* implícitas: la *acción* de inicio de la *operación* y la acción de término de la *operación*. La *acción* de inicio tiene siempre un *evento* requerido por un determinado cliente (quien solicita la *operación*). La *acción* de término tiene siempre un *evento* requerido a sí mismo, obligado e inmediatamente posterior a la ejecución de la última *acción* ejecutada por la *operación*.

El mantener implícito las *acciones* de inicio y término de *operación* simplifica la especificación. Al mismo tiempo, apoya la idea de que tanto *eventos* como *operaciones* sean tratados en la especificación bajo un concepto más amplio; el concepto de *servicio*. A continuación, se abordará el tratamiento de *operaciones*, sólo para mostrar la correspondencia con la formalización de *procesos* ya realizada (en la que no se discutió el tratamiento de *operaciones*).

Sea op una *operación* definida para la instancia. Denotaremos por $op^{\blacksquare}$ a la *acción* de inicio de la *operación* y por $op^{\odot}$ a la *acción* de término de la *operación*.

Si existiera una *precondición* como: $\neg\phi \longrightarrow [op]false$, se interpretaría implícitamente como una *precondición* para la acción de *inicio* de la *operación*, es decir, $\neg\phi \longrightarrow [op^{\blacksquare}]false$. Del mismo modo, una *evaluación* del tipo $\psi \longrightarrow [op]\phi$ se interpreta como una *evaluación* asociada a la *acción* de inicio de la *operación*, es decir, $\psi \longrightarrow [op^{\blacksquare}]\phi$.

Según lo anterior, el tratamiento de *operaciones* se reduce a la ejecución de las *acciones* de la *operación*-más las dos *acciones* implícitas de inicio y fin de la *operación*.

Queda por asegurar que al permitirse concurrencia intra-instancia (de *eventos* y *operaciones*), las *acciones* de un *paso* sean siempre consistentes y a la vez se cumpla la obligatoriedad de las *acciones* de una *operación* en ejecución.

La no-consistencia entre *acciones* es difícil de establecer. Sin embargo, la consistencia se cumple siempre si las *acciones* no están en conflicto[12]

[11] El problema es análogo al de control de concurrencia de transacciones, ampliamente tratado en bases de datos relacionales. Nótese que la definición previa de *paso* era aplicable sólo para *acciones* ejecutadas en un mismo instante. Este concepto debe ser extendido para considerar cierta duración asociada a la ejecución de una operación.

(Letelier, Sánchez, & Ramos, 1997). El conjunto de *campos* sobre el cual puede tener efecto una *operación* está determinado por la unión de los conjuntos de *campos* sobre los cuales tiene efecto las *acciones* incluidas en la *operación*.

Asociado a cada *operación*, existe un atributo implícito de la instancia que determina en qué estado de la *operación* se encuentra la instancia. Si la *operación* no está en ejecución entonces dicho *campo* toma el valor 0. Siendo *op* el c*ampo* que registra el *estado del proceso op*, entonces existe una evaluación implícita del tipo $[op^{\odot}](op = 0)$. Esta *evaluación* hace que inmediatamente después de ejecutarse la *acción* $op^{\odot}$ (*acción* de término del *proceso*), el *proceso op* se encuentre en su *estado de proceso* 0, es decir, el *proceso* vuelve a una situación de inactividad.

Por lo tanto, para asegurar que los *eventos* y *operaciones* (tratadas como *acciones*) de un *paso* no estén en conflicto, además de verificar que no estén en conflicto entre ellas, debe verificarse que no estén en conflicto con ninguna *operación* en ejecución (cuyo valor de atributo asociado es diferente a 0).

Capacidad para especificar procesos anidados

Cuando las especificaciones de *proceso* son extensas (muchos *estados de proceso*) es conveniente disponer de algún mecanismo de descomposición. Así, un *proceso* puede ser descompuesto en *subprocesos anidados*[13] especificados separadamente. Esto a su vez promueve la reutilización de especificaciones de *subprocesos* desde distintos *procesos* definidos en el tipo de un *flujo de trabajo*.

Ejemplo 8 Consideremos la especificación de un proceso L. Por simplicidad, supondremos que las acciones a_1 y a_2 son acciones cuyo servicio es un evento.

$$L \stackrel{\text{def}}{=} a_1.L_1$$
$$L_1 \stackrel{\text{def}}{=} a_2.L_1 + Q^{\blacksquare}.Q$$
$$L_2 \stackrel{\text{def}}{=} a_1.L_3$$
$$L_3 \stackrel{\text{def}}{=} 0$$
$$Q \stackrel{\text{def}}{=} a_2.Q_1 + a_1.Q_2$$
$$Q_1 \stackrel{\text{def}}{=} a_1.Q + a_2.Q_2$$
$$Q_2 \stackrel{\text{def}}{=} Q^{\odot}.L_2$$

[12] Dos *acciones* no están en conflicto si tienen efecto sobre conjuntos disjuntos de *campos*. Este criterio es bastante restringido respecto de la concurrencia intra-instancia. Sin embargo, es fácil de determinar: basta inspeccionar *evaluaciones* asociadas a las *acciones* y posibles *derivaciones de campos*.

[13] Por anidamiento entendemos la capacidad expresiva que permite que un *proceso* en su definición haga referencia a otro *proceso* como una forma de *transición compleja*, definida por otro *proceso*.

$Q^{\blacksquare}$ y $Q^{\odot}$ son dos acciones. Dichas acciones pueden verse como las acciones de inicio y de término para un cierto proceso (Q)*, respectivamente.*

Esto sugiere reescribir L como:

$L \stackrel{\text{def}}{=} a_1.L_1$
$L_1 \stackrel{\text{def}}{=} a_2.L_1 + q.L_2$
$L_2 \stackrel{\text{def}}{=} a_1.L_3$
$L_3 \stackrel{\text{def}}{=} 0$

Donde q es una acción que incluye al servicio no atómico Q. Así, el proceso L es definido utilizando la especificación del proceso Q, reescrito como:

$Q \stackrel{\text{def}}{=} a_2.Q_1 + a_1.Q_2$
$Q_1 \stackrel{\text{def}}{=} a_1.Q + a_2.Q_2$
$Q_2 \stackrel{\text{def}}{=} 0$

El proceso Q podría entonces ser utilizado en otras definiciones de proceso. A su vez, el proceso Q también podría utilizar otras definiciones de proceso.

Así, la justificación de este anidamiento de especificaciones de *proceso* se obtiene estableciendo la correspondencia de la especificación anidada con una equivalente sin anidamiento.

Plantilla de clase en Lógica Dinámica

Después de lo presentado, expresar la plantilla de una clase OASIS como un conjunto de fórmulas de Lógica Dinámica es inmediato. Siendo $\langle Campos|Eventos|Fórmulas|Procesos\rangle$ el tipo de un flujo de trabajo, el conjunto de fórmulas en Lógica Dinámica para el flujo de trabajo es el siguiente:

i. Para cada *evaluación* existirá una *fórmula de cambio de estado*:

$$\psi \longrightarrow [a]\phi$$

ii. Para cada $a \in A$ y para cada *fórmula de derivación* $(\phi \dashrightarrow \phi')$ existirá una fórmula del tipo:

$$[a](\phi \dashrightarrow \phi')$$

iii. Para cada *precondición* existirá una *fórmula de permiso*:

$$\neg\phi \longrightarrow [a]false$$

iv. Para cada *disparo* existirá una *fórmula de obligación*:

$$\phi \longrightarrow [\neg a] false$$

v. Para cada[14] $a \in A$ y siendo φ la conjunción de todas las *fórmulas de restricciones de integridad* existirá una fórmula del tipo:

$$[a]\varphi$$

vi. Para cada *proceso de obligación* R, definido por $\left(\kappa_R, A_R, \left\{\overset{a}{\rightarrow} : a \in A_R\right\}\right)$

- *Fórmulas por transiciones*

  Para cada $(E, E') \in \overset{a}{\rightarrow}$ la fórmula: $\Omega_{\mathrm{E}a} \longrightarrow [a](r = E')$.

  Además, la fórmula: $[a_{new}](r = 0)$.

- *Fórmulas por obligaciones*

  Para cada $a \in A_R$ la fórmula: $\Omega_{\mathrm{R}*a} \longrightarrow [\neg a] false$.

vii. Para cada *proceso de permiso* R, definido por $\left(\kappa_R, A_R, \left\{\overset{a}{\rightarrow} : a \in A_R\right\}\right)$

- *Fórmulas por transiciones*

  Para cada $(E, E') \in \overset{a}{\rightarrow}$ la fórmula: $\Omega_{\mathrm{E}a} \longrightarrow [a](r = E')$.

  Además, la fórmula: $[a_{new}](r = E_1)$, $E_1 \in \kappa_R$ es el estado de inicio del proceso.

- *Fórmulas por permisos*

  Para cada $a \in A_R$ la fórmula: $\neg\Omega_{\mathrm{R}*a} \longrightarrow [a] false$.

Representación OASIS del metamodelo de flujos de trabajo

La primera cuestión que se plantea a la hora de formalizar en OASIS el metamodelo de flujo de trabajo definido al principio de este capítulo 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 cualquier tipo de sistema y, en concreto, de flujos de trabajo. En concreto, la visión proceso y los distintos mecanismos para la especificación de características del estado y sus cambios en el tiempo permiten especificar flujos de trabajo sin añadir ninguna característica nueva (Penadés, 1999) (Penadés & Canós, Modelado Conceptual de Flujos de Trabajo, 2000).

[14] Que ocurre fuera del contexto de una *transacción* o que representa la *acción* de fin de *proceso* de la *transacción*

Así pues, 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, 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 subsiten 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 especialziación temporal y permanente. La siguiente figura muestra el modelo OASIS del metamodelo de flujo de trabajo propuesto (se ha utilizado notación UML pero las clases no son las del modelo orientado a objetos sino que representan clases OASIS):

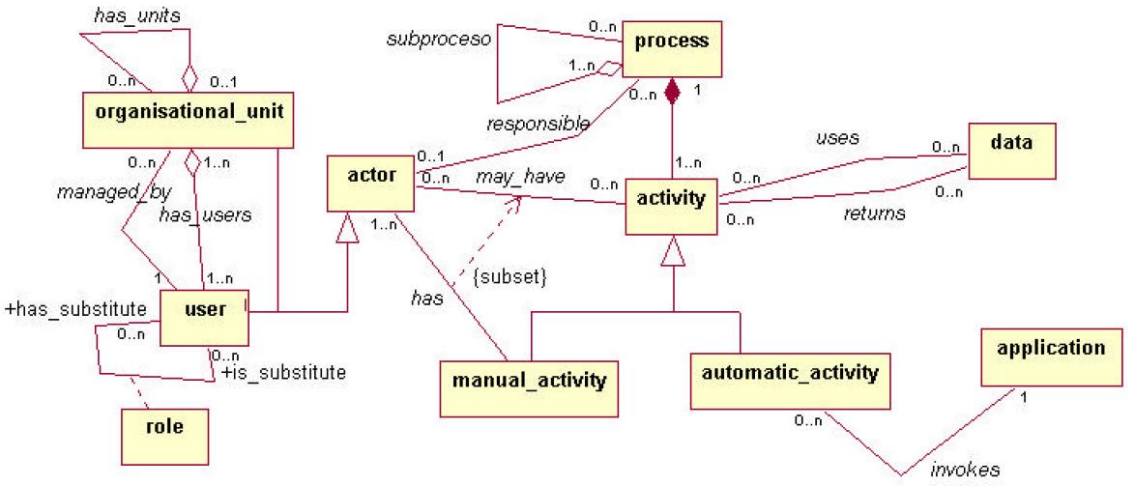

Figura 10 Modelo OASIS del metamodelo de flujo de trabajo de referencia

Al representar en OASIS el metamodelo de referencia, podemos observar que un flujo de trabajo puede estar compuesto por cero o más subprocesos y por cero o más actividades. Esto supone que un flujo de trabajo que representa parte de un proceso 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 flujos de trabajo).

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 tareas del flujo de trabajo así como la definición del orden de ejecución de las tareas queda dentro de la definición del propio proceso. Esto se detalla más en el capítulo siguiente.

A continuación describimos en más detalle las clases *process*, *activity*, *application*, *data* y *actor* del modelo OASIS para el metamodelo de flujos de trabajo de referencia.

La clase Process

En OASIS, un flujo de trabajo o proceso es un objeto activo, es decir, un programa, modificado y enriquecido con un conjunto de propiedades que comentamos a continuación.

Un programa OASIS representa un objeto activo, es decir, la vida de tal objeto siempre comenzará por un evento de creación (start) y terminará por uno de destrucción (stop) y entre ambos, el código o cuerpo (body) del programa contendrá los servicios a ejecutar constituyendo la vida del programa.

Así pues, a partir de la clase abstracta *program* se pueden definir programas con un conportamiento determinado (rellenando su conjunto de servicios a ejecutar) y cada instancia de dicha clase representará una ejecución distinta del mismo. Cada vez que un agente solicita a la clase *program* hija 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 cuerpo de la clase, va a actuar de agente y ejecutar el cuerpo del programa. Una vez finaliza, el propio objeto se destruye.

La estructura de un proceso, a partir de la clase *program*, 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, un nombre y una descripción; y como atributos variables, uno denominado estado cuyo valor cambiará para reflejar el estado del proceso según los eventos que vayan ejecutándose y de acuerdo con los posibles estados en los que se puede encontrar un proceso y un conjunto de variables que almacenarán los datos manipulados por una o varias actividades del 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 para 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.

En el siguiente capítulo se definirá cómo especificar un orden de ejecución entre el conjunto de actividades o subprocesos específicos que forman un proceso.

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 elemental. La plantilla de especificación, sin embargo coincide.

El apartado *processes* de la especificación define el comportamiento activo del objeto que representa la actividad, al igual que ocurre con los procesos. Su especificación coincide con la de la clase proceso, y al igual que antes, definir una actividad concreta supone definir las acciones que implica la ejecución de dicha actividad sólo que, 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 representan como especializaciones estáticas de la clase *activity* con nombres *manual_activity* y *automated_activity* respectivamente.

La aplicación que invoca una actividad auomática 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.

Por último, destacar que 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.

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. 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)*[15] y cuando este evento finalice, significa que la ejecución de la aplicación ha finalizado. En este mismo momento se dispara el evento *result_app(Result)* que almacena en el objeto el valor de retorno de la aplicación.

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.

[15] 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.

En este caso no se trata de objetos activos por lo que no hay predefinido una sección *processes,* aunque sí que se puede definir dependiendo de las características específicas del tipo de datos concreto.

La clase Actor

La clase *actor* es la generalización de la clase *user* y de la clase *organisational_unit*. Destacar el hecho de que no se trata de objetos, con las mismas consideraciones tenidas en cuenta para el caso de los datos.

Resumen

En este capítulo hemos presentado el metamodelo de referencia en lo que respecta a los conceptos fundamentales compartidos por todas las aproximaciones para la especificación de procesos, es decir, sin considerar la parte relativa a la especificación del orden de ejecución de las distintas tareas que lo componen, que se desarrolla en detalle en el siguiente capítulo.

También se ha presentado la formalización del metamodelo haciendo uso de los resultados obtenidos en OASIS en los que se utiliza la lógica deóntica para representar el comportamiento de un sistema como conjunto de fórmulas en una variante de la lógica dinámica.

En el siguiente capítulo completaremos el metamodelo y su formalización en lo que respecta a la especificación del orden de ejecución de las tareas, principal contribución de este trabajo ya que constituye el marco flexible para la especificación de procesos.

CAPÍTULO 4 – ESPECIFICACIÓN FLEXIBLE DE PROCESOS

Este capítulo define las distintas alternativas que ofrece el metamodelo de referencia para la especificación de procesos propuesto por este trabajo para la especificación del orden de ejecución de las tareas. El hecho de disponer de distintas alternativas, integradas en un marco común, para la especificación del orden de ejecución de las tareas proporciona la flexibilidad en tiempo de diseño de nuestra propuesta.

En primer lugar se presentan las alternativas basadas en dos dimensiones para la especificación del orden de ejecución de las tareas de un proceso y a continuación como queda reflejado y formalizado dicho orden en el metamodelo de referencia y el formalismo presentados en el capítulo anterior.

Captura de requisitos basada en 4 cuadrantes

En términos de expresividad, consideramos que el orden de ejecución de las tareas de un flujo de trabajo puede especificarse desde dos perspectivas complementarias. En cada instante pueden existir tareas obligadas de ejecutar, así como tareas cuya ejecución está permitida o prohibida. Además, el orden de ejecución de las tareas obligatorias y permitidas o prohibidas puede establecerse de la siguiente forma:

- Considerando el estado del proceso en dicho instante. Así, por ejemplo, la ejecución de una tarea en ciertos estados podría verse impedida o permitida por haberse especificado una prohibición o un permiso respectivamente. De forma análoga, en determinados estados del proceso la ejecución de una tarea podría estar obligada si se especifica una obligación.
- Considerando un orden predeterminado para la ejecución de las tareas. Una especificación de proceso permite expresar un ordenamiento de las tareas acontecidas, es decir, determina trazas posibles. Sin embargo, una especificación de proceso puede tener dos lecturas semánticas, determinando las trazas de tareas que durante la ejecución del proceso son ejecutadas obligatoriamente o, por otra parte, las trazas de ejecución de tareas que es posible o no ejecutar.

De esta forma, conseguimos cuatro perspectivas complementarias para especificar el orden de ejecución de las tareas de un flujo de trabajo, las cuales se representan por cada uno de los cuadrantes de la siguiente figura y se describen en más detalle a continuación.

Figura 11 Cuadrantes de especificación del orden de ejecución de las tareas de un flujo de trabajo

Cuadrante I: Proceso de Obligación

Se establece un proceso en el cual las acciones deben ejecutarse de manera obligatoria. Esta es la expresividad que más se acerca a la forma clásica usada en la especificación de procesos como el de la Figura 3.

Cuadrante II: Regla de Obligación

Cuando una condición sobre el estado del proceso determina un "salto" obligado en la secuencia de acciones que se deben ejecutar, una especificación de proceso como la del cuadrante I puede resultar poco adecuada. Frecuentemente se requiere esta expresividad para representar flujos de control alternativos durante la ejecución del proceso en cuyo caso se interrumpe el flujo normal para establecer un nuevo estado del proceso. En esta situación lo más apropiado es especificar una regla de obligación que complemente el proceso especificado por el cuadrante I. Por ejemplo, usando la información contenida en la Figura 3, podríamos añadir una regla para representar lo siguiente: "Si se detectan llamas y humo intenso avisar a los bomberos", sin tener que indicar explícitamente en cada actividad este comportamiento excepcional.

Cuadrante III: Proceso permisos/prohibiciones

En determinados estados de la gestión de una emergencia pueden existir secuencias de acciones que pueden ejecutarse o que están prohibidas. El cuadrante III permite especificar dichas secuencias como complemento al proceso del cuadrante I, pero con una semántica distinta, la cual establece secuencias de acciones válidas pero cuya ejecución no es obligatoria. Por ejemplo, en la especificación de la Figura 3, antes de ejecutar la actividad Realizar Reconocimiento

podría recomendarse recabar información histórica de incidencias en la zona de la emergencia y hacerla llegar al equipo de reconocimiento.

Cuadrante IV: Reglas de permiso/prohibición

Estas reglas permiten especificar de forma sencilla y flexible en qué estados de la respuesta a la emergencia está permitida o prohibida una acción. De este modo se puede asegurar condiciones esenciales que siempre deben cumplirse y que podrían no estar garantizadas con las especificaciones en los otros cuadrantes o condiciones bajo las cuales están permitidas ciertas acciones (sin ser obligatoria su ejecución). Por ejemplo, en la Figura 3, se podría añadir la siguiente regla de prohibición: "No apagar los ventiladores si se encuentra en el lugar el equipo de reconocimiento"

Así, una especificación de proceso o, siguiendo con el ejemplo utilizado en este trabajo, un Plan de Emergencia podrá contener una combinación de las expresividades ofrecidas por cada cuadrante. A continuación presentamos cómo se completa el metamodelo de proceso descrito en el capítulo anterior para dar soporte a la especificación del orden de ejecución de las tareas según cualquier de los cuadrantes expresivos presentados.

Metamodelo de Proceso de Referencia (con transiciones)

Para dar soporte a la expresividad anterior, en lo que respecta al orden de ejecución de las tareas, el metamodelo de proceso de referencia definido en el capítulo anterior debe ser completado con el concepto de transición entre tareas.

Una transición ordena dos o más tareas de un proceso, especificando la secuencia correcta de ejecución. Está formada por tres elementos:

- **Condición de inicio**. La condición de inicio es una expresión sobre el estado del proceso que determina el origen de la transición. Tras un cambio de estado del proceso, provocado por un cambio de estado de alguna de sus actividades o por la ejecución de un servicio de cambio de estado, el SGFT busca qué transiciones tienen como condición de inicio una expresión que se satisface en el estado actual.
- **Destino de la transición**. El destino de la transición es la tarea que debe ejecutar el proceso tras hacer efectiva la transición.
- **Tipo de transición**. Como hemos visto en la sección anterior, una transición puede ser obligatoria, permitida o prohibida. El tipo de transición determina este comportamiento. Así pues, en el caso de una transición de tipo obligatorio, tras hacer efectiva la transición se iniciará la ejecución de la tarea destino. En el caso de una transición

de tipo permitido, tras hacer efectiva la transición se permitirá la ejecución de la tarea destino (bajo demanda). En el caso de una transición de tipo prohibido, tras hacer efectiva la transición se lanzará una excepción en caso de que se intente iniciar la ejecución de la actividad por parte del usuario, es decir, se pasará automáticamente al estado *terminated* que representa una ejecución sin éxito de la actividad.

Para aquellos instantes en los que tras la finalización de una actividad o un cambio de estado se cumpla más de una condición de inicio de transición, por defecto, el SFGT ejecuta cada una de las transiciones en hilos de ejecución separados. Por otra parte, para aquellas actividades que son destino de varias transiciones, por defecto, el SFGT ejecutará la actividad destino cada vez que se haga efectiva una de esas transiciones. Sin embargo es posible modificar este comportamiento por defecto en la definición de las actividades.

Así pues, en la definición de una actividad se pueden añadir *condiciones de transición* que permiten establecer distintas alternativas o caminos a seguir para los casos en los que tras su ejecución se cumplan las condiciones de inicio de varias transiciones o sea destino de varias transiciones. En nuestro metamodelo, a las primeras las llamaremos condición de bifurcación y a las segundas condición 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 una o más tareas. El comportamiento general de la condición de bifurcación es el siguiente: cuando finaliza la ejecución de la tarea que tiene asociada la condición de bifurcación, se decide qué transiciones son válidas, y en consecuencia, qué actividades inician su ejecución. Se distinguen tres tipos de condiciones de bifurcación:

- AND (por defecto): se ejecutan todas las transiciones de forma concurrente y en hilos separados.
- XOR: se ejecuta la primera transición válida, descartando el resto. Sólo es válida para transiciones cuya ejecución es opcional.

Condiciones de unión

Son puntos de reunión dentro del proceso, en los que, para ejecutar el siguiente paso dentro del proceso, es necesario que hayan finalizado uno o más pasos previos que se están ejecutando de forma concurrente. Las condiciones de unión, por lo tanto, tiene el papel de puntos de sincronización

dentro del proceso. El comportamiento general de la condición de unión es el siguiente: cada vez que se cumple una transición a la actividad que tiene asociada la condición de unión, se decide si la actividad es iniciada en función del tipo de condición de unión. Se distinguen tres tipos de condiciones de unión:

- AND: la actividad no se inicia (o no está permitida) hasta que todas las transiciones que la tienen como destino son efectivas.
- OR (por defecto): la actividad se inicia (o está permitida) cada vez que una transición que la tiene como destino es efectiva.
- XOR: la actividad se inicia (o está permitida) sólo para la primera transición que la tiene como destino y es efectiva, descartando el resto.

A continuación describimos la formalización en OASIS de los cuatro cuadrantes expresivos.

Representación OASIS de las transiciones entre actividades y formalización como variante de la lógica dinámica

Como ya se comentó en el capítulo anterior, la especificación de las posibles vidas o trazas de ejecución de las tareas de un proceso se determinan en la sección *processes* de la clase process de una especificación OASIS que representa un flujo de trabajo. Más concretamente, las transiciones posibles entre las tareas de un proceso se especifican en el subproceso *actions*. Así pues, el conjunto de transiciones de un flujo de trabajo quedará representado en el subproceso *actions* de la especificación OASIS.

Por otra parte, el comportamiento obligatorio o permitido/prohibido determina si el objeto que representa el proceso debe tener un comportamiento activo o pasivo. Una transición obligatoria indica que es el propio objeto, con un comportamiento activo por medio de disparadores, debe iniciar la ejecución de la tarea destino. Por el contrario, una transición permitida o prohibida indica un comportamiento pasivo que ofrecerá el objeto en forma de servicios.

Las condiciones de transiciones de las actividades quedan reflejadas en distintas secciones de la especificación OASIS. Las transiciones de unión quedan reflejadas en las precondiciones de los servicios de las actividades. Así pues, por ejemplo, una condición de unión AND quedará reflejada como una precondición en la que se comprueba si las transiciones que tienen como destino la actividad en cuestión cumplen su conidición de inicio. Por el contrario, las transiciones de bifurcación quedan reflejadas en la sección

processes del proceso. Por ejemplo, una bifurcación AND indicará una ejecución concurrente de las vidas correspondientes a cada transición.

Así pues, dado que no se han introducido nuevos elementos, la formalización de las transiciones, y por tanto, el orden de ejecución de las tareas de un proceso se corresponde con la presentada en el capítulo anterior.

Resumen

En este capítulo hemos descrito el marco flexible para la especificación del orden de ejecución de las tareas de un proceso. Hemos visto como a través de dos dimensiones (flujo/reglas y obligación/permiso o prohibición) es posible abordar toda la expresividad necesaria.

También hemos visto como queda ampliado el metamodelo de referencia para la especificación de procesos utilizado en este proceso a través del concepto *transición* y las condiciones de transición (unión y bifurcación) de las actividades.

Por último, hemos visto como quedan formalizados estos conceptos a través de OASIS.

CAPÍTULO 5 – FLEXIBILIDAD EN TIEMPO DE EJECUCIÓN

La flexibilidad en tiempo de ejecución se consigue por medio del mecanismo de metamodelado de OASIS, desarrollado extensamente en la tesis de José A. Carsí (Carsí Cubel, 1999). Así pues, en este capítulo se describe brevemente dicho metanivel y su relación con el metamodelo de referencia para la especificación de procesos propuesto. Se plantea como trabajo futuro explotar esta característica de OASIS para soportar la evolución de los procesos en tiempo de ejecución (flexibilidad en tiempo de ejecución), el versionado de los procesos, reutilización, o guías metodológicas, entre otras. Estas características se presentan al final del capítulo.

La metaclase OASIS

En (Ramos, Pelechano, Penadés, Canós, & Pastor, 1995) (Ramos I. , Pelechano, Penadés, Bonet, Canós, & Pastor, 2995) (Canós, Penadés, Carsí, & Bonet, 1996) se introducen la noción de metaclase y se especifican los atributos y servicios de las clases OASIS vistas como objetos. En (Carsí Cubel, 1999) 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 (Bonner & Kifer, 1995). A continuación se da una visión general puesto que justifica la flexibilidad en tiempo de ejecución basada en evolución de las instancias de proceso.

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 servicio, añadir una transacción, añadir un disparo, cambiar una restricción, cambiar una precondición, cambiar la fórmula de evaluación de un servicio, borrar un atributo, borrar un servicio, 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 servicios, conjunto de evaluaciones, conjunto de precondiciones,* etc.

En (Ramos I. , Pelechano, Penadés, Bonet, Canós, & Pastor, 2995) aparece detallados todos los servicios y atributos de los metaobjetos. La Figura 12 muestra la representación gráfica del metaobjeto *usuario*, siguiendo esta aproximación.

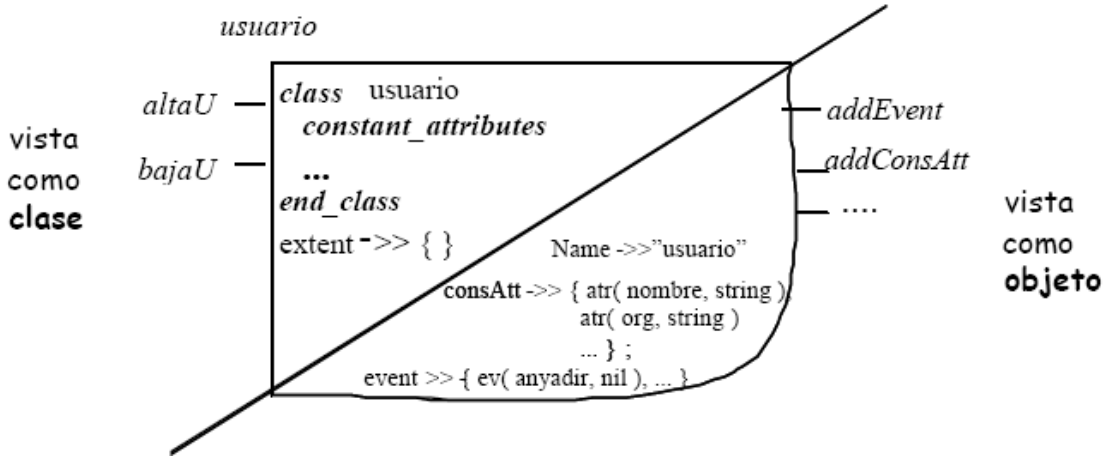

Figura 12 Doble visión de los metaobjetos

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 atirbutos 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 Metaclase OASIS

Se denomina *metaclase OASIS* al esquema conceptual[16] que define las plantillas de las metaclases que permiten crear metaobjetos que con su estado definen las características de las clases primitivas, elementales y complejas dentro del modelo OASIS.

[16] 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 correponde a lo que se conoce como *tiempo de edición* y la segunda al *tiempo de ejecución o animación*.

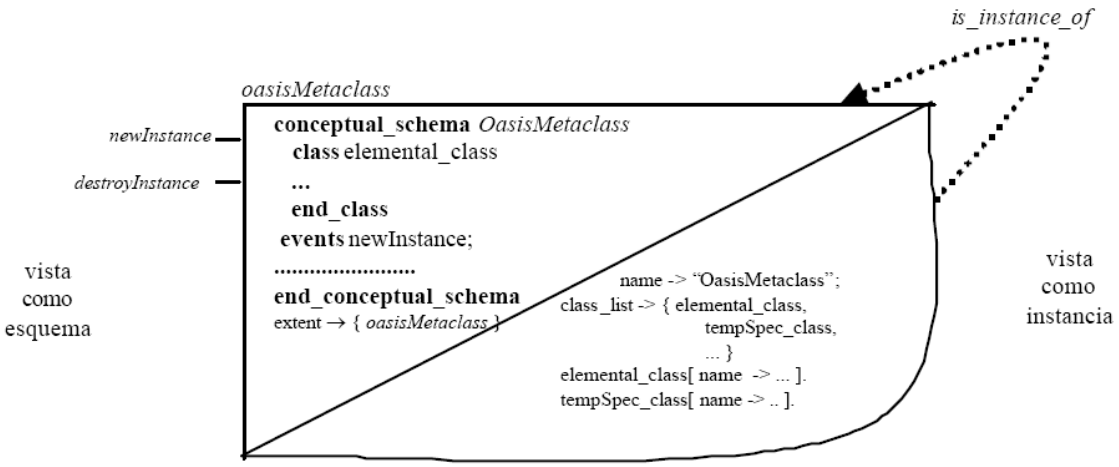

Figura 13 Doble visión de la metaclase OASIS

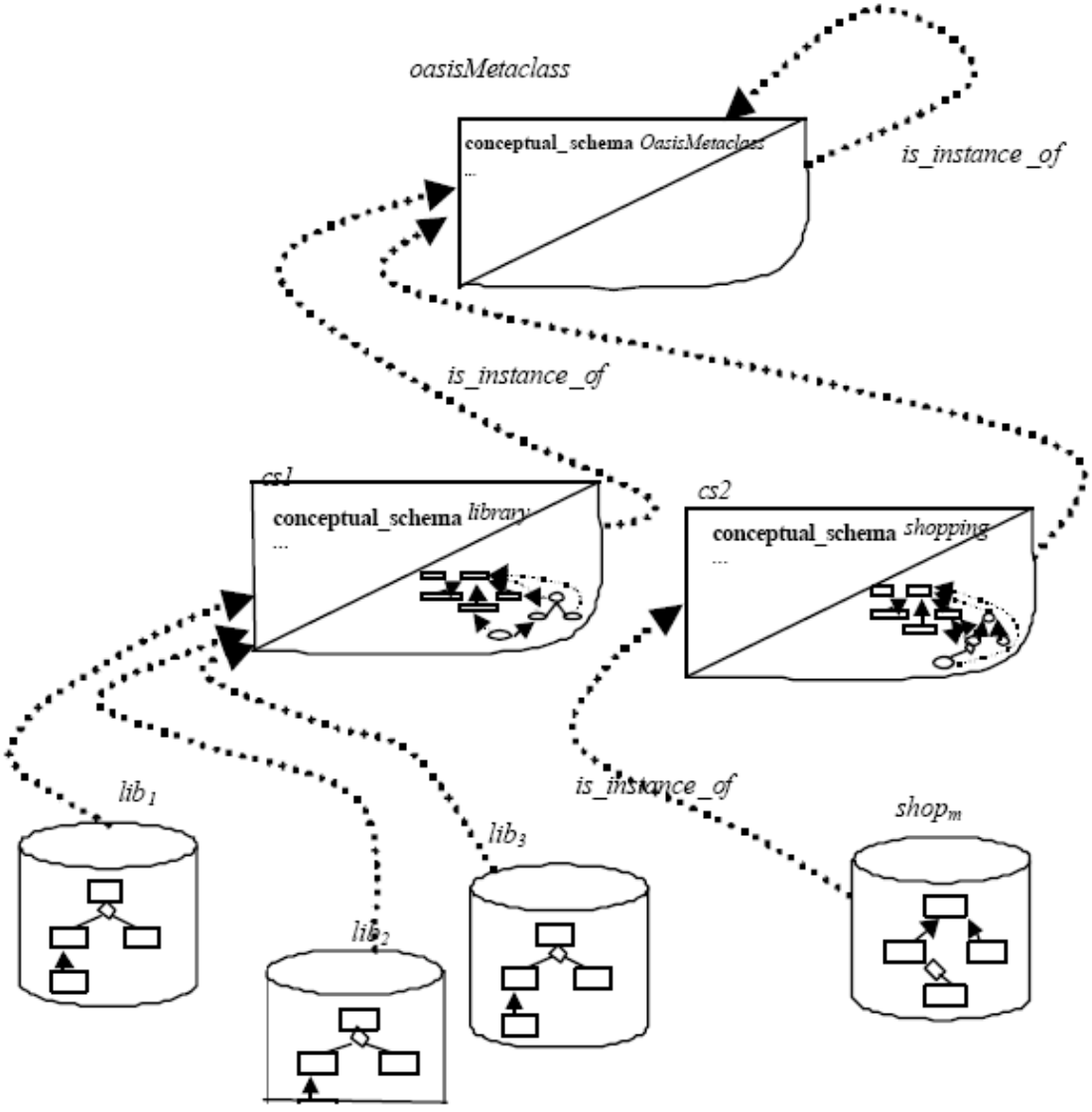

Figura 14 Jerarquía is_instance_of de la metaclase OASIS

La metaclase OASIS ofrece únicamente dos servicios: creación y destrucción de instancias[17]. 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 potencialmente infinita de metaclases se realiza el cierre reflexivo de la metaclase. Se considera que la metaclase OASIS es instancia 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 13 muestra la doble visión de la metaclase OASIS.

El proceso de creación de especificaciones OASIS haciendo uso de la metaclase OASIS se describe en (Carsí Cubel, 1999).Éste siembre se inicia con la invocación del servicio `newInstance` que crea una instanciación de la metaclase OASIS, inicialmente vacía 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 14 muestra la jerarquía de la relación *is_instance_of* de la metaclase OASIS (raíz de la jerarquía), dos esquemas conceptuales y diferentes instanciaciones de los esquemas.

Ingeniería del Software desde la visión de la metaclase

En los siguientes puntos se presentan algunos de los problemas de la Ingeniería del Software (IS), en general, y los procesos de negocio, en particular, y la vía de solución dentro del modelo OASIS presentado.

Evolución del Software

Un esquema conceptual en OASIS, es el estado de un determinado objeto (la metainstancia que lo define). El estado de cualquier objeto puede ser modificado en cualquier momento por el usuario a través de los servicios ofrecidos por el metaobjeto si el usuario está autorizado a ello. Estos cambios pueden seguir una aproximación temporal (marcando el tiempo en cada uno de los estados del objeto) o una dinámica (sólo se guarda el último estado). El efecto resultante de la ocurrencia de los eventos en el estado de los objetos se da de forma declarativa en la especificación de la metaclase. Una parte del estado de los metaobjetos es la plantilla que evoluciona. La otra parte la forma la población de la clase que deberá migrar para ser conforme con la primera. Ambos cambios se describen de la misma forma en la especificación de la metaclase.

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

Versionado del software

La evolución del software está motivada por la evolución de las reglas de negocio en el espacio del problema: "lo que es válido hoy no lo será mañana". Si se desea mantener una historia de los diferentes esquemas conceptuales, cada uno de ellos válido durante un periodo de tiempo, es necesario marcar el tiempo en los estados de los metaobjetos que codifican los esquemas conceptuales.

Cada estado de los metaobjetos está marcado por los periodos de tiempo en los que dicho estado permanece. En este sentido, el versionado de software es tratado en OASIS usando modelos temporales en el nivel meta.

Reutilización de software

El software desarrollado usando OASIS facilita su reutilización. Los *assets*[18] son las clases y los objetos. El repositorio formado por la extensión de la metaclase es la biblioteca de componentes que sirve para producir software por reuso.

La biblioteca de componentes tiene una estructura (relaciones de herencia y agregación) que permite la optimización de las búsquedas realizadas por las herramientas para recuperar los *assets* a diferentes niveles de especificación (relación de herencia) o granularidad (relación de agregación).

El lenguaje de consulta usado para recuperar los componentes es el mismo que el usado para recuperar los objetos en las aplicaciones de los usuarios.

Ayudas y guías metodológicas

En el área de las herramientas CASE, las metodologías subyacentes proporcionan lenguajes visuales y un tedioso conjunto de reglas que el desarrollador debe seguir para describir los modelos conceptuales.

Ésta es una de las razones del poco éxito de las herramientas CASE tradicionales. En la práctica, el desarrollador produce software basándose parcialmente en el método que le ofrecen las herramientas CASE y aplicándolo según su criterio.

Es obvio, que el único camino para que un método sea aplicado es implementarlo como un conjunto de ayudas y herramientas sobre el ordenador.

En OASIS, el protocolo de la metaclase puede determinar de forma clara y no ambigua la forma en la que se debe construir software. La metodología

[18] Los componentes reutilizables.

también está codificada en el modelo OASIS y las mismas herramientas que se usan para validar las aplicaciones sirven como ayuda y guía automática de la metodología.

Resumen

En este capítulo se ha esbozado el metanivel de OASIS. Dado que el metamodelo de flujos de trabajo presentado en este trabajo puede ser representado en OASIS, dicho metanivel también puede ser utilizado para la evolución, reutilización, versionado, guía metodológica, etc. de los procesos proporcionando la flexibilidad en tiempo de ejecución, segundo de los objetivos de este trabajo.

Como trabajo futuro se plantea estudiar en detalle esta aproximación.

CAPÍTULO 6 – CONCLUSIONES Y TRABAJO FUTURO

La especificación de procesos en dominios complejos y muy cambiantes requieren gran flexibilidad tanto en tiempo de diseño, dando soporte a la especificación de cualquier tipo de proceso por complejo que sea, como en tiempo de ejecución, permitiendo la evolución, versionado, reutilización, y resto de propiedades relacionadas con la modificación del proceso a posteriori.

Un ejemplo de dominio en el que se presentan estos requisitos son los sistemas para la resolución de emergencias. En este contexto claramente es necesario flexibilidad en tiempo de desarrollo, ya que el proceso de resolución es muy complejo teniendo que considerar numerosas condiciones excepcionales. Pero también flexibilidad en tiempo de ejecución, ya que los procesos de resolución de emergencias son muy difícilmente predecibles o, al menos, resulta muy difícil prever cualquier posible situación a la que puede evolucionar la emergencia, requiriendo por tanto modificar/evolucionar/reutilizar/etc. el proceso en tiempo de ejecución.

En este trabajo hemos presentado un Marco Flexible para la especificación de Procesos Flexibles. La primera flexibilidad se refiere a la flexibilidad en tiempo de diseño y se ha conseguido vía la integración de las aproximaciones clásicas basadas en diagramas de flujo y las basadas en reglas así como el formalismo basado en la lógica deóntica como variante de la lógica dinámica que aporta la semántica relativa a la ejecución opcional o, complementariamente, prohibida de tareas del proceso. La segunda flexibilidad se refiere a la flexibilidad en tiempo de ejecución y se ha conseguido representando el metamodelo de referencia de este trabajo para la especificación de flujos de trabajo en OASIS y reutilizando los resultados de la tesis de José A. Carsí que dotan de un metanivel a OASIS permitiendo evolucionar, versionar, reutilizar, guiar metodológicamente, y el resto de propiedades que implican modificación del proceso en tiempo de ejecución.

Así pues, la conclusión principal de este trabajo es que es posible definir marcos más expresivos que los actuales para la especificación de procesos de negocio o, en general, cualquier proceso proporcionando mayor flexibilidad tanto en tiempo de diseño como en tiempo de ejecución.

Como trabajos futuros se plantean los siguientes:

- **Notación textual y/o gráfica**. En los capítulos 3, 4 y 5 hemos descrito un marco flexible para la especificación de flujos de trabajo complejos basados tanto en diagramas de flujo, como en reglas, con

opcionalidad y prohibición de tareas, y flexibles en tiempo de ejecución. Hemos presentado el metamodelo de referencia y su formalización a través de OASIS y la variante de la lógica dinámica. Sin embargo, no hemos dicho nada respecto a la representación textual de dicho metamodelo y, ni mucho menos, su representación gráfica. Esto es un requisito previo al desarrollo de cualquier implementación relacionada con este trabajo por lo que se está trabajando en ello.

- **Ahondar en la flexibilidad en tiempo de ejecución**. En este trabajo, como ya se ha comentado en numerosas ocasiones, en lo que respecta a la flexibilidad en tiempo de ejecución nos hemos limitado a reutilizar los resultados relacionados con el metanivel de OASIS vía la especificación en ese modelo del metamodelo de flujos de trabajo de referencia descrito en este trabajo. Así pues, se plantea como trabajo futuro validar la aproximación adoptada respecto a la flexibilidad en tiempo de ejecución comparándola con otras alternativas establecidas en el area como las propuestas por Ellis mencionadas en este trabajo.
- **Soporte con herramienta.** Una herramienta que de soporte al marco aquí propuesto sería de gran utilidad para validar la propuesta. Conscientes de ello, estamos trabajando en desarrollar una aplicación que permite especificar flujos de trabajo basados en nuestro marco y posteriormente generar de forma automática el sistema correspondiente. En concreto, estamos desarrollando una herramienta para la especificación y ejecución de procesos de resolución de emergencias, como los comentados en este trabajo, y para ello estamos utilizando la tecnología de flujos de trabajo proporcionada por Microsoft en sus librerías de programación .NET Micorsoft Windows Workflow Foundation[19].
- **Líneas de trabajo adicionales**. Como líneas de trabajo que parten de este trabajo se plantea estudiar la validación y verificación formal de especificaciones e implementaciones de flujos de trabajo descritos con este marco así como la visualización y búsqueda avanzada de vidas posibles de un proceso a partir de una especificación basada en nuestro marco. Esta línea también se ha iniciado e incluso se han obtenido algunas publicaciones.

Resumen de contribuciones

La contribución principal de este trabajo es la definición de un marco flexible para la especificación de procesos de negocio flexibles y su demostración basada en OASIS y la variante de la lógica dinámica.

[19] http://wf.netfx3.com/

En el contexto de los sistemas cuyos procesos son realmente complejos y dinámicos, como es el de la resolución de emergencias, este marco aporta grandes ventajas respecto al resto de aproximaciones existentes. Prueba de ello es el premio Mike Meleshkin Award concedido por la organización internacional de sistemas de información para la gestión y resolución de emergencias (*ISCRAM Community*[20]) como reconocimiento de la contribución de este trabajo al dominio de las emergencias.

Publicaciones relacionadas

A la fecha de publicación, se han generado tres publicaciones relacionadas directamente con este trabajo. A continuación se añaden las referencias y se describen brevemente.

La primera de ellas (cronológicamente) fue publicada en las actas del congreso nacional *Jornadas de Ingeniería del Software y Sistemas de Información (JISBD)* en su décima edición celebradas en Granada, en el contexto del I Congreso Español de Informática. JISBD es el congreso español de referencia en lo que respecta a la Ingeniería del Software y las Bases de Datos y CEDI es el congreso de referencia en España para toda el área de informática. El título de la publicación fue *Un Enfoque Orientado a Procesos para la Especificación de Planes de Emergencia* y presenta el marco flexible para la especificación de flujos de trabajo descrito en este trabajo así como su relación con OASIS y, más concretamente, su formalización en la variante de la lógica dinámica. La referencia completa es:

> Un enfoque Orientado a Procesos para la Especificación de Planes de Emergencia
> **Manuel Llavador**, Patricio Letelier, Marcos R. S. Borges, José H. Canós, Mª Carmen Penadés, Carlos Solís
> X Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2005) dentro del I Congreso Nacional de Informática (CEDI'2005), pp. 171-178, Septiembre 2005
> ISBN 84-9732-434-X

La segunda publicación apareció en las actas del congreso internacional de sistemas de información para la gestión y resolución de emergencias (ISCRAM del inglés, *Information Systems for Crisis Response and Management*) en su tercera edición celebrada en el *New Jersey Institute of Technology* (Nueva Jersey, Estados Unidos) en el 2006. Esta conferencia es la más importante en el área de la gestión y resolución de emergencias y es en la que se nos concedió el premio Mike Meleshkin Award por el mejor artículo de la conferencia, que ya se ha comentado en la sección de contribuciones, cuyo título era *Precise jet Flexible Specification of Emergency*

[20] http://www.iscram.org/

Resolution Procedures, en el cual se presentaba el marco descrito en este trabajo desde el punto de vista de su aplicación a los sistemas para la resolución de emergencias. La referencia completa es:

> Precise yet Flexible Specification of Emergency Resolution Procedures
> **Mike Meleshkin award for Best Ph. D. Student Paper**
> **Manuel Llavador**, Patricio Letelier, Mª Carmen Penadés, José H. Canós, Marcos Borges, Carlos Solís
> 3rd International Conference on Information Systems for Crisis Response and Management (ISCRAM'2006), pp. 110-120, Mayo 2006
> ISBN 90-9020601-9

Por último, la tercera publicación se realizó también en las actas de JISBD pero en este caso en la edición XI celebrada en Sitges en el 2006. Este artículo tiene por título *Validación incremental de modelos usando escenarios y prototipado automático* y presenta una herramienta capaz de validar un conjunto de fórmulas de la variante de la lógica dinámica empleada en la formalización del metamodelo propuesto. La referencia completa es:

> Validación incremental de modelos usando escenarios y prototipado automático
> Ángel Roche, Patricio Letelier, Elena Navarro, **Manuel Llavador**
> XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06) Octubre 2006, pp.337-346, ISBN 84-95999-99-4

Este trabajo también se ha presentado, de forma menos directa, en otros eventos de difusión científica y cursos de formación a estudiantes de postgrado. A continuación se enumeran las referencias a las publicaciones de alguna manera relacionadas en las que hemos participado:

Capítulos de Libro

Parte 2, Módulo 4, Capítulo 5: Modelado de Procesos de Negocio con BizTalk Server 2004
Manuel Llavador
I Master en tecnologías Web "Desarrollo de Aplicaciones y Servicios para la Sociedad de la Información" Apuntes, presentaciones y conferencias, pp. 1-30, 2005
ISBN 84-689-3873-4, Depósito Legal: AB 498-2005

Revistas Internacionales

De Modelos de Proceso a Modelos Navegacionales
Carlos Solís, José H. Canós, **Manuel Llavador**, Maria Penadés
XI IEEE América Latina, Vol. 5, Issue 4, Edición especial Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 238-244, Julio 2007
ISSN 1548-0992

Conferencias Internacionales

Strong vs. Weak Links: Making Processes Prevail Over Structure in Navigational Design
POSTER and EXTENDED ABSTRACT
José H. Canós, Carlos Solís, María C. Penadés, **Manuel Llavador**
ACM Conference on Hypertext and Hypermedia (Hypertext'2007), Septiembre 2007
PENDIENTE DE PUBLICACIÓN

Model Driven Hypermedia Development Method
Carlos Solís, José H. Canós, María C. Penadés, **Manuel Llavador**
International Conference WWW/Internet (IADIS), Octubre 2006, pp. 321-328
ISBN 972-8924-19-4

Coordination in Software Architectures: an Aspect-Oriented Approach
POSITION PAPER
Jennifer Pérez, **Manuel Llavador**, José A. Carsí, José H. Canós, Isidro Ramos
5th Working IEEE/IFIP Conference on Software Architecture (WICSA'2005), pp. 219-220, Noviembre 2005
IEEE Computer Society, ISBN 0-7695-2548-2

Conferencias Nacionales

Coordinación y Acceso a Información en Gestión de Emergencias
José H. Canós, **Manuel Llavador**, Carlos Solís, Patricio Letelier, Mª Carmen Penadés, Marcos R. S. Borges
Actas V jornadas de trabajo DYNAMICA (DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures), pp. 103-108, Noviembre 2006
ISBN 84-690-2623-2

De modelos de proceso a modelos navegacionales
Seleccionado entre los 10 mejores artículos de la conferencia
Carlos Solís, José H. Canós, **Manuel Llavador**, Mª Carmen Penadés
XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06) Octubre 2006, pp. 44-53, ISBN 84-95999-99-4

Otras publicaciones informales o presentaciones sin publicación asociada

A Framework for Development of Emergency Management and Response Systems
Manuel Llavador, José H. Canós
Coloquio Internacional de Estudiantes de Doctorado ISCRAM'2007DS (celebrado junto a ISCRAM'2007), Mayo 2007

A Framework for Development of Emergency Management and Response Systems
Manuel Llavador, José H. Canós
Escuela de Verano ISCRAM-TIEMS Summer School'2006, Junio 2006

A Framework for Development of Emergency Management and Response Systems
Manuel Llavador, José H. Canós
Coloquio Internacional de Estudiantes de Doctorado ISCRAM'2006DS (celebrado junto a ISCRAM'2006), Mayo 2006

Un método de desarrollo de hipermedia dirigido por modelos
C. Solís, Mª Carmen Penadés, J. H. Canós, **M. Llavador**
IV Jornadas de Trabajo Proyecto DYNAMICA, pp. 12-20, Noviembre 2005

Codificación de procesos en conectores PRISMA: aplicación a los sistemas de gestión de emergencias
Manuel Llavador, José H. Canós, Marcos R. S. Borges, Patricio Letelier, Mª Carmen Penadés, Carlos Solís
III Jornadas de Trabajo Proyecto DYNAMICA, pp. 103-109, Abril 2005

BIBLIOGRAFÍA

Agha, G. A. (1986). *ACTORS: A model of Concurrent Computation in Distributed Systems.* The MIT Press.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Laymann, F., y otros. (2003). Bussiness Process Execution Language for Web Services.

Aqvist, L. (1984). *Deontic logic.* Reidel: In D.M. Gabbay and F.Guenthner, editors, Handbook of Philosophical Logic II, pages 605-714.

Bonner, A., & Kifer, M. (1995). *Transaction Logic Programming.*

Canós, J., Penadés, M., Carsí, J., & Bonet, B. (1996). *The R-OASIS Language.*

Carsí Cubel, J. A. (1999). *OASIS como marco conceptual para la evolución del Software.*

Casati, F., Ceri, S., Pernici, B., & Pozzi, G. (1996). Workflow Evolution.

Costa, J., Sernadas, A., Sernadas, C., & Ehrich, H. (1992). *Object Interaction.* Lisboa: INES.

Dayal, U., Hanson, E., & Widom, J. (1994). Active Database Systems.

Dubois, E., Du Bois, P., & Petit, M. (1993). *O-O Requirements Analysis: an agent perspective.* In Proc. of the 7th European Conference on Object Oriented Programming - ECOOP 93.

Ehrich, H., Coguen, J., & Sernadas, A. (1990). *A Categorical Theory of Objects as Observed Processes.* LNCS 489.

Ellis, C., & Keddara, K. (2000). ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems. *Computer Supported Cooperative Work* .

Ellis, C., Keddara, K., & Rozenberg, G. (1995). Dynamic change within workflow systems.

Genesereth, M. R., & Ketchpel, S. P. (1994). *Software Agents.* Commuications of the ACM, 37(7): pages 48-53.

Harel, D. (1984). *Dynamic Logic.* Reidel: In Handbook of Philosophical Logic II, editors D.M. Gabbay, F. Guenthner; pages 497-694.

Hollingsworth, D. (1995). *The Workflow Reference Model. Thecnical report TC00-1003.* WfMC.

Jablonski, S., & Bussler, C. (1996). *Workflow-Management: Modelling Concepts, Architecture and Implementation.* International Thomson Computer Press.

Kappel, G., Rausch-Schott, S., & Retschitzegger, W. (2000). A Framework for workflow management systems based on objects, rules and roles. *ACM Computing Surveys Symposium on Object-Oriented Application Frameworks.*

Letelier Torres, P., Sánchez Palma, P., Ramos Salavert, I., & Pastor López, O. *OASIS Versión 3.0. Un enfoque formal para el modelado conceptual orientado a objetos.*

Letelier, P., Sánchez, P., & Ramos, I. (1997). *Animación de Modelos Conceptuales para ayudar en la Validación de Requisitos.* Buenos Aires, Argentina: ASOO'97.

Leyman, F., & Roller, D. (2000). *Production Workflow. Concepts and Techniques.* Pertice Hall.

Llavador, M. (2007). *Un marco soporte para interoperabilidad basada en transformación de documentos.*

Meyer, J. (1988). *A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic.* In Notre Dame Journal of Formal Logic, vol. 29, pages 109-136.

Milner, R. (1989). *Communication and Concurrency.* Prentice Hall Series in Computer Science, C.A.R. Hoare, Series Editor.

Owen, M., & Raj, J. (2003). BPMN and Bussines Process Management, Popking Software.

Pastor, O. (1992). *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el Modelo Orientado a Objetos.*

Penadés Gramaje, M. *Una Aproximación Metodológica al Desarrollo de Flujos de Trabajo.*

Penadés, M. (1999). *Modelado de Flujos de Trabajo utilizando OASIS.*

Penadés, M., & Canós, J. (2000). *Modelado Conceptual de Flujos de Trabajo.*

Pernici, B. (1990). *Objects with Roles.* Cambridge, Mass.: In EEE/ACM Conference on Office Information Systems.

Ramos, I. e. (1993). Objects as Observable Processes. *4th International Workshop on the Deductive Approach to Information Systems and Databases.*

Ramos, I., Pastor, O., Cuevas, J., & Devesa, J. (1993). *Objects as Observable Processes.* Cataluña: Actas del 3rd. Workshop on the Deductive Approach to Information System Design.

Ramos, I., Pelechano, V., Penadés, M. B., Canós, J., & Pastor, O. (1995). *Análisis y Diseño Orientado a Objetos de un Entorno de Prototipación Automática.*

Ramos, I., Pelechano, V., Penadés, M., Bonet, B., Canós, J., & Pastor, O. (2995). *Especificación en R-OASIS de un Entorno de Prototipación Automática.*

Roche, A., Letelier, P., Navarro, E., & Llavador, M. (2006). Validación de modelos usando escenarios y prototipados automático.

Wainer, J. (2000). Logic representation of processes in work activity coordination. *ACM Symposium on Applied Computing, Coordination Track (1).*

Wegner, P., & Zdonik, S. (1988). *Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like.* LNCS 322, ECOOP pages 55-77.

Wieringa, R. J., & Meyer, J. (1993). *Actors, Actions and Initiative in Normative System Specification.* Annals of Mathematics and Artificial Intelligence, 7:289-346.

Wieringa, R., & Jonge, W. S. (1995). *Roles and dynamic subclasses: a modal logic approach.* The Netherlands: Vrije U.

Workflow Management Coallition. *Workflow Reference Model TC00-1003 Issue 1.1.*