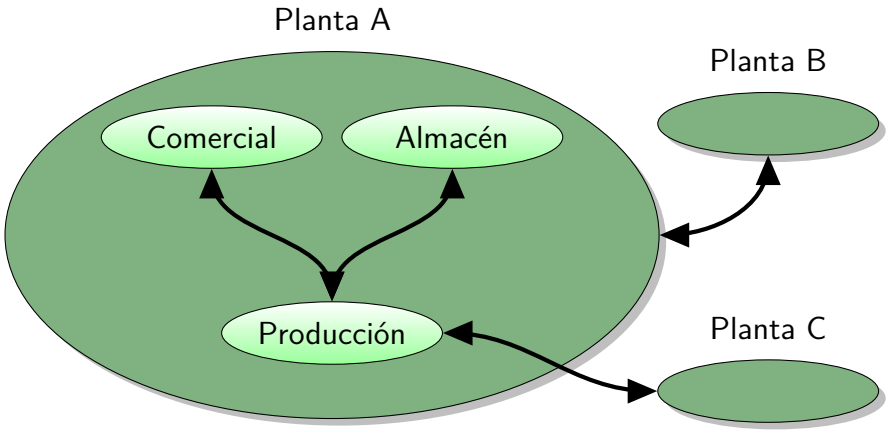
Universidad de Cádiz

Escuela Superior de Ingeniería

Máster Oficial en Ingeniería Fabricación

Trabajo de Aplicación

Estudio de viabilidad de una metodología dirigida por modelos para el desarrollo de los sistemas de información de una empresa de fabricación distribuida



Autor: Antonio García Domínguez
Tutor: Mariano Marcos Bárcena

Índice general

Índice de figuras

Índice de tablas

1

Introducción

1.1. Motivación

Hoy en día, las empresas se hallan bajo la necesidad de competir en un mercado en el que los ciclos de vida de los productos son cada vez más cortos y se exigen mayores niveles de flexibilidad y calidad a menor coste. Para ello, necesitan ser capaces de reorientar y mejorar continuamente sus procesos de negocio en función de la situación y de forma rentable. Sin embargo, las plataformas centralizadas normalmente usadas en las empresas de hoy en día no pueden cambiarse tan rápidamente como las situaciones lo requerirían, y finalmente son éstas las que definen las prácticas a seguir, más que la propia situación del mercado. Se necesita, por lo tanto, un enfoque distinto para estructurar los sistemas de información en la así llamada Siguiente Generación de Sistemas de Fabricación (SGSF).

Actualmente, se admite a nivel conceptual la necesidad de distribuir las actividades a lo largo de varios sistemas de información especializados y de posteriormente integrarlas en lo que se conoce como una empresa extendida Marcos et al. [2005]. Entre los protomodelos de empresa distribuida más conocidos, se encuentran las organizaciones holónicas, llamadas así por constituirse de una serie de actores semiautónomos que se interrelacionan entre sí a varios niveles, conocidos como holones Tharumarajah et al. [1998].

Una vez establecido el marco conceptual sobre el que modelar a la organización, resulta evidente la necesidad de plasmarlo en un sistema de información. Afortunadamente, desde hace varios años, una forma de estructurar los sistemas de información que se ajusta bien a estas ideas ha ido recibiendo cada vez más atención: las Arquitecturas Orientadas a Servicios o «Service-Oriented Architectures» (SOA). La idea central detrás de ellas es organizar los sistemas de información no como sistemas integrados unidad a unidad de negocio o proyecto a proyecto, como se ha hecho comúnmente, sino como servicios individuales que pueden ser reutilizados a lo largo de la organización, o incluso por otras organizaciones con las que se tiene relación.

Posteriormente, estos servicios individuales pueden ser integrados en servicios de nivel superior con su propio valor añadido, modelando procesos de negocio completos en vez de operaciones individuales. A nivel de implementación, estas arquitecturas suelen implementarse utilizando Servicios Web («Web Services» o WS), para así aprovechar las numerosas tecnologías, herramientas e infraestructuras ya implementadas para la World Wide Web y reducir las barreras tecnológicas. La información deja de concentrarse en «silos» de departamentos concretos y pasa a estar disponible a toda la organización en forma de servicios.

Este enfoque y su implementación mediante WS ofrecen grandes ventajas a nivel tecnológico y de negocios, pero no dejan de tener sus propias dificultades. Implantar una SOA es complejo, ya que afecta a toda la organización, y necesita tanto una buena visión global como una correcta implementación. La máxima flexibilidad se conseguiría con muchos servicios muy específicos, pero implantar cada servicio tiene un coste adicional respecto a la simple implementación de la funcionalidad, por lo que en términos de coste de desarrollo a corto plazo, interesa tener pocos servicios. El óptimo global se halla en una granularidad media de los servicios, pero es difícil de localizar. A esto se le añade la dificultad inherente en el desarrollo de cualquier sistema distribuido de esta magnitud.

Para hacer frente a este nivel de complejidad, han surgido metodologías orientadas a SOA como SOMA Ghosh et al. [2008], SOD-M de Castro [2007] o la propuesta por Stojanović Stojanović [2005]. Tratan de reducir el nivel de dificultad y asegurar resultados más controlables y repetibles en su implantación. Algunas de estas metodologías son revisiones de metodologías

de desarrollo tradicional de software orientado a objetos, y otras han sido creadas desde cero alrededor del concepto de «servicio» tal y como se entiende en SOA. En este trabajo se propone que estas nuevas metodologías son las que pueden dar los mejores resultados, ya que disponen de un nivel mayor de abstracción y reflejan de forma más directa los procesos de negocio a implementar.

Otro problema, independientemente de la metodología utilizada, es el hecho de que el mayor grado de integración y la mayor visibilidad de los servicios de la organización implican también una mayor dependencia en su correcto funcionamiento, y un mayor grado de impacto en caso de fallo. Esto se acentúa en el caso de SOA, ya que la integración de servicios internos o externos a la organización es central a dicho enfoque.

Este Trabajo Fin de Máster aplicará una metodología SOA a un caso práctico de una empresa de fabricación distribuida, por lo que habrá que definir una metodología que sea capaz de ayudar a controlar la complejidad en la gestión de una SOA y a reducir el riesgo de que en algún momento no funcione de la forma esperada.

1.2. Trabajos relacionados

En el contexto de la ingeniería de fabricación, la implementación de sistemas complejos a partir de la interacción de una serie de agentes simples es una estrategia común a nivel de planta Marcos et al. [2005]. La variedad radica en la forma en que se organizan estas entidades.

Algunos sistemas se estructuran de forma jerárquica, con mecanismos y sensores sin inteligencia propia que son manejados de forma remota con controladores de sección y de planta. Los controladores de planta realizan acciones de nivel superior utilizando la información que reciben de los controladores de sección. Otros sistemas más dinámicos integran agentes que interaccionan con su entorno, con los agentes en los alrededores y en algunos casos con supervisores, que más que enviar órdenes, proporcionan información y sugerencias. En muchos casos se combinan ambos enfoques, o se extienden utilizando técnicas de inteligencia artificial.

Por otro lado, las SOA se han utilizado normalmente a nivel de empresa para comunicar las distintas plantas entre sí y con la cadena de proveedores, uniéndolas en una empresa extendida Browne et al. [1998]. Conceptualmente puede verse como una implementación de los sistemas de información de acuerdo con el modelo empresarial holónico Tharumarajah et al. [1998]. En particular, Hyfinity Hyfinity [2003] utilizó el enfoque de las SOA para implementar un sistema que integraba la cadena de concesionarios de Nissan con sus plantas de fabricación y sus proveedores: una venta se convertía en un pedido de producción que generaba los pedidos de material necesarios, de forma que se minimizara el tiempo entre la realización del pedido y su entrega.

Otras organizaciones, como MESA, proponen usar las SOA en dos capas: una primera capa a nivel de la organización, y una segunda capa a nivel de planta IBM Corporation et al. [2008]. Esto se debe a los distintos requisitos de cada tipo de comunicación: pocas operaciones pero que implican mayor carga computacional y actúan a nivel de negocio, frente a muchas interacciones cortas con carga computacional mínima, que actúan a nivel de planta.

Antes de seguir, es necesario aclarar exactamente qué se entiende por SOA. La acepción original, que es la que se utilizará en este Trabajo Fin de Máster, fue descrita por Erl en Erl [2008] y sugiere que no es más que una forma de ver los sistemas de información como una colección de servicios que apoyan al negocio integrándose entre sí. Actualmente, sin embargo, muchas empresas desarrolladoras de software abusan del término SOA para hacer referencia a

una pila determinada de tecnologías.

Definir y mantener una SOA dentro de una organización no es nada trivial, por lo que han surgido diversas metodologías SOA con distintos alcances y objetivos. Engels propone una metodología que se limita a extraer del entorno los servicios que han de formar parte de la SOA Engels et al. [2008], mientras que la metodología sugerida por Stojanović define la funcionalidad esperada y cómo se implementará, pero no los datos que se manejarán Stojanović [2005]. La metodología SOD-M de Castro [2007], tras integrarse con MIDAS, permite describir la información que manejará el sistema además de su funcionalidad, pero no cubre todas las fases del ciclo de vida del software. Por otro lado, la metodología SOMA Ghosh et al. [2008] de IBM sí incluye todas las fases, pero tiene un importante coste de ejecución.

1.3. Objetivos

Los objetivos de este Trabajo Fin de Máster (TFM de ahora en adelante) son los siguientes:

- Seleccionar una metodología SOA dirigida por modelos que permita obtener la funcionalidad esperada del sistema a partir de la descripción de su negocio. Dicha metodología debería ser lo suficientemente flexible como para ser utilizada en organizaciones de menor tamaño, con menos recursos para realizar un modelado extensivo. Además, la metodología debería asistir en la generación parcial del código necesario para su implementación y/o pruebas.
- Revisar el grado de integración de técnicas de pruebas sobre la funcionalidad del sistema y demás características (como su rendimiento) en la metodología seleccionada. En caso de que el nivel de integración no sea satisfactorio, se propondrán las extensiones necesarias para cubrir dicha carencia.
- Realizar una revisión del estado del arte sobre pruebas en el contexto de los Servicios Web y SOA, reuniendo una serie de técnicas integrables en un enfoque dirigido por modelos. Estas técnicas podrían recibir sus entradas a partir de los modelos, o presentar sus resultados en ellos.
- Evaluar el grado de madurez de las herramientas existentes para la metodología seleccionada, extendiéndolas o reemplazándolas en función de las conclusiones extraídas.
- Aplicar el conjunto final de herramientas sobre un caso práctico basado en una empresa real, comprobando la viabilidad de la aplicación práctica de la metodología extendida.

1.4. Estructura del trabajo

En esta sección se listarán cada una de las partes en que se divide este trabajo, describiendo de forma general su contenido.

Conceptos previos En este capítulo se describen las ideas básicas sobre las que se asienta esta propuesta, tanto desde el punto de vista de la ingeniería de fabricación como desde el de la ingeniería del software. Se introduce el concepto de «empresa extendida» Browne et al. [1998], se relacionan con los modelos empresariales en la literatura y se proponen las arquitecturas

orientadas a servicios («Service-Oriented Architectures» o SOA) Erl [2008] como una forma de implementar sus sistemas de información. Se presentan las metodologías dirigidas por modelos Object Management Group [2003] como una forma de diseñar las SOA que permite mantener bajo control su gran envergadura.

Selección de una metodología SOA Tras haber definido los conceptos fundamentales, en esta parte del TFM se comparan algunas de las metodologías SOA existentes y se selecciona una de ellas, describiéndose en más detalle. Tras analizar sus carencias, se proponen una serie de extensiones dirigidas a modelar las características funcionales y no funcionales (rendimiento, p.ej.) de la SOA. Se expone un abanico de técnicas de prueba del software que pueden integrarse con estas especificaciones, de forma que se reduzca el riesgo de que el sistema no se comporte de la forma esperada en algún momento.

Extensión con restricciones de rendimiento de SOD-M Antes de poder evaluar la metodología, es necesario desarrollar las herramientas que permitan aplicarla de forma automatizada. Este capítulo comienza evaluando las herramientas para la metodología SOD-M recibidas en marzo del 2009 de sus desarrolladores, el grupo Kybele de la Universidad Rey Juan Carlos, y describiendo las tecnologías utilizadas y las posibilidades de extensión. Se estimó que era necesario mejorar el flujo de trabajo del desarrollo de dichas herramientas, por lo que se integraron una serie de nuevas tecnologías dirigidas a aumentar su agilidad y robustez. El resto del capítulo se dedica a los aspectos de diseño e implementación de las nuevas herramientas, que reemplazan a algunas de las originales e incluyen las extensiones antes propuestas para modelar el rendimiento esperado del sistema.

Caso práctico La metodología extendida e implementada es evaluada sobre un caso práctico inspirado en una importante empresa del sector tabaquero. Tras una descripción textual de sus sistemas de información, se describe el negocio en términos de intercambios de valor y del proceso seguido para que el consumidor final obtenga el tabaco empaquetado que desea. A partir de esta descripción de alto nivel se obtienen en última instancia modelos con las acciones a implementar en el sistema final, anotadas con el rendimiento esperado. De entre estos modelos, por limitaciones de espacio, se expande y detalla uno de ellos al nivel más concreto de la metodología SOD-M, seleccionando aquellas acciones que se ofrecerán o consumirán en forma de Servicios Web.

Bibliografía

J. Browne, I. Hunt, and J. Zhang. The Extended Enterprise (EE). In L. M. Camarinha-Matos, H. Afsarmanes, and V. Merik, editors, *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations*, pages 3–30. Kluwer Academic Publishers, Londres, 1998. 1.2, 1.3

María Valeria de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 1.1, 1.3

Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus Voß, and Johannes Willkomm. A method for engineering a true Service-Oriented Architecture. In José Cordeiro and Joaquim Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems*, pages 272–281, Barcelona, España, 2008. ISBN 978-989-8111-38-8. 1.3

Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823. 1.2, 1.4

S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008. 1.1, 1.3

Hyfinity. Nissan Motor Manufacturing Case Study, 2003. URL http://www.hyfinity.com/Nissan_Motor_Manufacturing.html. Fecha de última comprobación: 2 de noviembre de 2009. 1.2

IBM Corporation, MESA International, and Capgemini. SOA in Manufacturing Guidebook, May 2008. 1.2

Mariano Marcos, Francisco Aguayo, Manuel Sánchez Carrilero, Lorenzo Sevilla, and J. R. Lama. Toward the next generation of manufacturing systems. Frabiho: a synthesis model for distributed manufacturing. In *Proceedings of the First I*proms Virtual Conference*, pages 35–40. Elsevier, 2005. 1.1, 1.2

Object Management Group. MDA Guide version 1.0.1, June 2003. URL http://www.omg.org/mda/. Fecha de última comprobación: 2 de noviembre de 2009. 1.4

Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005. 1.1, 1.3

A. Tharumarajah, A.J. Wells, and L. Nemes. Comparison of emerging manufacturing concepts. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 325–331, California, EEUU, 1998. 1.1, 1.2

2

Conceptos previos

En esta sección se revisarán algunos conceptos básicos sobre los que se apoya el resto de este trabajo. Tras un recorrido general sobre el tratamiento del problema, se introducirán las ideas centrales a la propuesta: las arquitecturas orientadas a servicios y las metodologías de desarrollo dirigido por modelos.

2.1. Organizaciones virtuales: la empresa extendida

Ninguna empresa vive en un vacío: ha de relacionarse con el ecosistema formado por sus proveedores, diseñadores, fabricantes, subcontratistas, clientes y otras empresas competidoras. Para tener éxito, ha de añadir más valor a su producto final que los competidores, y esto se traduce, normalmente, en usar la información de forma que se pueda responder eficazmente a las demandas del mercado.

Sin embargo, hoy en día las empresas se ven obligadas a interactuar más allá del ámbito geográfico local al que se encuentran habituadas. El uso de sistemas de información interconectados es por lo tanto un paso crucial, así como la estandarización de las prácticas de cada participante y la resolución de diversos problemas de logística. Una vez se ha completado dicha integración, la organización resultante se conoce con el nombre de empresa extendida Browne et al. [1998]. Existe una amplia variedad de modelos para representar la estructura de una empresa extendida. Muchos de ellos pueden englobarse en una de tres categorías Tharumarajah et al. [1998]: modelos biónicos, fractales u holónicos.

Los modelos biónicos se hallan inspirados en sistemas biológicos. La empresa consiste en una serie de tejidos (correspondientes a procesos, productos o servicios), formados a su vez por células que realizan diversas operaciones y reciben y producen artefactos codificados de forma genética (partes y productos, por ejemplo). Manteniendo la analogía, una célula se autorregula mediante la secreción de enzimas (información interna de control), y puede influenciar o ser influenciada por las hormonas (información del entorno y otras unidades) que se hallen en el entorno. Ante situaciones urgentes, un sistema nervioso puede responder rápidamente.

Por otro lado, los modelos fractales parten de las formas geométricas del mismo nombre. Se centran en la construcción de la empresa a partir de entidades similares entre sí a distintos ámbitos. Se construye un fractal sobre otro cuando el fractal de nivel inferior no puede acometer todas las tareas que deben ser realizadas. De forma inversa, las metas de la organización descienden desde el fractal principal que modela a la organización completa, y se van concretando nivel a nivel. Cada fractal tiene un cierto grado de autonomía y capacidad de reorganización.

Finalmente, los modelos holónicos se inspiran en los estudios de los sistemas jerarquizados de Koestler, quien define el concepto de «holón» como una entidad que es a la vez un todo formado por agentes de nivel inferior, y una parte de varias holarquías (jerarquías de holones) de nivel superior. Dichos holones combinan cierta autonomía con un grado de dependencia en otros holones del mismo nivel o del nivel inmediatamente superior.

Estos tres modelos poseen distintos enfoques, pero tienen en común la visión de la empresa como una red dinámica de agentes con cierta autonomía y capacidad de colaboración. A nivel de implementación de estos modelos, uno de los mayores problemas de las empresas extendidas es el establecimiento de los canales de comunicación necesarios. Los sistemas de información suelen diseñarse e implementarse con planes a corto plazo específicos de cada unidad de cada organización, y en otras ocasiones existen aplicaciones cruciales que se diseñaron en una etapa anterior a la Web y resulta demasiado costoso reemplazar. Todo esto resulta en dificultades a la hora de establecer colaboraciones puntuales con empresas separadas geográficamente y

aprovechar los nuevos procesos y otras ventajas que ofrecen.

2.2. Arquitecturas orientadas a servicios

Normalmente, los sistemas de información utilizados por las empresas suelen dividirse en una serie de capas implementadas a lo largo de varias máquinas: una base de datos recoge toda la información, que es manipulada por una capa con la lógica de cada área de negocio y finalmente visualizada y manipulada por el usuario mediante una capa de presentación. Esta arquitectura ha sido utilizada con éxito para definir un número considerable de sistemas de gran envergadura de hoy en día. Encajan de forma natural con las empresas centralizadas, jerarquizadas y estables.

Sin embargo, no están exentos de problemas. Estos sistemas suelen estar desarrollados para satisfacer las necesidades de partes individuales de la organización a corto plazo. No tienen en cuenta la necesidad de cambiar las prácticas de negocio a medio o largo plazo: la lógica de negocio suele hallarse unida al diseño y la implementación. Tampoco se suelen diseñar con vistas a colaborar en el futuro con otras compañías originalmente no previstas. Con el tiempo, las empresas o bien han de rodear al sistema para poder innovar, o pierden la capacidad de reaccionar ante nuevas situaciones.

Por estas razones, un nuevo enfoque basado en arquitecturas orientadas a servicios («Service Oriented Architectures» o SOA) ha recibido un gran nivel de atención en los últimos años. Más que un conjunto de tecnologías, es una forma distinta de organizar los sistemas de información de una empresa Erl [2008]. Éstos dejan de ser estructuras rígidas y centralizadas para convertirse en una serie de servicios reutilizables y recombinables independientemente para definir nuevos procesos de negocio y/o refinar los existentes. Las ideas subyacentes son muy similares a las de los modelos holónicos o basados en agentes: un servicio de nivel inferior puede formar parte de servicios de nivel superior o procesos de negocio, posiblemente colaborando con otros servicios, y con un cierto nivel de autonomía. Una SOA sería una herramienta ideal para apoyar las operaciones de una empresa de fabricación distribuida.

Las tecnologías más utilizadas hoy en día para implementar SOA son las de los servicios web. Aprovechan muchas de las herramientas existentes y se basan en estándares abiertos para reducir las barreras tecnológicas.

Un fallo común a la hora de adoptar SOA es pensar que basta con adquirir y utilizar la última versión de la plataforma SOA escogida. Sólo se consiguen sus verdaderos beneficios una vez se hayan comprendido sus conceptos fundamentales, se haya definido un catálogo de servicios base de alto rendimiento y fiabilidad y exista una visión global de los servicios y procesos definidos desde los niveles más altos de la organización. Este proceso no es sencillo, por lo que surge la necesidad de definir metodologías que guíen su ejecución, al igual que cuando se desarrolla cualquier otro tipo de software. En una sección posterior se hará una comparativa de algunas de ellas.

2.3. Ingeniería dirigida por modelos

En la actualidad, un problema al desarrollar software es el hecho de que existe un vínculo muy débil entre los modelos de alto nivel (es decir, más cercanos a la forma de pensar de los humanos) que se crean de los sistemas y el código que los implementan. En la práctica, lo

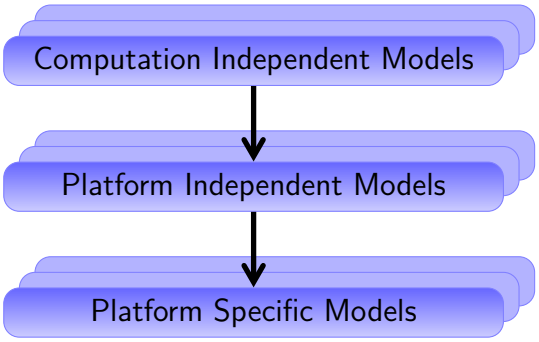


Figura 2.1. Propuesta MDA del OMG

que ocurre es que los lenguajes de modelado se usan solamente como vías de comunicación entre desarrolladores, y los modelos son de usar y tirar, ya que rápidamente quedan obsoletos al cambiar el código o los requisitos del sistema. Esto se traduce en una serie de problemas en varias actividades usuales, como por ejemplo comprobar si se han implementado los requisitos pedidos por el cliente, optimizar un diseño ante nuevas situaciones, aprovechar las últimas tecnologías existentes o verificar si el sistema cumple determinadas condiciones.

Una nueva perspectiva en el desarrollo de software conocida como ingeniería dirigida por modelos («Model Driven Engineering» o MDE) y defendida por el Object Management Group (OMG) bajo el nombre de arquitectura dirigida por modelos («Model Driven Architecture» o MDA) Object Management Group [2003a] trata de dar la vuelta a esta situación. Propone implementar el sistema a partir de una serie de modelos cada vez más detallados entre los cuales se puedan hacer correspondencias a distintos niveles de automatización. Así, volvería a elevarse el nivel de abstracción al que se implementan los sistemas, como se consiguió con los lenguajes de alto nivel frente al código ensamblador, o con el código ensamblador frente al código máquina.

En particular, la propuesta del OMG (véase la figura 2.1) define tres tipos de modelos para cualquier sistema. Los modelos independientes de computación («computation-independent models» o CIM) describen el negocio sin preocuparse de cómo se implementará el sistema. Los modelos independientes de la plataforma («platform-independent models» o PIM) dicen cómo cumplir los requisitos de negocio con el sistema, pero sin entrar en detalles de cómo se implementará en una plataforma software y hardware determinada. Por último, los modelos específicos de la plataforma («platform-specific models» o PSM) completan el resto de los detalles, para así simplificar la traducción final a código.

OMG ha definido el metamodelo de nivel 2 Meta-Object Facility (MOF) Object Management Group [2003b] para especificar los metamodelos de nivel 1 CIM, PIM y PSM, y la especificación Query/View/Transformation (QVT) Object Management Group [2008] para describir transformaciones automatizadas entre ellos.

A nivel práctico, dada la complejidad de MOF, se tiende a utilizar versiones simplificadas de dicha especificación, como la usada en la comunidad de desarrolladores Eclipse: Ecore Eclipse Foundation [2009b]. De forma parecida, en lugar de QVT, suelen usarse alternativas más limitadas pero maduras, como el ATLAS Transformation Language (ATL) Eclipse Foundation [2009a], un lenguaje similar a QVT.

Bibliografía

J. Browne, I. Hunt, and J. Zhang. The Extended Enterprise (EE). In L. M. Camarinha-Matos, H. Afsarmanes, and V. Merik, editors, *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations*, pages 3–30. Kluwer Academic Publishers, Londres, 1998. 2.1

Eclipse Foundation. ATLAS Transformation Language (ATL), 2009a. URL http://www.eclipse.org/m2m/atl/. Fecha de última comprobación: 2 de noviembre de 2009. 2.3

Eclipse Foundation. Eclipse Modeling Framework, 2009b. URL http://eclipse.org/modeling/emf/. Fecha de última comprobación: 2 de noviembre de 2009. 2.3

Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823. 2.2

Object Management Group. MDA Guide version 1.0.1, June 2003a. URL http://www.omg.org/mda/. Fecha de última comprobación: 2 de noviembre de 2009. 2.3

Object Management Group. Meta-Object Facility (MOF) 2.0, June 2003b. URL http://www.omg.org/spec/MOF/2.0/. Fecha de última comprobación: 2 de noviembre de 2009. 2.3

Object Management Group. Query/View/Transformation (QVT) 1.0, April 2008. URL http://www.omg.org/spec/QVT/1.0/. Fecha de última comprobación: 2 de noviembre de 2009. 2.3

A. Tharumarajah, A.J. Wells, and L. Nemes. Comparison of emerging manufacturing concepts. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 325–331, California, EEUU, 1998. 2.1

3

Selección de una metodología SOA

3.1. Introducción

Implementar una SOA es una labor compleja, ya que afecta a toda la organización. Sin una metodología apropiada, puede ser inabarcable incluso para empresas de fabricación de menor tamaño.

Por ello, han surgido una serie de metodologías SOA dirigidas a controlar la complejidad de las tareas asociadas. Estas metodologías varían en gran medida entre sí, por lo que es necesario compararlas siguiendo criterios de completitud, capacidad de automatización y coste de ejecución, entre otros.

Este capítulo se estructura de la siguiente forma: tras comparar algunas de las metodologías existentes, se selecciona una y se describe en mayor detalle. A continuación, se proponen una serie de extensiones para modelar los aspectos funcionales y no funcionales del comportamiento esperado del sistema.

3.2. Revisión de las metodologías existentes

En esta sección se realizará una revisión de algunas de las metodologías disponibles para desarrollar sistemas de información basados en arquitecturas orientadas a servicios. La tabla 3.1 resume los resultados.

Se descartan aquellas especificaciones y metodologías que sólo cubren pasos específicos del proceso de construcción del software, como Business Process Modeling Notation (BPMN) Object Management Group [2009a]. Esta especificación se ocupa únicamente de modelar los procesos de negocio, sin entrar en cómo se implementarán.

De la misma forma, la metodología presentada por Engels Engels et al. [2008] sólo establece la lista de servicios a crear y operaciones a definir en cada uno a alto nivel, sin llegar a implementarlos. Su objetivo es establecer un «sistema ideal» que vaya consiguiéndose con el tiempo a medida que el existente en la organización va siendo renovado. Para ello, utiliza una serie de modelos intermedios con correspondencias manuales sobre varias fuentes de información, utilizando un abanico de fuentes de información. Esta metodología podría asistir a cualquiera de las opciones de esta sección.

3.2.1. Antecedentes en el desarrollo basado en componentes

Ya se ha mencionado anteriormente que SOA no es una revolución, sino una evolución de las lecciones aprendidas en el campo de la Ingeniería del Software. Pueden establecerse varios paralelismos entre SOA y el paso conceptual inmediatamente anterior: desarrollo basado en componentes («component based development» o CBD).

Existe una gran variedad de definiciones del concepto de «componente», pero en general puede decirse que un componente se trata de una «caja negra» con entidad propia de la que se sabe qué servicio provee, pero no cómo lo hace, y que puede ensamblarse con otros componentes. En muchos aspectos, la idea es similar a la de un servicio.

La principal diferencia entre SOA y CBD es cómo se explota esa funcionalidad: con CBD, el componente pasa a formar parte inseparable del programa, mientras que con SOA sigue siendo un ente independiente, con el que se interacciona mediante mensajes. Además, un servicio de SOA ha de implementar una funcionalidad a nivel de negocio, mientras que un componente no tiene por qué, pudiendo ser algo más sencillo.

Aprovechando estas similitudes, Stojanović Stojanović [2005] define una metodología originalmente dirigida a CBD, que posteriormente extiende al caso de SOA. Existen otras metodologías basadas en componentes más conocidas, como Kobra Atkinson et al. [2008] o Catalysis D'Souza and Wills [1998], pero no tratan explícitamente el caso de SOA. Stojanović describe los conceptos a describir en cada modelo, pero no la notación exacta a utilizar: en su lugar, deja que el usuario elija de entre varias posibilidades.

El proceso seguido por esta metodología puede resumirse de la siguiente forma:

1. Se modelan los procesos de negocio, el entorno y sus participantes en unos modelos del dominio del negocio («Business Domain Models» o BDM).

2. Se identifican los casos de uso a nivel de negocio que debería implementar el sistema, y se agrupan en un modelo de componentes del negocio («Business Component Model» o BCM). Un componente del negocio reúne casos de uso que manejan los mismos elementos del universo de discurso, cambian en unísono, emplean los mismos sistemas ya existentes o forman parte de una misma transacción.

3. Identifica las aplicaciones a utilizar para implementar cada uno de los componentes del negocio seleccionados, produciendo los modelos de componentes de aplicación («Application Component Models» o ACM).

4. Concreta los ACM anteriores pasando de productos software de alto nivel a las partes que los forman: bibliotecas desarrolladas externamente o internamente, lógica del negocio, etc. En este paso se crean los modelos de componentes de implementación («Implementation Component Models» o ICM).

Los pasos son similares a la propuesta MDA del OMG: se modela el negocio, se crea un modelo del sistema de forma independiente de su implementación basado en agrupaciones de servicios y se consigue finalmente un modelo cercano a la implementación. La metodología no impone una notación particular, pero en su caso de estudio utiliza UML. Sin embargo, el paso de un modelo a otro es completamente manual, y no se establecen formalmente las correspondencias entre ellos.

3.2.2. Metodología IBM SOMA

La metodología SOMA (Service Oriented Modeling and Architecture), desarrollada por IBM, define un enfoque integrado sobre el proceso de desarrollo SOA que cubre desde su concepción hasta su monitorización y mantenimiento Ghosh et al. [2008]. Consiste en un ciclo iterativo de refinamiento dividido en varias fases.

En primer lugar, se define un modelo del negocio y una serie de plantillas para los distintos tipos de soluciones de integración posibles. A continuación se identifican los servicios que formarán parte de la arquitectura, combinando varias fuentes: las metas de la empresa, un modelo conceptual del entorno, y los sistemas informáticos ya implantados.

Posteriormente se estructurarán, racionalizarán y especificarán todos los servicios especificados en una arquitectura coherente. Puede que se haya identificado un gran número de servicios, por lo que se seleccionará el subconjunto que reporte el mayor retorno en inversión y se pospondrá el resto. Finalmente, se implementarán, depurarán e implantarán los servicios, tras los cuales se someterán a monitorización.

Tabla 3.1. Tabla comparativa de las metodologías SOA bajo estudio

Metodología	Alcance	Basado en	Notaciones	Automatización	Coste
Stojanović	Análisis y diseño funcional	Componentes, para posteriormente convertir a servicios	A elegir, UML (ligero)	Ninguna	Medio
SOMA	Ciclo de vida completo	Clases esterotipadas UML, para posteriormente convertir a servicios	Perfiles UML (completo)	Generación código y de documentos	Alto
SOD-M	Análisis y diseño funcional (ampliable si se integra con MIDAS)	Servicios, para posteriormente convertir a artefactos ejecutables	Diagramas de valor, perfiles UML (ligero)	Generación interfaces y modelos	Medio

Sus mayores aportaciones se hallan en los aspectos de ingeniería de requisitos, en que combinan tres distintas técnicas de obtención de servicios para conseguir un catálogo más completo:

- El modelado meta-servicio («Goal-Service Modeling» o GSM) obtiene servicios estudiando las metas a nivel de negocio de la organización, y viendo cómo el sistema puede impulsar dichas metas y sus métricas asociadas.
- La descomposición del dominio consiste en tomar los distintos conceptos que se manejan en el universo de discurso del negocio y listar las distintas funcionalidades que puede ofrecer el sistema en relación con ellos. Por ejemplo, si se tiene el concepto de un pedido, el sistema podría crearlos, consultarlos, modificarlos, crear una factura para ellos, etc.
- El análisis de los recursos existentes consiste en tomar los sistemas que ya estén implantados y ver cómo se podría reutilizar parte de su funcionalidad en otros puntos de la organización. Muchos de estos sistemas ya existentes son demasiado críticos como para ser reemplazados de un día para otro.

Los servicios obtenidos mediante estos métodos se estructuran en una jerarquía, sobre la que se aplica una prueba de paso («litmus test» en el original) personalizable para ver si realmente merece la pena incurrir en el coste adicional que supone exponerlo como servicio web.

La metodología se halla implementada como una serie de extensiones al Proceso Unificado de Rational («Rational Unified Process» o RUP)), utiliza el estándar Lenguaje Unificado de Modelado («Unified Modeling Language» o UML) Object Management Group [2009b] con algunas extensiones para modelado.

Aunque se ha validado a través de múltiples proyectos y se apoya sobre un proceso conocido, SOMA es una metodología compleja en la que se elaboran un gran número de documentos intermedios y se utilizan numerosas herramientas. Por ello, esta metodología puede no ser la mejor opción para empresas pequeñas y medianas de fabricación, con menos recursos para realizar este tipo de modelado extensivo. Una metodología más ligera sería más efectiva en estos casos.

3.2.3. Metodología SOD-M

La metodología Service Oriented Development Method (SOD-M) de Castro [2007]; Vara Mesa et al. [2008], desarrollada por el grupo Kybele de la Universidad Rey Juan Carlos, es también dirigida por modelos y centrada en servicios. Cubre los tres puntos de vista del enfoque MDA del OMG con perspectivas sobre el negocio (nivel CIM) y sobre el sistema (niveles PIM y PSM). Puede verse un esquema de las relaciones entre los distintos modelos en la figura 3.1.

La perspectiva del negocio incluye un modelo de la organización y su entorno especificado sobre los intercambios de valor entre las distintas partes de la organización Gordijn and Akkermans [2003], y un modelo de los procesos de negocio seguidos, utilizando diagramas UML de actividad. Estos diagramas son parecidos a los conocidos diagramas de flujo.

La definición de la perspectiva del sistema (nivel PIM), por otro lado, comienza por establecer modelos de casos de uso del sistema de información, que después se integran y estructuran en modelos de procesos de servicio del sistema. Estos modelos se integran y estructuran en una serie de modelos de casos de uso extendidos, que después se concretarán a modelos de procesos de servicio del sistema.

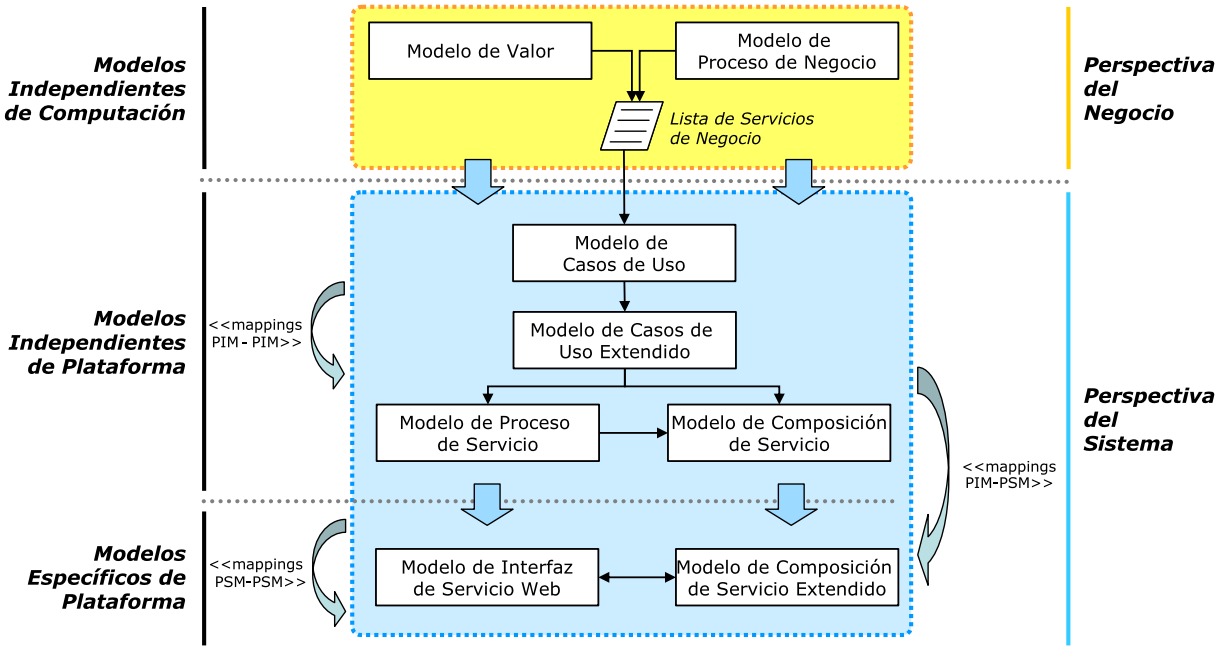


Figura 3.1. Esquema de los modelos utilizados en SOD-M

Los modelos de procesos de servicio detallan las actividades necesarias para su realización y las relaciones con los demás procesos. Posteriormente, esas actividades se reparten entre los participantes en modelos de composición de servicio. Estos dos últimos tipos de modelos también usan diagramas de actividad de UML.

Finalmente, en el nivel PSM de la perspectiva del sistema se decide en el modelo de composición de servicio extendido qué actividades de los modelos de composición de servicio extendido serán servicios web, y se definen sus interfaces.

Esta metodología resulta ser más ligera que en el caso de SOMA, y algunas de las correspondencias entre los distintos modelos llegan a estar automatizadas parcialmente Vara Mesa et al. [2008], a diferencia de otros métodos, como el de Stojanović Stojanović [2005]. Utiliza notaciones más sencillas, facilitando la comunicación entre clientes y desarrolladores.

Esta metodología es una buena candidata sobre la que basarse para definir una metodología completa para el desarrollo de sistemas de información orientados a servicios para empresas de fabricación distribuida. Pueden representarse de forma comprensible para las partes implicadas las prácticas actuales de negocio de la organización, y a partir de ellas descender hasta obtener un esqueleto de la SOA.

Es cierto que, a diferencia de SOMA, no incluye todos los pasos del proceso de desarrollo, y en particular no integra actividades de realización de pruebas sobre el software. Sin embargo, estas carencias pueden cubrirse extendiendo SOD-M en lo estrictamente necesario, e integrándolo en metodologías de nivel superior, como es el caso de MIDAS.

3.3. Metodología base escogida: SOD-M

En la anterior sección se seleccionó la metodología SOD-M, debido a que era capaz de modelar la SOA partiendo de la situación actual de la organización, al mismo tiempo que era suficientemente ligera como para aplicarse en organizaciones de menor tamaño.

En esta sección se describirán en más detalle los pasos seguidos por dicha metodología, describiendo en paralelo los conceptos subyacentes y las notaciones utilizadas. Inspirándose en la propuesta MDA del OMG Object Management Group [2003], la metodología parte de unos modelos de alto nivel del negocio, y va bajando el nivel de abstracción hasta obtener unos modelos cercanos a la implementación final. Los modelos utilizados se encuentran en la figura 3.1.

3.3.1. Diagramas UML utilizados

SOD-M define sus modelos basándose en el Lenguaje Unificado de Modelado («Unified Modeling Language» o UML) definido por el Object Management Group Object Management Group [2009b]. En particular, utiliza tres tipos de diagramas UML: diagramas de clases, diagramas de casos de uso y diagramas de actividad. En esta sección se expondrán los elementos de UML que SOD-M emplea de cada diagrama. SOD-M no considera todos los elementos definidos por UML, ya que se trata de una especificación considerablemente compleja.

Estos diagramas UML son extendidos utilizando perfiles, que añaden nuevos estereotipos con los que pueden marcarse los nodos y enlaces de los diagramas para reflejar aspectos semánticos. Estas extensiones se describirán en apartados posteriores.

Diagramas de clases

Los diagramas de clases permiten representar los distintos elementos que se procesarán en el sistema, de acuerdo con el paradigma de la programación orientada a objetos.

Una «clase» es un concepto abstracto que representa a toda una categoría de «objetos» que la instancian. Por ejemplo, el concepto abstracto *Coche* como «vehículo autopropulsado de cuatro ruedas» es una clase, y «ése coche» es un objeto de dicha clase.

Los objetos de una clase tienen asociados ciertos campos con información acerca de ellos (conocidos como «atributos»), y pueden incorporar ciertos comportamientos (conocidos como «métodos»). Los atributos y métodos se definen a nivel de clase.

Se dice que la clase A es subclase de una clase B cuando todos los objetos de A son también objetos de B. Es decir, los objetos de A son un tipo de los objetos de B. Volviendo al ejemplo anterior, *Turismo* sería una subclase de *Coche*, ya que todo *Turismo* es un *Coche*. Esta relación puede expresarse de otras formas: B es superclase de A, A hereda de B, B es padre de A, o A es hija de B. La ventaja de establecer estas relaciones es que las subclases heredan todos los atributos y métodos de sus superclases: por ejemplo, si *Coche* tuviera el atributo `cilindrada`, entonces *Turismo* también lo tendría.

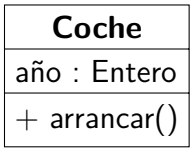
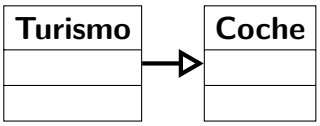
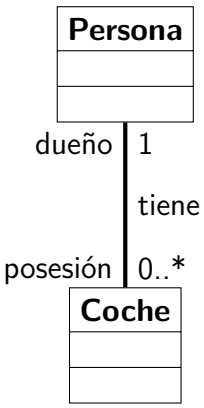
Existen varias formas de especificar las relaciones de herencia en UML, pero en estos diagramas consideraremos únicamente el tipo más común: herencia completa y disjunta. Es decir, no existen más subclases que las que indiquemos explícitamente, y un objeto sólo puede pertenecer a una subclase a la vez. Es decir, si sólo definimos las subclases *Turismo* y *Monovolumen*, entonces todo *Coche* será de uno y sólo uno de esos tipos. Así, no tendremos *Deportivo*s.

Puede que los objetos de una clase se relacionen con los objetos de otra clase. Un *Coche* tendrá una *Persona* que será su dueño (cardinalidad 1 en este extremo), y una *Persona* podrá ser dueño de ninguno o varios coches (cardinalidad cero o más). Cada clase juega un rol en dicha relación: aquí, *Coche* es una posesión, mientras que *Persona* es el dueño.

Un tipo más concreto de relación es el de composición, en el que se indica que un objeto contiene a otros objetos. Es decir, si desaparece el objeto de nivel superior, desaparecerán también los objetos de nivel inferior. Un *Motor* es parte de un *Coche*, y depende de éste: si el *Coche* se descarta, normalmente se descartará también el *Motor*.

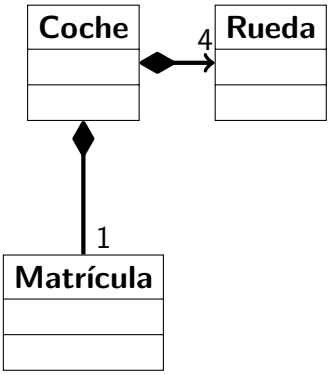
La notación utilizada en los diagramas de clase se describe en la tabla 3.2.

Tabla 3.2. Elementos de los diagramas UML de clases utilizados

Nombre	Notación	Significado
Clase		Categoría que engloba a una serie de objetos que se corresponden con un concepto abstracto. Todo objeto de dicha clase tiene asociado varios campos con información (atributos) y determinados comportamientos (métodos). Aquí tenemos una clase que representa a todos los *Coche*s, con un atributo que contiene su año de entrada a circulación, y un método que sirve para arrancarlo.
Relación de herencia		Relación de una clase origen A a una clase destino B, que indica que todo objeto de A es también un objeto de B, y hereda sus atributos y métodos. Por ejemplo, a la izquierda se dice que los *Turismo*s son un tipo de *Coche*s.
Relación de asociación		Relación entre dos clases (no necesariamente diferentes) que indica que cada objeto de una clase está asociado con un determinado número («cardinalidad») de objetos de la otra clase y viceversa. Cada uno de los objetos en cada extremo de la relación juega un determinado papel o «rol». En general, las asociaciones correspondientes a esta relación son bidireccionales, es decir, pueden seguirse en ambas direcciones. Si se pone una punta de flecha en alguno de los extremos, se indica que la relación es unidireccional. El nombre de la relación puede quedar implícito en ciertos casos, así como los roles o las cardinalidades (por omisión, se asumen iguales a 1).

Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
		El ejemplo de la izquierda muestra una relación de asociación bidireccional llamada «tiene», indicando que una *Persona* puede tener cero o más *Coche*s, y todo *Coche* es poseído por una sola *Persona*.
Relación de composición		Tipo de relación de asociación que al ir de una clase origen A a una clase destino B, indica que los objetos de A contienen objetos de B. Es decir, si desaparece el objeto de A, desaparecerán también los objetos de B. En el extremo de B puede aparecer una cardinalidad, indicando cuántos elementos están contenidos. En este caso, se tiene que un *Coche* tiene una *Matrícula* y cuatro *Rueda*s. Al no situar una punta de flecha en el extremo de *Matrícula* en su relación de composición, se está indicando que puede saberse el *Coche* a partir de una *Matrícula*.

Diagramas de casos de uso

Los diagramas de casos de uso representan las formas en que los participantes en la parcela de negocio en que opera el sistema (los «actores») usarán el sistema para conseguir un resultado tangible final (sus «casos de uso»).

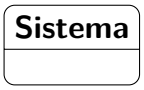
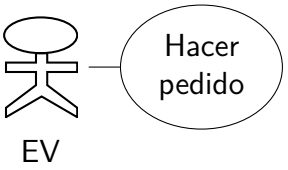
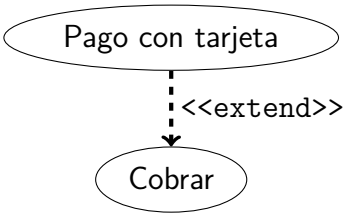
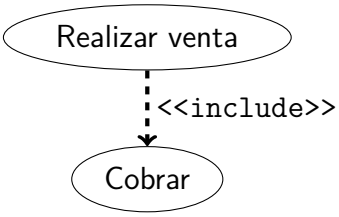
Estos diagramas enlazan a cada uno de los actores con los casos de uso en que son parte interesada, bien porque operan el sistema en algún punto, o porque afectan o son afectados por los resultados (p.ej. como el Ministerio de Hacienda cuando se realiza un cobro).

También pueden enlazar un caso de uso con otro, si se considera que su funcionalidad está relacionada: puede que un caso de uso incluya los pasos seguidos en otro caso de uso, o que los extienda.

Normalmente, los casos de uso se representan dentro de un elemento que simboliza los límites del sistema, y los actores se sitúan fuera de este límite. Esto ayuda a visualizar más fácilmente qué hace y qué no hace el sistema, y quién se ve afectado en alguna medida por su funcionamiento.

La notación seguida se describe en la tabla 3.3.

Tabla 3.3. Elementos de los diagramas UML de casos de uso utilizados

Nombre	Notación	Significado
Actor	 Empleado de Ventas	Persona o entidad que se ve afectada o que utiliza el sistema en alguna medida, como este *Empleado de Ventas* si se considerara un sistema de gestión de ventas, por ejemplo.
Límite del sistema		Representa todo lo que hará el sistema. Los casos de uso irán dentro de este contenedor, y los actores se quedarán fuera. Por restricciones de espacio, en esta tabla sólo los representaremos aquí.
Caso de uso	 EV	Secuencia de acciones que realiza el sistema que produce un resultado con un cierto de valor para uno o más de los actores que interactúan con el sistema. En este caso, *Registrar venta* produce un resultado para el *Empleado de Ventas* (EV) anterior, por lo que se enlazan entre sí. Opcionalmente, el enlace puede tener una punta de flecha en el extremo del caso de uso.
Relación de extensión		Relación entre dos casos de uso en la que uno de ellos extiende el comportamiento del otro. Aquí, *Cobrar con tarjeta* sólo extiende lo necesario de *Cobrar* para especificar las cosas que varían al usar este medio de pago.
Relación de inclusión		Relación entre casos de so en la que uno de ellos incluye el comportamiento del otro. Por ejemplo, para realizar una venta, una de las cosas que hay que hacer es cobrar los artículos.

Diagramas de actividad

Los diagramas de actividad UML recogen los pasos necesarios para llevar a cabo una acción en el sistema, indicando el orden en que se ejecutan, cómo se solapan estos pasos y bajo qué condiciones se ejecutan ciertos casos y no otros. En este sentido, se podrían ver como un tipo de diagrama de flujo.

En todos los diagramas de actividad, la ejecución empieza por un nodo inicial, y va progresando por una serie de flujos de control, hasta llegar a algún nodo final. Los nodos intermedios pueden

ser de varios tipos:

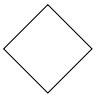
- Subactividades o acciones, que realizan alguna tarea concreta.
- Nodos de decisión, que escogen el flujo de control saliente cuya condición se cumpla.
- Nodos *fork* o de ramificación, que dividen la ejecución en dos o más ramas paralelas.
- Nodos *join* o de reunión, que esperan a que dos o más ramas terminen para proseguir con una sola rama.
- Nodos *merge* o de fusión, que reúnen varias ramas separadas por una condición en una sola.

En lugar de usar flujos de control, es posible usar flujos de objetos, que modelan la información que se pasa de una tarea a otra. Otra posible variación es la situación de los nodos en pistas o *swimlanes*, indicando qué participante es ocupado de realizar cada actividad.

Los diagramas de actividad UML completos incorporan otras características, como la capacidad de activar ciertos flujos de control ante ciertos eventos o la posibilidad de dividir el diagrama en varias partes utilizando nodos de referencia, pero no se utilizarán estas capacidades en este TFM.

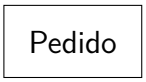
La notación usada se describe en la tabla 3.4.

Tabla 3.4. Elementos de los diagramas UML de actividad simplificados utilizados

Nombre	Notación	Significado
Nodo inicial		Único punto de partida de la ejecución.
Nodo final de actividad		Marca el final de la ejecución de la actividad completa.
Nodo final de flujo		Marca el final de la ejecución de la rama actual. El resto de ramas no se ven afectadas.
Nodo fork o de ramificación		Divide la rama actual de ejecución en varias, que se ejecutan a la vez. Tiene varios flujos entrantes y sólo uno saliente.
Nodo join o de reunión		Reúne varias ramas de ejecución que se dividieron a partir de un nodo *fork* en una sola, esperando a que terminen todas. Tiene varios flujos salientes y sólo uno entrante.
Nodo merge o de fusión		Reúne varias ramas de ejecución que se dividieron a partir de un nodo de decisión en una sola, esperando a que termine una de ellas. Tiene varios flujos entrantes y sólo uno saliente.

Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
Nodo de decisión		Hace que la ejecución prosiga por el flujo de control saliente para el que se cumpla la condición. Tiene varios flujos salientes y sólo uno entrante.
Actividad o acción		Representa una tarea conocida a llevar a cabo por el sistema o alguno de los participantes. La diferencia entre una actividad y una acción está en que una actividad se descompone en varios de nivel inferior, mientras que una acción es indivisible.
Nodo objeto		Simboliza un objeto o mensaje que se pasa entre dos actividades o acciones. Típicamente, se utiliza para indicar cómo la salida de una tarea se utiliza como entrada de la siguiente.
Flujo de control		Especifica que cuando haya concluido la ejecución del elemento del que parte, la ejecución proseguirá por su elemento destino.
Flujo de objeto		Si parte de una actividad o acción, indica que enviará el nodo objeto al que apunta. Si parte de un nodo objeto, señala la actividad o acción a la que se envía el objeto representado.

3.3.2. Modelos independientes de computación

En el anterior apartado se describieron los diagramas UML en que se basaba SOD-M. El resto de esta sección se dedicará a detallar cada uno de los pasos seguidos en SOD-M, así como la notación utilizada.

El primer paso en las metodologías basadas en el enfoque MDA de OMG, como SOD-M, es definir una serie de modelos del negocio, de forma que el sistema obtenido finalmente esté de acuerdo con la situación actual de la empresa que va a explotarlo.

Estos modelos no tienen por qué tener en cuenta ningún aspecto del sistema final. Sirven únicamente para formalizar cómo funciona la empresa actualmente, con qué entidades se relaciona y de qué manera.

Modelos de valor

Los primeros modelos creados en esta metodología son los *modelos de valor*, propuestos por Gordijn en Gordijn and Akkermans [2003] e implementados en Gordijn and Akkermans [2006].

Estos modelos sirven para describir las ideas de negocio sobre las que opera la empresa, en términos de los intercambios de valor que se producen entre los distintos participantes en su entorno. Los elementos de los modelos de valor utilizados en este TFM, su significado y la notación empleada se hallan en la tabla 3.5.

Estos intercambios se describen de forma abstracta, sin entrar en detalles de cómo se llevan a cabo físicamente. Por ejemplo, no se describe si una transacción monetaria se realiza en efectivo o con cargo a cuenta, ni el medio de transporte de una mercancía.

Para interpretar un modelo de valor, es necesario partir del estímulo inicial, un nodo especial que marca la generación de una necesidad del consumidor final. Esta necesidad desencadena una serie de intercambios de valor entre los participantes a lo largo del modelo, que acaban en varios nodos finales.

Para elaborar una primera versión del modelo de valor a partir de una descripción textual de la organización y su entorno, hay que realizar una serie de tareas:

1. Identificación del escenario operacional, es decir, qué productos, servicios o experiencias desean los consumidores del negocio.

2. Identificación de los actores, tanto si son los consumidores finales como si contribuyen a la realización de alguna necesidad de los primeros.

3. Identificación de los objetos, viendo qué envía y qué recibe cada uno de los actores.

4. Identificación de los caminos de dependencia, con sus estímulos finales y la secuencia de los intercambios necesarios para realizar la necesidad del consumidor final.

Modelos de proceso de negocio

Los modelos de proceso de negocio describen las distintas tareas que hay que realizar para obtener un resultado tangible de negocio. Las tareas son también a nivel de negocio, por lo que son demasiado abstractas como para ser implementadas directamente en un sistema de información. Estos modelos sólo sirven para conocer las prácticas actuales de la organización.

La notación utilizada es la misma que en los diagramas de actividad UML de §3.3.1. Para crear un modelo de proceso de negocio, hay que seguir estos pasos:

1. Identificación de las actividades del proceso de negocio, es decir, de las tareas necesarias para conseguir el resultado de negocio. Las actividades pueden variar en gran medida en nivel de detalle y complejidad.

2. Identificación de los flujos de control y de objeto, indicando qué actividades van antes de otras, y qué flujos son paralelos o alternativos.

Lista de servicios de negocio

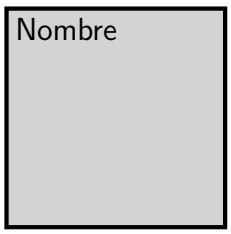
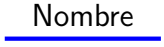
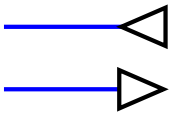
La lista de servicios de negocio consiste únicamente en una enumeración de todos los servicios que se ofrecen en la organización para satisfacer una necesidad del consumidor final. Cada entrada incluye el nombre del servicio y el consumidor final cuya necesidad resuelve.

A la hora de elaborar dicha lista, es necesario primero identificar quiénes son los consumidores finales de los servicios de la organización. Se puede partir del modelo de valor: todo actor que

contenga un estímulo inicial de algún camino de dependencias es un consumidor final, que para cumplir su necesidad causa una serie de intercambios de valor entre los participantes.

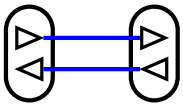
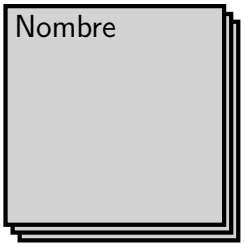
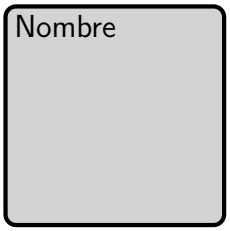
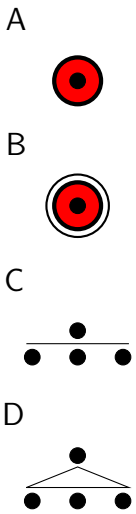
A continuación, hay que identificar los servicios que requieren dichos consumidores. Para ello, puede partirse de los estímulos iniciales del modelo de valor, o de las actividades del modelo de proceso de negocio.

Tabla 3.5. Elementos de los modelos de valor

Nombre	Notación	Significado
Actor		Entidad económica independiente, como una empresa o un consumidor final. Se entiende por económicamente independiente que puede conseguir beneficios tras un intervalo de tiempo (en el caso de una empresa), o que puede obtener un valor añadido (en el caso de un consumidor final). Para que la idea de negocio sea viable, todo actor debe cumplir alguna de las dos condiciones anteriores.
Objeto de valor		Tipo de servicio, producto o experiencia que tiene valor económico para al menos uno de los actores involucrados en el modelo.
Puerto de valor		Punto a partir del cual un actor envía (en el caso de un puerto de salida) o recibe (en el caso de un puerto de salida) objetos de valor.
Ofrecimiento de valor	(sin representación)	Reúne todo lo que un actor envía (en el caso de un ofrecimiento saliente) o recibe (para un ofrecimiento entrante) en un determinado intercambio con el entorno. Consiste en un conjunto de puertos de valor de la misma dirección, asociados a una misma interfaz de valor. La idea es que para realizarse, deben recibirse o enviarse objetos de valor a través de todos los puertos.

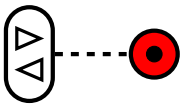
Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
Interfaz de valor		Reúne un ofrecimiento de valor saliente que hace un actor a su entorno, y el ofrecimiento de valor entrante que espera recibir. Para que se active, deben enviarse y recibirse todos los objetos a la vez a través de sus puertos de valor. Se dibujan sobre los extremos de los actores y segmentos de mercado.
Intercambio de valor		Conexión entre dos puntos de valor, indicando que los participantes están dispuestos a intercambiar objetos de valor entre sí.
Segmento de mercado		Reúne a una categoría de actores que comparten unas mismas interfaces de valor. Es posible que un actor determinado tenga más interfaces además de éstas, en cuyo caso se modelaría explícitamente aparte.
Actividad de valor		Actividades de un actor que le producen algún tipo de beneficio en términos de valor. Aparecen dentro de un actor, indicando que es el actor que la ejecuta.
Nodo de dependencia		Nodos que describen los distintos escenarios que ejemplifican la forma en que ciertos actores cubren sus necesidades, partiendo de un estímulo inicial (A) y acabando en una serie de estímulos finales (B). Es posible que un intercambio resulte en varios intercambios, que tengan que realizarse todos (*AND fork*, C) o al menos uno de ellos (*OR fork*, D). También puede que tras hacerse todos los intercambios de una lista determinada, tenga que hacerse otro (*AND join*, C). En el caso en que sólo sea necesario que se haga uno de dicha lista, se habla de un *OR join* (D).

Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
Camino de dependencia		Conecta un nodo de dependencia a una interfaz de valor.

3.3.3. Modelos independientes de plataforma

Mediante los modelos independientes de computación, se describió la empresa y su entorno y se obtuvo una lista de los servicios que necesitan los consumidores finales.

En esta segunda fase, se obtendrán una serie de modelos que describirán la funcionalidad que deberá tener el sistema, sin entrar en detalles específicos de implementación, como las tecnologías empleadas o el entorno en que se va a implantar.

Modelos de casos de uso

Los modelo de casos de uso representan en SOD-M las relaciones entre los servicios de negocio que ofrecerá el sistema y los consumidores finales que los utilizarán. Se usan diagramas de casos de uso como los de §3.3.1, aunque ligeramente modificados (véase la tabla 3.6).

No se usan los casos de uso originales de UML, sino un tipo específico, llamado «servicio de negocio». Cada servicio de negocio está enlazado con exactamente un «consumidor final», un tipo especial de actor. Al mismo tiempo, todo consumidor final utiliza un solo servicio de negocio.

Para elaborar el modelo de casos de uso, que ya es el primer modelo del sistema final, tendremos que considerar exactamente cuál será su alcance, y quién usará dicho sistema. Puede que no todos los servicios de negocio se ofrezcan como servicios del sistema, y puede que no sean los consumidores finales de dichos servicios los que interactúen con el sistema, sino unos intermediarios (normalmente personal de la empresa).

Una vez hayamos decidido cuáles serán los servicios de negocio que ofrecerá el sistema y quiénes son los consumidores finales del sistema de información, los relacionaremos entre sí con los enlaces oportunos.

Modelos de casos de uso extendidos

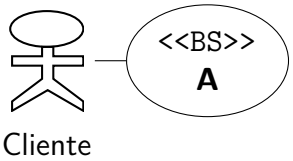
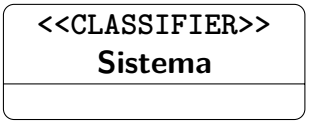
Los modelos de casos de uso extendidos toman los servicios de negocio anteriores y los descomponen en casos de uso «compuestos», que a su vez se dividen en varios casos de uso «básicos», que son más sencillos de implementar.

La descomposición de los servicios de negocio ha de hacerse apoyándose en la información de los modelos de valor y los modelos de proceso de negocio. Así, puede que se necesiten determinadas funcionalidades para llevar a cabo un intercambio de valor, o para implementar alguna de las acciones del modelo de proceso de servicio.

Además, se establecerán relaciones de inclusión y de extensión entre los casos de uso básicos y compuestos. Estas relaciones servirán para definir una primera aproximación de la estructura de los modelos de proceso de servicio, como se verá a continuación.

En SOD-M también se distingue entre los casos de uso «estructurales» y los casos de uso «funcionales». Los casos de uso estructurales se corresponden con la gestión de determinados

Tabla 3.6. Elementos nuevos o cambiados en los modelos de casos de uso de SOD-M

Nombre	Notación	Significado
Consumidor final	 Cliente	Tipo específico de actor que modela a un usuario que va a obtener un servicio mediante el sistema.
Servicio de negocio		Servicio que ofrece el sistema, produciendo un resultado con un valor tangible para el consumidor final.
Límite del sistema		En semántica es igual al límite del sistema original, pero la notación ha cambiado a la de la izquierda.

tipos de recursos presentes en el sistema (como «Crear reserva»), mientras que los funcionales implementan funcionalidades más complejas.

Las variaciones sobre la notación de los diagramas de casos de uso normales de UML se hallan en la tabla 3.7.

Modelos de proceso de servicio

Los modelos de caso de uso extendidos antes obtenidos dan una visión de qué funcionalidades proporciona el sistema. El siguiente paso es indicar cómo se reúnen estas funcionalidades para proveer los servicios que los consumidores finales desean. Los modelos de proceso de servicio incluyen esta información.

Los modelos de procesos de servicio son diagramas de actividad UML en los que se interconectan una serie de «actividades de servicio», un tipo específico de actividad. Para obtener una primera versión del modelo de proceso de servicio, se puede convertir cada caso de uso básico del modelo de casos de uso extendido en una actividad de servicio, y aplicar las reglas de la tabla 3.8 para obtener la forma en que se interconectan entre sí.

En los modelos de procesos de servicio no se utilizan pistas, ya que no se modelan aún a los distintos participantes del negocio.

Modelos de composiciones de servicio

Los modelos de composiciones de servicio toman los anteriores modelos de proceso de servicio y descomponen las actividades de servicio en acciones más concretas que ya pueden ser implementadas directamente en el sistema.

Además, reparten dichas acciones entre cada uno de los participantes o «colaboradores del negocio», que modelan como pistas o *swimlanes*. Estos colaboradores se comunican entre sí a través del intercambio de nodos objeto.

Los colaboradores del negocio se obtienen a partir del modelo de valor: cada actor o segmento de mercado que realiza una serie de actividades de valor relacionadas con la composición de servicio en cuestión ha de ser considerado.

Tabla 3.7. Elementos nuevos o cambiados en los modelos de casos de uso extendidos de SOD-M

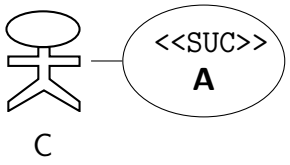
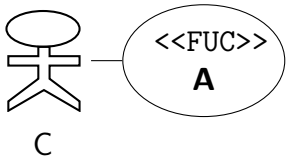
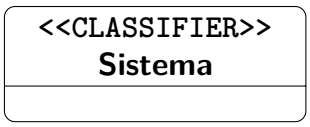
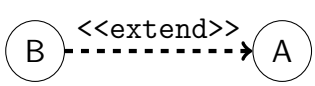
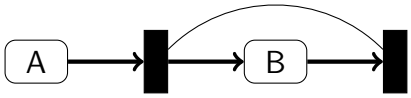
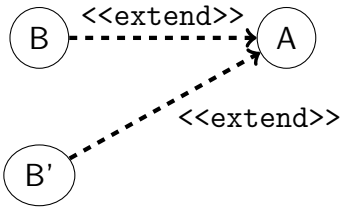
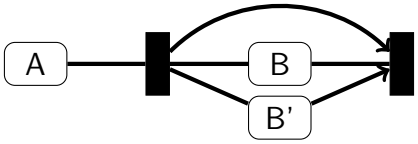
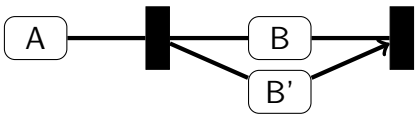
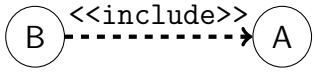
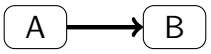
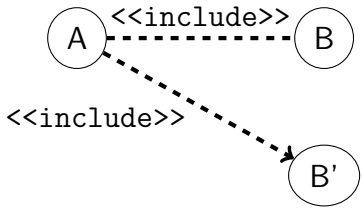
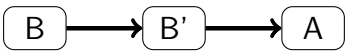
Nombre	Notación	Significado
Consumidor final	 Cliente	Tipo específico de actor que modela a un usuario que va a obtener un servicio mediante el sistema.
Caso de uso compuesto	Depende de si es funcional o estructural.	Conjunto de acciones requerido para prestar un servicio de negocio. Se descompone en varios casos de uso básicos, y puede ser estructural o funcional.
Caso de uso básico	Depende de si es funcional o estructural.	Conjunto de acciones requerido para prestar un servicio de negocio. Puede ser estructural o funcional.
Caso de uso estructural		Conjunto de acciones requerido para prestar un servicio de negocio, que manipula un determinado tipo de recurso del sistema. Puede ser compuesto o básico.
Caso de uso funcional		Conjunto de acciones requerido para prestar un servicio de negocio, que implementa una funcionalidad de nivel superior a la de un caso de uso estructural. Puede ser compuesto o básico.
Límite del sistema		En semántica es igual al límite del sistema original, pero la notación ha cambiado a la de la izquierda.

Tabla 3.8. Correspondencias de los modelos de casos de uso extendido a los modelos de proceso de servicio

Modelo de casos de uso extendido	Modelo de proceso de servicio
	
	 o bien 
	
	 o bien 

3.3.4. Modelos específicos de plataforma

La última parte del proceso de SOD-M consiste en definir una serie de modelos que sí dependen de las tecnologías usadas, por lo que se conocen como modelos específicos de plataforma. Se emplean dos tipos: los modelos de composiciones de servicio extendido y los modelos de interfaces de Servicio Web.

Los modelos de composiciones de servicio extendido son iguales a los modelos de composiciones de servicio del nivel PIM, pero marcan algunas de las acciones con el estereotipo <<WS>> (de *Web Service*), indicando que dicha acción se ofrecerá al resto de la SOA como un Servicio Web.

En este TFM no se describirán en profundidad los modelos de interfaces de Servicio Web, ya que por limitaciones de tiempo no han sido utilizados, y no ha sido necesaria su extensión para modelar aspectos de prueba. Baste decir que consisten en una correspondencia de los elementos de la especificación WSDL World Wide Web Consortium [2007] a diagramas de clases de UML. Cada modelo de interfaz de Servicio Web describe las operaciones que permite realizar cada acción ofrecida como un Servicio Web, así como el formato de los mensajes que acepta y envía.

3.4. Extensiones propuestas sobre SOD-M para integración de pruebas

Anteriormente, se ha seleccionado SOD-M como una candidata viable de la que partir para definir una metodología SOA ligera y completa, y se ha descrito en detalle su funcionamiento. Este análisis ha revelado una carencia importante en dicha metodología: no integra procesos de prueba.

En esta sección se proponen algunas mejoras sobre SOD-M para ayudar a cubrir dicha carencia y asistir a las pruebas sobre los sistemas de información de empresas de fabricación distribuida, ya publicadas en congresos de nivel nacional García Domínguez et al. [2009]; Medina Bulo et al. [2009].

A grandes rasgos, existen seis categorías de pruebas de sistemas de información Myers [2004]: pruebas de instalación, aceptación, sistema, función, integración y módulo. Se considerarán en este trabajo aquellos tipos de pruebas del software que requieren un tratamiento especial en SOA: las pruebas de sistema, función e integración. Las pruebas de aceptación podrían hacer uso de la trazabilidad entre los modelos de los niveles CIM y PIM de SOD-M: no es necesario extender estos modelos, sino refinar las herramientas.

Para aprovechar el enfoque dirigido por modelos, las pruebas deben integrarse en un ciclo de realimentación en el que los resultados de las pruebas y la calidad medida de los casos de prueba sirvan para revisar los modelos existentes.

3.4.1. Pruebas de sistema

Las pruebas de sistema cubrirían en el caso de SOA a la arquitectura completa, es decir, al ecosistema creado por todos los servicios expuestos y sus consumidores. Sería útil controlar el riesgo de que el sistema fuese incapaz en algún momento de cumplir ciertas propiedades deseables, como rendimiento (en transacciones/minuto), número máximo de accesos concurrentes, el volumen máximo de datos admitido, las restricciones de acceso impuestas, etc. Muchas veces,

estas propiedades forman parte del acuerdo de nivel de servicio («Service Level Agreement» o SLA).

Para las pruebas de sistema, una posibilidad es anotar los modelos de procesos de servicio del nivel PIM con información acerca del nivel de servicio esperado (rendimiento, tiempo de respuesta, etc.) y las restricciones de seguridad. Dichas anotaciones se harían sobre los procesos de servicio completos y posteriormente se extenderían a cada una de las actividades, en función de la estructura interna del proceso de servicio.

Una actividad de servicio podría ser reutilizada en varios procesos de servicio, y tener restricciones distintas de rendimiento en cada uno. Habría que reunir estas restricciones en una sola (normalmente más estricta que todas las demás) para obtener la restricción final a utilizar en la actividad de servicio correspondiente de los modelos de composiciones de servicio. Se obtendrían a partir de las restricciones de una actividad de servicio de una composición de servicio las restricciones de las acciones en que ésta se descompone.

Las restricciones de rendimiento de las acciones individuales se usarían para generar casos de prueba que comprueben si en algún momento el sistema no sería capaz de cumplir los requisitos de alto nivel. Estos casos de prueba consistirían en entradas para herramientas de pruebas ya existentes como JUnitPerf Clark [2009] (para clases Java) o soapUI eviware.com [2009] (para servicios web que definan interfaces SOAP).

En la figura 3.2 puede verse un ejemplo de un modelo de proceso de servicio para atender un pedido de un cliente. Se ha decorado el modelo mediante una serie de nodos estereotipados, indicando para las actividades el número mínimo de transacciones por segundo que han de poder ser procesadas y el tiempo límite para cada una de ellas. Para las transiciones condicionales, se ha indicado una estimación de la probabilidad de que se tome cada rama. En función de la información especificada por el diseñador (distinguida con comentarios con un fondo sombreado), se podría derivar nueva información (comentarios transparentes). Por ejemplo, dado que la probabilidad de aceptar el pedido es $p = 0{,}8$, cada uno de los dos flujos concurrentes en caso de aceptación ha de poder procesar un mínimo de $5p = 4$ transacciones por segundo.

3.4.2. Pruebas de función e integración

Las pruebas de función tratan de ver si el sistema implementa correctamente un servicio de negocio determinado. Un servicio de negocio se define en SOD-M de Castro [2007] como «un servicio que forma parte de un negocio y que se ofrece para satisfacer una necesidad de un consumidor final». Por ejemplo, «vender una entrada de cine» es un servicio de negocio, pero no lo sería «autenticar a un usuario», dado que no aporta valor al consumidor final (el usuario).

Para dar dicho servicio de negocio, hay que seguir un proceso de servicio, en el que se detallan las actividades de servicio individuales que han de ser realizadas, y de qué forma deben realizarse. Por el alto nivel de abstracción al que operan, pueden usarse notaciones específicas para definir estos procesos de servicio, además de lenguajes de uso general más conocidos como C++ o Java. Por ejemplo, se podría usar la Business Process Model Notation (BPMN) Object Management Group [2009a] o el Web Service Business Process Execution Language (WS-BPEL) OASIS [2007].

Por otro lado, una prueba de integración trabaja sobre la implementación de una de las actividades de servicio del proceso de servicio que proporciona el servicio de negocio. Cada actividad de servicio se divide a su vez en varias acciones, y se implementa uniendo varios módulos o componentes. Se emplean normalmente lenguajes de uso general para implementar las actividades de servicio y sus acciones.

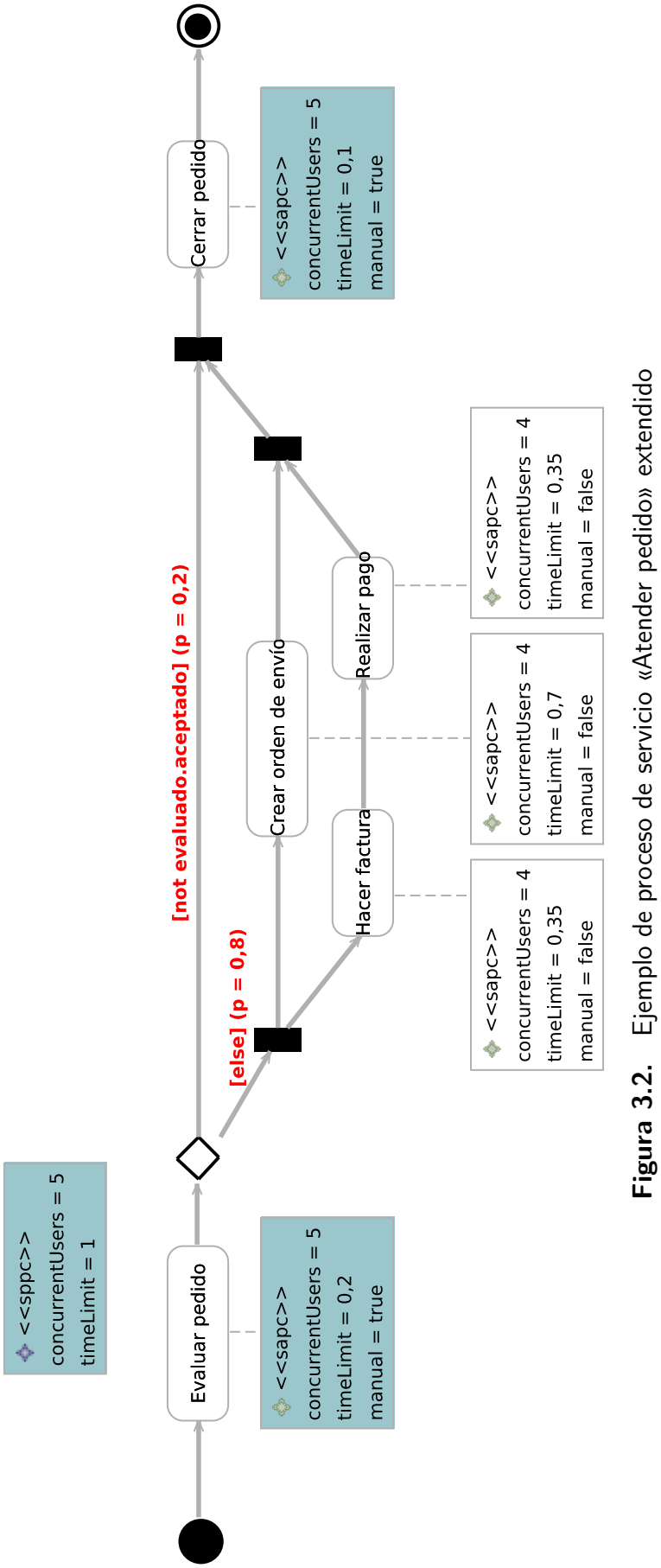


Figura 3.2. Ejemplo de proceso de servicio «Atender pedido» extendido

Para realizar las pruebas funcionales y de integración se puede tanto lanzar el código sobre unas entradas, viendo qué va haciendo y qué resultados produce, como sólo examinarlo para hacer varias comprobaciones. En ambos casos, necesitamos saber a nivel abstracto qué debe hacer cada acción del modelo de composiciones de servicio.

Para ello, hay que anotar las acciones de los modelos de composición de servicio con expresiones que relacionen las entradas, salidas y los cambios sobre la información almacenada. Desde el punto de vista de SOA, nos interesan sobre todo las acciones que serán implementadas como servicios web, marcadas con el estereotipo <<WS>> en el modelo PSM de composición de servicio extendido derivado del modelo PIM de composición de servicio. Las acciones a implementar como servicios web pasarán a formar parte del catálogo de servicios que serán reutilizados en el resto de la SOA, por lo que es necesario reducir el riesgo de que no se comporten de la forma esperada en algún momento.

SOD-M en sí no dispone de modelos para representar la información almacenada en el sistema, pero sus autores ya lo han tenido en cuenta al integrarlo con la metodología para sistemas de información vía Web MIDAS de Castro [2007], definiendo un modelo conceptual de datos a nivel PIM.

En la figura 3.3 puede verse el modelo de composición de servicio extendido derivado del modelo de proceso de negocio de la figura 3.2. Se han indicado con los estereotipos estándar <<precondition>> y <<postcondition>> de UML las expresiones booleanas escritas en el Object Constraint Language (OCL) Object Management Group [2006] a cumplir al inicio y final de la ejecución de la composición. Se han especificado con los estereotipos <<localPrecondition>> y <<localPostcondition>> las precondiciones (condiciones a ser cumplidas antes de su ejecución) y postcondiciones (condiciones a ser cumplidas tras su ejecución) de la acción «Hacer factura». Esta acción se implementará como un servicio web, ya que está marcada con el estereotipo <<WS>>.

Las precondiciones sobre la acción «Hacer factura», a implementar como servicio web, especifican que inicialmente el pedido ha de hallarse abierto, aceptado, no vacío, y no debe tener ya una factura asociada. Tras su ejecución, según las postcondiciones, debe haberse creado una factura nueva con los artículos, precios y valor total correctos, que se enviará a la acción «Realizar pago» para posterior procesamiento.

OCL no es el único lenguaje disponible para escribir las precondiciones y postcondiciones. Existen actualmente otras notaciones, como el Java Modeling Language (JML) Burdy et al. [2005] o Spec# Barnett et al. [2005] que tienen la ventaja de asemejarse más a lenguajes conocidos como Java o C#, respectivamente. Una especificación equivalente a la realizada en OCL en la figura 3.3, pero escrita en JML, tendría la forma del listado 3.1.

Por cuestiones de sintaxis ha sido necesario expresar la acción «Hacer factura» en forma de un método Java, pero el enfoque de JML puede adaptarse a otras tecnologías. En particular, Baresi Baresi et al. [2007] se basa en JML para definir el Web Service Constraint Language (WSCoL), expresando diversas restricciones sobre puntos de una definición de proceso WS-BPEL OASIS [2007].

JML y Spec# podrían ser especialmente útiles en las pruebas de integración. Ambos permiten decorar el código Java o C# final que implementará un determinado servicio web con contratos y realizar la compilación de forma que estos contratos se comprueben en tiempo de ejecución. Otro posible uso es como «oráculo» de la prueba: una prueba puede considerarse no superada si en la ejecución de algunos de los casos de prueba se ha violado uno de estos contratos.

En general, la decoración de las acciones individuales con precondiciones y postcondiciones a comprobar antes y después de su ejecución es más invasiva que una simple monitorización de

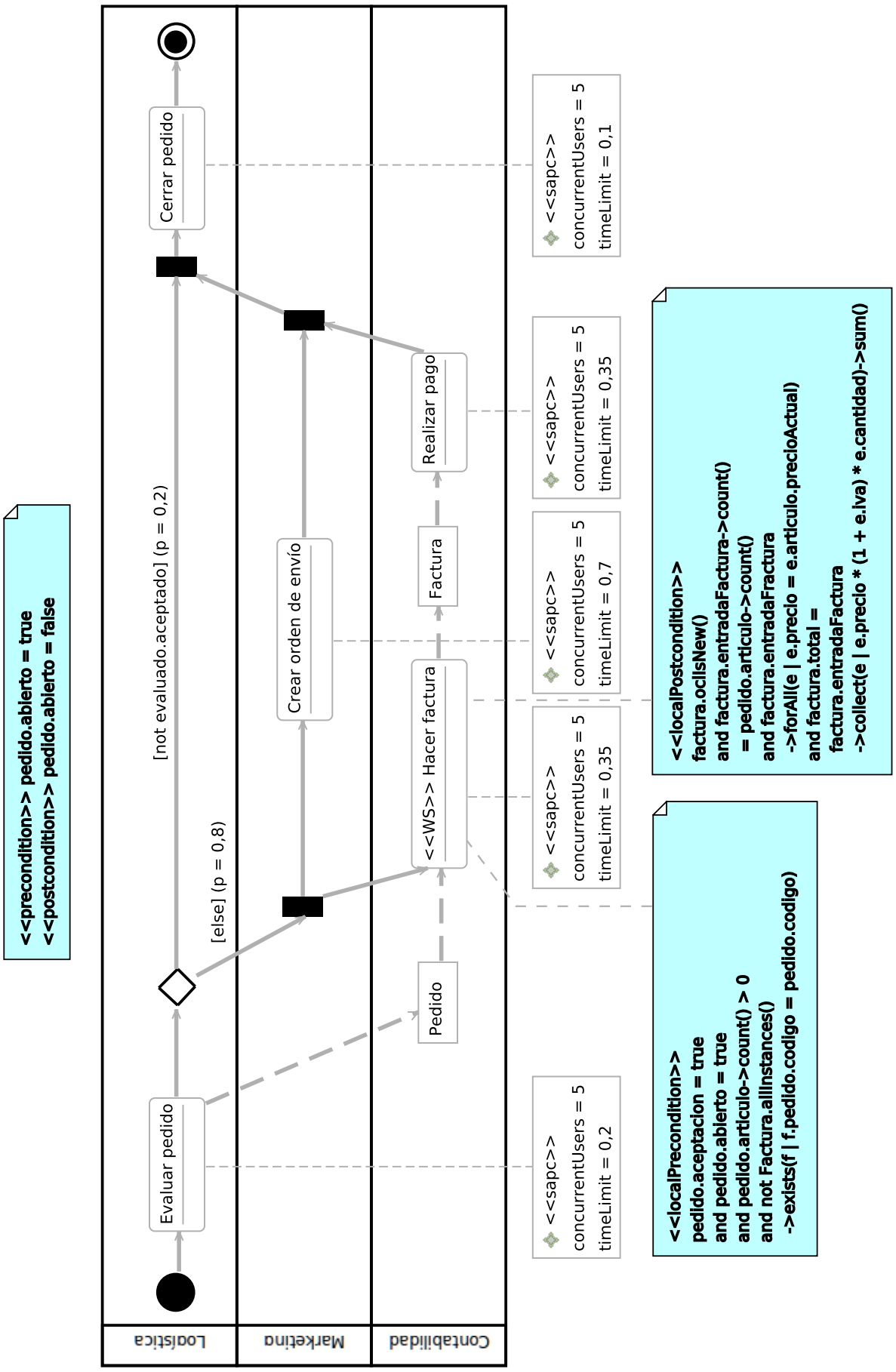


Figura 3.3. Ejemplo de composición de servicio «Atender pedido» extendida

Listado 3.1 Restricciones funcionales sobre «Hacer factura» en JML

```
/*@
  @ requires p.aceptacion == true
  @         && p.abierto == true
  @         && p.articulos.length > 0
  @         && !(\exists Factura f;
  @              Factura.listaFacturas.contains(f)
  @               ⇒ f.pedido.codigo == p.codigo);
  @
  @ ensures \fresh(\result)
  @        && \result.entradas.length == p.articulos.length
  @        && (\forall int i;
  @            i ≥0 && i < \result.entradas.length;
  @            \result.entradas[i].precio
  @               == p.articulos[i].precioActual)
  @        && (\sum int i;
  @             i ≥0 && i < \result.entradas.length;
  @            \result.entradas[i].precio) == \result.total)
  @*/
public Factura hacerFactura(Pedido p) {
   // ...
}
```

los mensajes entrantes y salientes, pero puede resultar en especificaciones de mayor calidad para servicios a reutilizar dentro de una organización, al ser refinadas y utilizadas tanto manualmente como por medio de un abanico de herramientas.

Por último, existen opciones que no utilizan asertos basados en expresiones booleanas. Sinha Sinha and Paradkar [2006] sugiere utilizar descripciones semánticas en base a una ontología, y Lohmann Lohmann et al. [2005, 2006, 2007] expone una notación gráfica basada en transformaciones de grafos. Esta última se acerca mucho al ideal del desarrollo basado en modelos, al ser fácil de entender y utilizar, pero parece no disponer de la misma potencia de expresión que otros lenguajes como OCL o JML.

Los contratos visuales se basan en una serie de reglas en las que el grafo a la izquierda de una flecha divisora (las precondiciones) se convierte en el grafo de la derecha (las postcondiciones). Cada operación de un servicio web puede tener asociada una o varias reglas. Para imponer condiciones que no deben cumplirse, los grafos se sitúan dentro de una caja gris con el estereotipo <<NAC>>. En general, los objetos que aparecen sólo a la izquierda han sido eliminados, y los que aparecen sólo a la derecha han sido añadidos. Los cuantificadores universales y existenciales pueden expresarse con multiobjetos, que representan colecciones de objetos de un determinado tipo y se denotan con múltiples rectángulos apilados.

Para comparar los contratos visuales con JML y OCL, se ha tratado de reescribir las restricciones anteriores en dicho formato, tal y como puede verse en la figura 3.4. Se puede comprobar la menor capacidad de expresión de esta notación: no se ha podido imponer que el total mostrado en la factura sea la suma de los precios individuales, o que los precios incluidos en la factura sean los precios más recientes establecidos sobre los productos.

3.4.3. Integración de técnicas de pruebas

Las especificaciones antes creadas sobre el rendimiento y comportamiento esperados de cada servicio y composición guiarán la aplicación de diversas técnicas de pruebas. En este apartado se discutirán técnicas basadas en la ejecución del código finalmente obtenido (pruebas dinámicas o simplemente «pruebas» según Ammann and Offutt [2008]), aunque se podría considerar utilizar pruebas estáticas («verificaciones» según la anterior referencia) sobre los modelos intermedios.

Los modelos servirán, fundamentalmente, para reunir un conjunto más exhaustivo de requisitos sobre los casos de prueba y presentar los resultados con un mayor nivel de abstracción. A su vez, los resultados de las pruebas se presentarán en los modelos de nivel superior, y permitirán su refinamiento.

Para asistir en la generación de casos de prueba, una posibilidad es derivar un grafo causa-efecto Myers [2004] de los modelos de composición de negocio, extrayendo requisitos sobre los casos de forma semiautomática. Paradkar Paradkar et al. [1997] describe un enfoque que opera sobre diagramas de estados UML. Habría que modificarlo para que operara sobre diagramas de actividades.

También puede realizarse una partición del espacio de entrada Lohmann et al. [2007] a partir de las definiciones de las precondiciones y postcondiciones, integrando varios enfoques como el uso de valores límite o el estudio de las intersecciones entre varias precondiciones. Sinha Sinha and Paradkar [2006] propone convertir las descripciones semánticas en una ontología a una máquina de estados extendida, que utiliza para generar casos de prueba.

Un problema con estas técnicas actualmente es que generan casos de prueba de forma independiente, y en diversos formatos. Sería interesante la definición de una notación genérica para los requisitos de los casos de prueba sobre un servicio web, que un componente externo de

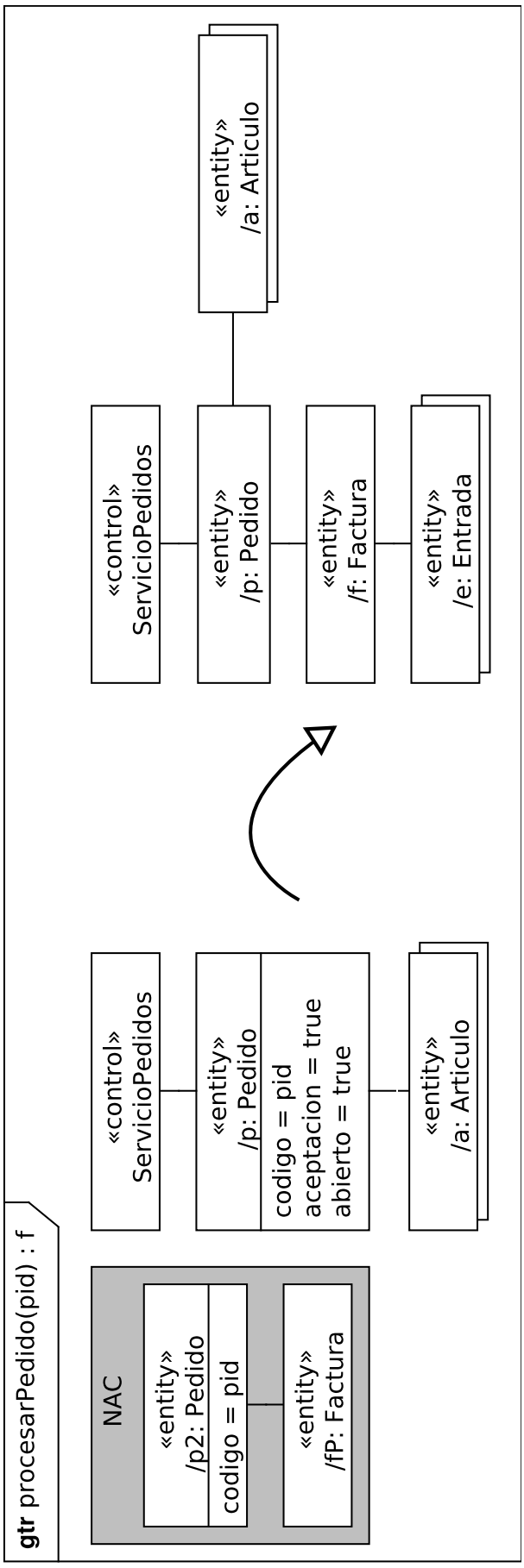


Figura 3.4. Ejemplo de un contrato visual basado en transformaciones de grafos para «Atender pedido»

generación pudiera recibir y ejecutar. Este componente sería el ocupado de contener la lógica específica relativa a la pila de tecnologías a utilizar y el formato XML de los mensajes.

Tras generar los casos de prueba para la implementación, habrá que evaluar su calidad. Existe un abanico de técnicas para ello, ya sea empleando los modelos originales de los que se derivó la implementación, o usando la propia implementación. Zhu realiza una revisión de ellas en Zhu et al. [1997].

Una elección popular es el análisis del grado de cobertura de los casos de prueba de las instrucciones y ramas condicionales del programa, como el realizado por la herramienta Cobertura Erdfelt et al. [2009] para programas Java. A la hora de presentar los resultados, se podría establecer una correspondencia entre la cobertura de nodos y transiciones del proceso WS-BPEL a la del modelo de composición de servicio extendido.

Otro tipo de técnica útil sería el análisis de mutaciones, ya sobre el código WS-BPEL Domínguez Jiménez et al. [2008] o ya sobre los contratos Jiang et al. [2005]. Esta técnica ayudaría a encontrar fallos de programación o violaciones en los contratos que los casos de prueba no serían capaces de detectar.

La generación dinámica de invariantes, finalmente, ayudaría a identificar propiedades Palomo Duarte et al. [2008] que fueran distintas de las esperadas a partir de las trazas de ejecución de los casos de prueba.

3.5. Conclusiones

Desarrollar una SOA es una tarea compleja que puede afectar a toda la organización, por lo que resulta necesario aplicar una metodología que guíe el proceso. En este capítulo se ha definido la metodología concreta que se utilizará en este TFM.

Se han comparado la metodología propuesta por Stojanović Stojanović [2005], la metodología SOMA de IBM Ghosh et al. [2008] y la metodología SOD-M del grupo Kybele de la Universidad Rey Juan Carlos de Castro [2007].

De entre estas tres, se ha escogido la metodología SOD-M como punto de partida para la metodología de este TFM, gracias a que combina la capacidad de alinear los sistemas de información con los modelos del negocio con un coste menor de aplicación, que la hace viable para organizaciones de menor tamaño.

Tras analizar SOD-M en detalle, pudo comprobarse que no permitía modelar directamente aspectos de prueba, por lo que se han propuesto diversas extensiones para modelar los aspectos funcionales y no funcionales del comportamiento esperado del sistema. Las tareas a realizar por el sistema podrían incluir condiciones a cumplirse antes y después de su ejecución, y se podrían anotar con restricciones acerca de su tiempo de respuesta ante una determinada carga.

Estas condiciones y restricciones podrían servir de entrada para un abanico de técnicas de prueba, adaptando herramientas existentes. Se ha seleccionado una serie de técnicas candidatas a ser integradas en un futuro, como los grafos causa-efecto Paradkar et al. [1997] o la generación dinámica de invariantes Palomo Duarte et al. [2008].

Bibliografía

Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 1 edition, 2008. ISBN 0521880386. 3.25

Colin Atkinson, Philipp Bostan, Daniel Brenner, Giovanni Falcone, Matthias Gutheil, Oliver Hummel, Monika Juhasz, and Dietmar Stoll. Modeling components and Component-Based systems in KobrA. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 54–84. Springer Berlin, Heidelberg, Alemania, 2008. ISBN 978-3-540-85288-9. doi: 10.1007/978-3-540-85289-6_4. 3.2

L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of WSCoL. In *Proceedings of the IEEE International Conference on Web Services, 2007 (ICWS 2007)*, pages 663–670, 2007. doi: 10.1109/ICWS.2007.25. 3.22

Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: an overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, pages 49–69. Springer Berlin, 2005. 3.22

Lilian Burdy, Yoonsik Cheon, and David R. Cok. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3):212–232, June 2005. 3.22

Mike Clark. JUnitPerf, October 2009. URL http://clarkware.com/software/JUnitPerf.html. Fecha de última comprobación: 2 de noviembre de 2009. 3.20

María Valeria de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 3.4, 3.20, 3.22, 3.27

Juan José Domínguez Jiménez, Inmaculada Medina Bulo, and Antonia Botaro Estero. A framework for mutant genetic generation for WS-BPEL. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, Spindleruv Mlýn, República Checa, 2008. 3.27

Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. Addison-Wesley Professional, October 1998. ISBN 0201310120. 3.2

Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus Voß, and Johannes Willkomm. A method for engineering a true Service-Oriented Architecture. In José Cordeiro and Joaquim Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems*, pages 272–281, Barcelona, España, 2008. ISBN 978-989-8111-38-8. 3.1

Joakim Erdfelt, John Lewis, Lukasik Grzegorz, Jiri Mares, and Jeremy Thomerson. Cobertura, September 2009. URL http://cobertura.sourceforge.net/. Fecha de última comprobación: 2 de noviembre de 2009. 3.27

eviware.com. Página principal de soapUI, 2009. URL http://www.soapui.org/. Fecha de última comprobación: 2 de noviembre de 2009. 3.20

Antonio García Domínguez, Inmaculada Medina Bulo, and Mariano Marcos Bárcena. Hacia la integración de técnicas de pruebas en metodologías dirigidas por modelos para SOA. In *Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA*, Madrid, España, October 2009. 3.19

S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008. 3.2, 3.27

Jaap Gordijn and Hans Akkermans. Value-based requirements engineering: exploring innovative e-commerce ideas. *Requirements Engineering*, 8(2):114–134, July 2003. doi: 10.1007/s00766-003-0169-x. 3.4, 3.11

Jaap Gordijn and Hans Akkermans. e3value™ toolset, August 2006. URL http://www.e3value.com/tools/. Fecha de última comprobación: 2 de noviembre de 2009. 3.11

Ying Jiang, Shan-Shan Hou, Jin-Hui Shan, Lu Zhang, and Bing Xie. Contract-based mutation for testing components. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005 (ICSM'05)*, pages 483–492, 2005. ISBN 1063-6773. doi: 10.1109/ICSM.2005.36. 3.27

M. Lohmann, S. Sauer, and G. Engels. Executable visual contracts. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 63–70, 2005. doi: 10.1109/VLHCC.2005.35. 3.25

M. Lohmann, G. Engels, and S. Sauer. Model-driven monitoring: generating assertions from visual contracts. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006 (ASE '06)*, pages 355–356, 2006. ISBN 1527-1366. doi: 10.1109/ASE.2006.52. 3.25

Marc Lohmann, Leonardo Mariani, and Reiko Heckel. A model-driven approach to discovery, testing and monitoring of web services. In *Test and Analysis of Web Services*, pages 173–204. Springer Berlin, 2007. ISBN 978-3-540-72911-2. doi: 10.1007/978-3-540-72912-9_7. 3.25

Inmaculada Medina Bulo, Antonio García Domínguez, Francisco Aguayo, Lorenzo Sevilla, and Mariano Marcos Bárcena. Propuesta metodológica para la implementación de una arquitectura orientada a servicios en entornos de sistemas de fabricación distribuida. In *Actas del III Congreso Internacional de la Sociedad de Ingeniería de Fabricación*, pages 346–353, Alcoy, España, June 2009. 3.19

Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2 edition, 2004. ISBN 0471469122. 3.19, 3.25

OASIS. Web Service Business Process Execution Language (WS-BPEL) 2.0, April 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Fecha de última comprobación: 2 de noviembre de 2009. 3.20, 3.22

Object Management Group. MDA Guide version 1.0.1, June 2003. URL http://www.omg.org/mda/. Fecha de última comprobación: 2 de noviembre de 2009. 3.6

Object Management Group. Object Constraint Language Specification (OCL) 2.0, May 2006. URL http://www.omg.org/technology/documents/formal/ocl.htm. Fecha de última comprobación: 2 de noviembre de 2009. 3.22

Object Management Group. Business Process Modeling Notation (BPMN) 1.2, January 2009a. URL http://www.omg.org/spec/BPMN/1.2/. Fecha de última comprobación: 2 de noviembre de 2009. 3.1, 3.20

Object Management Group. Unified Modeling Language (UML) 2.2, February 2009b. URL http://www.omg.org/spec/UML/2.2/. Fecha de última comprobación: 2 de noviembre de 2009. 3.4, 3.6

Manuel Palomo Duarte, Antonio García Domínguez, and Inmaculada Medina Bulo. Takuan: a dynamic invariant generation system for WS-BPEL compositions. In *Proceedings of the 6th IEEE European Conference on Web Services*, Dublin, Irlanda, November 2008. 3.27

A. Paradkar, M. A. Vouk, and K. C. Tai. Specification-based testing using cause-effect graphs. *Annals of Software Engineering*, 4:133–157, 1997. 3.25, 3.27

Avik Sinha and Amit Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 17–22, Portland, Maine, 2006. ACM. ISBN 1-59593-458-8. doi: 10.1145/1145718.1145721. 3.25

Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005. 3.2, 3.5, 3.27

Juan Manuel Vara Mesa, Esperanza Marcos, and María Valeria de Castro. Obteniendo modelos de sistemas de información a partir de modelos de negocios de alto nivel: un enfoque dirigido por modelos. In *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA*, pages 15–28, Sevilla, España, October 2008. 3.4, 3.5

World Wide Web Consortium. WSDL 2.0 part 1: Core Language, June 2007. URL http://www.w3.org/TR/wsdl20. Fecha de última comprobación: 2 de noviembre de 2009. 3.19

H. Zhu, P. Hall, and J. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, December 1997. ISSN 0360-0300. 3.27

4

Extensión con restricciones de rendimiento de SOD-M

4.1. Introducción

En esta sección se describen las tecnologías utilizadas y el proceso a través del cual se ha obtenido una versión de la metodología SOD-M de Castro [2007] (desarrollada por el grupo Kybele de la Universidad Rey Juan Carlos) que extiende los modelos de nivel PIM para describir las restricciones de rendimiento a exigir. Primero se establecerían restricciones sobre las actividades de servicio de los procesos de servicio, y que se agregarían para establecer las restricciones de las actividades de servicio de las composiciones de servicio. Cada una de estas actividades de servicio se dividiría en varias acciones, que incluirían sus propias restricciones, derivadas de sus actividades de servicio contenedoras.

El contenido de esta sección se estructura como sigue: en primer lugar se describe el estado inicial de las herramientas, y las tecnologías en que se apoya. A continuación se describen las modificaciones introducidas sobre la metodología y las nuevas herramientas creadas, primero a nivel general y luego a un nivel más técnico. Finalmente, se realiza una valoración del estado actual de las herramientas, y se esbozan las líneas de trabajo futuro con vistas a su actualización a la última versión de SOD-M y su aplicación para asistir en la generación de casos de pruebas.

4.2. Estado inicial de las herramientas y tecnologías usadas en SOD-M

Tras una comunicación inicial con el grupo de investigación Kybele, se tuvo acceso a una instantánea de las herramientas que habían definido para SOD-M en su momento. Estas herramientas servirían como punto inicial para la creación de la versión extendida de la metodología, tal y como se describió en el capítulo 3.4.

4.2.1. Plataforma base de desarrollo: Eclipse

El conjunto de herramientas definidas por el grupo Kybele fue desarrollado no a partir de cero, sino dentro de la plataforma de desarrollo Eclipse Eclipse Foundation [2009g]. Eclipse se trata de un proyecto de código abierto iniciado por IBM, y define no solamente un entorno de desarrollo integrado para la creación de software en diversos lenguajes de programación, como C++ o Java, sino todo un marco en el que construir aplicaciones a partir de múltiples componentes integrados de forma dinámica y automática. Para ello, Eclipse 3.1.2 y posteriores versiones contienen una implementación de la especificación OSGi 4.0 OSGi Alliance [2009], conocida como Eclipse Equinox.

Esta especificación define los requisitos que deben cumplir tanto cada uno de los componentes individuales como el núcleo del sistema que se ocupa de integrarlos. Existen otras implementaciones de este mismo estándar, como por ejemplo en las plataformas de gestión del software Java instalado en diversos móviles, como los modelos del fabricante Nokia equipados con el sistema operativo Symbian.

Eclipse delega prácticamente toda su funcionalidad a componentes externos, dejando sólo en el núcleo de Eclipse lo estrictamente necesario para integrar estos componentes. Esto permite el uso de Eclipse para crear aplicaciones completas, conocidas como aplicaciones Rich Client Platform (RCP), y su extensión en prácticamente cualquier aspecto.

En la comunidad de desarrolladores de Eclipse, estos componentes externos integrables se conocen como *plugins*, que pueden depender de otros y ofrecer puntos de extensión sobre los que nuevos *plugins* añadan más funcionalidades. Para ello, todo *plugin* incluye una serie de ficheros de configuración detallando estos aspectos.

La versión de Eclipse en la instantánea de desarrollo proporcionada por el grupo Kybele es la 3.4, conocida como Eclipse Ganymede, publicada en junio del 2008. Para el desarrollo de este TFM se ha utilizado la última versión disponible en el momento de su comienzo, la versión 3.5, conocida como Eclipse Galileo, y publicada en junio del 2009.

4.2.2. Definición de metamodelos: EMF

Al desarrollar utilizando modelos, es necesario definir los metamodelos sobre los que se basan. Estos metamodelos de nivel 1 son a su vez modelos de un metamodelo de nivel 2. Podrían añadirse tantos niveles como se quisiera, pero el proceso no puede repetirse indefinidamente. Por esta razón, el Object Management Group (OMG) ha definido el estándar Meta-Object Facility (MOF) Object Management Group [2003], un metamodelo de nivel 2 útil para definir metamodelos de nivel 1. MOF es muy potente, ya que se diseñó para definir la versión 2.0 del estándar Unified Modeling Language (UML), de amplio uso en el mundo de la ingeniería del software, pero es difícil de implementar en su totalidad.

Debido a esta dificultad, el proyecto de código abierto Eclipse Modeling Framework (EMF) Eclipse Foundation [2009b] no implementa todo MOF, sino sólo un subconjunto, al que denomina Ecore.

Los metamodelos basados en Ecore son ficheros con la extensión `ecore` que siguen el estándar XMI Object Management Group [2007] para serialización de modelos, con lo que podrían crearse a mano en un editor de textos. Sin embargo, esto sería muy lento y propenso a errores, por lo que EMF provee un editor gráfico para Eclipse basado en árboles que resulta más cómodo de utilizar. A partir del fichero `ecore` se crea un modelo de generación de código con extensión `genmodel`, con el que es posible generar y personalizar los *plugins* necesarios para gestionar los modelos del metamodelo Ecore definido. Los modelos se guardan también en XMI, pero la extensión varía según el nombre del metamodelo.

Se pueden generar hasta cuatro *plugins* con código Java a partir del modelo `genmodel`. A continuación se listan sus nombres en la documentación inicial, sus traducciones y para qué sirve cada uno:

- *Model plugin* (*plugin* del modelo): tiene las clases necesarias para representar un modelo del metamodelo en memoria.
- *Edit plugin* (*plugin* de edición): añade el código necesario para que varios participantes modifiquen los modelos en memoria.
- *Editor plugin* (*plugin* del editor): define un editor gráfico de modelos basado en árboles, para facilitar el trabajo de los usuarios finales del metamodelo.
- *Tests plugin* (*plugin* de pruebas): añade código útil para realizar pruebas de unidad sobre las clases Java que representan al modelo.

El proyecto EMF se centra exclusivamente en definir los modelos y proveer la infraestructura base para editarlos de forma sencilla, guardarlos en formato XMI y hacerlos accesibles a través

de múltiples mecanismos, como bases de datos relacionales o redes de computadoras. El resto de tareas son implementadas en otros subproyectos de la iniciativa Eclipse.

4.2.3. Creación de notaciones gráficas para modelos: GMF

El editor gráfico basado en árboles generado por EMF supone una mejora considerable respecto a editar los ficheros XMI a mano, pero sigue siendo bastante incómodo para manejar modelos de un cierto nivel de complejidad. Por ello, en muchos casos interesa crear un editor que permita situar los elementos en el espacio y enlazarlos entre sí, usando una notación más cercana a la forma de pensar de las personas. Este editor gráfico avanzado dispone de muchas de las facilidades normalmente esperadas, como la posibilidad de obtener una vista ampliada o reducida, colocar automáticamente los elementos, alinearlos o personalizar su apariencia (colores, tipos de letra, etc.).

Para asistir en el desarrollo de estos editores gráficos más avanzados, el proyecto Graphical Modeling Framework (GMF) Eclipse Foundation [2009e] implementa una metodología dirigida por modelos a partir de la cual se genera el código del *plugin* correspondiente a este editor avanzado. La metodología conlleva a grandes rasgos los siguientes pasos:

1. Creación de un modelo con extensión `gmfgraph` de las formas que se utilizarán en los diagramas. Si se van a usar rombos, círculos, o formas de cualquier tipo, aquí es donde deben ser especificadas, junto con cualesquiera etiquetas de texto a emplear o restricciones a imponer, como tamaños mínimos, máximos, etc.

2. Creación de un modelo con extensión `gmftool` de las herramientas a situar en la paleta del editor gráfico. Estas herramientas opcionalmente incorporarán sus propios iconos pequeños y grandes, incluirán una descripción textual y se organizarán en grupos para gestionar el espacio en pantalla.

3. Creación de un modelo con extensión `gmfmap` que establezca las correspondencias entre los elementos del metamodelo descrito en el fichero `ecore` (el universo de discurso), las formas gráficas del modelo `gmfgraph` y las herramientas del modelo `gmftool`. Así, por ejemplo, si el metamodelo especificaba un tipo de elemento llamado *Actor*, se podría relacionar con la herramienta «Crear Actor» del grupo «Objetos» y representarse con la forma gráfica de una persona, con una etiqueta indicando su nombre debajo.

4. Creación a partir del modelo `gmfmap` del modelo de generación de código para el *plugin* del editor gráfico avanzado, con extensión `gmfgen`. Este modelo permite personalizar la forma en que se genera el código, pudiendo controlar las plantillas usadas para convertir los elementos del modelo `gmfgen` a código o los analizadores usados para procesar lo que introduzca el usuario en las etiquetas de texto, entre otras cosas.

5. Generación del código del editor gráfico y su posterior personalización manual, retirando los comentarios con el texto `@generated` que indican que un fragmento determinado de código ha sido generado automáticamente, para evitar que en una posterior regeneración las personalizaciones sean sobreescritas.

Como puede verse, el proceso conlleva una cantidad importante de trabajo manual, en el que se crean por primera vez y/o retocan los 4 modelos, y se retoca el código finalmente generado,

ya que en muchos casos las opciones de los modelos anteriores no son suficientes, tal y como ha ocurrido en este TFM.

GMF define una correspondencia automática del modelo `gmfmap` a la primera versión del modelo `gmfgen`, y dispone de plantillas prefabricadas para generar el código producido en los pasos 4 y 5. Estas plantillas están escritas en el lenguaje Xpand2 Eclipse Foundation [2009h], uno de los lenguajes actualmente utilizados para hacer transformaciones de modelo a texto («Model to Text» o M2T)„ junto con el más antiguo Java Emitter Templates (JET) y el más reciente Acceleo.

Las plantillas pueden ser personalizadas por el usuario empleando la funcionalidad conocida como plantillas dinámicas, permitiendo que el código generado incluya automáticamente los retoques deseados. Usando estas plantillas dinámicas, se pueden redefinir algunos aspectos de la generación de código completamente, o añadir código antes o después de varias partes de las plantillas ya existentes, siguiendo el enfoque de la programación orientada a aspectos (aspect-oriented programming o AOP) Kiczales et al. [1997]. Se han usado plantillas dinámicas en este trabajo para añadir diversas funcionalidades importantes, como la validación automática al guardar los modelos.

4.2.4. Estado inicial de las herramientas de SOD-M

La instantánea de desarrollo recibida de Kybele en marzo de 2009 consistía en una distribución de Eclipse Ganymede para Windows, que incluía los *plugins* creados con EMF y GMF para editar los modelos de casos de uso, de casos de uso extendido, de procesos de servicio y de composiciones de servicio antes mencionados.

En dicha distribución no se incluyeron las transformaciones automáticas entre estos modelos que se definieron en de Castro [2007] y cuyo método de implementación se describió en Vara Mesa et al. [2008]. Al parecer, la generación de los ficheros WSDL y del código final aún no estaba implementada. Tampoco se habían llegado a definir mecanismos de validación de los modelos.

Los *plugins* para los modelos de caso de uso y de casos de uso extendido no incluían el código fuente ni los modelos de GMF, por lo que resultaba imposible retocarlos sin implementarlos desde cero. Sin embargo, no era necesario extenderlos en este TFM, y funcionaban sin problemas (excepto algunos defectos gráficos) en Eclipse Galileo, por lo que pudieron usarse sin cambios.

Por otro lado, los *plugins* para los modelos de procesos de servicio y composiciones de servicio sí incluían los modelos de GMF, pero no fue posible migrarlos a la versión de GMF (2.2) incluida en Eclipse Galileo. Dada la naturaleza de este trabajo como paso previo de la tesis, era fundamental comenzar a trabajar con la versión más reciente de las herramientas, para evitar la dependencia en versiones antiguas con menos funcionalidad de la plataforma Eclipse, EMF y GMF. Por esta razón y por la previsión de la posible necesidad de introducir modificaciones importantes en los modelos de proceso y composición de servicio, se decidió reimplementar estos dos *plugins* desde cero.

4.3. Tecnologías adoptadas para la extensión de SOD-M

En esta sección se describen las nuevas tecnologías adoptadas durante el reemplazo de los *plugins* de los metamodelos de procesos de servicio y de composiciones de servicio incluidos

en la distribución recibida del grupo Kybele, tras haber evaluado su estado inicial y el de las tecnologías de partida en la sección anterior.

Se comienza esta sección con una valoración de la metodología habitual de desarrollo de metamodelos EMF/GMF, que sirve como marco para introducir a continuación una descripción de las nuevas tecnologías utilizadas y los motivos detrás de su adopción.

4.3.1. Problemas en la metodología de desarrollo habitual en EMF y GMF

Existen diversos problemas a la hora de desarrollar metamodelos con EMF y posteriormente producir editores gráficos avanzados para ellos con GMF.

El primer problema se debe al hecho de que hay que pasar por 6 modelos intermedios antes de producir el código del editor gráfico: la descripción del metamodelo Ecore en un fichero con extensión `ecore`, la generación de código para EMF (extensión `genmodel`), las formas gráficas (`gmfgraph`), las herramientas de la paleta (`gmftool`), las correspondencias entre los anteriores modelos (`gmfmap`) y la generación de código para GMF con un fichero (`gmfgen`).

En principio, sólo hay correspondencias automáticas del metamodelo Ecore al modelo `genmodel` y del `gmfmap` al `gmfgen`. El resto de modelos de GMF han de ser creados desde cero y aunque no son particularmente complejos, su elaboración constituye un proceso bastante laborioso. Esto supone un problema si el metamodelo Ecore de partida no es especialmente estable, como el caso de este trabajo, en que se identificaron varias veces cambios importantes a realizar en los metamodelos.

Si el metamodelo Ecore cambia, puede que tenga que cambiar todo lo que le sigue, y revisar los modelos de GMF ante cambios en el metamodelo Ecore es muy propenso a fallos. Algunos son detectados por la validación interna en EMF y GMF, pero otros no, y producen errores difíciles de depurar. Es crucial mantener la consistencia entre todos los modelos producidos.

El propio código generado por GMF (no tanto el de EMF) ha de ser normalmente retocado para atender a diversos aspectos que las plantillas prefabricadas de generación de código no atienden. Esto conlleva a que en un mismo fichero se mezcle el código escrito a mano con código generado, dificultando la comprensión de las modificaciones introducidas y obligando a ir moviendo manualmente dicho código según se cambien alguno de los modelos anteriores.

Esta situación de fragilidad se agrava debido al segundo problema: GMF se sigue desarrollando activamente, y se identifican nuevos requisitos de forma continua. La versión 2.2 introdujo diversos cambios que exigían una migración de los modelos definidos en GMF 2.1, pero los mecanismos de migración automática no eran del todo fiables, y las diferencias no se hallaban bien documentadas. Es de esperar que esta situación se repita en posteriores versiones de GMF. Si se va a iniciar una labor de investigación de cierta duración, es conveniente estudiar previamente cómo evitar esta fragilidad a la hora de desarrollar las nuevas versiones de los *plugins* para procesos y composiciones de servicios.

4.3.2. Paso a notación textual para definir metamodelos Ecore: Emfatic

El editor gráfico de Eclipse basado en árboles utilizado comúnmente para editar metamodelos Ecore cumple su cometido, pero resulta incómodo y lento de usar, dada su dependencia en el manejo a través del ratón y su insistencia en el uso de menús anidados y páginas de propiedades.

Otra opción es basarse en código Java existente, definiciones XML Schema o diagramas UML, pero no se dispone de los dos primeros y el tercero es igualmente incómodo para modelos relativamente grandes. Además, resulta difícil visualizar claramente las diferencias introducidas entre una versión y otra del metamodelo, debido al formato XMI de los ficheros *ecore*. En general, existen muchas más herramientas disponibles para operar sobre ficheros de texto que sobre documentos XMI.

Con vistas a mejorar esta situación, los plugins de Emfatic Scharf [2008] para Eclipse dan la posibilidad de escribir un metamodelo Ecore mediante un fichero de texto plano con la extensión `emf`, utilizando un lenguaje específico de dominio («Domain-Specific Language» o DSL) diseñado para ello. El lenguaje sigue una sintaxis familiar a la de lenguajes conocidos como Java, pero a su vez dispone de una serie de características para describir aspectos específicos a EMF, como las referencias inversas de una referencia («jefe» es la referencia inversa de «subordinados» en la clase *Empleado*, por ejemplo), o el número de valores asociados a un atributo o referencia de una clase (es decir, se suele tener un solo jefe y cero o más subordinados).

Otro aspecto particularmente interesante es la posibilidad de añadir anotaciones a cualquiera de las clases del metamodelo. Estas anotaciones pasarán a formar parte del metamodelo Ecore generado, y podrán ser aprovechadas por otras herramientas, como EuGENia, que será descrita en una sección posterior.

Se ha utilizado Emfatic para escribir los ficheros `emf` con las descripciones textuales de los metamodelos de proceso de servicio y los modelos de composición de servicio, y así derivar sus metamodelos Ecore anotados. El hecho de que los ficheros `emf` sean simplemente texto ha permitido que los dos tipos de modelos aprovechen una parte común de forma automática, evitando duplicidades innecesarias y propensas a errores.

4.3.3. Unificación de lenguajes de manejo de modelos: Epsilon

La extensión de SOD-M para atender a aspectos de rendimiento requiere no sólo crear editores amigables para los metamodelos, sino además llevar a cabo de forma automática diversas operaciones sobre los modelos:

- Los modelos deberían validarse para comprobar que pueden ser procesados posteriormente por los algoritmos ocupados de añadir las restricciones no especificadas manualmente. El usuario debería ser capaz de ver qué problemas han habido, si son graves o no, y cómo podría arreglarlos.
- Los algoritmos que estiman las restricciones no especificadas manualmente deberían poder lanzarse fácilmente sobre los modelos de proceso de servicio y de composiciones de servicio a través de la interfaz gráfica de Eclipse. Si algo no quedara claro, se debería preguntar al usuario.
- Tras conseguir el modelo de proceso de servicio con las anotaciones de rendimiento, el usuario debería poder obtener una primera aproximación del modelo de composición de servicio con dichas restricciones.

En un futuro, además, se necesitará generar código y otros documentos a partir de estos modelos, y es posible que se desee comparar modelos o unificar dos versiones de un mismo modelo.

Estas diversas tareas han recibido mucha atención entre los desarrolladores de metodologías dirigidas por modelos para Eclipse, dando lugar a la creación de un amplio abanico de lenguajes específicos de dominio para cada tarea, con sus herramientas asociadas. A continuación se listan algunos de los lenguajes más conocidos, clasificándolos según la tarea para la que fueron diseñados:

- Transformar un modelo en otro («Model to Model» o M2M): QVTO Object Management Group [2008] o ATL Eclipse Foundation [2009a] son los proyectos más estables dedicados a esta tarea.
- Validar un modelo: los modelos `gmfgen` permiten poner restricciones sobre los elementos a crear en el editor gráfico. Estas restricciones están escritas en el Object Constraint Language (OCL) Object Management Group [2006] del OMG.
- Generar texto a partir de un modelo («Model to Text» o M2T): los Java Emitter Templates (JET) representan uno de los primeros lenguajes usados, aunque muchos proyectos han migrado a Xpand2 Eclipse Foundation [2009h] (como GMF) y otros usan Acceleo.
- Generar un modelo a partir de texto («Text to Model» o T2M): Xtext Eclipse Foundation [2009j] permite definir una gramática a través de la cual construir instancias de metamodelos Ecore de forma textual.

Además, generalmente se utilizan programas Java para consultar los modelos, utilizando las facilidades ofrecidas por los *plugins* generados a partir de sus modelos `genmodel`. Como puede verse, la comunidad de desarrollo en torno a Eclipse es muy activa y prolífica, dando lugar a una gran variedad de enfoques para solucionar los distintos problemas planteados. Sin embargo, esta gran variedad dificulta su aprendizaje y el mantenimiento de estos sistemas, ya que cada uno tiene un diseño, estado de madurez y planificación distintos.

A diferencia de los proyectos anteriores, el proyecto Epsilon Kolovos et al. [2009] trata de definir no un solo lenguaje para una tarea específica, sino una familia completa de lenguajes para las diversas tareas necesarias a la hora de manipular modelos. Para ello, define una capa de compatibilidad llamada Epsilon Model Connectivity Layer (EMC Layer) para poder manipular modelos definidos en base a metamodelos creados con diversas tecnologías, como Ecore o XML Schema. Además, implementa un lenguaje base llamado Epsilon Object Language (EOL) que se inspira en OCL y lo extiende con asignaciones a variables, bucles, llamadas a función y demás construcciones usuales del paradigma imperativo de programación. Este lenguaje base puede usarse por sí mismo para consultar y modificar modelos siguiendo un paradigma imperativo, en el estilo de los lenguajes de programación más conocidos, como C++ o Java, aunque a un mayor nivel de abstracción. EOL constituye la raíz de una jerarquía completa de lenguajes específicos de dominio para realizar diversas tareas:

- Epsilon Comparison Language (ECL): comparación de modelos.
- Epsilon Generation Language (EGL): transformaciones M2T.
- Epsilon Merging Language (EML): fusión de modelos.
- Epsilon Transformation Language (ETL): transformaciones M2M de un modelo a otro modelo separado.

- Epsilon Validation Language (EVL): validación de modelos, situando marcadores de aviso y error en los sitios apropiados y ofreciendo al usuario mecanismos automatizados para corregir sus causas cuando sea posible.
- Epsilon Wizard Language (EWL): transformaciones M2M sobre un mismo modelo, fácilmente accesibles desde un menú, y sensibles a contexto.
- Flock: migración de modelos entre distintas versiones de un metamodelo.
- Human-Usable Textual Notation (HUTN): implementa parcialmente la especificación HUTN del OMG Object Management Group [2004] para escribir modelos usando una notación textual. A diferencia de Xtext, no hay que especificar ninguna gramática, ya que es generada automáticamente a partir del metamodelo. Se distingue de Emfatic en que describe modelos de un metamodelo ya existente, y no un nuevo metamodelo.

Como puede verse, Epsilon proporciona una buena base para definir nuevos lenguajes específicos para diversas tareas, e incluye de serie varios lenguajes cuyas funcionalidades son comparables o superan a las de los anteriormente mencionados, como en el caso de ETL frente a ATL, o de EVL frente a OCL. En particular, en este proyecto se han empleado EVL, EWL, ETL y EOL, y en un futuro se considera utilizar EML y ECL. Además, Epsilon dispone de documentación de gran calidad, con un libro electrónico disponible en PDF que documenta la mayor parte de las tecnologías Epsilon y abundantes ejemplos y vídeos demostrativos (*screencasts*). Otra ventaja de contar con un lenguaje base común es la posibilidad de aprovechar parte del mismo código para las transformaciones y validaciones, por ejemplo.

Por otro lado, Epsilon se encuentra en un estado menos maduro de desarrollo que las alternativas anteriores. Una parte considerable del tiempo dedicado al desarrollo de este TFM se dedicó a la localización y colaboración en la corrección de defectos. En particular, se participó en la elaboración de los siguientes informes de fallos, y en varios casos se propusieron soluciones iniciales.

- https://bugs.eclipse.org/bugs/show_bug.cgi?id=285912 (arreglado en Epsilon 0.8.7)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286017 (arreglado en 0.8.7)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286126 (arreglado en 0.8.7)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286215 (parche propuesto)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286385 (se evita dicho fallo de forma temporal)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286412 (se evita)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286549 (arreglado en 0.8.7)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286849 (arreglado en 0.8.7)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=286992 (arreglado en 0.8.7)

- https://bugs.eclipse.org/bugs/show_bug.cgi?id=287542 (molestia menor, pendiente de escalar a un proyecto de nivel superior)
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=287756 (arreglada en 0.8.8)

Actualmente, las herramientas creadas en este trabajo funcionan sin problemas a partir de la versión 0.8.7 oficial de Epsilon.

4.3.4. Generación automática de los modelos de GMF: EuGENia

Anteriormente se mencionó el hecho de que la creación de los modelos GMF era a partir de cero, y que cambiar el metamodelo Ecore inicial implicaba cambiar los modelos GMF, siendo un proceso propenso a fallos.

EuGENia Eclipse Foundation [2009c] se trata de una herramienta creada en el proyecto Epsilon que trata de paliar este problema. Para ello, procesa a través de una serie de guiones EOL un metamodelo Ecore debidamente anotado (escrito con Emfatic o con el editor basado en árboles incluido en EMF, por ejemplo) para producir una primera aproximación de los modelos `gmfgraph`, `gmftool` y `gmfmap`, y retocar el modelo `gmfgen` finalmente producido.

Las anotaciones disponibles se hallan documentadas en Eclipse Foundation [2009d]. Existen anotaciones para definir nodos, enlaces, compartimentos, diagramas y nodos adjuntos (fijados al borde de otro nodo), y para establecer qué características deberían tener.

Listado 4.1 Descripción en Emfatic con anotaciones de la clase *InitialNode*

```
@gmf.node(figure="es.uca.modeling.figures.FgFilledCircleFigure",
border.color="0,0,0", size="30,30", tool.name="Start")

abstract class InitialNode extends FlowNode {}
```

Sirva como ejemplo el fragmento Emfatic del listado 4.1, que describe a la clase abstracta *InitialNode* (nodo inicial) de la parte común a los metamodelos de proceso de servicio y composición de servicio. Extiende a la clase *FlowNode*, a la que pertenecen todos los nodos que forman parte del flujo. La anotación `gmf.node` indica que debe ser representado como un nodo del grafo. El atributo `figure` indica que se usará un círculo rellenado con el color de primer plano, que según `border.color` será negro por omisión, y medirá 30 píxeles de ancho y alto inicialmente, gracias a `size`. Las herramientas que aparecerán en la paleta para crear cada uno de los elementos de las subclases de *InitialNode* se llamarán en principio «Start», tal y como dice `tool.name`, a menos que dichas subclases cambien el valor de `tool.name` por su cuenta.

Además, EuGENia es capaz de ejecutar automáticamente los guiones EOL que defina el usuario para realizar los retoques sobre los modelos GMF que normalmente se harían manualmente, si existen. De esta forma, no hay que llevar el control de los modelos GMF, sino sólo del código fuente Emfatic y los guiones EOL con los retoques sobre lo que genere EuGENia. Así se deja de depender de una versión determinada de GMF y pueden reutilizarse dichos retoques desde varios sitios. Para ello, pueden opcionalmente situarse en el mismo directorio que el metamodelo Ecore anotado de partida los siguientes ficheros:

FixGenModel.eol retoca el modelo `genmodel`.

ECore2GMF.eol retoca los modelos `gmfgraph`, `gmftool` y `gmfmap`.

FixGMFGen.eol retoca el modelo `gmfgen`.

4.3.5. Automatización del flujo de trabajo: Apache Ant

Aunque el proceso de desarrollo de los editores gráficos avanzados para los metamodelos extendidos había mejorado notablemente con la adopción de Emfatic, Epsilon y EuGENia, la generación del código final con ellos requería de la ejecución manual de los siguientes pasos, en este mismo orden:

1. Generar el metamodelo Ecore a partir de la descripción Emfatic.
2. Incluir el metamodelo Ecore en el registro de *EPackage*s de EMF, para su acceso posterior usando únicamente su dirección URL identificativa, sin indicar a cada momento dónde está.
3. Generar el modelo de generación de código `genmodel` a partir del metamodelo Ecore.
4. Retocar el anterior modelo con los guiones `FixGenModel.eol` de EuGENia y del usuario (si existe).
5. Producir el código para los *plugins* de manipulación de modelos, de edición y del editor basado en árboles.
6. Generar los modelos `gmfgraph`, `gmftool` y `gmfmap` de GMF usando EuGENia, y retocarlos con los guiones.
7. Retocar los anteriores modelos con los guiones `ECore2GMF.eol` de EuGENia y del usuario (si existe).
8. Generar el modelo `gmfgen` a partir del modelo `gmfmap`, usando la correspondencia definida en el propio GMF. Parte de la correspondencia ha de ser modificada redefiniendo y envolviendo las plantillas Xpand2 de GMF.
9. Retocar el modelo `gmfgen` con los guiones `FixGMFGen.eol` EOL de EuGENia y del usuario (si existe).
10. Producir el código para el *plugin* del editor gráfico avanzado a partir del modelo `gmfgen`, volviendo a redefinir parte de las plantillas Xpand2 para que produzcan el código deseado.

Como puede imaginarse, la ejecución manual de estos 10 pasos cada vez que se retocaba la descripción inicial en Emfatic suponía un proceso repetitivo y propenso a errores. Un error exigía volver a comenzar el proceso desde el principio. Además, si el cambio había sido lo suficientemente grande, normalmente se hacía necesario eliminar los productos de la anterior ejecución para que todo funcionase, cuidando de no dejar alguno inadvertidamente.

Tras un tiempo, se consideró necesaria la automatización de estos pasos en un flujo de trabajo integrado con el entorno Eclipse. Existían de antemano lenguajes para escribir flujos de trabajo con modelos, como es el del Modeling Workflow Engine (MWE), parte del proyecto Eclipse Modeling Framework Technology (EMFT) Eclipse Foundation [2009i], pero requerían de componentes que no existían para Epsilon, y que habrían tenido que ser creados desde cero.

Sin embargo, Epsilon sí que disponía de herramientas para automatizar la ejecución de guiones escritos en algunos de sus lenguajes, en forma de «tareas» para Apache Ant.

Apache Ant Apache Foundation [2008] es un sistema de construcción automática de programas ampliamente conocido en el mundo Java, y muy maduro y estable. Es usado en un gran número de proyectos de todo tipo y envergadura. Permite definir una serie de «objetivos» que dependen unos de otros, y que requieren la ejecución de determinadas acciones o «tareas» para su consecución. Los usuarios pueden integrar fácilmente nuevas tareas dentro del ya amplio conjunto definido por Apache Ant. Además, las instrucciones de construcción (conocidas como «guión Ant») de un elemento determinado pueden reutilizar las instrucciones de otro, especificando solamente los aspectos que sean distintos.

Eclipse utiliza internamente Ant para ciertas tareas, y puede lanzar un guión determinado de forma automática cuando se producen cambios en ciertos elementos: sólo hay que añadir un constructor Ant («Ant Builder» en inglés) a las opciones del proyecto, y configurarlo debidamente.

Existían tareas Ant para automatizar muchas de las acciones antes realizadas manualmente a través de la interfaz de Eclipse. Había ciertos defectos en las tareas definidas para generar los *plugins* de manipulación, edición y el editor gráfico basado en árboles a partir del modelo `genmodel`, pero fueron rápidamente resueltos tras su notificación García Domínguez [2009].

En el caso de las tareas de Epsilon, se identificó la necesidad de ejecutar guiones EOL no accesibles directamente a través del sistema de ficheros, sino incluidos en *plugins* instalados en Eclipse, y se asistió en la implementación de esta funcionalidad y su integración en la versión oficial 0.8.7.

Por otro lado, hubo que implementar una tarea Ant para generar el metamodelo Ecore a partir de su descripción textual Emfatic.

Finalmente, se integró la tarea Ant desarrollada por Ecliptical para la generación del código del editor gráfico avanzado basado en GMF. En el futuro la tarea de Ecliptical se reemplazará por la tarea oficial de GMF dedicada al mismo propósito, que aparecerá Eclipse Foundation [2009f] en la próxima versión 2.3, aún no publicada.

El guión Ant definido incluye también un objetivo para realizar la limpieza de los productos del anterior proceso de generación de forma sistemática y fácilmente repetible, evitando que al usuario se le olvide limpiar cualquier detalle.

Un último aspecto interesante es que se ha utilizado la capacidad de Ant de copiar un fichero de texto sustituyendo partes de él por otras cadenas de texto para hacer algo que no podía hacerse directamente con un metamodelo normal en Ecore, aprovechando el uso de una notación basada en texto como la de Emfatic: los metamodelos de proceso de servicio y de composición de servicio incluyen un mismo núcleo común, sin llegar a referenciar uno al otro, puesto que las definiciones Emfatic comunes han sido incluidas de un fichero común. El guión Ant se ocupa de asegurar de forma transparente que ambos siempre usen la última versión del núcleo común.

4.4. Creación de las herramientas mejoradas para SOD-M

En las dos anteriores secciones se han descrito las diversas tecnologías utilizadas para implementar las extensiones de modelado de rendimiento sobre SOD-M. En particular, en la

sección §4.3 se describieron varias tecnologías adoptadas para paliar los problemas inicialmente identificados de fragilidad y falta de flexibilidad de la metodología de desarrollo de editores basados en EMF y GMF.

En esta sección se hablará de la implementación de los reemplazos de los antiguos *plugins* para los metamodelos de proceso de servicio y de composición de servicio, junto con otros cambios introducidos sobre la propia metodología SOD-M. Se expondrán los metamodelos extendidos, su notación gráfica, los mecanismos de validación introducidos, los algoritmos de estimación de restricciones de rendimiento implementados y, por último, la transformación automática de los modelos de proceso de servicio a los modelos de composición de servicio.

4.4.1. Cambios en los metamodelos

Los modelos de procesos de servicio y composición de servicio de SOD-M (véanse §3.3.3 y §3.3.3) se basan en los diagramas de actividad de UML descritos en §3.3.1.

Entre otros cambios, las antiguas actividades son reemplazadas por actividades de servicio en los procesos de servicio y por acciones en las composiciones de servicio. Las pistas desaparecen en los procesos de servicio y pasan a conocerse como «colaboradores del negocio» en las composiciones de servicio.

Por lo demás, no se llegaron a hacer grandes cambios respecto a lo incluido en el estándar UML. Sin embargo, en la versión de este TFM se han introducido muchos cambios, motivados en su mayor parte por aspectos de implementación de las herramientas.

Separación de un núcleo común a ambos metamodelos El primer cambio en los metamodelos ha sido dividirlos en un núcleo común a ambos metamodelos, y una parte específica de cada metamodelo que reutiliza este núcleo común. El uso de un núcleo común permite aprovechar los mismos algoritmos de estimación para ambos metamodelos, de forma que cada metamodelo sólo tenga que especificar las partes que varíen.

El núcleo común se halla dedicado a modelar una variante sobre los diagramas UML de actividad estándar (véase §3.3.1), en la que pueden especificarse condiciones de activación de un flujo de control con estimaciones de probabilidad, y se pueden establecer restricciones sobre el tiempo límite de respuesta ante el acceso concurrente de un determinado número de usuarios. En este núcleo común también se definen una serie de tipos de datos para expresar el lenguaje en que están escritas las condiciones, junto otros tipos útiles a nivel de implementación, y se revisa la semántica de algunos elementos. Puede verse un diagrama de clases UML del metamodelo en la figura 4.1.

A grandes rasgos, los diagramas UML de actividad modificados utilizados en este trabajo consisten en grafos cuya ejecución comienza por un *InitialNode*, y se propaga a otros *FlowNode* (nodos del flujo de ejecución) a través de los *FlowEdge* (aristas del flujo de ejecución). Se evitó utilizar los nombres *ActivityNode* y *ActivityEdge* propios del metamodelo original UML para evitar confusiones con las actividades de servicio (*ServiceActivity*).

Hay dos tipos de *FlowEdge*: aquellos que suponen el paso de un objeto entre dos nodos (*ObjectFlow*), y aquellos que únicamente indican el nodo a iniciar tras terminar la ejecución de la actual (*ControlFlow*). La mayoría de los *FlowNode* realmente no llevan a cabo ninguna funcionalidad, sino que controlan la forma en que se propagan los mensajes o se pasa de una tarea a otra. Los distintos tipos de elementos, la notación usada y su significado en esta variante de los diagramas UML de actividad se listan en la tabla 4.1.

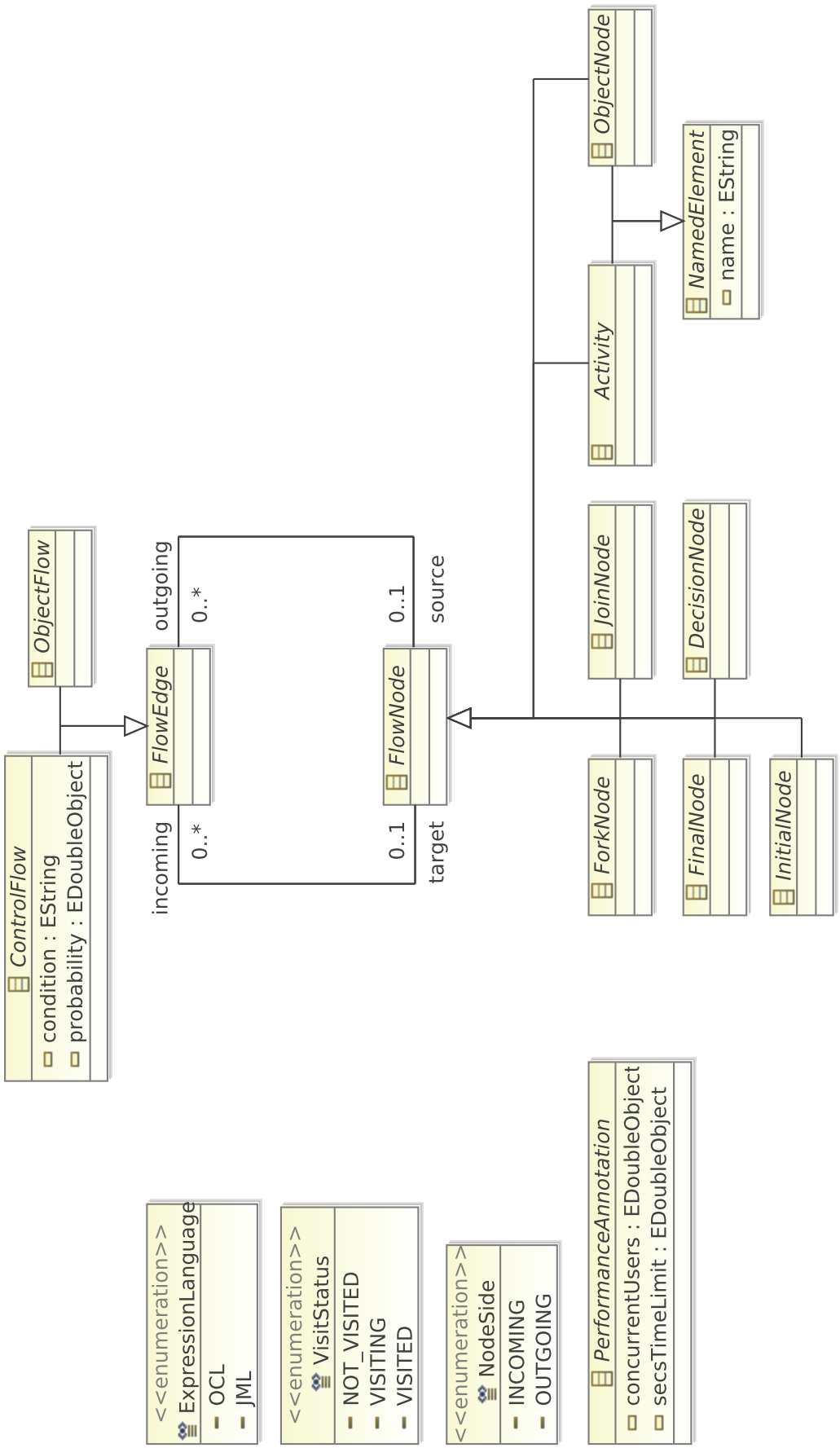


Figura 4.1. Diagrama de clases UML con el metamodelo común para los diagramas de actividad UML modificados

Como puede verse, se ha mantenido la mayor parte de los elementos y significados originales de los diagramas de actividad UML, salvo por dos aspectos, encaminados a simplificar la notación:

- Los *JoinNode* incluyen tanto a los nodos de reunión originales de UML como a sus nodos de fusión, en un esfuerzo por simplificar la notación.

  Queda pendiente evaluar si supone una ventaja, y si debería de aplicarse el mismo criterio sobre los *DecisionNode* y *ForkNode*, o si por lo contrario debería volverse a la división original de roles.

- Los *FinalNode* incluyen tanto a los nodos de final de actividad como a los nodos de final de flujo de UML, en otro intento de simplificar la notación. Es la propia estructura del grafo la que indica si sólo se termina la rama, o si termina toda la actividad.

  Al llegar a un *FinalNode* en principio sólo acaba la ejecución de la rama actual. Si era la única rama, entonces efectivamente ha terminado la ejecución de la actividad completa.

Metamodelo de procesos de servicio Una vez queda definida la parte común de ambos metamodelos, resulta mucho más sencillo describir los metamodelos de proceso y composición de servicio.

Los elementos del metamodelo de procesos de servicio se hallan resumidos en la tabla 4.2, y la figura 4.2 contiene su diagrama de clases UML simplificado. Este metamodelo añade un elemento raíz que contiene a todas las instancias del resto de clases, llamado *ServiceProcess*. Este elemento contiene, además de los arcos y nodos del grafo (especializaciones de los elementos de la parte común), especializaciones de *PerformanceAnnotation* para modelar las restricciones de rendimiento de los procesos de servicio completos (*ProcessPerformanceAnnotation*) y de actividades de servicio individuales (*ActivityPerformanceAnnotation*).

En particular, se distingue entre las restricciones de actividad de servicio creadas manualmente y aquellas creadas de forma automática, con el objetivo de guiar a los algoritmos de estimación de las restricciones no especificadas sobre qué restricciones pueden ser libremente revisadas (al haberse generado en una ejecución anterior) y qué restricciones sólo pueden ser revisadas con consentimiento explícito del usuario (al haberse especificado manualmente).

Finalmente, este metamodelo no considera los flujos de objetos, ya que en él no llegan a modelarse todavía los distintos participantes, por lo que no hay necesidad de especificar qué mensajes u objetos se intercambian entre ellos aún.

Metamodelo para composiciones de servicios El metamodelo para composiciones de servicio es similar al anterior. La tabla 4.3 resume sus elementos, y el diagrama de clases UML simplificado correspondiente se halla en la figura 4.3. Parte de una clase principal *ServiceComposition* que contiene a todos los demás elementos, pero en este caso se usan dos jerarquías de flujos y de nodos, basadas en los elementos tanto del metamodelo común como del metamodelo de proceso de servicio anterior.

La razón detrás de esta aparente duplicidad es la necesidad de modelar explícitamente el hecho de que una actividad de servicio se compone de varias acciones, para así en un futuro poder realizar la transformación inversa de composiciones de servicio a procesos de servicio y agilizar el trabajo del modelador. Esto implicaba que las actividades de servicio en este metamodelo (*ServiceActivity*) no serían simples elipses con etiquetas de texto en su interior,

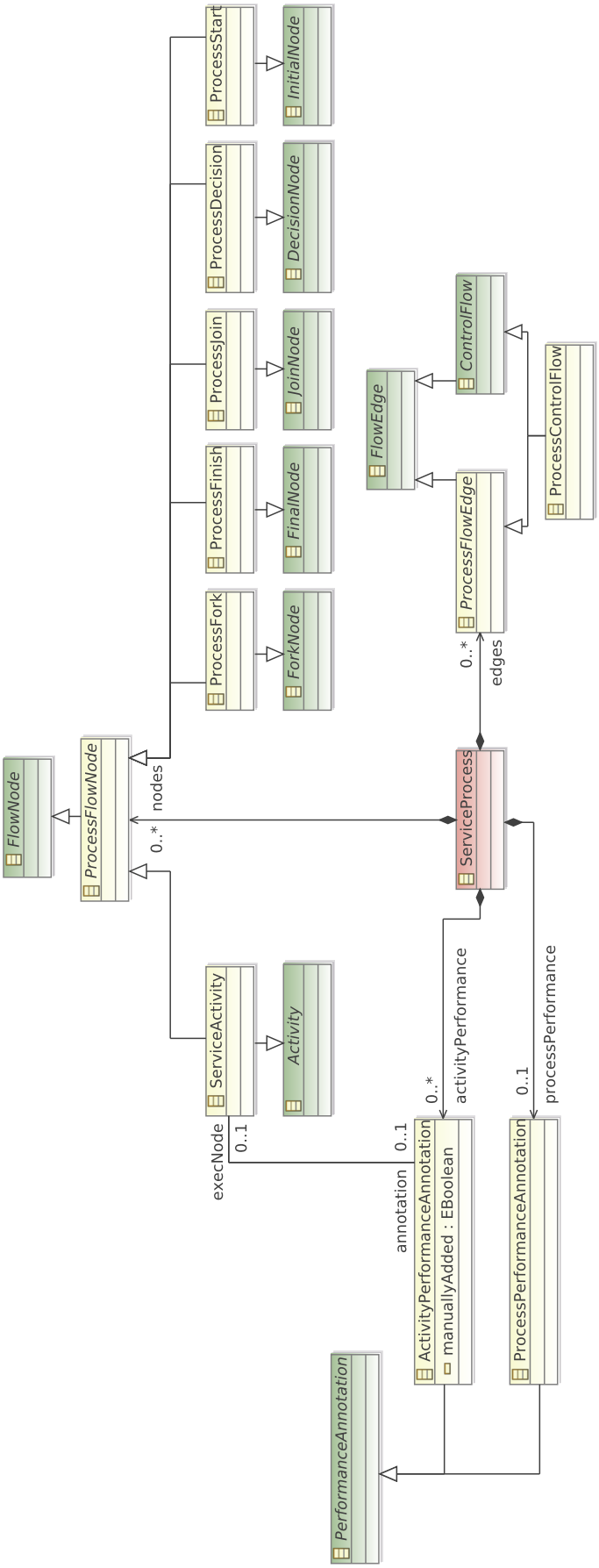


Figura 4.2. Diagrama simplificado de clases UML para el metamodelo de procesos de servicio extendido

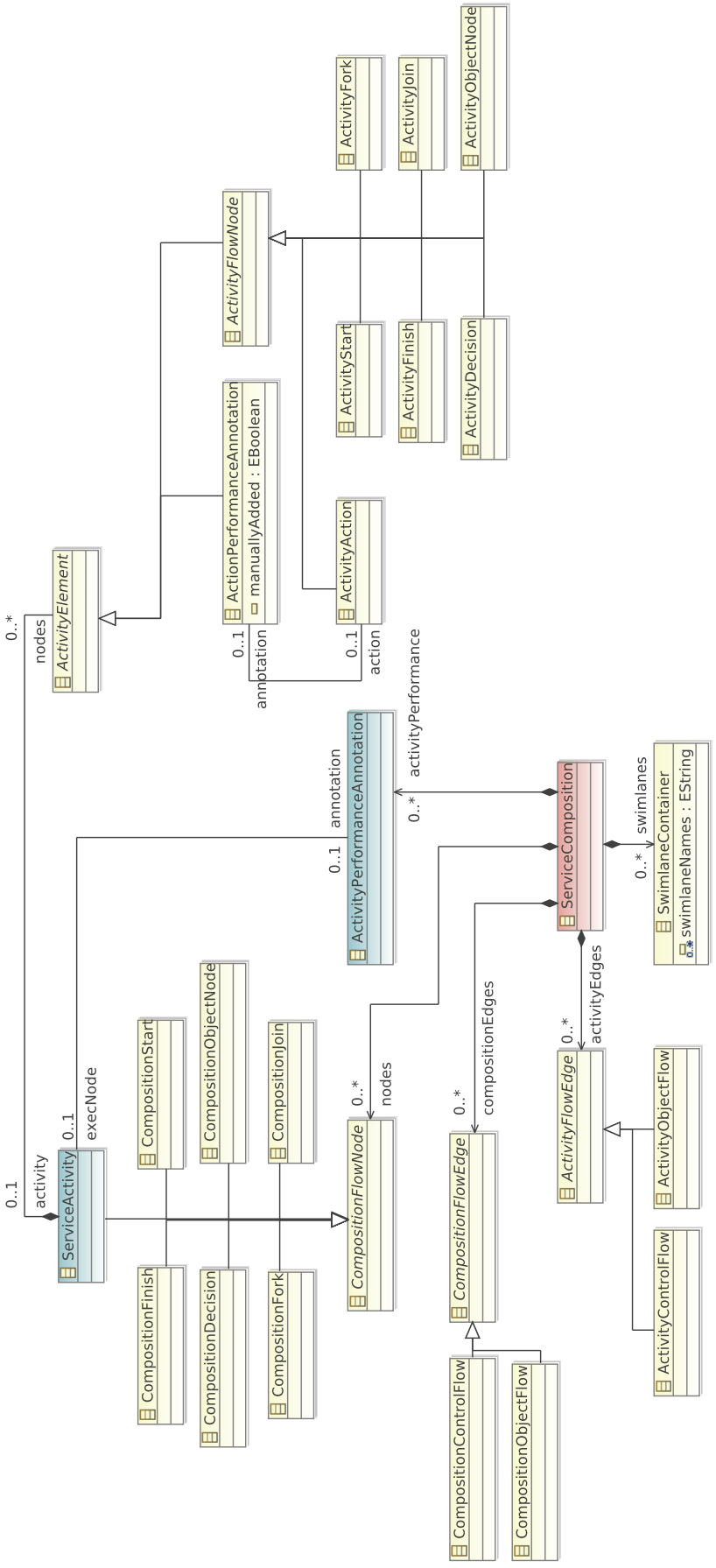


Figura 4.3. Diagrama simplificado de clases UML para el metamodelo de composiciones de servicio extendido

como pasaba en el metamodelo de procesos de servicio, sino que contendrían grafos anidados con diagramas de actividad UML. En resumen, los elementos dedicados al grafo externo usan el prefijo «Composition» en sus nombres (siendo hijos de una *ServiceComposition*), mientras que los elementos dedicados a los grafos anidados en cada *ServiceActivity* usan el prefijo «Activity» (al ser hijos de una *ServiceActivity*).

Otra diferencia en el metamodelo de composición de servicio es que las restricciones de rendimiento para las actividades de servicio (*ActivityPerformanceAnnotation*) ya se consideran siempre manuales. En las composiciones de servicio los algoritmos de estimación de restricciones no especificadas operan no sobre el grafo de nivel superior, sino sobre cada uno de los grafos anidados de acciones contenidos en las diversas actividades de servicio. Así, el atributo `manuallyAdded` ha pasado a las *ActionPerformanceAnnotation*, que indican las restricciones de rendimiento de una determinada acción dentro de una actividad de servicio.

Al examinar el modelo con cuidado, puede verse que los *ActivityFlowEdge* no están contenidos directamente en sus actividades de servicio, sino a nivel global en la composición de servicio. Esto se debe a un problema actual en la implementación de EuGENia, que no maneja correctamente los enlaces contenidos dentro de un compartimento.

Los «Colaboradores del Negocio» de los modelos de composición de servicio de SOD-M han sido reemplazados por objetos de la clase *Swimlanes*. Un objeto *Swimlanes* reúne a los distintos participantes de una composición de servicio en un solo elemento, en el que se dibujan sus respectivas pistas. La idea detrás de este cambio era simplificar el trabajo de los modeladores al reducir el número de elementos a gestionar. Sin embargo, en la versión actual los *Swimlanes* no llegan a contener a las actividades de servicio ni a ningún otro elemento del grafo, usándose solamente a nivel de documentación. La razón de esta limitación es la complejidad que supone implementar a nivel de GMF que las diversas acciones del grafo anidado de una actividad de servicio puedan asignarse a distintas pistas. Dadas las limitaciones de tiempo, se ha pospuesto esta mejora a versiones posteriores, dado que no era necesaria para este trabajo.

Tabla 4.1. Resumen de elementos en el metamodelo común

Nombre	Notación	Significado
InitialNode		Se corresponde con un nodo inicial de un diagrama de actividad UML.
FinalNode		Se corresponde con un nodo final de un diagrama de actividad UML.
ForkNode		Se corresponde con un nodo de ramificación de un diagrama de actividad UML.
DecisionNode		Se corresponde con un nodo de decisión de un diagrama de actividad UML.

Continúa en la siguiente página

Continuado desde la página anterior

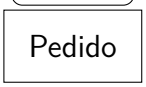
Nombre	Notación	Significado
JoinNode		Reúne varias ramas de ejecución que se dividieron a partir de un mismo nodo en una sola. Si las ramas se dividieron en un *DecisionNode* sólo espera a aquella por la que se prosiguió, y si se dividieron en un *ForkNode* espera a todas. Supone una simplificación respecto a los diagramas de actividad UML originales, en que se usan los nodos de reunión sólo para el primer papel, y los nodos de fusión para el segundo papel.
Activity		Se corresponde con una actividad de un diagrama de actividad UML.
ObjectNode		Se corresponde con un nodo objeto de un diagrama de actividad UML.
ControlFlow		Se corresponde con un flujo de control de un diagrama de actividad UML.
ObjectFlow		Se corresponde con un flujo de objeto de un diagrama de actividad UML.

Tabla 4.2. Resumen de elementos en el metamodelo de proceso de servicio

Nombre	Notación	Significado
ProcessStart	Véase *InitialNode* en el metamodelo común.	
ProcessFinish	Véase *FinalNode* en el metamodelo común.	
ProcessFork	Véase *ForkNode* en el metamodelo común.	
ProcessDecision	Véase *DecisionNode* en el metamodelo común.	
ProcessJoin	Véase *JoinNode* en el metamodelo común.	
ServiceActivity	Véase *Activity* en el metamodelo común.	
ProcessControl-Flow	Véase *ControlFlow* en el metamodelo común.	

Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
Process-Performance-Annotation	<<sppc>> concurrentUsers = 1 timeLimit = 120	Indica que las tareas de todos los caminos que parten del nodo inicial del grafo deberían completar su ejecución bajo el tiempo límite especificado (aquí 120 segundos), suponiendo que se da servicio al número de usuarios especificado (aquí 1).
Activity-Performance-Annotation	<<sapc>> concurrentUsers = 1 timeLimit = 18 manual = false	Indica que esta tarea particular debería tardar no más de un tiempo determinado (aquí 18 segundos), atendiendo a un cierto número de usuarios que acceden concurrentemente al sistema (aquí 1). Es posible que la restricción no haya sido añadida manualmente, sino por uno de los algoritmos de estimación automática de restricciones no especificadas, como ocurre en este caso.

Tabla 4.3. Resumen de elementos en el metamodelo de composición de servicio

Nombre	Notación	Significado
CompositionStart	*InitialNode* del grafo de la composición completa.	
CompositionFinish	*FinalNode* del grafo de la composición completa.	
CompositionFork	*ForkNode* del grafo de la composición completa.	
Composition-Decision	*DecisionNode* del grafo de la composición completa.	
CompositionJoin	*JoinNode* del grafo de la composición completa.	
ServiceActivity		*Activity* del grafo de la composición completa. Contiene un grafo anidado formado por una serie de elementos *ActivityFlowNode*.
Composition-ControlFlow	*ControlFlow* del grafo de la composición completa.	
Composition-ObjectFlow	*ObjectFlow* del grafo de la composición completa.	
ActivityStart	*InitialNode* del grafo anidado de una actividad de servicio.	
ActivityFinish	*FinalNode* del grafo anidado de una actividad de servicio.	

Continúa en la siguiente página

Continuado desde la página anterior

Nombre	Notación	Significado
ActivityFork	*ForkNode* del grafo anidado de una actividad de servicio.	
ActivityDecision	*DecisionNode* del grafo anidado de una actividad de servicio.	
ActivityJoin	*JoinNode* del grafo anidado de una actividad de servicio.	
ActivityAction	*Activity* del grafo anidado de una actividad de servicio.	
ActivityControl-Flow	*ControlFlow* del grafo anidado de una actividad de servicio.	
ActivityObject-Flow	*ObjectFlow* del grafo anidado de una actividad de servicio.	
Activity-Performance-Annotation	<<sapc>> concurrentUsers = 1 timeLimit = 5	Indica que las tareas accesibles desde todos los caminos que parten del *ActivityStart* contenido en el grafo anidado de la *ServiceActivity* correspondiente deben terminar su ejecución dentro del límite de tiempo establecido (aquí 5 segundos), suponiendo que un determinado número de usuarios accede a la vez (aquí 1).
Action-Performance-Annotation	<<apc>> concurrentUsers = 1 timeLimit = 5,75 manual = false	Indica que esta acción, parte de una actividad de servicio determinada, completará su ejecución en menos de un cierto tiempo (aquí 5,75 segundos), suponiendo que acceden a ella un determinado número de usuarios concurrentemente (aquí 1). La restricción sobre la acción puede haberse añadido de forma manual, o haberse estimado de forma automática (como en este caso).

4.4.2. Validación de los metamodelos

Tras definir los elementos de los metamodelos extendidos para procesos y composiciones de servicio, se hizo necesario definir las restricciones a cumplir por estos modelos. De esta forma se podrían diseñar los algoritmos de estimación de restricciones de rendimiento bajo la suposición de que se cumplieran dichas restricciones, simplificando su implementación.

Las restricciones se organizaron aprovechando la división en parte común y parte específica, usando la parte específica sólo cuando fuera estrictamente necesario. Estas restricciones se hallan implementadas como guiones EVL, y se comprueban de forma automática cuando el usuario guarda un modelo a disco, gracias a la definición de los puntos de extensión apropiados en el descriptor del *plugin* Eclipse que los contiene, y a la redefinición de partes de la plantilla Xpand2 `xpt/editor/DocumentProvider.xpt` para que genere el código apropiado, usando

el mecanismo de plantillas dinámicas visto anteriormente.

Las restricciones en el guión EVL `model/Common.evl` del *plugin es.uca.modeling.eol*, a cumplir por los elementos de la parte común de cada metamodelo y sus elementos derivados en la parte específica, son los de la tabla 4.4. En general, se dice que una restricción en EVL se cumple cuando es aplicable (es decir, se cumple la guarda, que se describe textualmente empezando con «A menos que» en dicha tabla) y cuando se cumple la propiedad que describe. Algunas restricciones sólo son aplicables si otras restricciones se cumplen en primer lugar, para simplificar su definición, reducir el número de errores y avisos presentados al usuario y evitar que el guión EVL no acabe nunca, al introducirse en un bucle infinito.

En algunos casos se exige que ciertas cantidades estén en intervalos aproximados y no que sean equivalentes a ciertos números, como podría esperarse. Esto se debe a la naturaleza aproximada de la representación de números reales utilizando números de coma flotante IEEE-754 Goldberg [1991].

En la actualidad, se impone la restricción de que no existen ciclos en los grafos utilizados, dado que entonces se tendrían en principio infinitos caminos posibles por el grafo. Esta restricción puede parecer muy fuerte para un modelo de un programa, en el que los bucles normalmente juegan un rol muy importante, pero puede rodearse tomando el cuerpo del bucle completo como una sola actividad de servicio o acción. Queda pendiente como trabajo futuro analizar la posibilidad de imponer un umbral de probabilidad sobre el número de iteraciones de un ciclo (por ejemplo, suponiendo que en el 90 % de los casos basta con hacer 20 iteraciones como mucho), de forma que pueda limitarse el número de caminos a un número finito.

Alguna de las restricciones impuestas al inicio de este TFM fueron retiradas a lo largo de su ejecución. Tal es el caso de la restricción de que hubiera un solo nodo final, que fue retirada tras revisar el estándar UML Object Management Group [2009].

Los conjuntos de restricciones específicos a los procesos de servicio y composiciones de servicio, respectivamente, se hallan en las tablas 4.5 y 4.6. Como puede verse, son notablemente más cortos, y existe un paralelismo entre las restricciones de un proceso de servicio (en el que debe de haber una restricción manual global y un único nodo inicial) y las restricciones de una actividad de servicio dentro de una composición de servicio (que también debe tener una restricción de rendimiento y un nodo inicial para su grafo anidado).

Un problema con la implementación de estas validaciones en EVL es que sólo se aplican cuando el usuario guarda el documento, y por lo tanto no impiden que el usuario realice cambios que violen alguna de las restricciones. En general resulta imposible hacerlo para todas las restricciones, dado que ralentizaría la interacción con el usuario. Sí que se han impuesto ciertas restricciones OCL para evitar que se creen flujos de control o datos no válidos, añadiendo los elementos apropiados en el modelo `gmfmap`.

Estas restricciones OCL son un subconjunto muy pequeño, pero útil, de todas las restricciones. En general, evitan que se produzcan ciclos directos (arcos en que el origen y el destino son el mismo) y que se conecten elementos *ObjectNode* a otros elementos mediante arcos *ControlFlow*. También evitan que se intercambien elementos *ObjectNode* entre elementos que no sean dos *Activity* separados. Las restricciones OCL son añadidas al modelo `gmfmap` utilizando el guión `Ecore2GMF.eol` invocado por EuGENia y el flujo de trabajo automatizado con Ant, tal y como se describió en el capítulo anterior.

Tabla 4.4. `Common.evl`: restricciones a cumplir por el metamodelo común y otros elementos compartidos a nivel de implementación

Elemento	Restricción	Correcciones
FlowNode	A menos que sea un *InitialNode*, debe tener al menos un arco entrante.	Ninguna.
	A menos que sea un *JoinNode* o haya ciclos en el grafo o los arcos entrantes sean de distintos tipos, debe tener como mucho un arco entrante.	Reasignar los múltiples arcos entrantes a un *JoinNode*.
	A menos que sea un *FinalNode*, debe tener al menos un arco saliente.	Ninguna.
	A menos que sea un *ForkNode* o sea un *DecisionNode* o haya ciclos en el grafo o los arcos entrantes sean de distintos tipos, debe tener como mucho un arco saliente.	Reasignar los múltiples arcos salientes a un *ForkNode* o a un *DecisionNode*.
	A menos que no se cumpla la restricción sobre el número mínimo de arcos entrantes, todos los arcos entrantes deberían ser del mismo tipo.	Ninguna.
	A menos que no se cumpla la restricción sobre el número mínimo de arcos salientes, todos los arcos salientes deberían ser del mismo tipo.	Ninguna.
	A menos que no se cumplan las restricciones de mismo tipo para los arcos entrantes o salientes o se trate de un *Activity*, el tipo de todos los arcos entrantes debería ser el mismo que el de los arcos salientes.	Ninguna.
FlowEdge	Ambos extremos deben estar asociados a un *FlowNode*.	Ninguna.

Continúa en la siguiente página

Continuado desde la página anterior

Elemento	Restricción	Correcciones
	El origen y el destino de un arco no puede coincidir, ya que originaría un ciclo directo y la ejecución nunca acabaría.	Borrar el arco.
ControlFlow	Si el arco contiene una condición, debe partir de un *DecisionNode*.	Retirar la condición.
	Si el arco parte de un *DecisionNode*, debe tener una condición asociada.	Poner «dummy» como condición, para indicar al usuario que debe rellenarla.
	A menos que no se cumplan las restricciones sobre condiciones o el arco no tenga condición asignada, su probabilidad deberá hallarse en el intervalo real (0, 1).	Ninguna.
	A menos que no se cumplan las restricciones sobre condiciones o el arco sí tenga condición asignada, su probabilidad deberá ser 1 ± 10^{-3}.	Asignar la probabilidad 1 al arco.
ObjectFlow	A menos que alguno de los extremos no esté definido, exactamente uno de los nodos asociados a los dos extremos se trata de un *ObjectNode*, y el otro ha de ser un *Activity*.	Ninguna.
Performance-Annotation	El número de usuarios accediendo concurrentemente para el que se impone el límite de tiempo debe ser mayor que 0.	Ninguna.
	El tiempo límite requerido debe ser mayor de 0.	Ninguna.
InitialNode	A menos que haya ciclos en el grafo, todos los caminos realizables desde este nodo deben llegar a un nodo final.	Ninguna.
	No debe haber arcos entrantes.	Eliminar todos los arcos entrantes.

Continúa en la siguiente página

Continuado desde la página anterior

Elemento	Restricción	Correcciones
	A menos que no se cumplan las restricciones sobre el número máximo de arcos salientes, ninguno de los arcos salientes puede apuntar a un *FinalNode*.	Añadir un *Activity* entre este nodo y el *FinalNode* problemático.
	No debe haber ciclos a partir del nodo actual.	Eliminar uno de los arcos que forma el primer ciclo encontrado.
FinalNode	No debe haber ningún arco saliente.	Eliminar los arcos salientes.
NamedElement	El nombre no puede estar vacío.	Ninguna.
ForkNode	A menos que no se cumplan las restricciones sobre el número mínimo de arcos entrantes y salientes de *FlowNode*, debe haber más de un arco saliente.	En caso de que se aplique la restricción y no se cumpla, se tiene un *ForkNode* con un arco entrante y otro saliente, por lo que se puede sustituir por un simple arco, ya que no hay división en múltiples ramas concurrentes.
JoinNode	A menos que no se cumplan las restricciones sobre el número mínimo de arcos entrantes y salientes de *FlowNode*, debe haber más de un arco entrante.	En caso de que se aplique la restricción y no se cumpla, se tiene un *JoinNode* con un arco entrante y otro saliente, por lo que se puede sustituir por un simple arco, ya que no hay división en múltiples ramas concurrentes.
	Todos los pares de ramas unidas deben tener el mismo ancestro común más cercano («least common ancestor» o LCA).	Ninguna.
DecisionNode	A menos que en alguno de los arcos salientes no se cumplan las restricciones relativas a probabilidades, el total debe estar en el intervalo 1 ± 10^{-3}.	Ninguna.
Activity-Performance-Annotation	La restricción debe estar asociada a algún elemento.	Ninguna.
	A menos que no esté asociada a ningún elemento, la restricción debe estar asociada a un *Activity*.	Ninguna.

Tabla 4.5. `ServiceProcess.evl`: restricciones adicionales a cumplir por el metamodelo de procesos de servicio

Elemento	Restricción	Correcciones
ServiceProcess	Debe haber exactamente un elemento *ProcessStart* en todo el diagrama.	Ninguna.
	Debe haberse creado la restricción de rendimiento global para todo el proceso.	Crear la restricción de rendimiento global del proceso con valores por omisión para el número de transacciones por segundo y su tiempo límite.
	Debe haber al menos un elemento *ProcessFinish* en todo el diagrama.	Ninguna.
Process-Performance-Annotation	Debe haber exactamente un elemento de este tipo en el diagrama.	Retirar este elemento.

4.4.3. Algoritmos de estimación de restricciones de rendimiento

Una vez se ha definido la estructura, notación general y restricciones para los metamodelos de procesos y composiciones de servicio, se puede proceder a la implementación de los algoritmos necesarios para poder estimar las restricciones de rendimiento no especificadas a partir de la información manualmente indicada. La utilidad de estos algoritmos es doble: en primer lugar, permite ahorrar una cantidad de tiempo considerable al modelador, que no tiene por qué expresar todo manualmente, y en segundo lugar, permite comprobar que las restricciones especificadas manualmente por el modelador no se contradicen entre sí.

Estos dos algoritmos se hallan implementados en el lenguaje EOL, aunque se invocan desde dos asistentes escritos en EWL y accesibles desde la interfaz gráfica de Eclipse. Se apoyan sobre otros dos algoritmos que resuelven dos problemas comunes al tratar con grafos: la detección de ciclos y el cálculo del ancestro común más cercano («least common ancestor» o LCA) de dos nodos. El algoritmo del cálculo del LCA de dos nodos es reutilizado para la validación, aprovechando las posibilidades que ofrece Epsilon de importar código EOL tanto desde EVL como desde EWL.

Algoritmo de detección de ciclos

Un camino en un grafo se define como una secuencia de nodos para la que hay un arco de cada nodo al siguiente en dicho grafo. Si el nodo inicial y el nodo final coinciden, se dice que se trata de un ciclo.

Los grafos cíclicos (aquellos que contienen ciclos) complican considerablemente todos los algoritmos que han de operar sobre ellos, dado que en principio tienen un número infinito de caminos distintos. En esta primera versión de la herramienta se realizará la suposición simplificadora de que los grafos son acíclicos (es decir, que no contienen ciclos), por lo que hay

Tabla 4.6. `ServiceComposition.evl`: restricciones adicionales a cumplir por el metamodelo de composiciones de servicio

Elemento	Restricción	Correcciones
Service-Composition	Debe haber exactamente un elemento del tipo *CompositionStart* en todo el diagrama.	Ninguno.
ServiceActivity	Debe tener asociada una restricción de rendimiento, preferiblemente derivada del modelo de proceso de servicio.	Crear la restricción de rendimiento local a la actividad de servicio con valores por omisión para el número de transacciones por segundo y su tiempo límite.
	Debe tener al menos un elemento de tipo *ActivityStart*.	Añadir un elemento *ActivityStart* al grafo anidado de este elemento.
	Debe tener como mucho un elemento de tipo *ActivityStart*.	Ninguna.
ActivityFlowEdge	El nodo origen y el nodo destino deben pertenecer a la misma actividad de servicio.	Eliminar el arco actual.
	El nodo origen y el nodo destino deben pertenecer a la jerarquía de los *ActivityFlowNode*.	Ninguna.
CompositionFlow-Edge	El nodo origen y el nodo destino deben pertenecer a la jerarquía de los *CompositionFlowNode*.	Ninguna.

que implementar la comprobación automática de esta condición antes de aplicar el resto de algoritmos. Para ser más exactos, los algoritmos trabajarán con grafos acíclicos orientados, ya que se considerarán distintos los arcos $A \rightarrow B$ y $B \rightarrow A$, para dos nodos A y B cualesquiera del grafo.

El algoritmo usado para detectar ciclos recibe un *FlowNode* inicial i, devuelve un conjunto de arcos E que puede estar vacío o tener un solo elemento, y se trata de un recorrido primero en profundidad a partir de i con marcado de los nodos por los que se ha pasado. El marcado se realiza usando una propiedad extendida de EOL Kolovos et al. [2009] llamada `visitStatus`, evitando tener que añadir un atributo a todos los *FlowNode* del metamodelo común sólo para realizar esta operación. Sigue estos pasos:

1. Se marcan todos los nodos como no visitados, usando el valor `NOT_VISITED` de la enumeración *VisitStatus* definida en el metamodelo común.

2. Se comprueba si el nodo ha sido visitado, es decir, si el nodo actual i está marcado con `VISITED`. En ese caso, se ha comprobado previamente que el subgrafo del nodo actual (i) y sus descendientes no tiene ningún ciclo, por lo que se devuelve $E = \emptyset$ como resultado del algoritmo.

3. Se marca el nodo actual indicando que está siendo visitado (`VISITING`).

4. Se recorren todos los arcos salientes $a \in i.outgoing$ del nodo actual, ejecutando estos pasos para cada uno:

   4.1. Se comprueba si el nodo destino del arco actual está marcado como siendo visitado (`VISITING`). Si es así, se devuelve $E = \{a\}$ como resultado del algoritmo: ha detectado un ciclo, y a es uno de los arcos que lo forma.

   4.2. Se invoca recursivamente este algoritmo sobre el nodo destino del arco actual, empezando por el paso 2 y obteniendo la salida E_R. Si $E_R \neq \emptyset$, entonces se devuelve el resultado $E = E_R$.

5. Se marca el nodo actual indicando que ha sido recorrido en su totalidad sin encontrar ciclos (`VISITED`).

Algoritmo de cálculo del LCA de dos nodos de un grafo orientado acíclico

Existen múltiples definiciones equivalentes para el concepto de ancestro común más cercano («least common ancestor» o LCA). En este trabajo, se define el conjunto de ancestros comunes más cercanos («set of least common ancestors» o SLCA) como las hojas del subgrafo que contiene a todos los ancestros comunes de los dos nodos bajo consideración. Puede decirse que un nodo A es ancestro de B, y que B es descendiente de A, cuando hay un camino de A a B. Si el camino recorre un solo arco (es decir, es de longitud 1), se dice que A es el «padre» de B, y que B es el «hijo» de A. Se dice que un nodo es una «hoja» cuando no tiene ningún descendiente aparte de sí mismo.

Se han publicado varios algoritmos para calcular el LCA de dos nodos de tanto árboles como grafos orientados acíclicos Bender et al. [2001]. Muchos de los algoritmos existentes preprocesan el árbol o grafo original para conseguir que las consultas posteriores consistan en una rápida búsqueda en una tabla. Estos algoritmos se encuentran diseñados para grafos muy grandes y densos (es decir, con muchos más arcos de los estrictamente necesarios para

conectar a todos los nodos). Este tipo de grafos es particularmente común en campos como la bioinformática, por ejemplo.

Sin embargo, en los modelos de este trabajo no se llegarán a tener grafos de tal tamaño, y no van a ser especialmente densos. Además, no se necesitará conocer el LCA de todo par de nodos, sino sólo de los padres de los *JoinNode*. De hecho, los resultados obtenidos por Bender et al. Bender et al. [2001] indican que el algoritmo más simple de implementar (basado en la definición anterior de LCA) tiene un rendimiento aceptable para grafos con estas características, aunque existen algoritmos más eficientes. De todos modos, en un futuro se podría sustituir este algoritmo por otro de mejor comportamiento asintótico sin impactar al resto de las herramientas.

El algoritmo recibe dos nodos A y B de un grafo G y devuelve $lca(A, B)$, otro nodo del grafo, siguiendo estos pasos:

1. Se calculan las profundidades $P(A)$ y $P(B)$ de los nodos A y B, respectivamente y se obtiene $M = \text{máx}\{P(A), P(B)\}$.

   En este trabajo, se define de forma general la profundidad de un nodo de un grafo orientado acíclico como la máxima longitud de los caminos que van desde un nodo sin arcos entrantes a dicho nodo. Dado que las restricciones EVL aseguran que los *InitialNode* no tienen arcos entrantes y que sólo puede haber uno en cada grafo, puede concretarse la definición y decir que se trata de la longitud del camino más largo a partir del nodo inicial.

   Las profundidades de cada nodo se calculan una sola vez, ya que el resultado de la invocación a la operación EOL `getDepth` responsable es almacenado en memoria, gracias a su anotación `@cached`.

2. Se asignan los antecesores de A con profundidad menor o igual a M al conjunto A_A.

3. Se obtiene A_B análogamente a partir de B.

4. Se calcula $A = A_A \cap A_B$ y se devuelve uno de los elementos del conjunto $SLCA(A, B) = \{e | P(e) = max_{a \in A}\{P(a)\}\}$, formado por los elementos comunes de A_A y A_B que tienen la máxima profundidad.

Al implementar este algoritmo se ha supuesto que $|SLCA(A, B)| = 1$ para todo par de nodos A y B de cualquiera de los grafos que superan las restricciones EVL, es decir, que sólo tienen un antecesor común con la máxima profundidad. Esto sólo es cierto en general para árboles, y no para grafos, por lo que se han añadido restricciones a los *JoinNode* para asegurar que esta suposición se cumple.

La figura 4.4 describe este algoritmo de forma gráfica: para obtener el LCA de D y F, se obtienen los antecesores de cada uno. Estos antecesores se hallan divididos en antecesores comunes (círculos) y antecesores no comunes (pentágonos para D y rombos para F). De entre los antecesores comunes, se toma aquel que tenga la mayor profundidad: B, destacado con un borde grueso.

Algoritmo de estimación de restricciones de usuarios concurrentes

Este algoritmo recibe un grafo que cumple todas las restricciones EVL antes descritas y cuyos elementos son especializaciones de los elementos del metamodelo común, y añade a

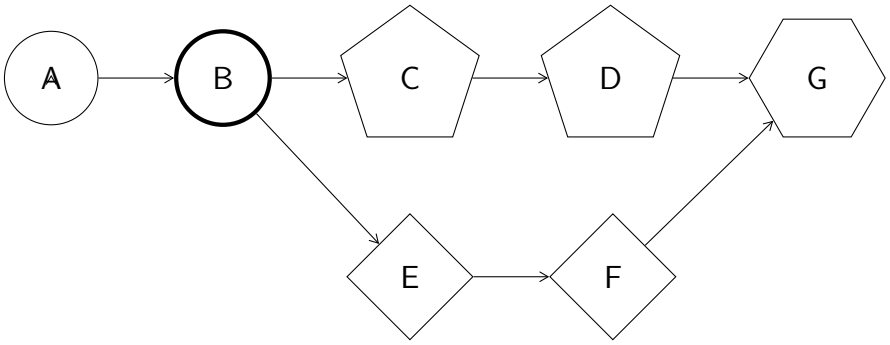


Figura 4.4. Descripción gráfica del cálculo de $B = lca(D, F)$

los *Activity* e anotables que no tengan ninguna restricción la información necesaria sobre el número de usuarios concurrentes estimados automáticamente, $U_A(e)$, a los que deberían ser capaces de dar servicio bajo el tiempo límite. Las anotaciones que hayan sido generadas en una ejecución anterior son actualizadas.

Por lo general, se respetan las restricciones indicadas manualmente $U_M(e)$ sobre los *Activity* correspondientes. La única excepción es cuando el usuario impone unas restricciones demasiado relajadas y $U_A(e) > U_M(e)$, en cuyo caso se ofrece al usuario la posibilidad de corregir la situación de forma automática, aprovechando las capacidades interactivas que ofrece EWL. Si el usuario rechaza la corrección, el proceso completo de anotación es cancelado. La ejecución del algoritmo de anotación tiene una semántica transaccional, gracias a EMC: o se realizan todos los cambios propuestos, o no tiene efecto alguno.

Es importante notar que, aunque sólo interesan realmente los valores de $U_A(e)$ para los *Activity*, se necesita definir su valor para todos los *FlowNode* y *FlowEdge* que participan en los grafos, para poder propagar la información disponible desde el nodo inicial hasta cada uno de los *Activity*. Sin embargo, $U_M(e)$ sólo se encuentra definido para los tipos derivados de *Activity* de cada metamodelo, puesto que son los únicos que el usuario puede anotar.

Otro detalle de interés es que aunque en un mismo nodo se disponga de $U_A(e)$ y $U_M(e)$, sólo se propaga el valor de $U_A(e)$ a sus descendientes, ya que éste será menor. La idea detrás de esta limitación es evitar que una restricción manual más estricta de lo necesario sobre un solo elemento (normalmente hecha pensando en otro proceso más exigente o en necesidades futuras) acabe afectando al subgrafo completo de sus descendientes. En futuras versiones de la herramienta, se podría dejar que el usuario controlara este comportamiento de propagación.

El algoritmo sigue estos pasos:

1. Se relajan todas las restricciones automáticas (con `manuallyAdded` a `false`), para permitir que se actualicen fácilmente, asignando 0 al campo `concurrentUsers` de cada una.

2. Para asegurar que las correcciones automáticas hechas sobre las restricciones manuales del usuario se propaguen a lo largo del grafo, se recorren sus *FlowNode* en orden ascendente de profundidad (es decir, se realiza un recorrido primero en anchura), ya que el número de usuarios concurrentes por segundo de cualquier elemento del grafo sólo depende de sus arcos entrantes y el subgrafo compuesto por sus antecesores. Cada uno de los elementos visitados e se procesa de la siguiente forma:

   2.1. Se calcula el valor de $U_A(e)$ para dicho nodo, usando las definiciones de la tabla 4.7,

donde se utiliza la notación $A.B$ para indicar la lectura del atributo B del objeto A. Este valor es almacenado en memoria para agilizar los cálculos de nodos posteriores. En EOL, esto se consigue anotando con `@cached` la operación que devuelve su valor.

2.2. Si e tiene una restricción manual sobre el número de usuarios concurrentes $U_M(e)$ impuesta, se comprueba si $U_A(e) > U_M(e)$. En dicho caso, se ofrece al usuario la posibilidad de hacer que $U_M(e) = U_A(e)$ cambiando la restricción. Si el usuario rechaza la propuesta, el proceso completo de anotación se cancela, revirtiendo todos los cambios hechos desde el inicio de la ejecución del algoritmo. Si por el contrario ocurre que $U_M(e) \geq U_A(e)$, entonces esto no supone ningún problema, y se pasa al siguiente nodo, volviendo al paso 2.

2.3. Se anota e con el valor de $U_A(e)$, creando la anotación si no existe y actualizando su campo `concurrentUsers` si existe.

Tabla 4.7. Definiciones de $U_A(e)$ para cada tipo de elemento de los metamodelos

Tipo más específico del elemento e	**Valor de** $U_A(e)$
ControlFlow	$U_A(e) = e.\text{probability} \cdot U_A(e.\text{source})$
FlowEdge	$U_A(e) = U_A(e.\text{source})$
JoinNode	Sean: $I = e.\text{incoming}$ $P = \{i.\text{source} \mid i \in I\}$ $SLCA = \{lca(a,b) \mid a,b \in P, a \neq b\}$ En un grafo cualquiera, *SLCA* podría tener varios elementos, pero la validación EVL sobre *JoinNode* comprueba que el $lca(a,b)$ de todo par $(a,b) \in P$ sea siempre el mismo nodo L. Entonces, utilizando el predicado `isKindOf(T)` de EOL que indica si un elemento es de una clase hija o equivalente a T: $U_A(e) = \begin{cases} \sum_{i \in I} U_A(i), & \text{L.isKindOf(ForkNode)} \\ \text{mín}_{i \in I} \{U_A(i)\}, & \text{L.isKindOf(DecisionNode)} \end{cases}$

Continúa en la siguiente página

Continuado desde la página anterior

Tipo más específico del elemento e	**Valor de** $U_A(e)$
FlowNode	Utilizando la operación `first` de toda colección de EOL, que devuelve su primer elemento, se tiene que: $U_A(e) = U_A(e.\text{incoming.first}())$ El hecho de usar solamente el primer elemento no supone pérdida de información, ya que las restricciones EVL especifican que sólo los *JoinNode* pueden tener varios arcos entrantes, y dichos nodos se tratan en su propio caso.
ProcessStart	Siendo P es la primera y única *ProcessPerformanceAnnotation* del modelo, se tiene que: $U_A(e) = P.\text{concurrentUsers}$
ActivityStart	Se accede a la propiedad `activity` de todo *ActivityFlowNode*, que señala al *ServiceActivity* en el que se halla contenido, consultando su anotación: $U_A(e) = e.\text{activity.annotation.concurrentUsers}$

Algoritmo de estimación de restricciones de tiempo límite

Este algoritmo, de forma similar al anterior, recibe un grafo G que cumple todas las restricciones EVL mencionadas en §4.4.2 y anota los *Activity* e que no tengan restricciones manuales con información sobre el tiempo límite $T_L(e)$ del que disponen para completar su ejecución, de forma que se cumplan estas condiciones:

- La suma de los tiempos límite de todos los caminos disponibles desde el nodo inicial hasta cualquiera de los nodos finales ha de ser menor o igual al tiempo límite $T_L(G)$ del grafo completo.

  Es decir, si C es el conjunto de todos los caminos desde el nodo inicial hasta cada uno de los nodos finales, y se define $E(c)$ como el conjunto de los *Activity* que aparecen en un camino c, entonces debe cumplirse que:

$$\forall c \in C \sum_{e \in E(c)} T_L(e) \leq T_L(G) \tag{4.1}$$

- Los tiempos límite de cada *Activity* han de ser números reales comprendidos en el intervalo $(0, T_L(G)]$:

$$\forall c \in C \, \forall e \in E(c) \quad 0 < T_L(e) \leq T_L(G) \tag{4.2}$$

- Los tiempos límite asignados a cada nodo deben ser equitativos en la medida de lo posible. Es decir, que si hay N actividades en un camino que no debería tardar más de T segundos en ejecutarse, el tiempo límite para cada actividad debería ser de T/N segundos a falta de otras restricciones. La idea detrás de esta restricción es no imponer un mayor rendimiento a ninguna parte del sistema a falta de otra información. En otros términos, se desea minimizar el valor de:

$$\sum_{c \in C} \sum_{a,b \in E(c)} |T_L(a) - T_L(b)| \tag{4.3}$$

- Los tiempos límite asignados a cada nodo deberían tan grandes como fuese posible. En otros términos, si se define la holgura $H(c, G)$ existente en cada camino c del grafo G de la siguiente forma:

$$H(c, G) = T_L(G) - \sum_{e \in E(c)} T_L(e) \tag{4.4}$$

  La expresión a minimizar es la siguiente:

$$\sum_{c \in C} H(c, G) \tag{4.5}$$

Si un *Activity* no disponía de restricción de rendimiento se le creará una, y si ya disponía de una restricción automática se actualizará su campo `timeLimit`. El algoritmo notificará al usuario si ha planteado restricciones contradictorias en su modelo, pero en este caso no existe forma de corregir dichos problemas de forma automática: se cancelará la ejecución del algoritmo y se revertirán todos los cambios que haya hecho hasta el momento.

Tras definir las condiciones sobre sus resultados y su ejecución, se plantearon varias opciones para implementar el algoritmo. La primera opción considerada fue basarse en alguno de los algoritmos para tiempos en grafos ya existentes, como CPM o PERT, pero estos algoritmos reciben los tiempos como entrada, en vez de producirlos como salida, por lo que no habrían servido. Otra posibilidad, que se desarrolló en más profundidad, pero finalmente se descartó, fue la de transformar el problema basado en grafos a un problema de programación lineal («linear programming» o LP). A continuación se presentan dos de las formulaciones probadas, tras lo cual se pasará a definir el algoritmo heurístico finalmente utilizado.

Primer intento de formulación mediante LP La primera formulación probada representa los tiempos límite $T_L(e)$ a establecer sobre cada actividad de servicio o acción e como variables reales x_e. Para asegurar que se cumple (4.1), habría que añadir una restricción de la siguiente forma por cada camino $c \in C$:

$$\forall c \in C \sum_{e \in E(c)} x_e \leq T_L(G)$$

Si se añade la siguiente restricción, se cumplirá además (4.2):

$$\forall c \in C \, \forall e \in E(c) \quad x_e > 0$$

Por último, se le añadirían las restricciones manuales M sobre los tiempos de alguna de las actividades de servicio o acciones. Estas restricciones consisten en pares ordenados de la forma (e, m), donde e es el elemento a anotar y m su tiempo manual $T_M(e)$.

$$\forall (e, m) \in M \quad x_e = m$$

Sin embargo, la minimización de (4.3) requeriría en principio usar una función objetivo no lineal. Considerando que

$$|x| = \sqrt{x^2}, x \in \mathbb{R} \tag{4.6}$$

y sustituyendo en (4.3), ignorando la raíz (ya que su valor exacto no importa, sino su magnitud relativa respecto a otras asignaciones de valores para las variables x_e), se obtiene la función objetivo (4.7):

$$\begin{aligned} \text{mín} \sum_{c \in C} \sum_{a,b \in E(c)} |x_a - x_b| &= \text{mín} \sum_{c \in C} \sum_{a,b \in E(c)} (x_a - x_b)^2 \\ &= \text{mín} \sum_{c \in C} \sum_{a,b \in E(c)} x_a^2 + x_b^2 - 2x_a x_b \end{aligned} \tag{4.7}$$

Sin embargo, (4.7) no es una restricción lineal, ya que incorpora términos de grado 2 y productos entre las distintas variables. Se trataría de un problema de programación no lineal («non-linear programming» o NLP), cuya resolución puede tener un coste excesivo en términos de tiempo de implementación y tiempo de ejecución.

Segundo intento de formulación mediante LP Otra forma de acercarse a una posible formulación en términos de un problema LP es utilizar como variables no los tiempos finales, sino las diferencias d_e de $T_L(e)$ respecto a un tiempo equitativo $T_E(e)$ resultado de repartir por igual los tiempos límite entre los nodos de cada camino.

$T_E(e)$ debe definirse como la asignación mínima equitativa de tiempo de entre todos los caminos, ya que un mismo nodo puede aparecer como parte de varios caminos de distinta longitud. Cuanto más largo sea el camino, menor tiempo se le asignará como resultado de un reparto equitativo.

Sea $C_I(e)$ el conjunto de los caminos que pasan por c. Las definiciones de $T_E(e)$ y d_e son las siguientes:

$$\begin{aligned} C_I(e) &= \{c \in C \mid e \in c\} \\ T_E(e) &= \frac{T_L(G)}{\text{máx}_{c \in C_I(e)}|c|} \\ d_e &= T_L(e) - T_E(e) \end{aligned}$$

Empezaremos por imponer estas restricciones, para cumplir (4.1):

$$\forall c \in C \sum_{e \in E(c)} d_e \leq T_L(G) - \sum_{e \in E(c)} T_E(e)$$

Suponiendo que $T_E(e)$ es el menor valor que tomarán los $T_L(e)$, para así cumplir (4.2) y partir de un reparto equitativo que le dé un valor pequeño a (4.3), ha de darse que:

$$\forall c \in C \, \forall e \in E(c) \quad d_e \geq 0$$

Finalmente, para las restricciones manuales:

$$\forall (e, m) \in M \quad d_e = m - T_E(e)$$

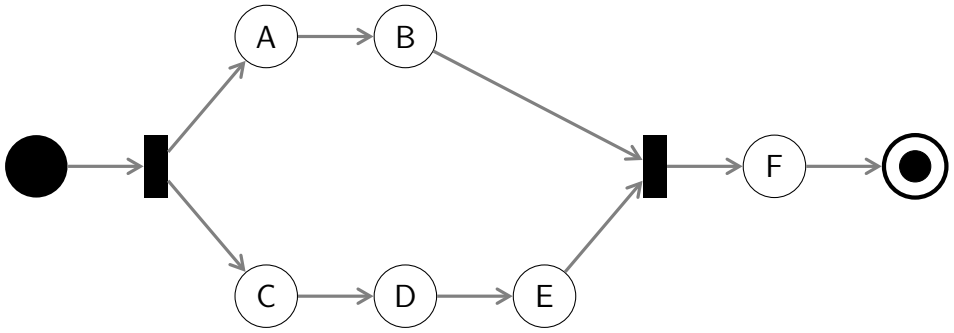


Figura 4.5. Grafo problemático para la segunda formulación con programación lineal

La función objetivo se obtiene operando ligeramente sobre (4.5) y usando la definición de holgura de (4.4):

$$\begin{aligned} \text{mín} \sum_{c \in C} H(c, G) &= \text{mín} \sum_{c \in C} \left(T_L(G) - \sum_{e \in E(c)} T_L(e) \right) \\ &= \text{mín} \sum_{c \in C} \left(T_L(G) - \sum_{e \in E(c)} T_E(e) - \sum_{e \in E(c)} d_e \right) \end{aligned} \tag{4.8}$$

Al retirar aquellos términos que no dependen de las variables d_e de (4.8), se obtiene la función objetivo final (4.9):

$$\begin{aligned} \text{mín} \sum_{c \in C} \left(- \sum_{e \in E(c)} d_e \right) &= -\text{mín} \sum_{c \in C} \sum_{e \in E(c)} d_e \\ &= \text{máx} \sum_{c \in C} \sum_{e \in E(c)} d_e \end{aligned} \tag{4.9}$$

Esta formulación sí es lineal y asegura que se cumple (4.1), tomando como base una solución basada en un reparto equitativo (con un valor bueno en (4.3), aunque no necesariamente el óptimo), y trata de optimizar el valor de (4.5). Supone una mejora considerable respecto a la situación anterior, pero sigue sin realmente resolver el problema.

La cuestión es que maximizar la función objetivo no asegura que los valores finales de x_e sigan minimizando (4.3), sino que potencialmente asignaría todo el tiempo sobrante en un camino a un solo nodo, aumentando innecesariamente el valor de (4.3).

Sirva de ejemplo el grafo de la figura 4.5. Suponiendo que $T_L(G) = 1$ y que no se imponen restricciones manuales, el grafo se correspondería con el siguiente problema:

Objetivo Maximizar $d_a + d_b + d_c + d_d + d_e + 2d_f$

Restricciones

$$\begin{aligned} d_a, d_b, d_c, d_d, d_d, d_e, d_f &\geq 0 \\ d_a + d_b + d_f &\leq 1 - \frac{1}{3} - \frac{1}{3} - \frac{1}{4} && = \frac{1}{12} \\ d_c + d_d + d_e + d_f &\leq 1 - 4\frac{1}{4} && = 0 \end{aligned}$$

Una de las soluciones óptimas es:

$$d_a = 1/12$$
$$d_b = d_c = d_d = d_e = d_f = 0$$

Sin embargo, esta solución aumenta el valor de (4.3) en $1/12$, cuando se podría haber evitado. Existe otra solución, también óptima, que minimiza tanto (4.3) como (4.5): repartir ese $1/12$ de segundo por igual entre A y B:

$$d_a = d_b = 1/24$$
$$d_c = d_d = d_e = d_f = 0$$

Definición de un algoritmo heurístico de refinamiento iterativo Tras estos dos intentos de crear formulaciones basadas en problemas de programación lineal, se decidió implementar un algoritmo desde cero para resolver el problema. Este algoritmo es de tipo heurístico e iterativo, revisando y completando su solución hasta acercarse a una solución buena, que no necesariamente tiene por qué ser óptima. Se ha diseñado para que cumpla (4.1), al mismo tiempo que trata de minimizar simultáneamente (4.3) y (4.5), y para que dé más información al usuario en caso de que algunos de los requisitos manuales se contradigan entre sí.

El algoritmo requiere de las siguientes definiciones:

- Se considera que un nodo se halla «libre» cuando no tiene una restricción manual de tiempo límite impuesta, es decir, cuando la restricción no existe o ha sido generada automáticamente.
- Se dice que un *Activity* e con tiempo límite actual x_e es «actualizable» a un nuevo tiempo límite x'_e cuando es libre y se cumple que $x'_e \leq x_e$. Es decir, siempre se puede actualizar el tiempo límite a un valor igual o más pequeño, pero no a un valor más grande, ya que entonces se correría el riesgo de no cumplir (4.1) para alguno de los caminos del grafo.
- $A(c, l)$ representa al conjunto de los elementos actualizables a un nuevo límite l en un camino $c \in C$. De esta definición se deriva que $A(c, 0)$ es el conjunto de los elementos libres en el camino c, ya que sus tiempos límite han de ser estrictamente mayores que 0, por (4.2). También se puede ver que $E(c) - A(c, l)$ representa al conjunto de los *Activity* no actualizables a l en el camino c, bien porque tienen restricciones manuales, o bien porque su límite de tiempo es menor que l.
- El tiempo manual del camino c, $T_M(c)$, se define como la suma de los tiempos límite asignados manualmente dentro de dicho camino:

$$T_M(c) = \sum_{e \in E(c), (e,m) \in M} m \tag{4.10}$$

- El tiempo fijado en el camino c cuando el límite a intentar aplicar es l, $T_F(c, l)$, se define como la suma de los tiempos de los nodos no actualizables al límite l en dicho camino:

$$T_F(c, l) = \sum_{e \in E(c) - A(c,l)} T_L(e) \tag{4.11}$$

Los pasos del algoritmo son los siguientes:

1. Se calcula el conjunto C con todos los caminos realizables desde el nodo inicial hasta cada uno de los nodos finales.

2. Se obtiene el tiempo global límite para el grafo, $T_L(G)$, a partir de su restricción global de rendimiento. Esta información se almacena a nivel del grafo completo en los modelos de proceso de servicio, y a nivel de actividad de servicio en los modelos de composición de servicio.

3. Para cada camino $c \in C$:

   3.1. Se calcula la suma de los tiempos límite asignados manualmente dentro de dicho camino, $T_M(c)$, y se almacena en memoria.

   3.2. Se almacena en memoria la holgura disponible para las restricciones automáticas, $H_A(c, G)$, definida como:

   $$H_A(c, G) = T_L(G) - T_M(c) \tag{4.12}$$

   3.3. Se relajan todas las restricciones automáticas, dando un valor arbitrariamente grande a sus tiempos límite. De esta forma, todos los nodos libres serán actualizables la primera vez que sean visitados.

4. Para cada camino $c \in C$, en orden ascendente según el valor de $H_A(c, G)/(1 + A(c, 0))$, de forma que se visiten los caminos con los requisitos más fuertes primero, se realizan los siguientes pasos:

   4.1. Si $H_A(c, G) = 0 \pm 10^{-3}$, se comprueba si además $|A(c, 0) > 0|$. En dicho caso, no queda holgura que asignar y hay al menos un nodo libre, con lo que se informa de la situación al usuario y se cancela el proceso completo. De lo contrario, se pasa al siguiente camino (paso 4), ya que no queda tiempo que repartir.

   4.2. Si $H_A(c, G) < 0$, entonces por (4.12) se tiene que $T_L(G) < T_M(c)$. Es decir, el usuario ha especificado unos tiempos manuales en el camino que superan al tiempo límite establecido a nivel global en el grafo, haciendo que no se cumpla la restricción (4.1). En dicho caso, se informa de la situación al usuario y se cancela el proceso completo.

   4.3. Si no se aplicaba ninguno de los dos casos anteriores, entonces es que $H_A(c, G) > 0$. Si además $|A(c, 0)| > 0$, entonces se puede repartir la holgura existente en el camino entre sus nodos libres. El proceso de reparto se describe a continuación.

Por su relativa complejidad, el algoritmo de reparto de una determinada holgura $H_A(c, G)$ sobre los nodos libres $A(c, 0)$ del camino c se describe aparte. El algoritmo va revisando su solución hasta que se estabiliza, momento en el que la aplica.

En este algoritmo se usará la notación $x \leftarrow b$ para indicar que se asigna el valor b a la variable x, con lo que lecturas posteriores de esta variable obtendrán el valor b hasta que se realice otra asignación. Los pasos seguidos son los siguientes:

1. Se calculan los dos primeros términos de la sucesión de los límites l_i a aplicar sobre los nodos actualizables:

$$l_0 \leftarrow \frac{T_L(G) - T_M(c)}{|A(c,0)|}$$
$$l_1 \leftarrow \frac{T_L(G) - T_F(c,l_0)}{|A(c,l_0)|}$$
$$i \leftarrow 1$$

   Como puede verse, se hace un reparto equitativo de acuerdo con (4.3) de toda la holgura de que se dispone, tal y como exige (4.5), pero sólo entre los nodos que sean actualizables al nuevo límite, ya que de lo contrario se podría dejar de cumplir (4.1) en algún momento.

   Hay que asegurarse de que en todo momento $|A(c,l_i)| > 0$, para prevenir una división por cero. Si resulta que $|A(c,l_i)| = 0$ en algún momento, entonces es que no se ha podido repartir la holgura, y se termina sin más la ejecución del algoritmo. Esta comprobación debe repetirse también dentro del paso 2.

2. De forma indefinida, hasta que se salga del bucle explícitamente:

   2.1. Si $|l_i - l_{i-1}| \leq 10^{-3}$ (es decir, $l_i \simeq l_{i-1}$), entonces se sale del bucle, saltando al paso 3: se ha calculado el límite final $l_f = l_i$ a aplicar sobre los nodos actualizables a él en este camino.

   Si el límite se ha estabilizado, ello quiere decir que el número de nodos actualizables a él también, por lo que se ha llegado al reparto que afecta a más nodos de la holgura entre los nodos de este camino, sin invalidar (4.1).

   2.2. De lo contrario, se genera el siguiente término y se vuelve al paso anterior:

$$l_{i+1} \leftarrow \frac{T_L(G) - T_F(c,l_i)}{|A(c,l_i)|}$$
$$i \leftarrow i + 1$$

3. Se hace $T_L(e) \leftarrow l_f$ sobre cada uno de los $e \in A(c,l_f)$.

4.4.4. Transformación de modelos de proceso de servicio a modelos de composición de servicio

Para maximizar el retorno sobre la inversión en tiempo necesaria para elaborar un modelo de proceso de servicio, se implementó un guión ETL con una transformación automatizada de los modelos de proceso de servicio extendidos a los modelos de composición de servicio. Esta transformación serviría para obtener un primer boceto del modelo de composición de servicio, manteniendo las anotaciones de rendimiento estimadas a través de los algoritmos anteriores, de forma que se pudieran reutilizar para obtener las restricciones sobre las acciones específicas.

Esta transformación se ha implementado como un nuevo *plugin* de Eclipse, que extiende su interfaz gráfica con un nuevo elemento en el menú contextual producido al pulsar con el botón secundario del ratón sobre un fichero de modelo de proceso de servicio (con extensión `serviceprocess`) en su navegador de ficheros.

Las correspondencias entre los elementos no abstractos de los metamodelos de proceso de servicio y de composición de servicio se hallan descritas en la tabla 4.8. Se ha utilizado la notación o para hacer referencia al elemento original y d_i para hacer referencia al i-ésimo elemento correspondiente del metamodelo destino. Cuando haya sólo un elemento, se utilizará simplemente d. De la misma forma que en la tabla 4.7 con las definiciones de $U_A(e)$, se emplea la notación $A.B$ para indicar que se lee el atributo B del objeto A.

Tabla 4.8. Correspondencias del metamodelo de proceso de servicio al metamodelo de composición de servicio

Elemento del proceso de servicio (o)	**Elemento(s) de la composición de servicio (d_i)**
ServiceProcess	*ServiceComposition*, donde: ▪ d.nodes contiene las correspondencias de tipo CompositionFlowNode de los elementos en o.nodes, ▪ d.compositionEdges contiene las correspondencias de los elementos en o.edges, ▪ d.activityEdges contiene las correspondencias de tipo ActivityFlowEdge de los elementos en o.nodes, ▪ d.swimlanes contiene un nuevo elemento de tipo SwimlaneContainer, con una sola pista llamada «Main», y ▪ d.activityPerformance contiene el elemento correspondiente a o.activityPerformance.
ActivityPerformanceAnnotation	*ActivityPerformanceAnnotation*, copiando los atributos `concurrentUsers` y `secsTimeLimit` y asignando la correspondencia de o.execNode en d.execNode.
ProcessControlFlow	*CompositionControlFlow*, copiando los atributos `probability` y `condition` y asignando la correspondencia de o.target a d.target y de o.source a d.source.

Continúa en la siguiente página

Continuado desde la página anterior

Elemento del proceso de servicio (o)	**Elemento(s) de la composición de servicio (d_i)**
ServiceActivity	Un *ServiceActivity* (d_1) y dos *ActivityControlFlow* (d_2, d_3). d_1 se inicializa mediante las siguientes acciones: ▪ Se copia o.name a d_1.name. ▪ Se asigna la correspondencia de o.annotation a d_1.annotation. ▪ Se asignan las correspondencias de los elementos de o.incoming a d_1.incoming. ▪ Se asigna la correspondencia de o.outgoing a d_1.outgoing. ▪ Se inicializa d_1.nodes añadiendo tres nuevos elementos: un nodo *ActivityStart* un nodo *ActivityAction* (cuyo atributo `name` recibe el valor o.name) y un nodo *ActivityFinish*. Para inicializar d_2 y d_3, se utilizan los tres nuevos elementos antes creados: ▪ se referencia al nodo *ActivityStart* creado desde d_2.source, ▪ se referencia al nodo *ActivityAction* creado desde d_2.target y d_3.source, y ▪ se referencia al nodo *ActivityFinish* creado desde d_3.target.
ProcessStart, ProcessFinish, ProcessDecision, ProcessFork y ProcessJoin	Elemento correspondiente de la jerarquía con raíz en *CompositionFlowNode*. Por ejemplo, a un nodo *ProcessStart* le corresponde un nodo *CompositionStart*. En todos ellos, se asignan d.incoming y d.outgoing a las correspondencias de los elementos de o.incoming y o.outgoing.

4.5. Conclusiones

En las anteriores secciones se han descrito las modificaciones actualmente realizadas sobre las herramientas proporcionadas por el grupo Kybele para llevar a cabo el modelado de acuerdo con la metodología SOD-M en marzo de 2009.

Los resultados son positivos a nivel técnico: se han reimplementado parte de las herramientas de SOD-M, añadiéndoles lo necesario para modelar las restricciones de rendimiento propuestas y mejorando la experiencia del desarrollador y del usuario. Además, se ha elaborado un algoritmo que propaga las restricciones del tiempo límite bajo una carga de un número determinado de usuarios accediendo concurrente al sistema de una parte de los elementos del modelo al resto. Dado que los algoritmos requieren que el modelo cumpla ciertas condiciones de integridad, se implementó previamente su validación automática, con sugerencias de corrección en un número considerable de casos.

Este algoritmo es usado tanto en los modelos de proceso de servicio como en los de composición de servicio. El usuario crearía los modelos de proceso de servicio, ejecutaría el algoritmo de estimación y luego usaría la correspondencia automática implementada para pasar a los modelos de composición de servicio. Tras terminar de especificar el resto de detalles en el modelo de composición de servicio, podría usar una vez más el algoritmo de estimación, para conseguir las restricciones sobre las acciones concretas. Estas restricciones podrían usarse para generar casos de prueba en el lenguaje final en que se implementara cada una de las acciones.

Existen diversas líneas de trabajo futuro en las que profundizar. Por un lado, el grupo Kybele ha revisado la metodología SOD-M desde el inicio del trabajo, por lo que resulta necesario evaluar el impacto de estos cambios sobre las extensiones realizadas. Por otro lado, es imprescindible implementar la generación de código (o al menos de un esqueleto) a partir de los modelos especificados, ya bien sea a un lenguaje de propósito general (como Java o C++) como a lenguajes de modelado de procesos de negocio (como WS-BPEL o BPMN). Esto permitirá generar los casos de prueba de rendimiento para el lenguaje escogido, completando así la secuencia de transformaciones desde los modelos de alto nivel hasta los productos finales. También posibilitará el comienzo de la implementación de la segunda propuesta de extensión de SOD-M: el modelado del comportamiento funcional de las acciones del metamodelo de composiciones de servicio, y su combinación con otras técnicas de prueba existentes.

Es necesario además revisar las propias extensiones de modelado de rendimiento realizadas. Durante el desarrollo de estos trabajos de extensión, se han observado ciertos aspectos mejorables en las extensiones propuestas. Es posible que el usuario no desee especificar a la vez manualmente el número máximo de usuarios concurrentes estimados o el tiempo límite de respuesta, sino sólo uno de ellos a la vez. Además, puede que el usuario no tenga una estimación absoluta, sino relativa, de la carga que supone una determinada actividad de servicio o acción. Es decir, que sepa que su ejecución requiere el doble de recursos que el resto de actividades en los caminos que participa, en promedio. Por último, el usuario podría estar interesado en controlar la forma en que se propagan sus restricciones manuales al resto del grafo.

Finalmente, se podría mejorar la usabilidad del editor gráfico, mejorando el algoritmo de cálculo de las posiciones finales de los elementos y modelando explícitamente la contención de una acción de una actividad de servicio en una pista. Sería interesante evaluar la posibilidad de simplificar la implementación de los metamodelos, evitando la necesidad de utilizar grafos anidados y múltiples jerarquías de nodos en los modelos de composición de servicio. Si los modelos empezaran a crecer considerablemente, sería necesario reemplazar el algoritmo trivial

usado actualmente para calcular el LCA de dos nodos por uno con un mejor rendimiento asintótico. Independientemente del tamaño de los modelos, habría que validar con más pruebas el algoritmo de estimación de las restricciones no especificadas.

Bibliografía

Apache Foundation. Apache Ant 1.7.1, June 2008. URL http://ant.apache.org/. Fecha de última comprobación: 2 de noviembre de 2009. 4.11

Michael A Bender, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Finding least common ancestors in directed acyclic graphs. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 845–853, 2001. doi: 10.1.1.15.9161. 4.27, 4.28

María Valeria de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 4.1, 4.4

Eclipse Foundation. ATLAS Transformation Language (ATL), 2009a. URL http://www.eclipse.org/m2m/atl/. Fecha de última comprobación: 2 de noviembre de 2009. 4.7

Eclipse Foundation. Eclipse Modeling Framework, 2009b. URL http://eclipse.org/modeling/emf/. Fecha de última comprobación: 2 de noviembre de 2009. 4.2

Eclipse Foundation. EuGENia, 2009c. URL http://www.eclipse.org/gmt/epsilon/doc/eugenia/. Fecha de última comprobación: 2 de noviembre de 2009. 4.9

Eclipse Foundation. EuGENia GMF tutorial, 2009d. URL http://www.eclipse.org/gmt/epsilon/doc/articles/. Fecha de última comprobación: 2 de noviembre de 2009. 4.9

Eclipse Foundation. Graphical Modeling Framework, 2009e. URL http://eclipse.org/modeling/gmf/. Fecha de última comprobación: 2 de noviembre de 2009. 4.3

Eclipse Foundation. Eclipse Bugzilla: bug 282331 - Ant build script for generating GMF diagram editors, July 2009f. URL https://bugs.eclipse.org/bugs/show_bug.cgi?id=282331. Fecha de última comprobación: 2 de noviembre de 2009. 4.11

Eclipse Foundation. Eclipse.org home, 2009g. URL http://eclipse.org/. Fecha de última comprobación: 2 de noviembre de 2009. 4.1

Eclipse Foundation. Página principal del proyecto Model to Text (M2T), 2009h. URL http://www.eclipse.org/modeling/m2t/. 4.4, 4.7

Eclipse Foundation. Página principal del proyecto Eclipse Modeling Framework Technology (EMFT), 2009i. URL http://www.eclipse.org/modeling/emft/?project=mwe. Fecha de última comprobación: 2 de noviembre de 2009. 4.10

Eclipse Foundation. Xtext, 2009j. URL http://www.eclipse.org/Xtext/. Fecha de última comprobación: 2 de noviembre de 2009. 4.7

Antonio García Domínguez. Eclipse Community Forums: EMF → Ecore2Java: differences with the GenModel context menu?, August 2009. URL http://www.eclipse.org/forums/index.php?t=tree&th=152228&S=1c86da94a73bf76a8e1b91ed0388287a. Fecha de última comprobación: 2 de noviembre de 2009. 4.11

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163. 4.21

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Akcsit Mehmet and Satoshi Matsuoka, editors, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, Jyväskylä, Finlandia, June 1997. ISBN 978-3-540-63089-0. 4.4

Dimitrios Kolovos, Richard Paige, Louis Rose, and Fiona Polack. The Epsilon book, 2009. URL http://dev.eclipse.org/svnroot/modeling/org.eclipse.gmt.epsilon/trunk/doc/org.eclipse.epsilon.book/. Fecha de última comprobación: 2 de noviembre de 2009. 4.7, 4.27

Object Management Group. Meta-Object Facility (MOF) 2.0, June 2003. URL http://www.omg.org/spec/MOF/2.0/. Fecha de última comprobación: 2 de noviembre de 2009. 4.2

Object Management Group. Human-Usable Textual Notation (HUTN) 1.0, August 2004. URL http://www.omg.org/technology/documents/formal/hutn.htm. Fecha de última comprobación: 2 de noviembre de 2009. 4.8

Object Management Group. Object Constraint Language Specification (OCL) 2.0, May 2006. URL http://www.omg.org/technology/documents/formal/ocl.htm. Fecha de última comprobación: 2 de noviembre de 2009. 4.7

Object Management Group. XML Metadata Interchange (XMI) 2.1.1, December 2007. URL http://www.omg.org/spec/XMI/2.1.1/. Fecha de última comprobación: 2 de noviembre de 2009. 4.2

Object Management Group. Query/View/Transformation (QVT) 1.0, April 2008. URL http://www.omg.org/spec/QVT/1.0/. Fecha de última comprobación: 2 de noviembre de 2009. 4.7

Object Management Group. Unified Modeling Language (UML) 2.2, February 2009. URL http://www.omg.org/spec/UML/2.2/. Fecha de última comprobación: 2 de noviembre de 2009. 4.21

OSGi Alliance. Specifications Home Page, 2009. URL http://www.osgi.org/Specifications/HomePage. Fecha de última comprobación: 2 de noviembre de 2009. 4.1

Michael Scharf. Emfatic plugin download, October 2008. URL http://scharf.gr/eclipse/emfatic/download/. Fecha de última comprobación: 2 de noviembre de 2009. 4.6

Juan Manuel Vara Mesa, Esperanza Marcos, and María Valeria de Castro. Obteniendo modelos de sistemas de información a partir de modelos de negocios de alto nivel: un enfoque dirigido por modelos. In *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA*, pages 15–28, Sevilla, España, October 2008. 4.4

5

Caso práctico

5.1. Introducción

En esta sección se describe la aplicación de la metodología SOD-M antes extendida sobre una parte del modelo de negocio de una importante empresa del sector tabaquero. Para abreviar, de ahora en adelante se la denotará por EBE (empresa bajo estudio).

Antes de aplicar dicha metodología, se expondrá la información recopilada sobre EBE y su sistema de información actual. Posteriormente, se seleccionará un subconjunto representativo de su modelo de negocio y se desarrollarán de forma razonada sus modelos CIM, PIM y el modelo de composición de servicio extendido de nivel PSM.

Mientras que la anterior sección concluía con una evaluación a nivel técnico de las extensiones realizadas, esta sección se cerrará evaluando su aplicabilidad actual a una organización real, como metodología.

5.2. Descripción de los sistemas de información de la organización modelada

La EBE se trata de una empresa de fabricación distribuida, en la que múltiples plantas realizan distintos pasos de la elaboración del tabaco. En particular, EBE dispone de plantas para realizar el pretratado del tabaco, su liga y su empaquetado final. El pretratado (por ejemplo, la expansión de su volumen para reducir la densidad de la nicotina y el añadido de diversos aromas) suele hacerse en cajas de 180-200 kilogramos, que son posteriormente enviadas a una planta que realiza la liga y también empaqueta. El tabaco empaquetado puede entonces ser recogido por las distribuidoras mayoristas, que envían el tabaco a las distribuidoras minoristas (estancos, por ejemplo), donde el cliente final compra el tabaco que va a consumir.

EBE realiza el pretratado no sólo para producción propia, sino que además acepta pedidos de otras empresas tabaqueras. Estos pedidos externos representan hasta un 30 % de su producción mensual, reportándole unos considerables beneficios. Normalmente, tras localizar una planta de pretratado que satisfaga en un nivel aceptable sus requisitos (dado que cada planta de pretratado procesa variedades distintas de tabaco), la planta de liado y empaquetado solicita visitar las instalaciones para realizar una prueba de calidad del tabaco in situ mediante un comité de expertos. Si se supera el control de calidad de los expertos, se realiza el pedido correspondiente, que incluye las especificaciones que debe cumplir el tabaco finalmente producido en temperatura, humedad, concentrado de salsas, aromas, nivel de tostado, color, etc.

EBE forma parte de un grupo mayor de empresas. La integración de la información relevante a nivel del grupo completo se realiza utilizando el sistema ERP (Enterprise Resource Planning) SAP R/3. Éste incluye los pedidos realizados y entregados a empresas del grupo y a empresas externas, los datos de producción de cada planta, y las existencias de material.

Al mismo tiempo, cada planta mantiene de forma independiente la información que es únicamente necesaria para sus propias actividades, como el estado de la maquinaria y los informes de mantenimiento. Para evitar una dependencia excesiva en el correcto funcionamiento del sistema SAP R/3 central, algunas plantas utilizan su propio sistema de gestión de existencias, que sincronizan manualmente con las de SAP R/3 periódicamente. Esto puede suponer que las existencias reales y las listadas en el sistema central se desvíen temporalmente en algunos casos, por lo que los procesos de negocio que dependan de ello han de incluir una comprobación final de las existencias a través del responsable de la planta en cuestión.

La información específica de cada planta es mantenida utilizando una serie de aplicaciones desarrolladas internamente. En particular, en una de las plantas de EBE se dispone de cuatro sistemas con los siguientes roles:

- Sistema de supervisión visual en tiempo real del estado de la planta, al estilo de un sistema SCADA, que informa del estado actual de funcionamiento de la maquinaria y los consumos producidos, generando las gráficas apropiadas para resumir la situación.
- Sistema de generación de informes de producción periódicos, que dan información más detallada que la usualmente dada por el sistema SAP R/3.
- Sistema de gestión de recetas y control de los PLC (Programmable Logic Controllers) que regulan el funcionamiento de las distintas máquinas en la planta, y generan diversas métricas de control.
- Sistema de gestión de almacenes, que mantiene sincronizados los pedidos al almacén y los pedidos enviados al sistema SAP R/3. Se ocupa de gestionar las órdenes de entrada y salida en el almacén.

Los envíos de tabaco sin pretratar, pretratado y tabaco empaquetado son gestionados por una empresa de logística que forma parte del mismo grupo que EBE. Cada planta se mantiene en contacto con esta empresa de logística y hace el seguimiento de sus envíos a través de sus responsables de almacén y expertos en logística.

5.3. Modelos de negocio

Tras estudiar la estructura general de los sistemas de información de EBE, se seleccionó la parte de su modelo de negocio correspondiente a la provisión de tabaco pretratado y empaquetado a otras plantas y a distribuidoras mayoristas, respectivamente. Para esta versión del caso de estudio se decidió excluir los sistemas propios de cada planta, ya que no requerían integración con el resto de plantas.

5.3.1. Modelo de intercambios de valor

El primer modelo de la metodología SOD-M se trata de un modelo de intercambios de valor según la metodología e^{3}value™ Gordijn and Akkermans [2003, 2006] de Gordijn. Un modelo de intercambio de valor (descrito en más detalle en §3.3.2) representa la forma en que los participantes de cada sector de mercado, en un caso de negocio exitoso, consigue cumplir sus necesidades. A partir de un estímulo inicial de uno de los participantes, se desencadena una serie de intercambios entre los participantes que concluyen con la realización de su necesidad. La descripción de estos intercambios es únicamente a nivel del valor añadido que recibe cada una de las partes, y no entra en detalles de qué elementos físicos se intercambian o qué proceso se utiliza.

El modelo de valor para la parcela del negocio de EBE seleccionada se halla en la figura 5.1. Los participantes individuales se simbolizan a través de rectángulos con borde sencillo, mientras que los sectores de mercado se simbolizan con varios rectángulos apilados. Cada participante y sector de mercado dispone de unos nodos con flujos entrantes y salientes de valor, que simbolizan

cada uno de los intercambios necesarios para llevar a cabo sus actividades, simbolizadas como elipses. Estos intercambios se disparan a partir de un estímulo inicial generado por el deseo de los clientes de consumir tabaco, representado por un círculo con borde sencillo en su interior, y concluyen en una serie de estímulos finales (círculos con borde doble) en la empresa de logística y el sector de los productores del tabaco.

Los intercambios modelados son los siguientes:

- Los clientes obtienen el tabaco empaquetado que desean pagando con dinero en última instancia a las plantas productoras del tabaco. Como es común en los diagramas de intercambios de valor, no se representan todos los detalles de cómo se produce físicamente el intercambio. En particular, no se han modelado los intercambios entre las plantas ligadoras y empaquetadoras y las distribuidoras mayoristas y minoristas.
- Las plantas ligadoras y empaquetadoras consiguen el tabaco pretratado de las plantas de pretratado a cambio de dinero, y realizan sus envíos a cambio de dinero mediante la empresa de logística del grupo al que pertenece EBE.
- Las plantas de pretratado obtienen el tabaco sin tratar de los agricultores a cambio de dinero.

5.3.2. Modelo de proceso de negocio

Mientras que el anterior modelo describía la forma en que se beneficiaba cada participante en el caso de negocio de EBE, sin entrar en qué pasos seguían ni qué objetos físicos se manejaban, el modelo de proceso de negocio descrito por la metodología SOD-M como un diagrama de actividad UML se centra en los pasos seguidos en una ejecución completa del proceso de negocio. Los modelos de proceso de negocio no deben confundirse con los modelos de proceso de servicio: operan a ámbitos y niveles de detalle muy distintos.

Un modelo de proceso de negocio describe el funcionamiento de un caso de negocio completo (por ejemplo, «vender tabaco a un cliente final»), mientras que un proceso de servicio atiende únicamente a uno de los servicios de negocio que ha de posibilitar el sistema de información de la organización para que pueda llevarse a cabo el proceso de negocio anterior (por ejemplo, «vender tabaco pretratado»). Los modelos de proceso de negocio se describen en más detalle en §3.3.2.

El modelo de proceso de negocio de la parcela de EBE bajo estudio se halla en la figura 5.2. Puede verse cómo, a partir de un nodo inicial (simbolizado por un círculo sólido negro), a medida que se avanza en el proceso, se llevan a cabo los intercambios antes mencionados, además de otras acciones, como controles de calidad o consulta de existencias. Ciertos flujos son de tipo condicional, como el paso de «Probar tabaco in situ» a «Pretratar tabaco», que depende de que la prueba haya sido superada. Otros flujos se realizan de forma paralela, como el pago del pedido y el envío del tabaco pretratado, por ejemplo. Estas restricciones sobre el orden de las distintas actividades y cómo pueden solaparse serán importantes posteriormente a la hora de elaborar los modelos de proceso de servicio individuales.

5.4. Modelos de casos de uso

Tras describir la parcela del negocio de EBE a tratar usando los modelos de SOD-M de la perspectiva del negocio en el nivel CIM, se ha de pasar a especificar qué funcionalidades de

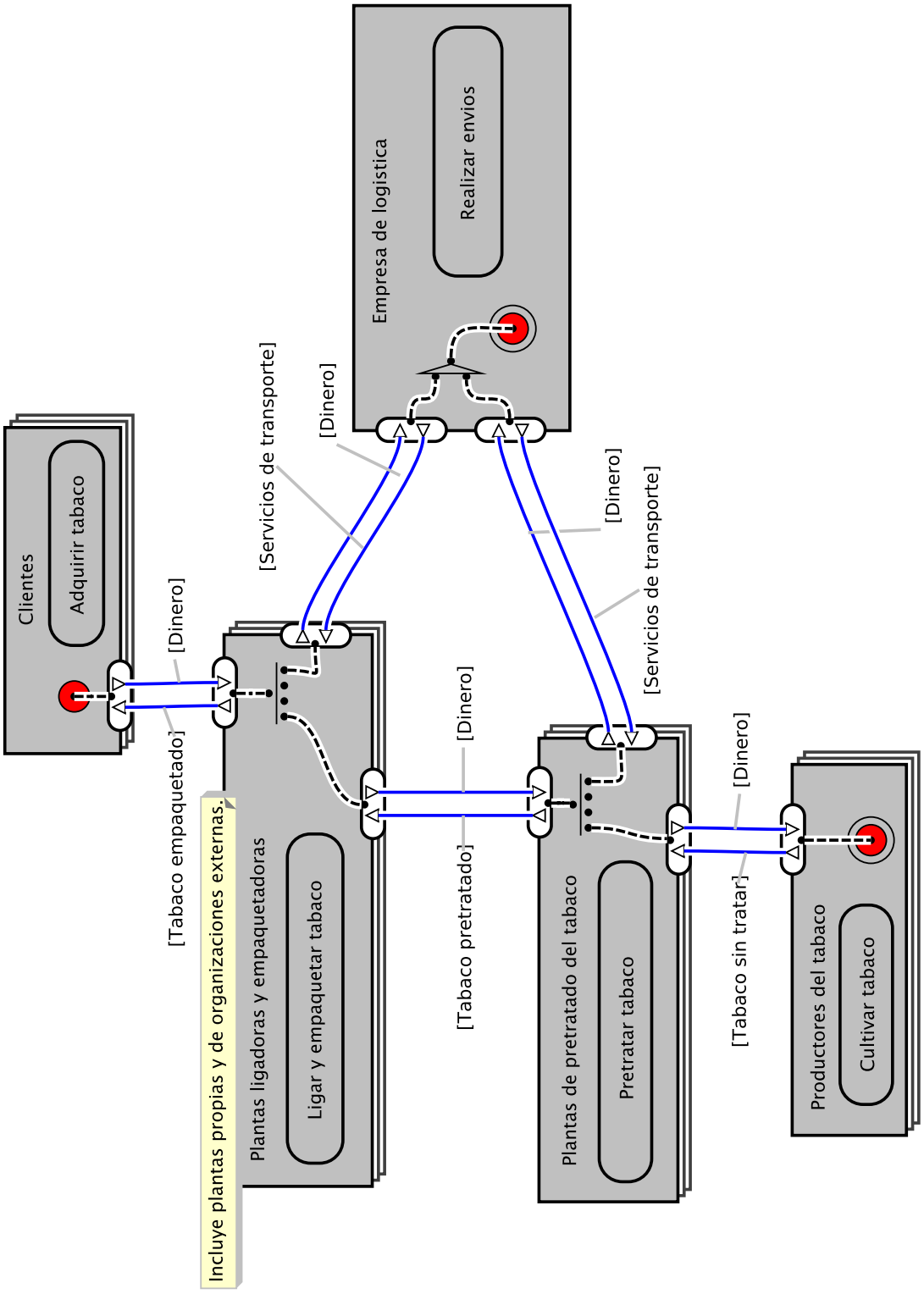


Figura 5.1. Modelo de intercambios de valor en EBE

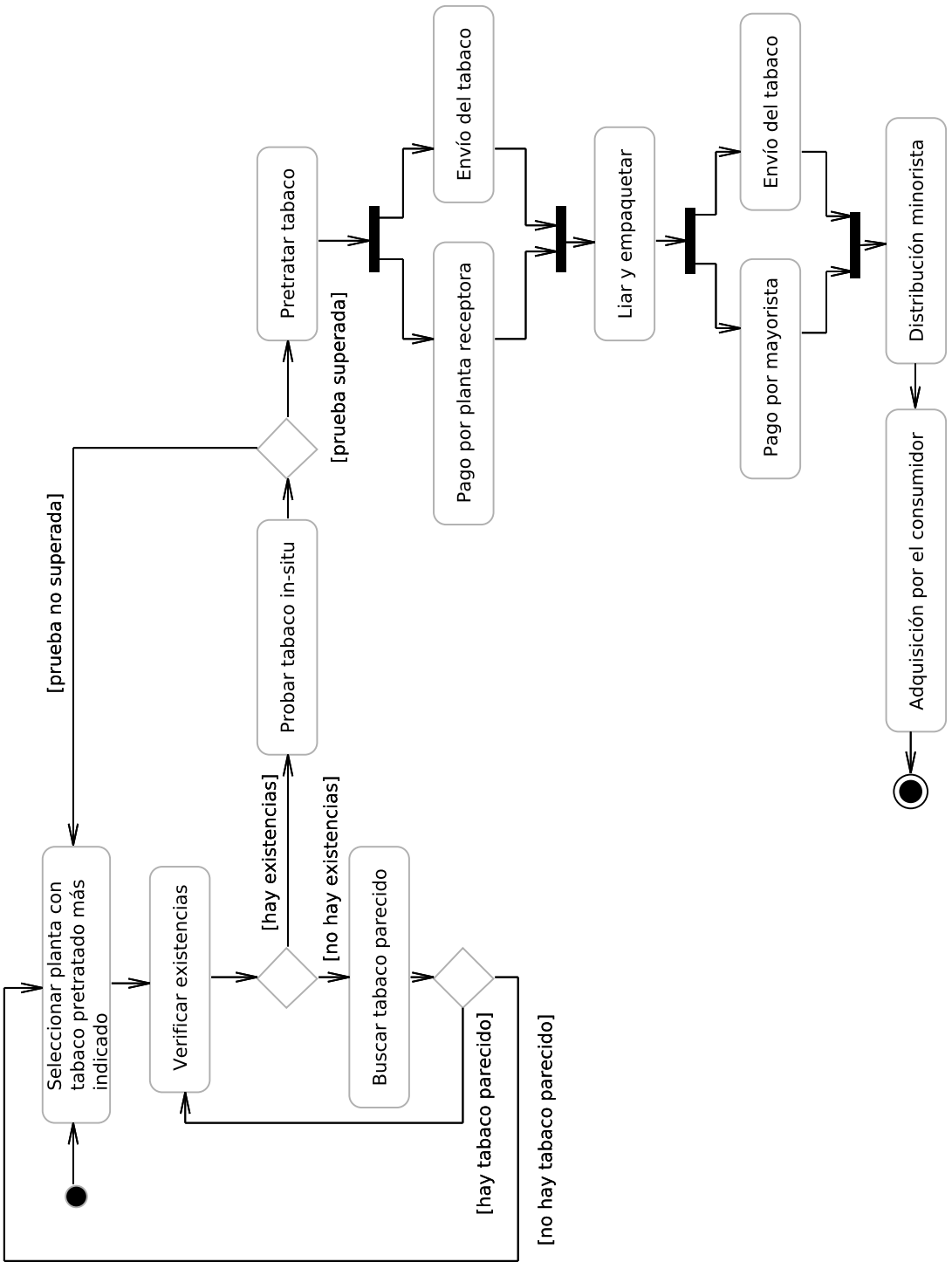


Figura 5.2. Modelo de proceso de negocio de la parcela seleccionada de EBE

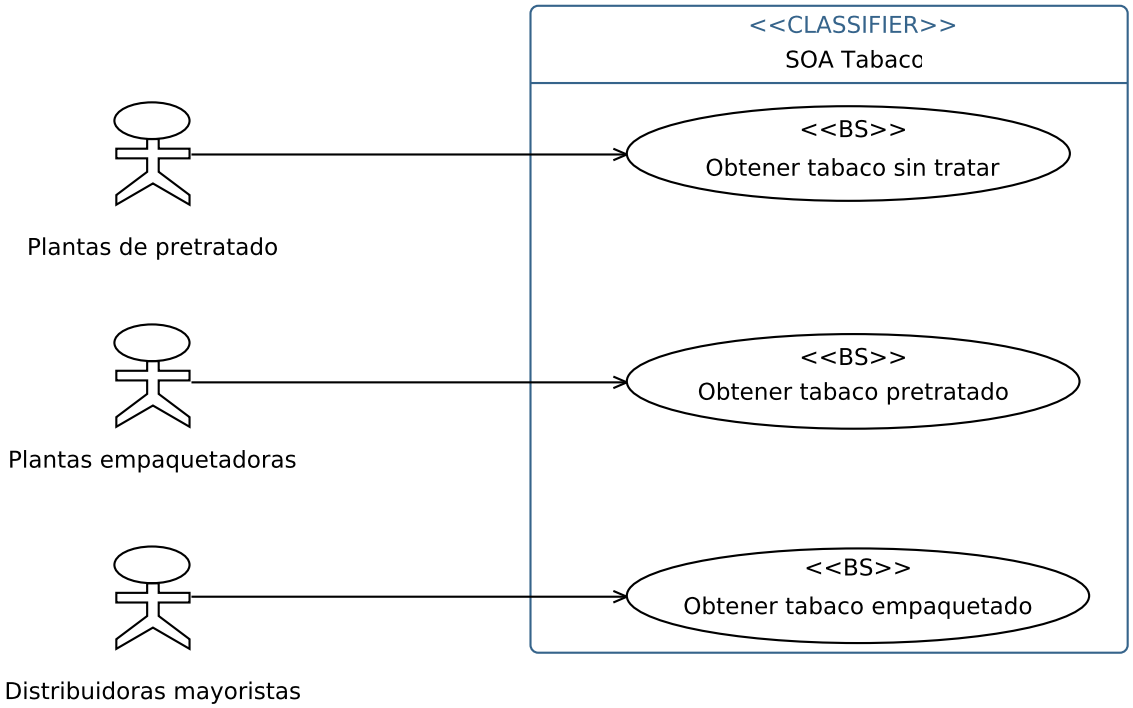


Figura 5.3. Modelos de casos de uso para EBE

alto nivel ofrecerá la arquitectura SOA a desarrollar. Para ello, se comienza con un modelo de casos de uso UML (descrito en 3.3.3) que lista los escenarios que deberá atender. El modelo de casos de uso para el presente sistema se halla en la figura 5.3.

Cada uno de los tres casos de uso o «servicios de negocio» (elipses con el estereotipo `<<BS>>`) que la SOA para la empresa tabaquera (representado con un rectángulo con el estereotipo `<<classifier>>`) implementa es iniciado por un consumidor final del servicio prestado. En última instancia, el sistema ha de posibilitar los intercambios de valor descritos en §5.3.1, siguiendo el modelo de proceso de negocio antes descrito. Es importante notar que no aparecen como consumidores finales ni los agricultores, ni las distribuidoras minoristas ni los propios consumidores finales. Estos participantes no harán uso directo del sistema.

Estos casos de uso son «básicos», pues aún son de demasiado alto nivel como para ser implementados directamente. Hay que descomponerlos en casos de uso más sencillos, que sean reutilizables en varios puntos del sistema y se puedan implementar más fácilmente. Estos casos de uso se conocen como «estructurales», y forman parte de cada uno de los modelos de casos de uso extendidos, descritos en más detalle en 3.3.3 y mostrados en las figuras 5.4 a 5.6.

Algunos son incluidos como prerrequisitos de otros (señalados con enlaces `<<include>>`), y otros añaden acciones al final de un caso de uso de nivel superior (enlaces `<<extend>>`). Esta estructuración sirve también, por lo tanto, para establecer los primeros requisitos sobre la secuenciación de las actividades.

En los modelos de casos de uso extendidos existen una serie de casos de uso comunes, como «Realizar pedido» o «Realizar pago». Además, en dos de los modelos aparecen búsquedas exactas y aproximadas, por lo que podría pensarse en su posterior implementación sobre una base común.

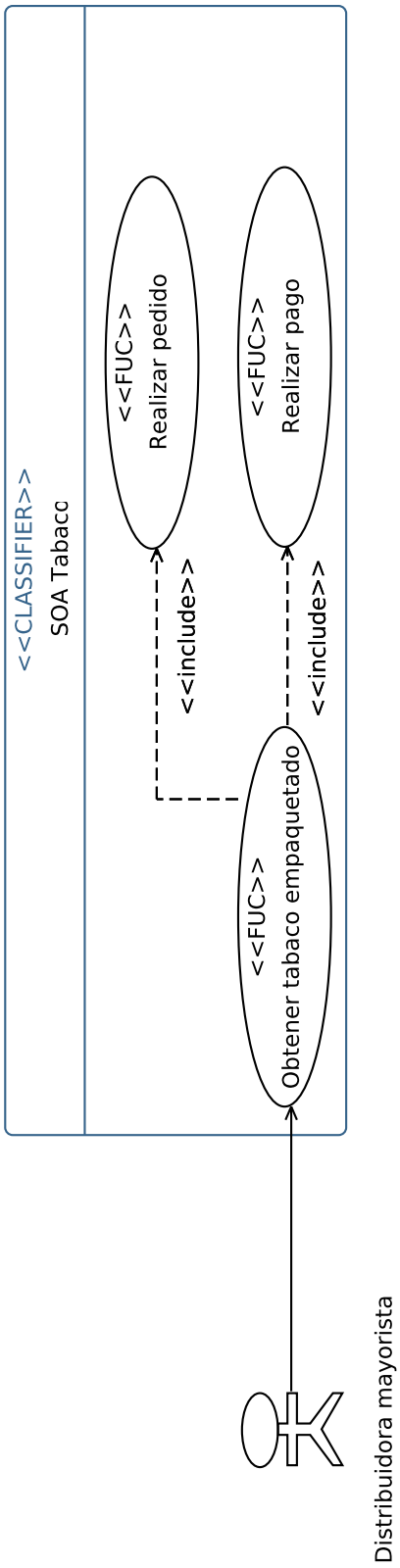


Figura 5.4. Modelo de casos de uso extendido «Obtener tabaco empaquetado»

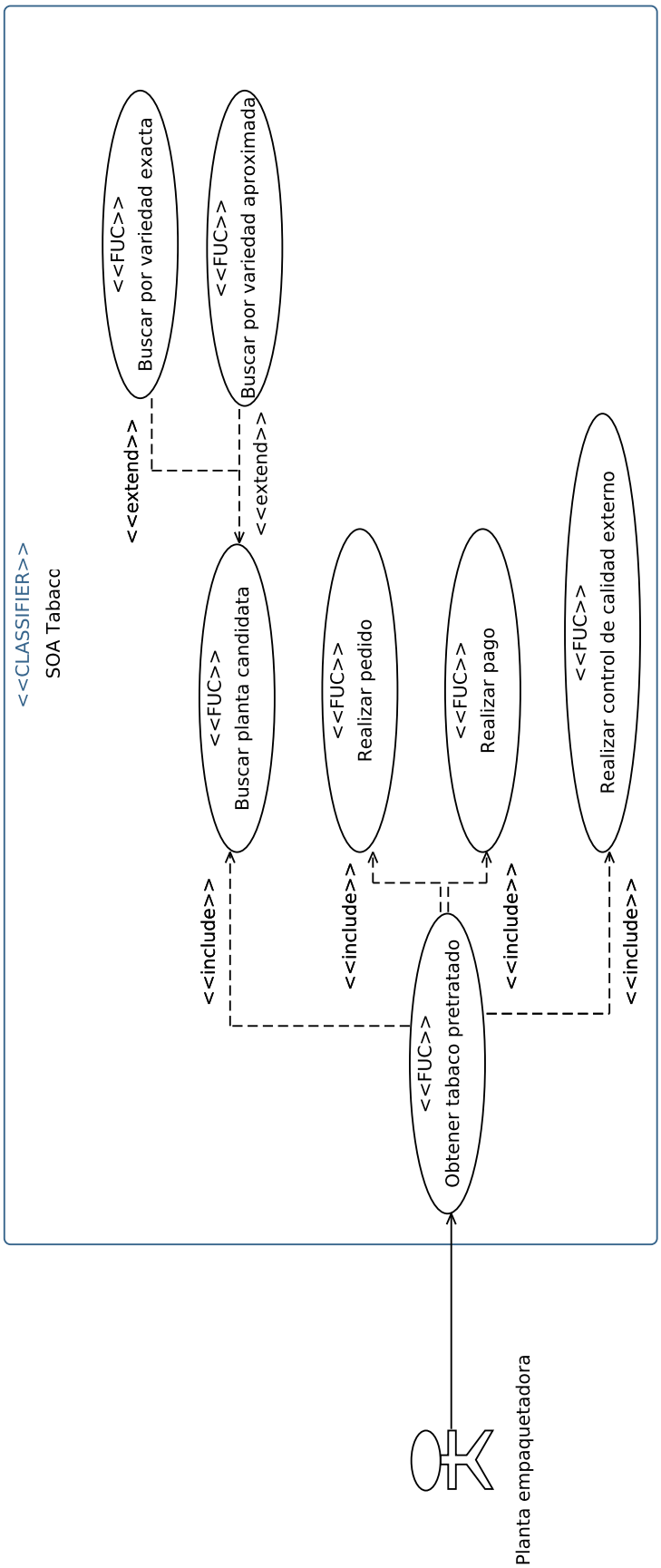


Figura 5.5. Modelo de casos de uso extendido «Obtener tabaco pretratado»

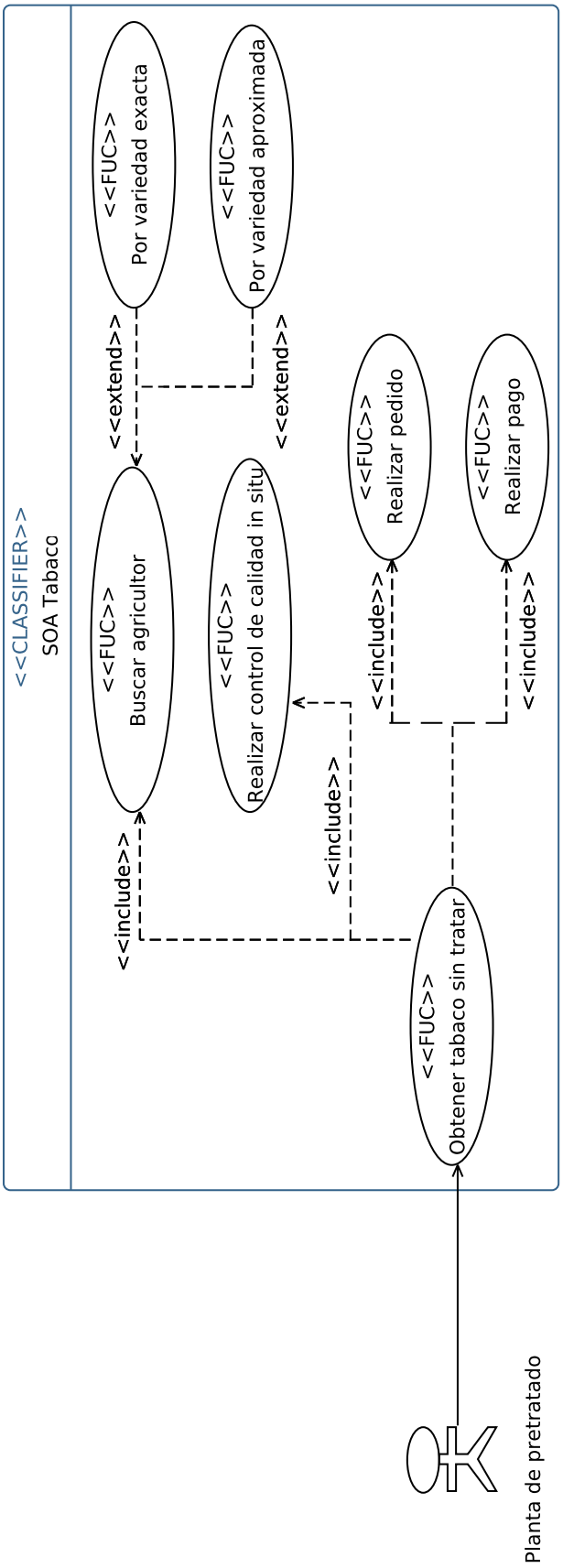


Figura 5.6. Modelo de casos de uso extendido «Obtener tabaco sin tratar»

5.5. Modelos de proceso de servicio

En los modelos de caso de uso extendido se describieron las distintas funcionalidades de alto nivel que debía implementar el sistema, y se especificaron cómo se relacionaban entre ellas. Algunas actividades incluían a otras como prerrequisitos (mediante enlaces <<include>>), y algunas extendían a otras añadiendo ciertas acciones tras su ejecución (mediante enlaces <<extend>>).

En los modelos de proceso de servicio (descritos en más detalle en §3.3.3 y modificados en §4.4.1), se toman los casos de uso básicos y estructurados de los modelos de casos de uso extendidos y se completa su secuenciación, ya que en los modelos anteriores sólo estaba definida de forma muy general. Si un caso de uso incluía a otros dos al mismo nivel, no se podía determinar cuál de esos dos debería venir antes, o si incluso deberían ejecutarse en paralelo, aunque antes que el caso de uso principal. De la misma forma, si un caso de uso era extendido por dos o más casos de uso de menor nivel, no era posible saber de antemano si deberían ir en un determinado orden tras el caso de uso de nivel superior, si se harían en paralelo, o si serían excluyentes entre sí.

Es a partir de los modelos de procesos de servicio cuando se pueden aplicar las extensiones realizadas sobre la metodología SOD-M. Los tres modelos de proceso de servicio correspondientes a los tres modelos de casos de uso extendidos de la sección anterior se incluyen en las figuras 5.7 a 5.9. Se ha retocado la secuenciación originalmente derivada a través de las indicaciones del párrafo anterior, permitiendo que en «Obtener tabaco sin tratar» se obtenga el tabaco y se pague a la vez.

Puede verse cómo cada actividad de servicio se halla anotada con una restricción acerca de su rendimiento, estimada de manera automática a partir de la estructura del grafo, de las restricciones globales impuestas manualmente y de la estimación de las probabilidades de cada rama condicional. La herramienta extendida no ha hallado problemas a la hora de tratar las múltiples ramificaciones condicionales y paralelas que aparecen en los modelos «Obtener tabaco sin tratar» y «Obtener tabaco empaquetado».

En este TFM, las estimaciones de número de usuarios concurrentes, tiempo límite y probabilidades se han basado en la experiencia con el manejo de otros sistemas, más que en datos fiables estimados a partir de la práctica. De todos modos, la validación de esta metodología no depende de sus valores concretos, siempre que se propaguen correctamente.

Se debe recalcar que el tiempo límite (120 segundos en el caso de «Obtener tabaco sin tratar») no quiere decir que se reciba el tabaco en 120 segundos, sino que en total el sistema le debería dedicar 120 segundos de tiempo de procesamiento, contando los accesos a la base de datos, las conexiones realizadas, el acceso a otros dispositivos y el procesamiento interno requerido, entre otros aspectos.

5.6. Modelos de composición de servicio

Los modelos de proceso de servicio incluyen toda la información necesaria acerca del orden en que deben de realizarse las actividades de servicio, y cómo se solapa su ejecución. Sin embargo, estas actividades de servicio siguen siendo demasiado abstractas como para ser implementadas, y no modelan los mensajes intercambiados entre las distintas partes ni cómo se reparte la funcionalidad requerida entre los distintos participantes.

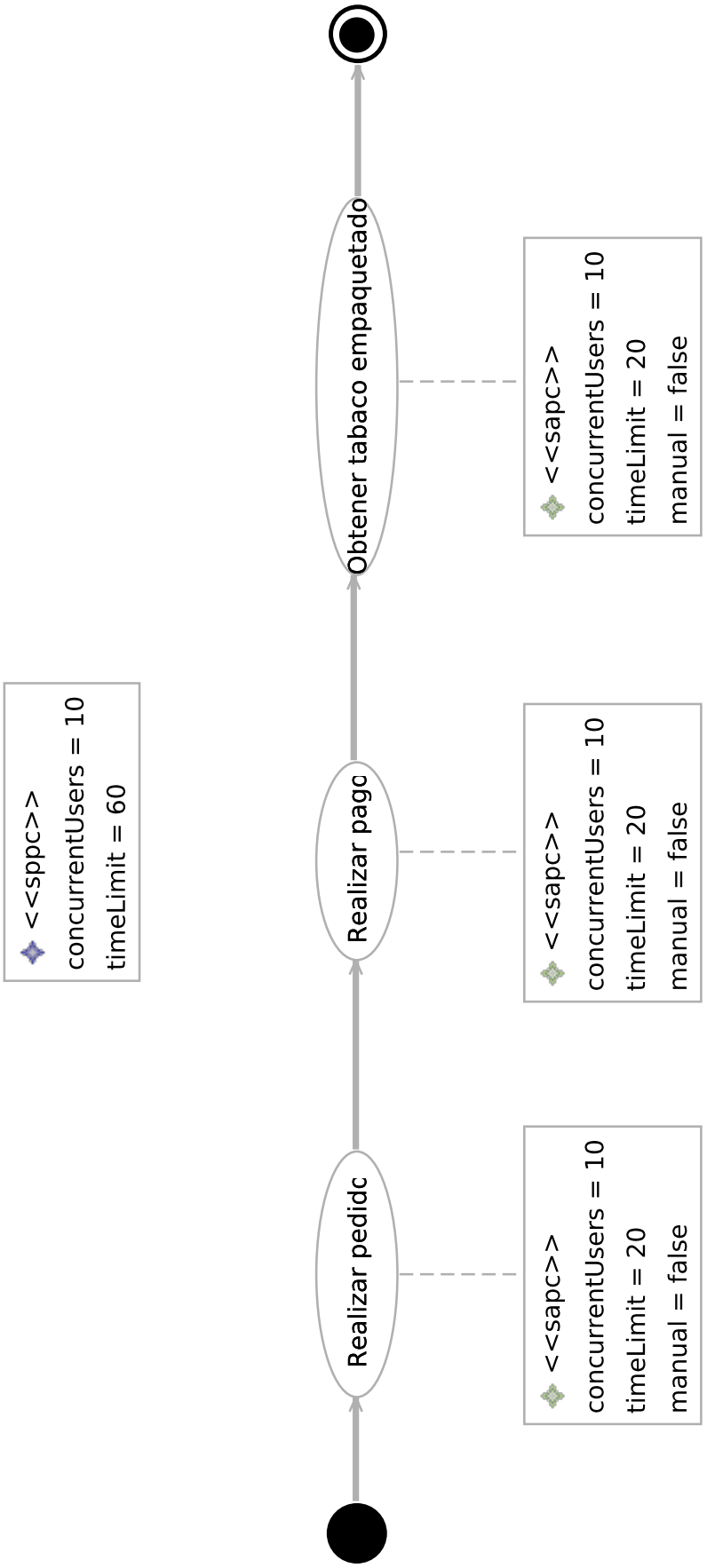


Figura 5.7. Modelo de proceso de servicio «Obtener tabaco empaquetado»

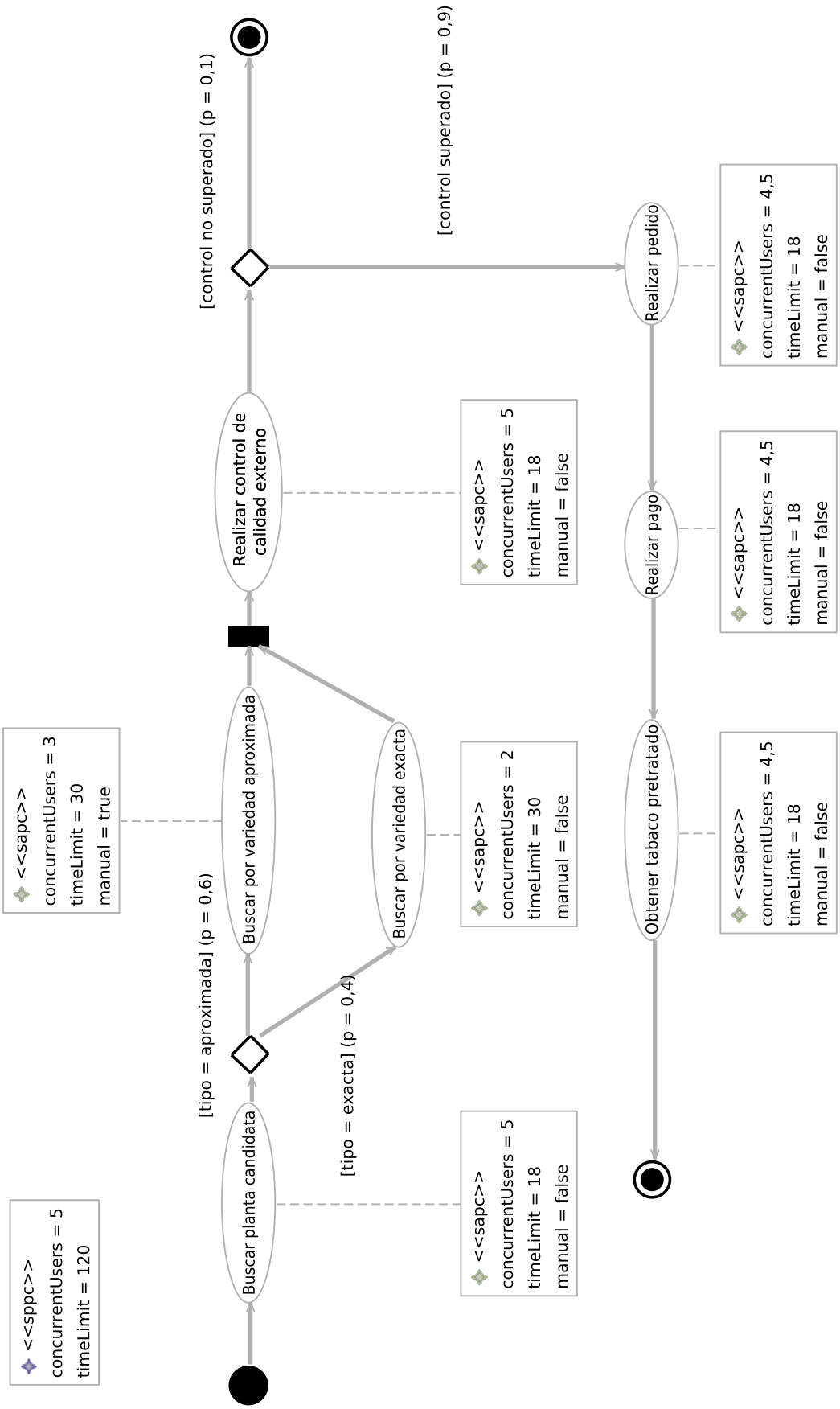


Figura 5.8. Modelo de proceso de servicio «Obtener tabaco pretratado»

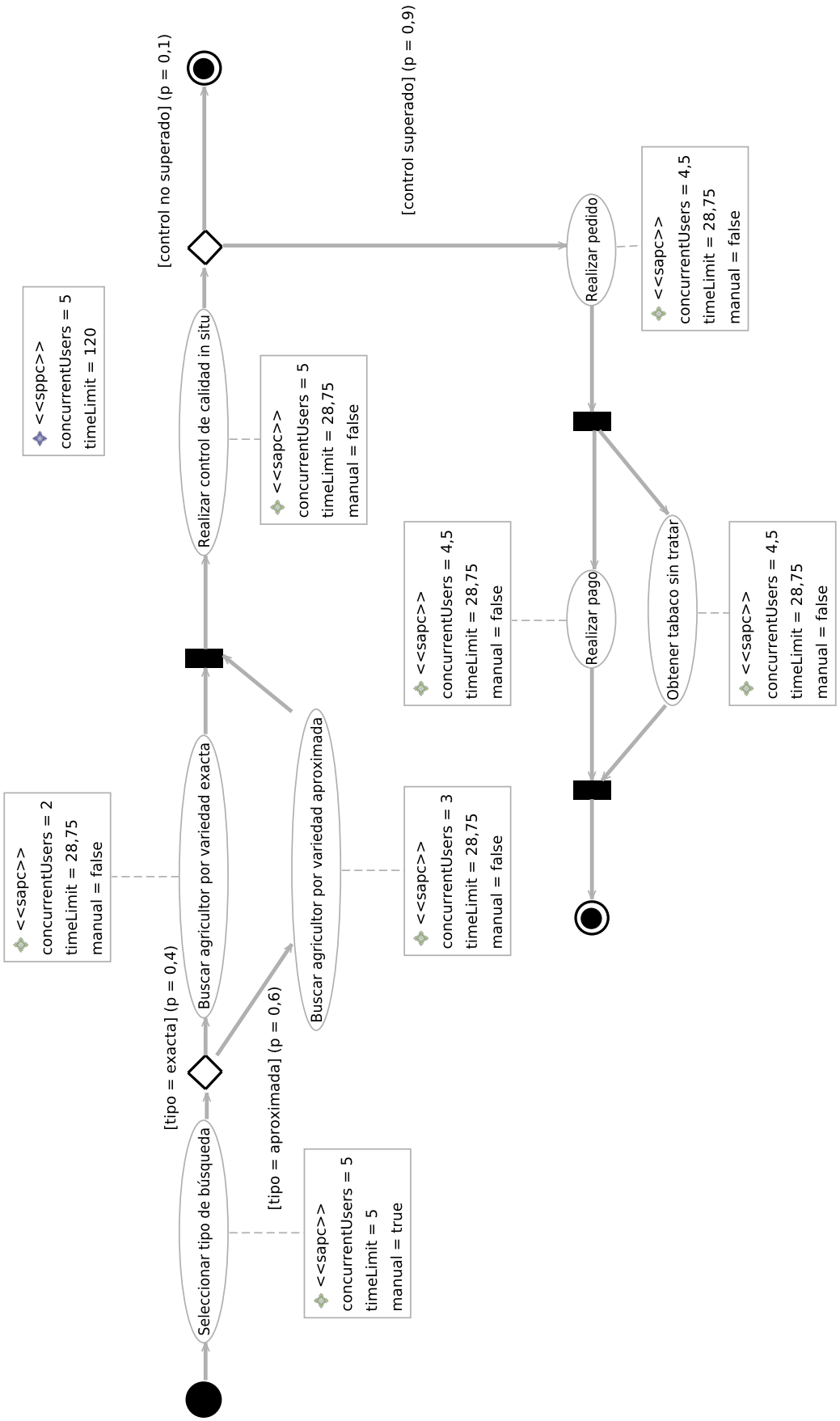


Figura 5.9. Modelo de proceso de servicio «Obtener tabaco sin tratar»

Éste es el papel de los modelos de composición de servicio, ya descritos en §3.3.3 y modificados en §4.4.1. Por restricciones de tiempo, se ha elaborado únicamente el modelo de composición de servicio completo para «Obtener tabaco sin tratar», incluyendo los mensajes a intercambiar. Para ello, se utilizó el resultado de la transformación ETL antes implementada como punto de partida, refinándose hasta conseguir el modelo final.

Por limitaciones de espacio, no es posible describir en una sola figura el modelo de composición de servicio. La estructura de alto nivel es la misma que la del modelo de proceso de servicio (véase la figura 5.9), aunque cada actividad de servicio ha sido ampliada con un grafo anidado que detalla las acciones concretas a realizar por cada participante y los mensajes intercambiados entre ellos. Las actividades de servicio ampliadas se hallan en las figuras 5.10 a 5.16.

Al realizar la descomposición de las actividades de servicio en acciones concretas, aparecen nuevos aspectos a considerar, como la necesidad de iniciar sesión y autenticar al usuario para que realice la búsqueda, gestione la visita al agricultor, realice el pedido o lleve a cabo el pago. La arquitectura SOA implementa la mayor parte de la funcionalidad necesaria, pero también ha de interactuar con otros sistemas, como el del agricultor (para obtener un listado de fechas factibles), el de la entidad financiera (para gestionar la realización del pago del tabaco sin tratar del agricultor), o el de la empresa de logística (para conseguir una estimación de la fechas y solicitar la recogida de la mercancía).

El algoritmo de estimación de requisitos ha sido ejecutado de nuevo, pero esta vez en cada uno de los grafos anidados en cada actividad de servicio. De esta forma, se tienen restricciones para todas las acciones a realizar por el sistema y el resto de participantes. En el caso de las acciones de sistemas externos, esto puede dar una idea de con qué nivel de servicio se está contando a la hora de integrarlos en la arquitectura SOA propia. Es posible que, si el nivel de servicio sobre el entorno externo no puede alcanzar los valores estimados, haya que revisar la estimación global o restringir las estimaciones locales para el resto de acciones. En este TFM no ha sido necesario, al no contar con datos de tiempos de sistemas reales, pero en la práctica podría darse tal situación.

El contenido de cada uno de los tipos de mensajes procesados (simbolizados a través de sus *ObjectNode*) se halla descrito textualmente en la tabla 5.1. Esta descripción textual sería utilizada posteriormente en combinación con las acciones a implementar como servicios web para elaborar sus descripciones formales en lenguajes como el Web Service Description Language (WSDL) World Wide Web Consortium [2007], utilizado en la metodología SOD-M para los modelos PSM de interfaces de servicios web.

Se ha identificado la necesidad de definir en un futuro de forma más clara la semántica de los flujos de objeto, y cómo se propaga la ejecución del proceso de servicio a través de ellos. En particular, no queda claro cómo se trata el caso en que un mismo *ObjectNode* se envía a varias acciones o es enviado desde varias acciones. Habría sido útil realizar el envío de las confirmaciones de pago desde la entidad financiera de forma que fueran recibidas simultáneamente por la planta de pretratado y el agricultor, pero dado que estos aspectos no estaban bien definidos, se optó por que el agricultor reenviara la confirmación a la planta de pretratado.

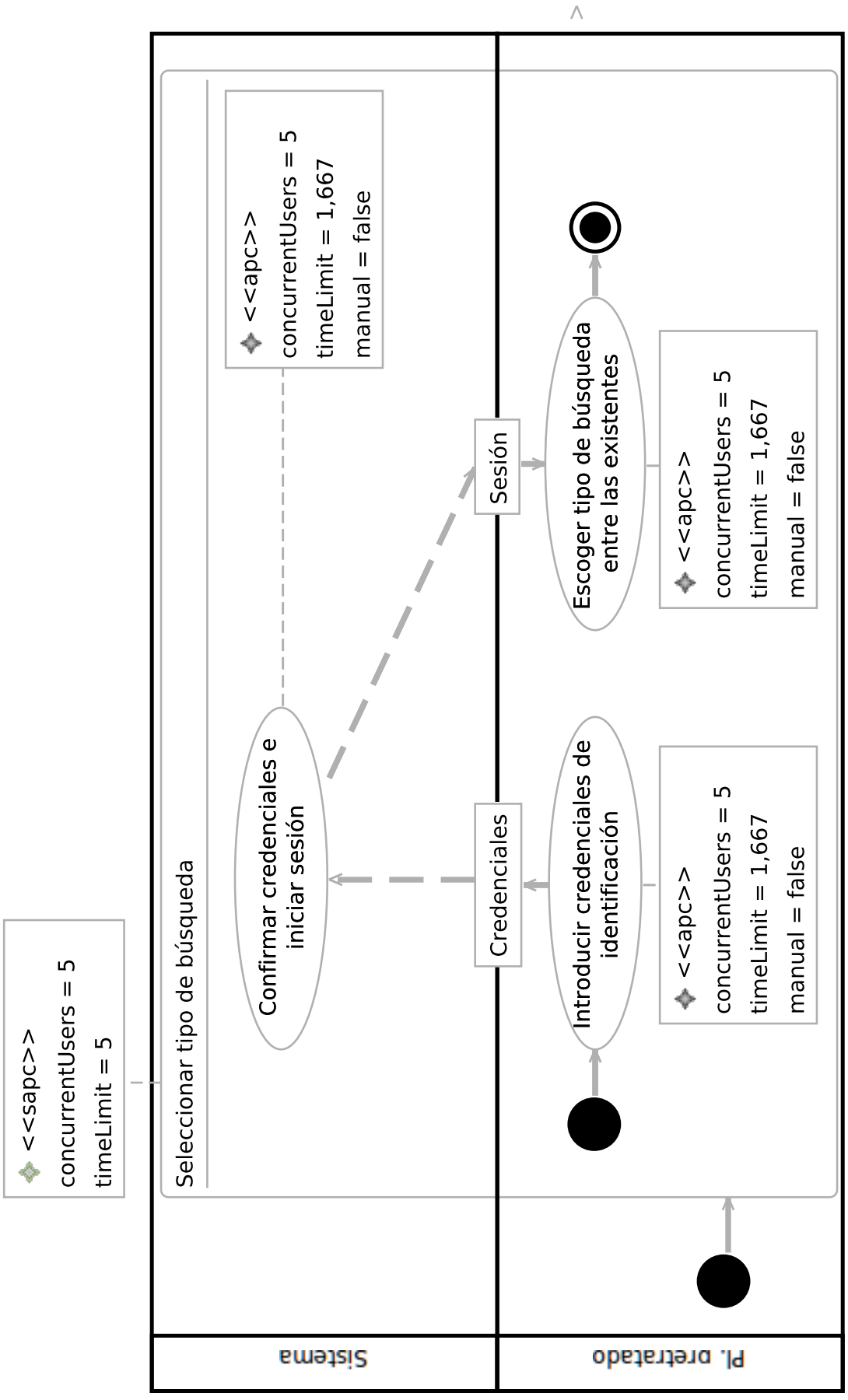


Figura 5.10. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio « Seleccionar tipo de búsqueda»

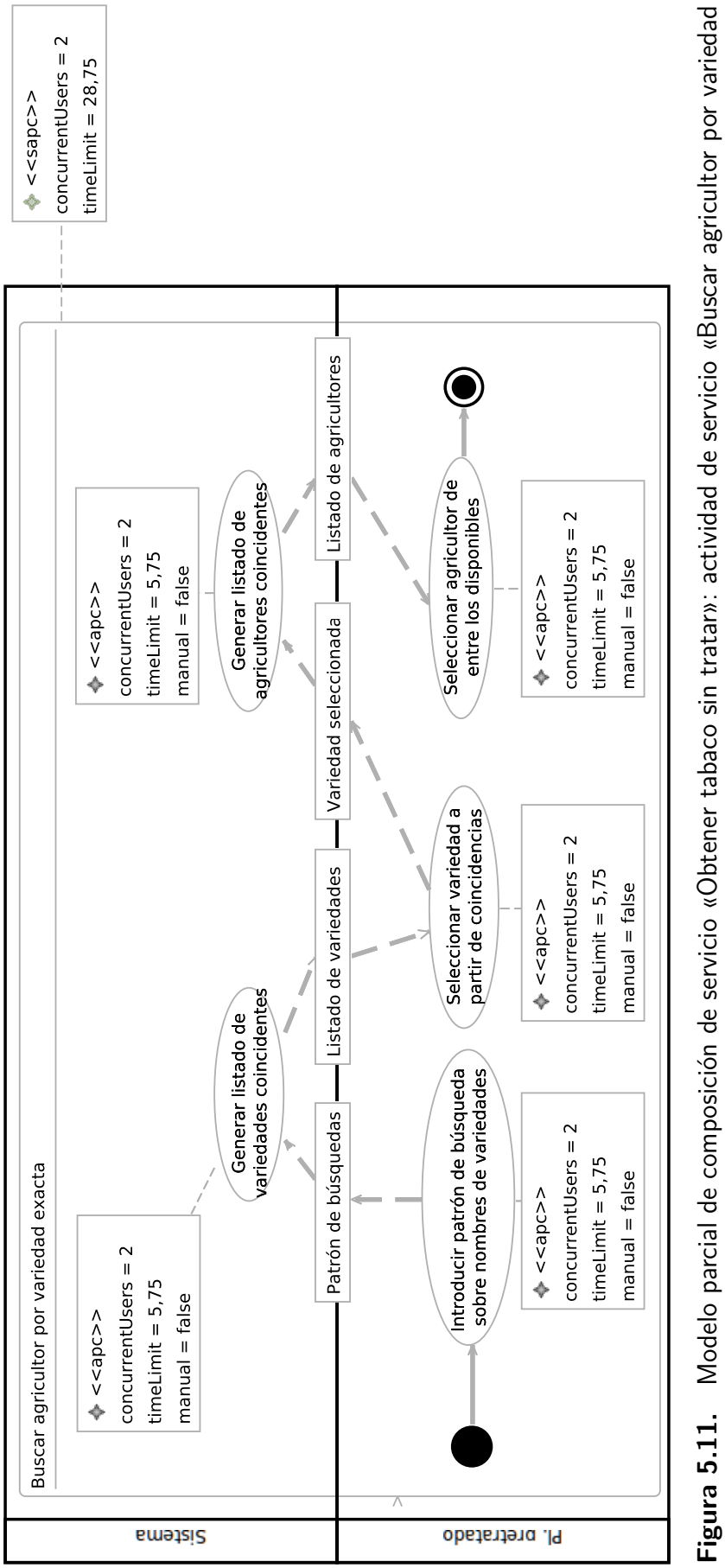


Figura 5.11. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Buscar agricultor por variedad exacta»

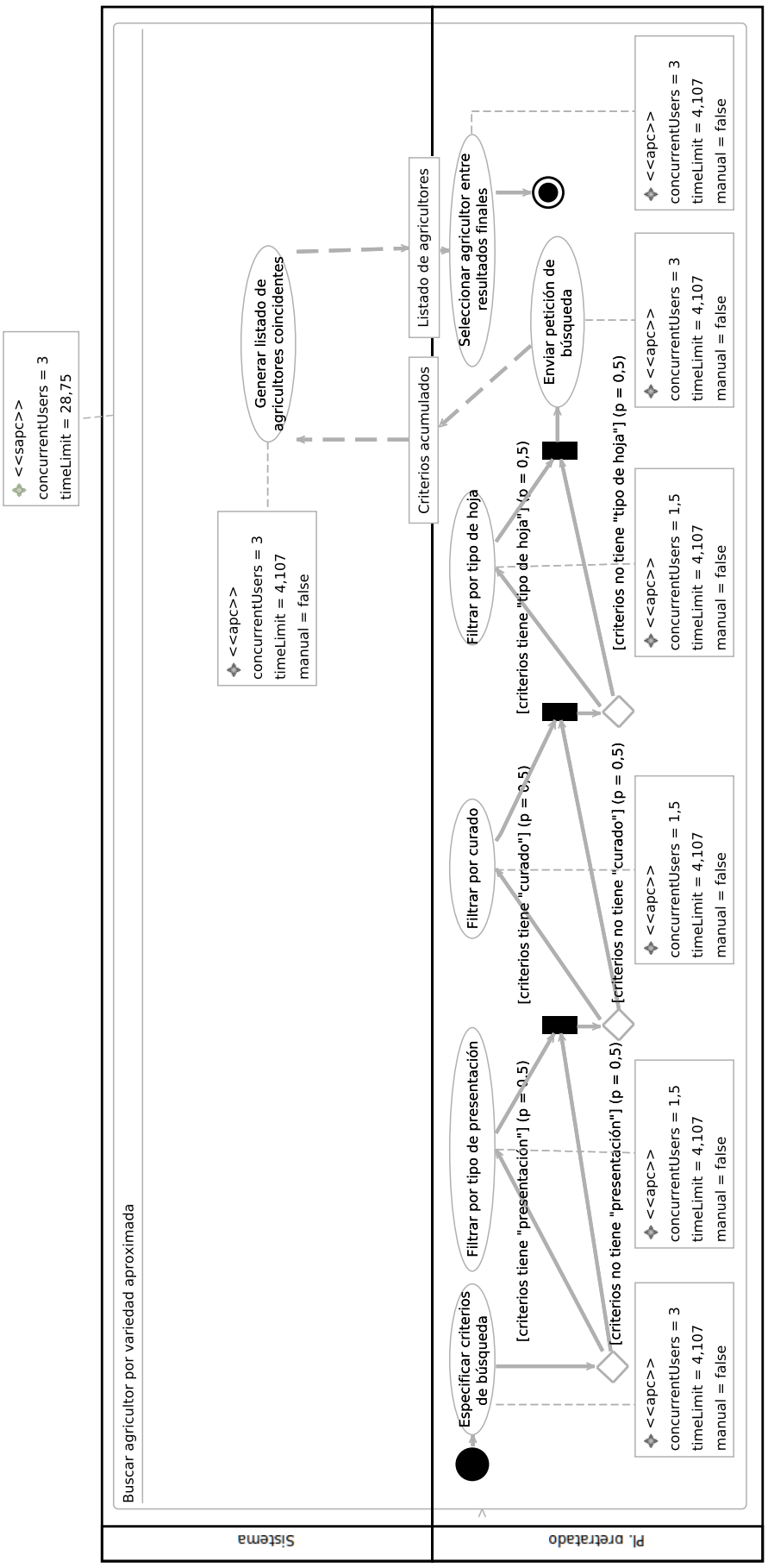


Figura 5.12. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Buscar agricultor por variedad aproximada»

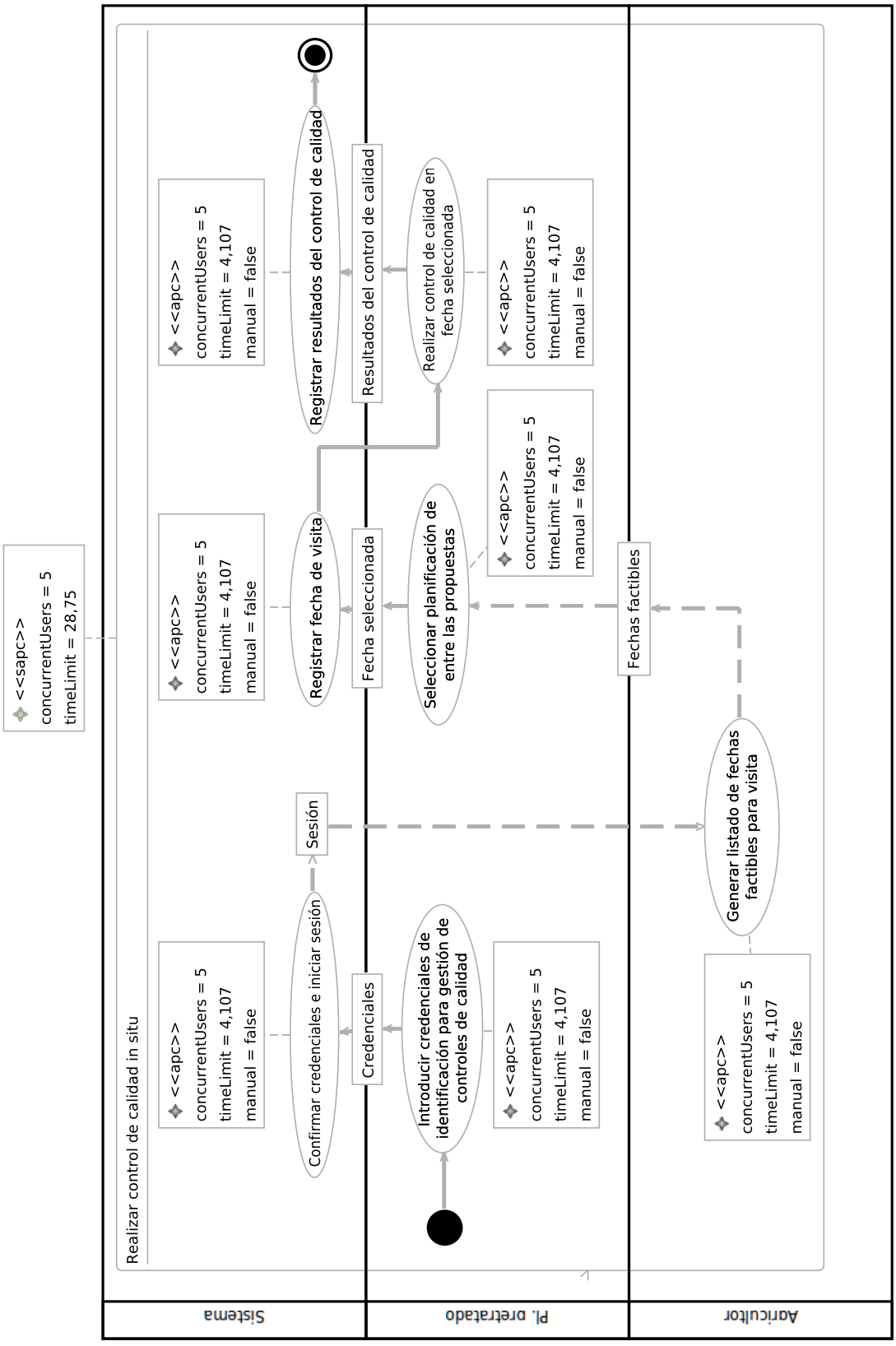


Figura 5.13. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Realizar control de calidad in situ»

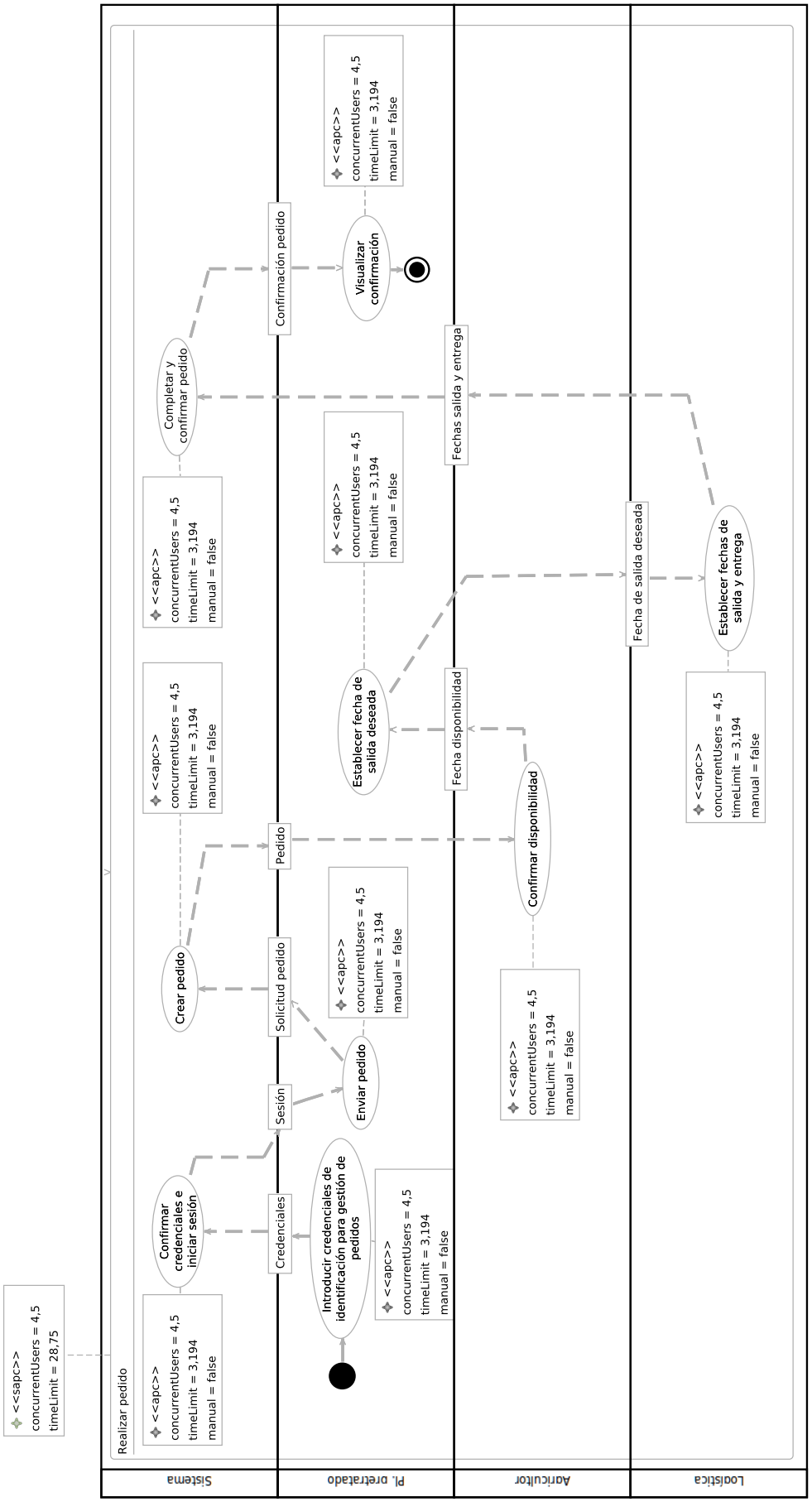


Figura 5.14. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Realizar pedido»

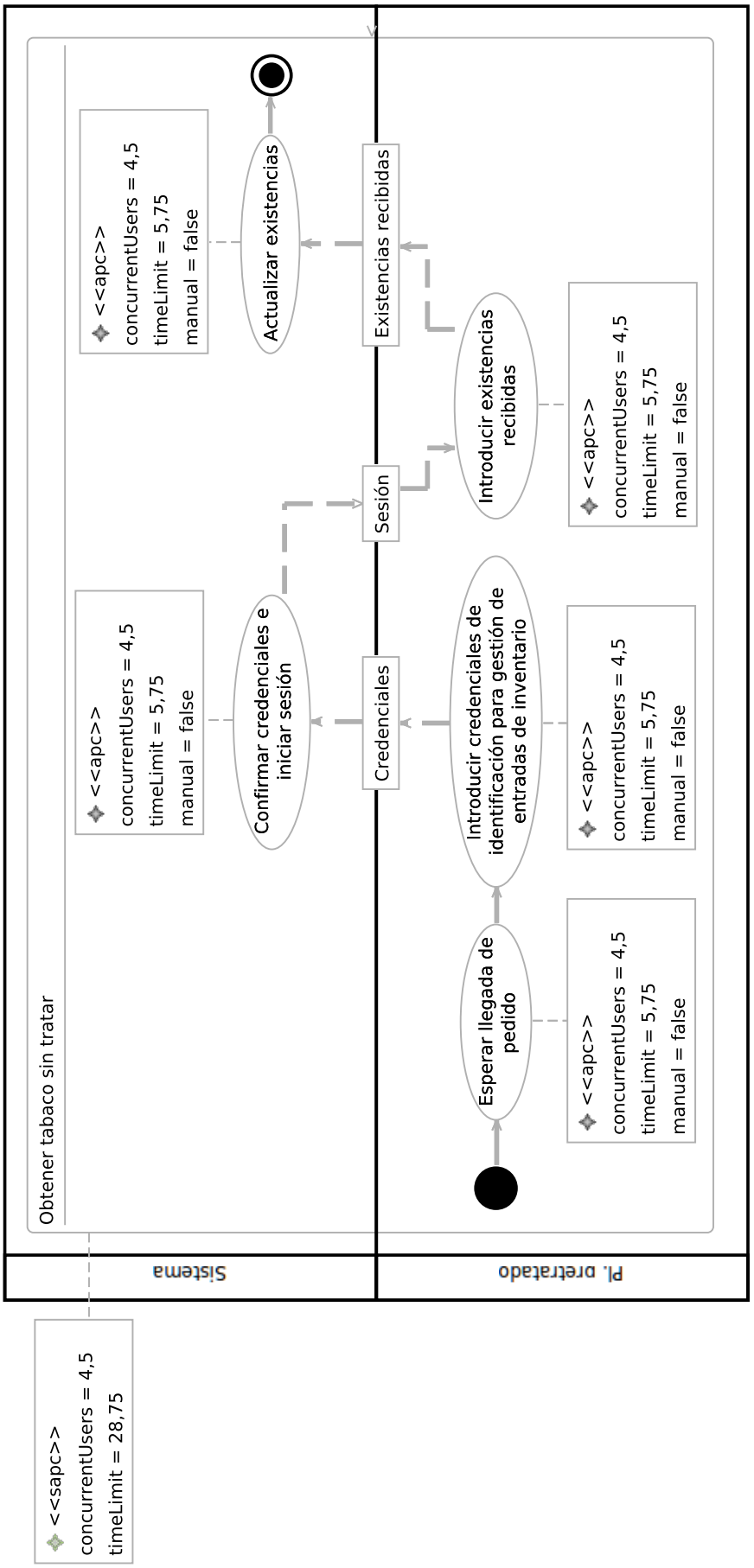


Figura 5.15. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Obtener tabaco sin tratar»

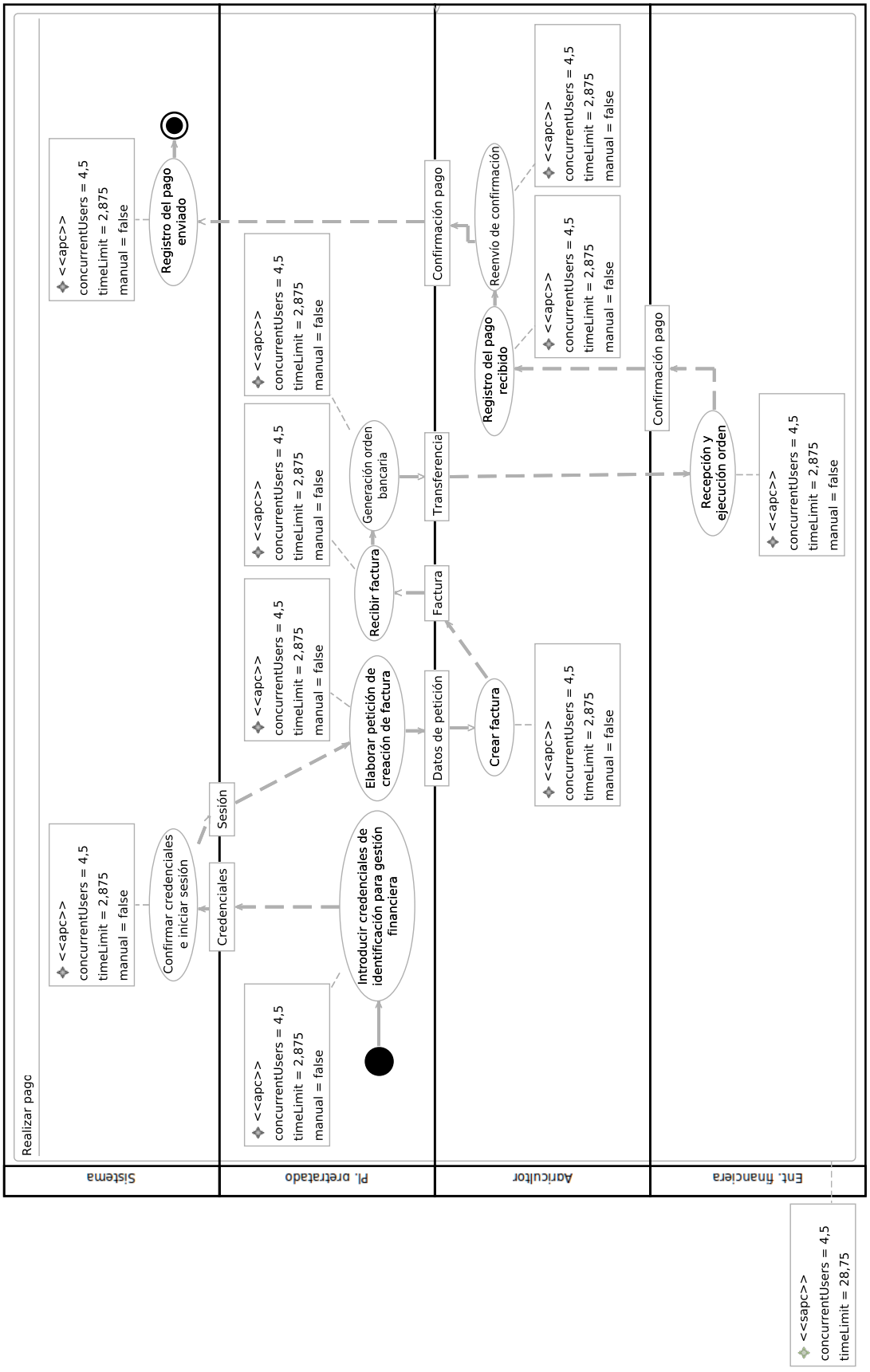


Figura 5.16. Modelo parcial de composición de servicio «Obtener tabaco sin tratar»: actividad de servicio «Realizar pago»

Tabla 5.1. Contenidos de los mensajes intercambiados en la composición de servicio «Obtener tabaco sin tratar»

Nombre del mensaje	Contenidos y requisitos de transmisión
Confirmación del pago	Confirmación de la realización del envío de una cantidad de dinero entre dos cuentas bancarias. Esta confirmación es enviada por la entidad financiera, como intermediario neutral de la transacción, y variará según el diseño de sus sistemas de información.
Confirmación del pedido	Repetición de la información del pedido, indicando que todo está conforme y se puede continuar con el proceso.
Credenciales	Nombre de usuario y contraseña con el que se intenta iniciar sesión para realizar una determinada acción. Este mensaje debería estar cifrado o, como mínimo, debería enviarse solamente un resumen criptográfico de la contraseña.
Criterios acumulados	Conjunto de pares nombre-valor con el nombre del criterio de búsqueda de variedades y los valores aceptados para dicho criterio. Es posible combinar varios criterios para realizar una búsqueda más estricta, o no especificar ninguno para conseguir un listado completo.
Datos de petición	Solicitud al agricultor de la generación de una factura para el suministro de la cantidad señalada de una determinada variedad de tabaco. Incluye un identificador de correlación del pedido.
Existencias recibidas	Mensaje con la cantidad finalmente recibida de la variedad indicada, con la fecha y hora de salida y entrega reales.
Factura	Mensaje con los conceptos por los que se efectuará el cobro de las cantidades monetarias indicadas, desglosando el IVA y detallando los datos fiscales del agricultor, junto con el resto de información requerida por el marco legal vigente.
Fecha de disponibilidad	Fecha a partir de la cual se podrá recoger el tabaco sin tratar solicitado.
Fecha de salida deseada	Fecha en la que la planta de pretratado del tabaco desea que se recoja el tabaco sin tratar solicitado. El mensaje incluye además las direcciones de recogida y destino, para asistir a la empresa de logística.
Fecha seleccionada	Fecha en la que se solicita realizar el control de calidad in situ.

Continúa en la siguiente página

Continuado desde la página anterior

Nombre del mensaje	Contenidos y requisitos de transmisión
Fechas de salida y entrega	Fechas finales de recogida del tabaco sin tratar y de su llegada a la planta de pretratado, confirmadas por la empresa de logística.
Fechas factibles	Conjunto de fechas en las que podría realizarse el control de calidad in situ.
Listado de agricultores	Conjunto de identificadores unívocos de los agricultores que fabrican alguna variedad que cumpla los criterios de la búsqueda (si se usa una búsqueda aproximada) o que fabriquen exactamente la variedad proporcionada (si se usa una búsqueda exacta).
Listado de variedades	Conjunto de identificadores unívocos de las variedades que cumplen el patrón de búsqueda de variedad.
Patrón de búsqueda (de variedad)	Cadena que se usará para filtrar las variedades existentes. Una variedad coincidirá con dicho patrón si su nombre contiene este patrón de búsqueda.
Pedido	Solicitud del pedido ampliada con la información necesaria para llevar un control de los pedidos: identificador de pedido, fecha y hora de su envío y responsable.
Resultados del control de calidad	Registro con los valores esperados y los valores obtenidos del control de calidad, junto con su fecha, hora y responsable. Se indicará además si ha sido exitoso o no.
Sesión	Identificador único pseudoaleatorio conocido solamente por el sistema y el agente que ha iniciado sesión, que permite a dicho agente demostrar su identidad ante al sistema durante un período de tiempo determinado. El sistema relaciona este identificador de sesión con las acciones permitidas en dicha sesión.
Solicitud pedido	Mensaje formado por los identificadores unívocos de la variedad deseada y el agricultor seleccionado, la cantidad de tabaco deseada (en cajas de 180-200 kilogramos).
Transferencia	Solicitud de la realización del envío de una determinada cantidad de dinero entre dos cuentas bancarias. El formato exacto de este mensaje variará en gran medida según la pasarela implementada por la entidad financiera contratada por la planta de pretratado.
Variedad seleccionada	Identificador unívoco de la variedad de tabaco seleccionada por el usuario.

5.7. Selección de las acciones de servicio a implementar mediante servicios web

Con los modelos de composición de servicio se ha completado la especificación del nivel PIM. A continuación, se debe elaborar el modelo de composición de servicio extendido, tal y como se describió en §3.3.4. Este modelo prácticamente es una copia del anterior, salvo por el hecho de que algunos de los elementos serán distinguidos con el estereotipo <<WS>>, indicando que serán implementados como servicios web, en el caso de las acciones del sistema a desarrollar, o que están disponibles en forma de servicio web, en el caso de las acciones del resto de participantes.

Para seleccionar las actividades a implementar como servicios web, SOD-M recomienda de Castro [2007] comprobar si son reutilizables, si ya están disponibles como tales o si permiten la comunicación con el resto de participantes, En particular, se han seleccionado las siguientes acciones para su implementación como servicio web:

- «Confirmar credenciales e iniciar sesión», por reutilizarse en varias actividades de servicio.
- «Generar listado de variedades coincidentes», para reutilizar en el futuro la lógica de búsqueda de variedades en otras partes del sistema.
- «Generar listado de agricultores coincidentes», para su reutilización, como en el caso anterior.
- «Registrar resultados del control de calidad», para posibilitar el envío de los resultados desde varios puntos (PDA y estación de trabajo del usuario).
- «Completar y confirmar pedido», para recibir la información de la fecha de entrada y salida finales a partir de la empresa de logística y validar automáticamente la situación final.
- «Registro del pago enviado», para comunicarse con el resto de participantes que cobren o paguen a EBE.

Así mismo, se asume que las siguientes acciones del resto de participantes se modelarán como servicios web:

- «Generar listado de fechas factibles para visita» y «Confirmar disponibilidad», por parte del agricultor.
- «Establecer fechas de salida y entrega», por parte de la empresa de logística.
- «Recepción y ejecución de orden», por parte de la entidad bancaria. Es posible que haya que utilizar mecanismos específicos de la pasarela del banco. En estos casos, se podría plantear modelar la interacción del banco a través de una composición WS-BPEL, que oculte los detalles específicos de la pasarela, aunque habría que evaluar su viabilidad técnica.

5.8. Conclusiones

En las anteriores secciones se ha desarrollado un caso práctico sobre una parcela del negocio de una importante empresa del sector tabaquero, aplicando la versión extendida de SOD-M propuesta e implementada anteriormente en este TFM.

En el nivel de los modelos independientes de computación, se describió la perspectiva del negocio con los intercambios de valor existentes que justifican su viabilidad y el modelo de proceso de negocio que esboza su funcionamiento general. En el siguiente nivel de modelos independientes de la plataforma se comenzó a especificar el alcance del sistema a implementar, y cómo llevaría a cabo sus acciones. En este nivel, las extensiones sobre SOD-M permitieron modelar con éxito cuánto tiempo debería dedicarse bajo una carga determinada a cada parte de los servicios que el sistema proporcionaría, empezando por las actividades de servicio de los modelos de proceso de servicio y concluyendo en las acciones concretas de los modelos de composición de servicio. Finalmente, en el nivel específico de plataforma se seleccionaron del modelo de composición desarrollado las acciones que pasarían a ser servicios web.

A nivel de metodología, puede decirse que SOD-M ha dirigido con éxito el proceso iterativo de refinamiento requerido, aunque por limitaciones de tiempo no se ha llegado a especificar formalmente el formato de los mensajes en un lenguaje como WSDL, sino de manera informal textualmente. Como trabajo futuro, queda pendiente la elaboración de los últimos modelos específicos de plataforma de SOD-M, los modelos de interfaz de servicio web.

Aunque la experiencia con SOD-M ha sido positiva, se han identificado varios aspectos mejorables en ella durante la elaboración de este caso práctico. En la versión de SOD-M sobre la que se basó este trabajo no se incluyeron las correspondencias automáticas entre los distintos modelos descritas en de Castro [2007], por lo que fue necesario hacer los modelos de caso de uso extendido y de proceso de servicio desde cero, en vez de a partir de sus modelos de nivel superior. Se podría plantear implementar estas transformaciones mediante ETL, como en el caso de la correspondencia entre procesos de servicio y composiciones de servicio implementada en este trabajo.

Sería igualmente deseable añadir a SOD-M la capacidad de realizar transformaciones no sólo descendentes, sino también ascendentes. Es posible que el modelador se dé cuenta de una discrepancia entre sus expectativas y el modelo de nivel inferior obtenido a partir del modelo de nivel superior. Idealmente, debería ser posible modificar el modelo de nivel inferior y hacer que los cambios se propagaran a los modelos de nivel superior. Para implementar esta mejora, sería necesario hacer una transformación con ETL, y reunir las versiones antigua y nueva del modelo de nivel superior usando EML, posiblemente mostrando las diferencias con ECL. A un nivel más concreto, se ha identificado la necesidad de aclarar mejor la semántica de los flujos de objeto y cómo se propaga la ejecución del programa a través de ellos.

Por otro lado, SOD-M parece haber cambiado considerablemente en los últimos meses, como pudo comprobarse durante la asistencia a las V Jornadas Científico-Técnicas en Servicios Web (JSWEB 2009) García Domínguez et al. [2009]. Se ha pasado de usar diagramas de actividad UML a emplear una extensión de la Business Process Modeling Notation (BPMN) en la que se distinguen las acciones realizadas por los usuarios y las realizadas por el sistema. Los actuales modelos de proceso de servicio pasan a integrarse en los propios procesos de negocio, haciéndolos más concretos y accesibles a la generación de código final. A pesar de la magnitud de estos cambios, la metodología sigue apoyándose al fin y al cabo en grafos en los que la ejecución va progresando a través de flujos de control y paso de mensajes, por lo que se podrán

reutilizar las extensiones propuestas e implementadas en este trabajo tras las correspondientes revisiones.

Una vez se realicen estas adaptaciones, se volverán a priorizar las líneas de trabajo futuras esbozadas anteriormente sobre la metodología SOD-M y la implementación parcial desarrollada en este trabajo.

Bibliografía

María Valeria de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 5.24, 5.25

Antonio García Domínguez, Inmaculada Medina Bulo, and Mariano Marcos Bárcena. Hacia la integración de técnicas de pruebas en metodologías dirigidas por modelos para SOA. In *Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA*, Madrid, España, October 2009. 5.25

Jaap Gordijn and Hans Akkermans. Value-based requirements engineering: exploring innovative e-commerce ideas. *Requirements Engineering*, 8(2):114–134, July 2003. doi: 10.1007/s00766-003-0169-x. 5.2

Jaap Gordijn and Hans Akkermans. e3value™ toolset, August 2006. URL http://www.e3value.com/tools/. Fecha de última comprobación: 2 de noviembre de 2009. 5.2

World Wide Web Consortium. WSDL 2.0 part 1: Core Language, June 2007. URL http://www.w3.org/TR/wsdl20. Fecha de última comprobación: 2 de noviembre de 2009. 5.14

6

Conclusiones y trabajos futuros

Las empresas de fabricación requieren, para mantenerse competitivas, integrar las últimas tecnologías en procesos de fabricación y revisar sus prácticas de negocio de acuerdo a la demanda del mercado. Para ello, la empresa pasa de ser un único ente centralizado y jerarquizado a ser un conjunto de elementos interrelacionados de forma dinámica, integrando a proveedores, clientes, diseñadores, subcontratistas y otros participantes en lo que se conoce como una empresa extendida. La importancia de este hecho puede verse en la aparición de modelos de empresa distribuida en el campo de la Ingeniería de Fabricación, como son las empresas holónicas, fractales y biónicas.

En este trabajo se propone que una empresa extendida estructurada de tal forma debería tener un sistema de información que acompañara a esta visión, siguiendo una arquitectura orientada a servicios, en la que el sistema completo se estructurara como un ecosistema de servicios, proveedores y consumidores, de tal forma que su libre reconfiguración permitiera la rápida revisión de las prácticas de negocio y la integración con sistemas externos. Sin embargo, no puede hacerse a la ligera: se trata de un esfuerzo a realizar por toda la organización, y un fallo en un servicio ampliamente reutilizado podría tener graves consecuencias. Por esta razón, se hace evidente la necesidad de seguir una metodología bien definida que facilite la comunicación y garantice el desarrollo de una arquitectura cuyos servicios cumplan los requisitos impuestos. Muchas de estas metodologías parten de modelos abstractos de alto nivel y van concretándolos hasta su implementación, con diversos grados de automatización.

En relación con los objetivos propuestos en §1.3, se han conseguido los siguientes resultados en este TFM:

- Tras comparar la metodología basada en componentes de Stojanović Stojanović [2005], la metodología SOMA Ghosh et al. [2008] de IBM y la metodología SOD-M de Castro [2007] del grupo Kybele de la Universidad Rey Juan Carlos, se ha seleccionado SOD-M como base para la metodología de este TFM.

  SOD-M es una metodología dirigida por modelos para SOA que alinea los sistemas de información con las necesidades del negocio y que tiene un coste de aplicación menor que otras metodologías. Permite producir las interfaces WSDL World Wide Web Consortium [2007] de los aspectos del sistema que se vayan a ofrecer como Servicios Web.

- SOD-M no incluye aspectos de modelado de pruebas, así que se han propuesto diversas extensiones para solventar esta carencia. Estas extensiones han recibido difusión en la *Third Manufacturing Engineering Society International Conference* Medina Bulo et al. [2009] (MESIC'09) y las V Jornadas Técnicas en Servicios Web y SOA García Domínguez et al. [2009] (JSWEB'09).

  Se ha propuesto modelar la funcionalidad que el sistema debe ofrecer con condiciones a cumplir antes y después de cada actividad realizada por el sistema. Estas condiciones podrían estar escritas en un lenguaje inspirado en alguno de los existentes para otras tecnologías, como JML Burdy et al. [2005], OCL Object Management Group [2006], Spec# Barnett et al. [2005] o WS-CoL Baresi et al. [2007]. Otras posibilidades incluyen utilizar transformaciones de grafos, como propone Lohmann Lohmann et al. [2006] o descripciones semánticas, como sugiere Sinha Sinha and Paradkar [2006].

  El rendimiento del sistema se modelaría mediante unas anotaciones que indicarían el tiempo límite de respuesta del sistema ante una determinada carga. No todo el sistema ha de ser anotado manualmente: las restricciones que falten se estimarán automáticamente.

- Se han identificado una serie de técnicas y herramientas que podrían integrarse con los modelos extendidos de SOD-M.

  La generación de los casos de prueba concretos con las entradas a enviar y las salidas esperadas puede hacerse apoyándose en grafos causa-efecto como los propuestos por Myers en Myers [2004] y utilizados por Paradkar en Paradkar et al. [1997], o realizando una partición del espacio de las posibles entradas, tal y como hace Lohmann en Lohmann et al. [2007].

  Tras crear los casos de prueba combinando las anteriores técnicas con la experiencia del desarrollador, es necesario evaluar la calidad del conjunto final de casos de prueba. Para ello, puede usarse qué proporción del programa es recorrida por todos los casos de prueba, como hace la herramienta Cobertura Erdfelt et al. [2009] para programas Java. Otras opciones incluyen el análisis de mutaciones Domínguez Jiménez et al. [2008]; Jiang et al. [2005] o la generación dinámica de invariantes Palomo Duarte et al. [2008].

  La realización de las pruebas sobre el rendimiento del sistema puede apoyarse en herramientas ya existentes como JUnitPerf Clark [2009] o soapUI eviware.com [2009]. La herramienta concreta variará según la tecnología que se utiliza para implementar cada actividad del sistema.

- Se han analizado las herramientas utilizadas en las herramientas proporcionadas por el grupo Kybele para su metodología SOD-M. El estado original de las herramientas dificultaba su extensión para modelar los aspectos de prueba, ya que no se disponía de todo el código fuente y parte de las herramientas no funcionaban en la versión actual de la plataforma de desarrollo Eclipse.

  Por estas razones, se decidió reemplazar parte de las herramientas con nuevas versiones desarrolladas desde cero, que incluyeran las extensiones propuestas para modelar el rendimiento esperado del sistema. En particular, se reemplazaron los componentes dedicados a elaborar modelos de proceso de servicio y de composición de servicio.

  Se integraron nuevas tecnologías que mejoraban la flexibilidad y capacidad de actualización del proceso de desarrollo, como Emfatic Scharf [2008], Epsilon Kolovos et al. [2009] y EuGENia Eclipse Foundation [2009], y se automatizó parte del flujo de trabajo empleando Apache Ant Apache Foundation [2008].

  Los nuevos editores de modelos de proceso de servicio y de composición incorporan validación automática de los modelos, e integran algoritmos para estimar las restricciones no especificadas manualmente sobre el número de usuarios esperados y el tiempo límite, utilizando la estructura del grafo y ciertas anotaciones manuales. Se han añadido elementos a la interfaz gráfica de Eclipse para transformar automáticamente los modelos de proceso de servicio a una primera aproximación de sus modelos de composición de servicio.

- Se ha aplicado la metodología SOD-M extendida en este TFM a una parcela del negocio de una importante empresa del sector tabaquero. La experiencia ha resultado positiva, pudiendo conseguir de forma metódica una descripción detallada de la funcionalidad requerida para proporcionar los servicios esperados usando el sistema, una lista de los servicios web a utilizar y una descripción de los mensajes que deben ser intercambiados.

Actualmente, existen diversas líneas de trabajo abiertas para mejorar la metodología de este TFM y las herramientas obtenidas:

- Actualizar la metodología de este TFM a la última versión de SOD-M: el grupo Kybele ha continuado el desarrollo de SOD-M en paralelo con el desarrollo de este TFM, introduciendo nuevos modelos y revisando otros Hermann et al. [2009].

  Entre otras modificaciones, se han reemplazado los diagramas de actividad UML por modelos basados en la Business Process Modeling Notation (BPMN) para expresar los modelos de proceso de negocio. Es necesario estudiar cuáles de estos cambios podrían integrarse de nuevo en la metodología extendida en este TFM.

  Un primer análisis sugiere que, aunque se ha cambiado la notación concreta, las extensiones propuestas y los algoritmos definidos podrían adaptarse sin dificultades, ya que el enfoque fundamental de la metodología no ha cambiado: siguen utilizándose grafos cuyos elementos se van concretando de forma descendente.

  De hecho, el cambio a BPMN podría ser beneficioso, ya que existen propuestas con transformaciones parciales automatizadas (como la presentada por Ouyang et al. Ouyang et al. [2006]) de modelos escritos en BPMN a programas ejecutables escritos en el lenguaje WS-BPEL OASIS [2007].

- Generalizar los algoritmos de estimación de restricciones no especificadas para retirar la restricción de que no haya bucles, entre otras. Se podría añadir información acerca de un número máximo de iteraciones, o de la probabilidad de que se recorra el arco que vuelva al punto de partida del bucle.

- Implementar la generación de código a partir de los modelos usados: actualmente, las herramientas disponibles no producen código, y no permiten modelar las interfaces WSDL de los Servicios Web seleccionados. Es imprescindible llegar a generar código, para poder ejecutar las pruebas y obtener realimentación sobre la efectividad de las técnicas y herramientas integradas.

- Permitir un control más fino por el usuario sobre qué restricciones de rendimiento imponer en cada nodo. Por ejemplo, es posible que el usuario sólo desee especificar parcialmente algunas restricciones, o que sólo tenga una idea de las proporciones de tiempo entre las actividades: sabe que una tarda aproximadamente el doble que otra, pero no cuánto tarda exactamente.

- Mejorar la usabilidad del editor, haciendo que la situación automática de los nodos produzca distribuciones más agradables, y simplificando la notación usada.

- Si los modelos empiezan a crecer considerablemente, puede ser necesario reemplazar algunos de los algoritmos utilizados por alternativas más eficientes ante entradas grandes.

- Seleccionar una notación (Burdy et al. [2005] es una primera posibilidad) para especificar el comportamiento del sistema en base a las relaciones entre sus entradas y salidas y los cambios sobre los datos almacenados.

- Una vez se escoja una notación, habrá que integrar algunas de las técnicas seleccionadas para generar casos de prueba y evaluar su calidad a partir de los resultados producidos por su ejecución.

  En particular, para la evaluación de la calidad de los casos de prueba pueden utilizarse herramientas como las presentadas en Domínguez Jiménez et al. [2008]; Palomo Duarte

et al. [2008], que tienen en cuenta la funcionalidad del programa, más que únicamente su estructura.

- Realizar casos de estudio sobre empresas de otras industrias, como la aeronáutica, y expandir el alcance del caso de estudio actual sobre una empresa del sector tabaquero.

Bibliografía

Apache Foundation. Apache Ant 1.7.1, June 2008. URL http://ant.apache.org/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of WSCoL. In *Proceedings of the IEEE International Conference on Web Services, 2007 (ICWS 2007)*, pages 663–670, 2007. doi: 10.1109/ICWS.2007.25. 6.1

Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: an overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, pages 49–69. Springer Berlin, 2005. 6.1

Lilian Burdy, Yoonsik Cheon, and David R. Cok. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3):212–232, June 2005. 6.1, 6.3

Mike Clark. JUnitPerf, October 2009. URL http://clarkware.com/software/JUnitPerf.html. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

María Valeria de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 6.1

Juan José Domínguez Jiménez, Inmaculada Medina Bulo, and Antonia Botaro Estero. A framework for mutant genetic generation for WS-BPEL. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, Spindleruv Mlýn, República Checa, 2008. 6.2, 6.3

Eclipse Foundation. EuGENia, 2009. URL http://www.eclipse.org/gmt/epsilon/doc/eugenia/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

Joakim Erdfelt, John Lewis, Lukasik Grzegorz, Jiri Mares, and Jeremy Thomerson. Cobertura, September 2009. URL http://cobertura.sourceforge.net/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

eviware.com. Página principal de soapUI, 2009. URL http://www.soapui.org/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

Antonio García Domínguez, Inmaculada Medina Bulo, and Mariano Marcos Bárcena. Hacia la integración de técnicas de pruebas en metodologías dirigidas por modelos para SOA. In *Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA*, Madrid, España, October 2009. 6.1

S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008. 6.1

Elisa Hermann, María Valeria de Castro, and Esperanza Marcos. Reduciendo el gap entre el modelado de proceoss de negocios y las aplicaciones .Net basadas en servicios. In *Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA*, pages 211–224, Madrid, October 2009. 6.3

Ying Jiang, Shan-Shan Hou, Jin-Hui Shan, Lu Zhang, and Bing Xie. Contract-based mutation for testing components. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005 (ICSM'05)*, pages 483–492, 2005. ISBN 1063-6773. doi: 10.1109/ICSM.2005.36. 6.2

Dimitrios Kolovos, Richard Paige, Louis Rose, and Fiona Polack. The Epsilon book, 2009. URL http://dev.eclipse.org/svnroot/modeling/org.eclipse.gmt.epsilon/trunk/doc/org.eclipse.epsilon.book/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

M. Lohmann, G. Engels, and S. Sauer. Model-driven monitoring: generating assertions from visual contracts. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006 (ASE '06)*, pages 355–356, 2006. ISBN 1527-1366. doi: 10.1109/ASE.2006.52. 6.1

Marc Lohmann, Leonardo Mariani, and Reiko Heckel. A model-driven approach to discovery, testing and monitoring of web services. In *Test and Analysis of Web Services*, pages 173–204. Springer Berlin, 2007. ISBN 978-3-540-72911-2. doi: 10.1007/978-3-540-72912-9_7. 6.2

Inmaculada Medina Bulo, Antonio García Domínguez, Francisco Aguayo, Lorenzo Sevilla, and Mariano Marcos Bárcena. Propuesta metodológica para la implementación de una arquitectura orientada a servicios en entornos de sistemas de fabricación distribuida. In *Actas del III Congreso Internacional de la Sociedad de Ingeniería de Fabricación*, pages 346–353, Alcoy, España, June 2009. 6.1

Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2 edition, 2004. ISBN 0471469122. 6.2

OASIS. Web Service Business Process Execution Language (WS-BPEL) 2.0, April 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Fecha de última comprobación: 2 de noviembre de 2009. 6.3

Object Management Group. Object Constraint Language Specification (OCL) 2.0, May 2006. URL http://www.omg.org/technology/documents/formal/ocl.htm. Fecha de última comprobación: 2 de noviembre de 2009. 6.1

Chun Ouyang, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. From business process models to process-oriented software systems: the BPMN to BPEL way. Technical Report 5266, Faculty of Science and Technology, Queensland University of Technology, Brisbane, Australia, 2006. URL http://eprints.qut.edu.au/5266/. 6.3

Manuel Palomo Duarte, Antonio García Domínguez, and Inmaculada Medina Bulo. Takuan: a dynamic invariant generation system for WS-BPEL compositions. In *Proceedings of the 6th IEEE European Conference on Web Services*, Dublin, Irlanda, November 2008. 6.2, 6.3

A. Paradkar, M. A. Vouk, and K. C. Tai. Specification-based testing using cause-effect graphs. *Annals of Software Engineering*, 4:133–157, 1997. 6.2

Michael Scharf. Emfatic plugin download, October 2008. URL http://scharf.gr/eclipse/emfatic/download/. Fecha de última comprobación: 2 de noviembre de 2009. 6.2

Avik Sinha and Amit Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 17–22, Portland, Maine, 2006. ACM. ISBN 1-59593-458-8. doi: 10.1145/1145718.1145721. 6.1

Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005. 6.1

World Wide Web Consortium. WSDL 2.0 part 1: Core Language, June 2007. URL http://www.w3.org/TR/wsdl20. Fecha de última comprobación: 2 de noviembre de 2009. 6.1

A

Instrucciones de instalación y uso

Los dos resultados principales de este trabajo consisten en un conjunto de herramientas que implementan la metodología dirigida por modelos propuesta, y en una serie de modelos resultantes de aplicar dichas herramientas a una parcela del negocio de una importante empresa del sector tabaquero.

Aunque dichos resultados se discuten extensamente en este trabajo, se ha considerado conveniente incluirlos en forma de un DVD con este trabajo, para permitir una evaluación más a fondo del trabajo realizado. En esta sección se describe la estructura de sus contenidos, y se dan instrucciones acerca de su instalación y uso.

A.1. Contenidos del DVD

A.1.1. Directorio raíz

La raíz del disco contiene los siguientes elementos:

- `herramientas-linux32.tar.bz2`, que contiene la distribución de Eclipse Galileo 3.5.1 (véase §4.2.1) para sistemas GNU/Linux de 32 bits con todas las dependencias preinstaladas.
- `herramientas-windows.zip`, equivalente a la anterior, pero preparada para sistemas Microsoft Windows.
- `herramientas-fuentes.zip`, con el código fuente de cada una de las nuevas herramientas desarrolladas en este trabajo. Para más información, véase §4.4.
- `modelos.zip`, con los modelos desarrollados para el caso práctico desarrollado en el capítulo 5.

A.1.2. Distribuciones Eclipse

En el DVD se incluyen dos distribuciones Eclipse Galileo 3.5.1: una es para sistemas GNU/Linux de 32 bits (`herramientas-linux32.tar.bz2`) y la otra (`herramientas-windows.zip`) es para sistemas basados en Microsoft Windows. Por lo demás, incorporan la misma funcionalidad y herramientas:

- EMF Eclipse Foundation [2009a] 2.5.0 y GMF Eclipse Foundation [2009b] 2.2.0, las versiones más recientes de los marcos de trabajo sobre los que se han implementado las herramientas. Ambos marcos se exponen de forma más extensa en §4.2.2 y §4.2.3, respectivamente.
- Emfatic Scharf [2008], en su versión más reciente: la 0.3.0, publicada el 13 de octubre del 2008. Implementa el lenguaje específico de dominio usado para implementar los metamodelos descritos en este trabajo. En §4.3.2 se describe en mayor profundidad.
- La última instantánea de desarrollo publicada hasta la fecha (26 de octubre de 2009) de Epsilon Kolovos et al. [2009], basada en su versión 0.8.7. Implementa una familia de lenguajes de manejo de modelos en el que se ha implementado una parte considerable de las herramientas. Epsilon incluye a EuGENia, que asiste en la generación de los editores

gráficos avanzados. Véanse §4.3.3 y §4.3.4 para más detalles sobre Epsilon y EuGENia, respectivamente.

- La versión 1.1.0 de las tareas Apache Ant Apache Foundation [2008] para GMF desarrolladas por Ecliptical. Estas tareas permiten automatizar la generación del código de los editores gráficos avanzados, uno de los pasos del flujo de trabajo automatizado descrito en §4.3.5.
- Los editores proporcionados por Kybele en marzo del 2009 para los metamodelos de casos de uso y de casos de uso extendido de la versión original de la metodología SOD-M.

Por razones técnicas, esta versión de Eclipse no incluye directamente las nuevas herramientas creadas en este TFM para la versión extendida de la metodología SOD-M, descritas en 4.4. En su lugar, se debe iniciar Eclipse, señalar el directorio principal obtenido al descomprimir `herramientas-fuentes.zip` como espacio de trabajo, y lanzar desde dicho entorno (con *Run* → *Run Model Workspace*) el espacio de trabajo dedicado a los modelos. Las nuevas herramientas se hallarán disponibles en dicho espacio de trabajo.

A.1.3. Código fuente

El archivo comprimido `herramientas-fuentes.zip` contiene un espacio de trabajo Eclipse que incluye los siguientes proyectos:

- *es.uca.modeling.ant*: define las tareas Apache Ant necesarias para automatizar la invocación de Emfatic y la generación del modelo `gmfgen` a partir del modelo `gmfmap`.
- *es.uca.modeling.eol*: incluye los guiones EOL, EVL y EWL necesarios para validar los modelos de la versión extendida de SOD-M descrita en este trabajo (véase §4.4.2) y para estimar las restricciones no especificadas manualmente (véase §4.4.3).
- *es.uca.modeling.eol.contextmenu*: añade la entrada «Map to Service Composition» al menú contextual abierto sobre los ficheros con la extensión `serviceprocess` desde la vista Navegador de Eclipse. Esta acción realiza la correspondencia descrita en §4.4.4 a través de un guión ETL, también incluido en el proyecto.
- *es.uca.modeling.eol.tools*: extiende Epsilon, añadiéndole la capacidad de clonar fácilmente objetos a través de su clase Java *ECoreUtilsTool*.
- *es.uca.modeling.figures*: implementa las formas gráficas descritas en las tablas 4.1 y 4.3, que no se ajustaban a ninguna de las que GMF incorporaba por defecto.
- *ServiceProcess*: este proyecto incluye el metamodelo extendido de los procesos de servicio de SOD-M, obtenido a partir de la unión de `model/serviceprocess.emf.template` y `model/common.emf.fragment` en un solo guión Emfatic, `model/serviceprocess.emf`.

  A partir de este guión se produce `model/serviceprocess.ecore`, metamodelo Ecore del que se obtienen en el mismo directorio los modelos para generación de los editores. Para ello se utiliza un flujo de trabajo automatizado por el guión Apache Ant contenido en `generate-code.xml`.

El directorio `src` incluye el código fuente generado automáticamente para la lectura de modelos de procesos.

El directorio `templates` incluye las plantillas dinámicas necesarias para retocar automáticamente el código fuente generado para el editor gráfico avanzado.

- *ServiceProcess.diagram*: incluye el código generado automáticamente para el editor gráfico avanzado de modelos de procesos de servicio.
- *ServiceProcess.edit*: incluye el código para facilitar la edición de los modelos de procesos de servicio a través de Java.
- *ServiceProcess.editor*: incluye el código del editor básico de modelos de procesos de servicio basado en árboles.
- *ServiceComposition*: define el metamodelo de composiciones de servicio Ecore combinando el fichero `model/common.emf.fragment` de *ServiceProcess* con su propio fichero `model/servicecomposition.emf.template`.

  De forma análoga a *ServiceProcess*, este metamodelo Ecore servirá para generar los editores gráficos, cuyo código se retocará mediante las plantillas dinámicas de *ServiceProcess* y las de este propio *plugin*, situadas en los directorios `templates-gmfgraph` y `templates-gmfgen`.

  El guión Apache Ant `generate-code.xml` automatiza el flujo de trabajo de este proyecto, basándose en el guión Apache Ant del mismo nombre de *ServiceProcess*, que extiende con nuevas funcionalidades dirigidas a reutilizar el código reutilizable de *ServiceProcess*.
- *ServiceComposition.diagram*: incluye el código del editor gráfico avanzado de modelos de composiciones de servicio.
- *ServiceComposition.edit*: incluye el código para facilitar la edición de los modelos de composiciones de servicio a través de Java.
- *ServiceComposition.editor*: incluye el código del editor básico de modelos de composiciones de servicio basado en árboles.

A.1.4. Modelos

En el fichero `modelos.zip` se incluye un espacio de trabajo Eclipse con un único proyecto llamado *ModelosTabaco*, en el que se incluyen los modelos descritos en el caso práctico de §5 y sus diagramas correspondientes. Este proyecto incluye los siguientes directorios:

- `casos-uso`, con los modelos de casos de uso `.uc` y sus diagramas `.uc_diagram`.
- `casos-uso-ext`, con los modelos de casos de uso extendidos `.euc` y sus diagramas `.euc_diagram`.
- `composiciones-servicio`, con los modelos `.servicecomposition` de composiciones de servicio y sus diagramas `.servicecomposition_diagram`.

- `proceso-negocio`, con el diagrama de actividad UML correspondiente al modelo de proceso de negocio de SOD-M. El fichero `ProcesoNegocioVentaTabaco.uml` contiene su modelo propiamente dicho, y `ProcesoNegocioVentaTabaco.umlact` contiene el diagrama correspondiente.

- `procesos-servicio`, que incluye cada uno de los 3 modelos de proceso de servicio de §5.5 en sendos ficheros `.serviceprocess`. Sus diagramas correspondientes se hallan en los ficheros `.serviceprocess_diagram` que llevan el mismo nombre.

A.2. Instalación y uso

Instalar e iniciar el entorno usado varía según se use GNU/Linux o Windows, pero el uso posterior es exactamente el mismo. Se asumen ciertos conocimientos básicos acerca del uso de cada sistema operativo, como la capacidad de descargar ficheros desde el navegador, extraerlos, manipularlos, instalar programas y ejecutar acciones bajo la línea de órdenes.

Si tras iniciar por primera vez Eclipse aparecen fallos en los proyectos (elementos de nivel superior bajo el panel *Navigator*), es necesario actualizar la imagen de Eclipse del sistema de ficheros. Para ello, se puede hacer clic con el botón derecho en cada uno de los proyectos (carpetas de nivel superior en el panel *Navigator*) y elegir la opción *Refresh*, o pulsar F5. Esto no será necesario en ejecuciones posteriores.

A.2.1. Instalación en Microsoft Windows XP

En primer lugar, se necesita una versión reciente del kit de desarrollo para Java («Java Development Kit» o JDK) de Sun. Se recomienda tener la versión 6.0, pero puede funcionar con la versión 5.0.

La última versión del JDK de Sun puede descargarse desde `http://java.sun.com/javase/downloads/index.jsp#jdk`: sólo es necesario seguir los pasos presentados por pantalla.

Una vez se haya instalado el JDK, se han de descomprimir `herramientas-windows.zip`, `herramientas-fuentes.zip` y `modelos.zip` a un mismo directorio (preferiblemente `C:\`).

Por limitaciones de algunas de las tecnologías empleadas, deben evitarse los directorios con espacios en los nombres para este cometido, o de lo contrario las herramientas no funcionarán.

Para iniciar Eclipse, se accederá al directorio `eclipse-windows` obtenido tras descomprimir `herramientas-windows.zip`, y se ejecutará su fichero `eclipse.exe`. Se le indicará a Eclipse que utilice el directorio `workspace-metamodelos` resultante de descomprimir `herramientas-fuentes.zip`.

A.2.2. Instalación en Ubuntu Linux 9.04

Para instalar un JDK compatible, puede ejecutarse la siguiente orden bajo una línea de órdenes, que pedirá nuestra contraseña:

```
sudo aptitude install openjdk-6-jdk
```

A continuación, se han de descomprimir al mismo directorio (evitando espacios en la ruta) los ficheros `herramientas-linux32.tar.bz2`, `herramientas-fuentes.zip` y `modelos.zip`.

Al terminar de descomprimir, en dicho directorio aparecerá un guión llamado `eclipse-linux32.sh`, que servirá para lanzar Eclipse. No es necesario especificar el espacio de trabajo en este caso, ya que el guión se ocupa de ello.

Si aparecen fallos, es posible que sea necesario actualizar la imagen de Eclipse del sistema de ficheros. Para ello, se puede hacer clic con el botón derecho en cada uno de los proyectos (carpetas de nivel superior en el panel *Navigator*) y elegir la opción *Refresh*, o pulsar F5.

A.2.3. Uso general

Eclipse es una aplicación muy avanzada, por lo que no se puede dar una descripción breve de su funcionamiento completo. En esta sección se recogen las líneas generales para trabajar con la aplicación.

Para únicamente examinar las herramientas implementadas en el trabajo, pero no usarlas, basta con navegar por el código fuente disponible justo tras iniciar Eclipse con las instrucciones anteriores. Tras hacer cambios en las herramientas, lo mejor es limpiar todos los proyectos mediante *Project* → *Clean*, y dejar que los constructores Ant automáticos hagan el resto del trabajo.

Para visualizar los modelos producidos en el caso de estudio y emplear las herramientas desarrolladas, más que examinar su implementación y diseño, hay que ejecutar el lanzador «Run Model Workspace» que se halla bajo el menú *Run*. Se abrirá una segunda instancia de Eclipse que empleará como espacio de trabajo el directorio dedicado a los modelos.

NOTA: el resto de instrucciones de este apartado son sólo aplicables estando dentro de esta segunda instancia.

Para crear los modelos definidos en este trabajo desde la segunda instancia, se debe hacer clic con el botón secundario del ratón en un directorio o proyecto del panel *Navigator* y elegir la opción *New...*. Los tipos de modelos creados en este trabajo son fácilmente localizables introduciendo la cadena «Service» en la caja de búsqueda del diálogo que aparece a continuación. Se recomienda crear diagramas directamente, al ser más cómodo, pero si se crea solamente un modelo, se puede hacer clic con el botón secundario del ratón y derivar un diagrama de él con la opción *Initialize diagram file* del menú contextual.

Para ejecutar los algoritmos implementados en este trabajo desde la segunda instancia, se ha de abrir el diagrama de un modelo de proceso de servicio o de un modelo de composición de servicio, y hacer clic con el botón secundario del ratón en el fondo del diagrama. Bajo la entrada de *Wizards* aparecerán dos elementos: *Infer performance annotations* se ocupa de crear y actualizar todas las anotaciones de rendimiento no especificadas manualmente, mientras que *Remove all inferred annotations* las retira todas.

Finalmente, para transformar un modelo de proceso de servicio a un modelo de composición de servicio desde la segunda instancia, se debe hacer clic con el botón secundario del ratón en su entrada del panel *Navigator* y escoger *Map to Service Composition*. El fichero con el modelo de composición de servicio correspondiente aparecerá automáticamente bajo el mismo directorio.

Bibliografía

Apache Foundation. Apache Ant 1.7.1, June 2008. URL http://ant.apache.org/. Fecha de última comprobación: 2 de noviembre de 2009. A.2

Eclipse Foundation. Eclipse Modeling Framework, 2009a. URL http://eclipse.org/modeling/emf/. Fecha de última comprobación: 2 de noviembre de 2009. A.1

Eclipse Foundation. Graphical Modeling Framework, 2009b. URL http://eclipse.org/modeling/gmf/. Fecha de última comprobación: 2 de noviembre de 2009. A.1

Dimitrios Kolovos, Richard Paige, Louis Rose, and Fiona Polack. The Epsilon book, 2009. URL http://dev.eclipse.org/svnroot/modeling/org.eclipse.gmt.epsilon/trunk/doc/org.eclipse.epsilon.book/. Fecha de última comprobación: 2 de noviembre de 2009. A.1

Michael Scharf. Emfatic plugin download, October 2008. URL http://scharf.gr/eclipse/emfatic/download/. Fecha de última comprobación: 2 de noviembre de 2009. A.1