Un marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos de software

Memoria de tesis doctoral presentada para la obtención del grado de Doctor por la Universidad de Sevilla

Doctoranda:

Laura García Borgoñón

Dirigido por:

Doctora Dña. María José Escalona Cuaresma
Doctor D. Manuel Mejías Risoto

Sevilla, 19 de noviembre de 2015

Un marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos de software

Memoria de tesis doctoral presentada para la obtención del grado de Doctor por la Universidad de Sevilla

Doctoranda:

Laura García Borgoñón

Dirigido por:

Doctora Dña. María José Escalona Cuaresma
Doctor D. Manuel Mejías Risoto

Sevilla, 19 de noviembre de 2015

A Inés, Rocío y Marina,
mi INROMA personal

Los trenes son maravillosos.
Es importante saber a dónde van,
pero lo más importante es decidirse a subir a ellos.
The Polar Express

Agradecimientos

Llegado este momento, el de escribir el apartado de agradecimientos de la memoria, se acumulan en mi interior una mezcla de sensaciones. Por un lado alivio y satisfacción, puesto que por fin ha terminado el trabajo, el esfuerzo, el final del túnel, un punto que hasta hace pocas semanas pensaba imposible de alcanzar. Por otro lado tristeza, porque aunque parezca mentira, sé que la voy a echar de menos. Y por último, agradecimiento a todas aquellas personas que me han acompañado en el camino.

A mis tutores, María José y Manuel, por sus consejos, dedicación y paciencia. Porque han estado siempre animándome para que no tirara la toalla, algo que en múltiples ocasiones se me ha pasado por la cabeza. Me gustaría mencionar de manera especial a María José que, desde el día que aparecí en su despacho de Relaciones Internacionales, llegada de la otra punta de España y sin conocernos de nada, para pedirle que fuera mi directora he tenido en ella siempre una mano amiga y un apoyo constante, convirtiéndome, a pesar de la edad, en una más de sus niñas.

A mis padres, Ángel y Margarita, por enseñarme el valor del esfuerzo y del trabajo bien hecho, y por hacer más fáciles mis ausencias. También a mi hermana Ana, porque su capacidad de superación siempre ha sido un estímulo para mí.

A mis niñas, Inés, Rocío y Marina, porque sé que esta tesis les ha privado tiempo de su mami. Porque han sabido respetar mis momentos de estrés y, con sus juegos y sus risas han conseguido que, a pesar del esfuerzo, siempre pudiera acabar el día con una sonrisa. Porque tienen reservada para mi el disfraz de la Dra. Juguetes. De manera muy especial a Miguel Ángel, mi marido, mi compañero de viaje y mi gran apoyo. Por las horas sin fin de charlas o en silencio siendo, según la ocasión, tan reconfortante lo uno como lo otro. Por ayudarme a demostrarme día a día que era capaz de conseguirlo, y siempre estar ahí para cuando lo he necesitado.

A mi amigo Guillermo por su paciencia infinita durante todo este tiempo, por su ayuda en los aspectos más técnicos y por aportarme una gran dosis de sensatez en los momentos de crisis. Por dejarme incorporar a su lenguaje la palabra metamodelo, aunque él ha prometido que nunca hará la tesis.

A mis compañeros del grupo IWT2 del departamento de Lenguajes y Sistemas Informáticos, que me recibieron con los brazos abiertos, siempre dispuestos a ofrecerme su ayuda en este

tiempo. Francis, Manolo, Julián, Alberto... son muchos, no quiero mencionarlos a todos porque seguro que me olvido de alguno. En especial a Julián, porque juntos empezamos el recorrido de la tesis y, aunque con unos meses de diferencia, también lo hemos terminado. Porque es para mí un ejemplo de fuerza y superación.

A los profesores Margaret Ross y Geoff Staples por acogernos en nuestra estancia en Southampton. Por su paciencia con el idioma y por sus sabios consejos aportando un punto de vista muy diferente. Y al profesor Gustavo Rossi, a Silvia y a todo su equipo del laboratorio LIFIA, por su hospitalidad y amabilidad en nuestra visita en La Plata en aquellos fríos días de Agosto.

A mis amigos más cercanos, María, Pilar, Irune, María, Esther, Kiko, Elena, Izaskun... porque me han apoyado, me han escuchado y preguntado a cada paso que he dado en este tiempo, aunque la temática les fuera completamente ajena. Unas tardes de risas compartidas son siempre un gran descanso.

Estoy segura de que he olvidado mencionar a muchos que, de alguna manera, han formado parte de este viaje que hoy termina. A todos ellos, mi agradecimiento de corazón, porque este trabajo nunca podría haberlo hecho sin vosotros.

Resumen

Hoy en día los sistemas software son cada vez más complejos y su desarrollo se convierte en un desafío continuo para las empresas de software que deben adaptar su forma de trabajar al entorno cambiante, dinámico y globalizado que las caracteriza. En aras de la fabricación de productos software de calidad, en un tiempo de acceso al mercado adecuado y con un coste competitivo, los procesos de software se han convertido en uno de los activos fundamentales de cualquier empresa del sector de las tecnologías de la información y las comunicaciones (TIC).

Son muchos los estándares y modelos de referencia, que la industria del software usa y aplica, que establecen la importancia de tener definidos y documentados los procesos de software, estableciendo qué características o requerimientos deberían cumplir, pero no indican cómo definir esos procesos. La búsqueda de la mejor forma de representar y definir los procesos de software dentro de las organizaciones, para garantizar su uso de forma sistemática e institucionalizada, ha sido objeto de estudio desde hace décadas, mediante la creación de diferentes lenguajes de modelado de procesos de software.

Desde los años 90 más de una treintena de lenguajes significativos han sido desarrollados con el objetivo de modelar los procesos de software. Cada uno de estos lenguajes era creado para solventar algún problema existente que no estaba resuelto con los anteriores, pero con el mismo objetivo: obtener los modelos de procesos y sistematizar su uso en las empresas de software. Varias han sido las tendencias sobre las tecnologías base para su modelado, desde la creación de nuevos lenguajes de programación específicos para procesos, pasando por la formalidad de las redes de Petri, hasta los más modernos enfoques basados en modelos de acuerdo con el paradigma de la ingeniería dirigida por modelos (Model Driven Engineering, MDE).

Sin embargo, a pesar de todos los esfuerzos por generar un lenguaje de modelado de procesos de software que sobresaliera por encima del resto y se convirtiera en un estándar de uso generalizado por las organizaciones de software a nivel mundial, ninguno ha tenido una aceptación muy superior al resto. Ni siquiera el respaldo de los más importantes organismos de estandarización en el ámbito del software ha conseguido cambiar esta inercia, ya sea por el conocimiento existente en los ingenieros de procesos o por la dinámica heredada por el sector en el que una empresa se mueve. Cada organización selecciona el lenguaje de modelado de procesos de software que considera más adecuado en sus inicios, estableciendo un vínculo tan fuerte que es difícil de romper más adelante, de forma que si las necesidades evolucionan con el tiempo, es preciso un cambio de lenguaje o, simplemente, es necesario que los procesos de una organización interopereren con los de otras para llevar a cabo algún proyecto específico, la actividad de transformación

o traducción a otros lenguajes se convierte en una tarea muy costosa en tiempo y esfuerzo de personal, generadora de errores e inconsistencias.

A diferencia de las propuestas de estandarización más extendidas, el planteamiento en este trabajo de tesis aboga por mantener la diversidad de lenguajes de modelado de procesos de software en la organizaciones de la forma en la que éstas lo consideren oportuno, siendo su objetivo fundamental el desarrollo de un marco de referencia para facilitar, mejorar y agilizar la interoperabilidad y mantenibilidad de los modelos de procesos de software, independientemente del lenguaje elegido para su modelado. Para alcanzarlo, el marco de referencia está basado en tres pilares fundamentales.

El primero de ellos consiste en un lenguaje de modelado de procesos de software adecuado para el marco, que se considera como un lenguaje base y al que hemos denominado INROMA (INteROperabilidad y MAntenibilidad). Dicho lenguaje se caracteriza por ser de fácil aprendizaje y por contener únicamente aquellos conceptos necesarios comunes para la definición y modelado de cualquier proceso de software, siguiendo la norma ISO/IEC TR 24744:2007, por lo que se convierte en una pieza clave para la interoperabilidad y mantenibilidad de los mismos. Estas características le convierten en un lenguaje de fácil acceso para cualquier organización, y no privilegia ningún lenguaje de modelado de procesos de software existente frente al resto. En el desarrollo de INROMA se han definido tanto su sintaxis abstracta, siguiendo el paradigma MDE, mediante un metamodelo, algo básico para obtener las funcionalidades del marco, como su sintaxis concreta, en su uso como lenguaje. El segundo de los pilares fundamentales es el método mediante el que se establecen las bases teóricas que permiten incorporar nuevos lenguajes de modelado de procesos de software al marco de referencia. Por último, el tercero de los pilares son las transformaciones que formalizan las correspondencias y se constituyen como el nexo entre cualquier lenguaje de modelado de procesos de software incorporado al marco de referencia e INROMA. Los tres elementos conforman el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los procesos de software.

Teniendo en cuenta que uno de los principales propósitos en el desarrollo de este trabajo de tesis es lograr su utilización en entornos empresariales, para conseguir un importante impacto en estos ámbitos todo este fundamento teórico se completa con MONETA, una herramienta de soporte para el marco de referencia, proporcionando asistencia y automatización en su uso. Dicha herramienta ha sido validada mediante casos de estudio reales extraídos de proyectos de transferencia con empresas en los que se observa cómo la propuesta desarrollada ha sido de gran utilidad en los mismos.

En definitiva, esta tesis doctoral plantea el desarrollo de un marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos de software, abordando tanto el planteamiento teórico que los sustenta como la parte práctica mediante una herramienta de soporte para su utilización en empresas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hoy en día los sistemas software son cada vez más complejos. Desarrollar productos de calidad, con un tiempo de acceso al mercado aceptable y con un coste competitivo es un desafío al que las empresas de software se enfrentan continuamente. Además, a este desafío se debe añadir que el tiempo en el que el desarrollo de los productos estaba completamente localizado en un emplazamiento concreto ha pasado. El mundo globalizado en el que vivimos nos facilita el acceso a los principales expertos internacionales en todas las materias, y esto es algo especialmente evidente en las tecnologías del software. Una propiedad intrínseca a estas tecnologías es que se encuentran continuamente en cambio y evolucionando, y las organizaciones de desarrollo de software se enfrentan diariamente al reto de trabajar manejando la complejidad de aquello que producen. La participación de múltiples equipos de proyecto situados en diferentes puntos del planeta, con conocimientos distintos y diversidad de herramientas, supone una complicada combinación que es necesario evaluar a la hora de plantear nuevas formas de trabajar, en las que se incluyen el conjunto de tecnologías, mejores prácticas y estrategias que nos permitan gestionar de manera efectiva el desarrollo realizado por grandes equipos de ingenieros de software a lo largo de todo el ciclo de vida de los productos. Las formas de trabajo que incorporan todos estos aspectos es lo que denominamos proceso de software y, cada día más, se han convertido en uno de los activos fundamentales de cualquier organización dedicada al desarrollo de software [Bendraou *et al.*, 2010].

A continuación, en la primera sección, ahondaremos en el concepto de proceso de software, ya que este trabajo de tesis se enmarca en torno a este dominio. En ella se esbozan los distintos enfoques desde los que se pueden abordar los procesos de software, así como la necesidad de describirlos, marcada en gran medida por su carácter de activo de una organización que debe comunicarse y transmitirse fielmente y que queda reflejado en los distintos estándares relacionados con procesos de software que existen en la actualidad. En la siguiente sección se señala la existencia de un gran número de lenguajes de modelado de procesos de software que se han ido definiendo desde hace años, y las dificultades que conlleva la elección del más adecuado, así como la problemática en la comunicación y operación interorganizacional derivada de la misma. A raíz de esta problemática, en la tercera sección se expone el enfoque que se propone en este trabajo de tesis y que va a ser profundizado a lo largo de toda esta memoria. A continuación, se describe cuál va a ser la estructura de este documento y, finalmente, se presentan unas breves

conclusiones de este capítulo.

1.1. Los procesos en la industria del software

El concepto de *proceso* no es algo nuevo ni específico del entorno del software. En el ámbito industrial los procesos existen desde la aparición de las primeras manufactureras en los siglos XVIII y XIX, mucho tiempo antes de que se diseñara el primer ordenador [McCartney, 1999]. Con ellos se establecía el flujo de trabajo y utilización de las máquinas dentro de las fábricas, así como la forma de interacción y comunicación entre los responsables de las diferentes actividades que se llevaban a cabo.

La International Organization for Standardization (ISO)[1] proporciona una definición genérica de proceso, estableciendo que *un proceso consiste en el uso de los recursos para transformar entradas en salidas debido a que algún tipo de trabajo, actividad o función se ha llevado a cabo.* La promesa de un coste efectivo y mejor calidad en los productos que se fabricaban fue un estímulo para convertirlos en una importante área de estudio. Los resultados obtenidos en su aplicación y la mayor competitividad de las organizaciones que hacían uso de ellos propició que, con la intención de beneficiarse de la experiencia de la industria manufacturera, otros dominios incorporaran los procesos como ámbito de investigación.

En lo que al software se refiere, los procesos pueden ser considerados desde el punto de vista del negocio (Business Process Management, BPM) o de la ingeniería del software (Software Process Engineering, SPE) [Bendraou y Gervais, 2007]. En ambos dominios los elementos clave de estudio son los mismos, los procesos y las tecnologías asociadas, pero cada uno de ellos establece un enfoque diferente y, por tanto, la evolución individual es también distinta. El hecho de disponer de diferentes perspectivas puede convertirse en causa de confusión a la hora de elegir cuál es el enfoque más adecuado para una organización de software en particular, por lo que resulta necesario conocer qué comprende cada una de ellas.

Desde el punto de vista del negocio, un proceso se define como *el conjunto de tareas relacionadas de manera lógica que se deben realizar para proporcionar valor a los clientes o cumplir otros objetivos estratégicos* [Strnadl, 2006]. La gestión de procesos de negocio se refiere al conjunto de métodos, técnicas y herramientas que soportan el diseño, aprobación, gestión, análisis y mejora de los procesos de negocio [Van Der Aalst *et al.*, 2003b].

En [Bendraou y Gervais, 2007] se resumen los principales objetivos en la adopción de BPM en los siguientes:

- Organizar el negocio alrededor de los procesos y centrarse en la satisfacción del usuario.
- Clarificar y documentar los procesos.
- Monitorizar el rendimiento y cumplimiento de los mismos.

[1] http://www.iso.org

- Identificar continuamente oportunidades para su mejora y despliegue.

Con este enfoque los procesos de desarrollo y mantenimiento de software se plantean de igual manera que cualquier otro proceso de organización, usando los mismos conceptos, ideas e intereses.

Si nos planteamos los procesos desde la perspectiva de la ingeniería del software, el enfoque cambia y el interés que suscitan también.

Una forma habitual de definir el proceso en ingeniería del software es considerarlo como *el conjunto parcialmente ordenado de pasos, de artefactos relacionados, recursos, tanto humanos como aplicaciones informáticas, estructuras organizacionales y restricciones, que tienen como objetivo producir y mantener software* [Lonchamp, 1993]. Humphrey [Humphrey, 1988] lo define como *el conjunto total de actividades de ingeniería del software necesarias para transformar los requerimientos de usuario en software*. Un aspecto interesante que aporta esta definición respecto a la anterior es que la calidad de un producto software viene determinada por el buen entendimiento de los requerimientos de usuario. Más recientemente, aunque muy similar a la definición de Humphrey, Sommerville [Sommerville, 2007] define procesos de software como *el conjunto de actividades y resultados asociados que producen un producto de software*.

Sea como fuere, el proceso de software se considera un pilar fundamental en la ingeniería del software, ya que constituye un elemento que armoniza las tres dimensiones críticas de cualquier organización: a) los *métodos*, que especifican cómo se construye técnicamente el software, b) las *herramientas*, que proporcionan un soporte automático o semi-automático en su construcción, y c) las *personas*, que son las que construyen ese software [Chrissis *et al.*, 2011]. En la figura 1.1 se muestran estas relaciones.

A pesar de tener características comunes, los procesos de software tienen particularidades que los hacen diferentes a los típicos procesos de producción. El desarrollo de software es una actividad altamente creativa y no completamente formalizable ni automatizable. En [Ruiz-González y Canfora, 2004] se identifican las principales características que determinan la naturaleza de los procesos de software:

- Son complejos.
- Están dirigidos por excepciones, altamente impredecibles ya que dependen de mucha gente y sus circunstancias.
- No todas las actividades pueden automatizarse con herramientas y algunas de ellas pueden ser incompletas e informales.
- Se basan en la obtención de evidencias y dependen de la comunicación, coordinación y cooperación de personas y tecnologías.
- Su éxito depende de la involucración del usuario y la coordinación de muchos roles.
- Suelen tener una larga vida, siendo susceptibles de cambios para su mejora.

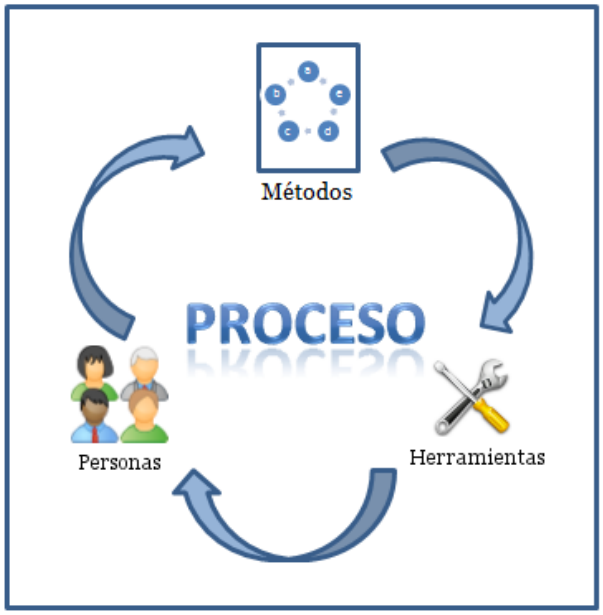


Figura 1.1: Dimensiones en la ingeniería del software.

En definitiva, eligiendo un enfoque de ingeniería del software se recalcan las particularidades de los procesos de software frente al resto de procesos existentes en una organización.

A pesar de las diferencias en los puntos a enfatizar, en ambos ámbitos, BPM y SPE, se establece como una necesidad fundamental el hecho de que una organización de software disponga de procesos, como soporte para conseguir que los recursos, métodos y herramientas de una organización se dispongan de la manera más adecuada para conseguir desarrollar productos software de calidad. La definición de los procesos de software, la forma en la que se documentan, se almacenan, se comunican y se institucionalizan dentro de una organización es clave para el buen funcionamiento de la misma [Ahern *et al.*, 2008].

Sirva como ejemplo lo establecido por Pressman [Pressman y Maxim, 2014], según el cual un proceso se define tal y como se muestra en la figura 1.2. Establece un marco de trabajo común, donde existen actividades que se aplican a todos los proyectos de software, independientemente de su tamaño o complejidad, y unas actividades de protección, que son transversales en una organización y no corresponden específicamente a ningún proyecto concreto. Es en el conjunto de tareas donde se ponen de manifiesto las características específicas de un proyecto determinado.

Este esquema de definición de procesos de software ha sido comúnmente seguido por un gran número de estándares, metodologías y modelos desarrollados en los últimos años, los cuales se han centrado en la gestión, desarrollo y evaluación de procesos de software. Dichos estándares han sido desarrollados para estructurar, describir y prescribir qué elementos deberían ser incluidos en la definición de los procesos de software de una organización. Algunos de estos modelos hablan de mejora continua y evaluaciones, otros hablan de definiciones propiamente dichas, y la mayoría suelen estar relacionados unos con otros. Como ocurre en otros ámbitos, estos estándares se han convertido en la referencia utilizada por los clientes para conocer el uso que

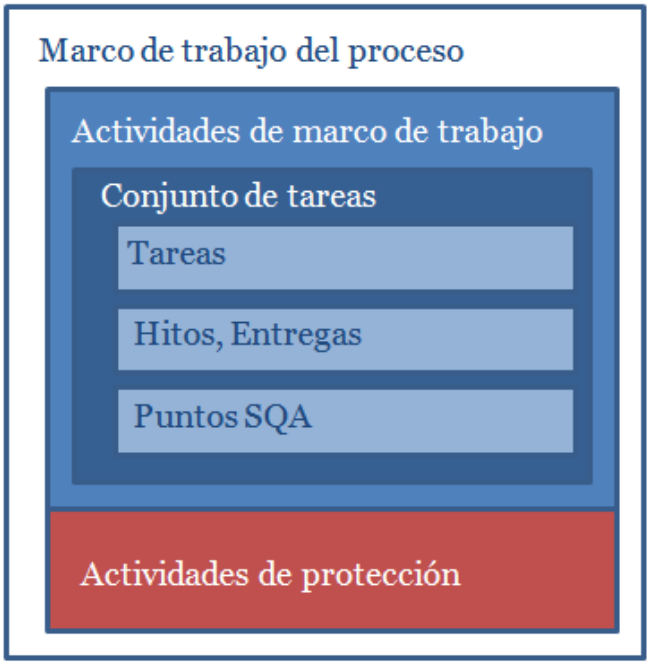


Figura 1.2: El proceso de software.

un proveedor de servicios software hace de los procesos, a modo de garantía de calidad. En la tabla 1.1 se muestra un resumen orientativo de los principales estándares y modelos de referencia más utilizados en la industria del software, así como su ámbito de aplicación, llevado a cabo por el comité de Tecnologías de la Información de la Asociación Española para la Calidad (AEC)[2]. Intentar hacer un listado más exhaustivo de todos los que hacen referencia al software es una tarea altamente complicada, porque muchos se especializan por la tipología del software a la que se refiere, criticidad, el ámbito o sector al que se dirige, etc.

Cabe destacar que, a la vista de los aspectos abarcados en las diferentes normativas y estándares mostrados en el resumen, en muchas ocasiones se mezclan los enfoques de negocio e ingeniería del software de los propios procesos, lo que confunde notablemente a las personas de las organizaciones encargadas de su adopción e implantación.

Para finalizar, como conclusión de todo lo que en esta sección se ha referido podemos extrapolar dos cosas fundamentalmente: la primera es la importancia de disponer de procesos en una organización de desarrollo de software, refrendada por la gran cantidad de estándares y modelos de referencia existentes manejados hoy en día por el mercado y la industria del software; la segunda es que dichos procesos deben estar establecidos (o definidos) y documentados (o mantenidos) de la forma más adecuada posible para que sean utilizados de forma sistemática en las organizaciones. Este hecho ha supuesto la causa por la que, tal y como veremos en la siguiente sección, numerosos lenguajes de modelado de procesos de software han sido propuestos como medio de definición e institucionalización en las empresas.

[2]http://www.aec.es

Estándares y Modelos de buenas prácticas	Dominios de Aplicación				
	Calidad	Desarrollo de Sistemas	Gestión de Proyectos	Gobierno y Servicios TIC	Negocio
ISO/IEC 12207	✓	✓	✓		
ISO/IEC 15288		✓	✓		
ISO/IEC 15939	✓		✓		
ISO/IEC 20000				✓	
ISO/IEC 38500				✓	✓
ISO/IEC 25000 SQuaRE	✓				
ISO/IEC 27001				✓	✓
ISO/IEC 21500			✓		
ISO/IEC 15504 (SPICE)	✓	✓	✓		
COBIT				✓	✓
ITIL			✓	✓	
PCI DSS					✓
BS 25999 (ISO 22301)					✓
BCI					✓
PMBOK			✓		
CMMI	✓	✓	✓		✓
EFQM					✓
TMMi	✓	✓	✓		✓
PRINCE2			✓		

Tabla 1.1: Principales estándares y modelos de buenas prácticas, y su dominio de aplicación.

1.2. Los lenguajes de modelado de procesos de software: una difícil elección

Con el fin de tener la capacidad de definir los procesos de software y sistematizar su uso, un gran número de lenguajes de modelado de procesos de software han sido desarrollados desde hace décadas [García-Borgoñón *et al.*, 2014a]. La aparición de cada uno de ellos ha venido motivada por la necesidad de satisfacer expectativas relacionadas con su uso y que los lenguajes ya existentes no eran capaces de cubrir. Entre estas expectativas o requerimientos se puede hablar de facilidad de uso, capacidad de ejecución, riqueza semántica, etc. Debido al hecho de que se trata de un campo de estudio con varias décadas de trabajo, desde una perspectiva temporal podemos ver la evolución de los enfoques propuestos para dichos lenguajes de una manera similar a la llevada a cabo por la ingeniería del software en general, partiendo de propuestas que eran un mero lenguaje de programación específico para el modelado de procesos, hasta las más actuales basadas en el paradigma Model Driven Engineering (MDE).

Si a los lenguajes de modelado de procesos de software, especificados bajo la perspectiva de la singularidad de éstos frente al resto de los procesos existentes, se incorporan los lenguajes de modelado de procesos de negocio, también utilizados en determinadas organizaciones para cubrir esta necesidad, el número resultante de lenguajes con los que una empresa de software puede trabajar para modelar sus procesos se incrementa de forma notable.

Las últimas tendencias en las propuestas de nuevos lenguajes de modelado de procesos de

software se orientan hacia la definición de un estándar de uso común en todas las organizaciones, que cubra todos los aspectos que se puedan plantear en relación a los procesos de software y que, como estándar, facilite el trabajo y la comunicación entre empresas. Sin embargo, de la gran cantidad de lenguajes definidos se desprende que ninguno de los existentes ha llegado a ser considerado estándar de facto por encima del resto, y que cada uno de ellos atiende y es adecuado para unos requerimientos concretos de uso. Hoy por hoy no existe ninguno que solucione todos los requisitos más habituales, puesto que algunos de ellos pueden llegar a ser incompatibles, como podría ser el caso de formalidad frente a flexibilidad o facilidad de uso. Una organización debería tener la posibilidad de modelar sus procesos de software utilizando el lenguaje que considere más adecuado en cada momento. Sin embargo, una vez que se ha elegido uno de ellos para afrontar una determinada necesidad, se establece un vínculo muy fuerte, y los procesos de software quedan altamente acoplados al mismo, siendo muy difícil utilizar otro lenguaje en otro contexto. Si a esta problemática le añadimos el hecho de tener que elegir el dominio de aplicación para poder incorporar los conceptos adecuados, ya sea gestión de procesos de negocio o ingeniería del software, este vínculo supone un hándicap aún mayor, puesto que si en un mismo ámbito cambiar de lenguaje es complicado, si nos trasladamos a un dominio diferente todavía lo es más. Por lo tanto, un primer problema que se plantea es la capacidad de mantenibilidad, por ejemplo en el tiempo, de esos procesos de software.

Ni que decir tiene que si a esta necesidad le añadimos la circunstancia de que dos organizaciones diferentes, por ejemplo, departamentos diferentes en una misma empresa o, incluso, empresas completamente separadas y distintas, tengan que colaborar y operar entre ellos con sus procesos de software y éstos se encuentren modelados en lenguajes diferentes, el problema ante el que nos enfrentamos se acrecienta considerablemente.

Una vez expuesta la problemática subyacente ante la existencia de tantas opciones de lenguajes de modelado de procesos de software, en la siguiente sección se sugiere cuál va a ser el planteamiento de la propuesta que se profundizará en este trabajo de tesis.

1.3. Una solución para el modelado de procesos de software

Recapitulando los puntos tratados en las secciones anteriores hemos razonado la importancia de los procesos en las organizaciones de software y la existencia de múltiples lenguajes de modelado para procesos de software que existen en la actualidad, aunque ninguno de ellos tenga una prevalencia sobre el resto, ni fuera sencillo operar entre organizaciones diferentes que modelan sus procesos de software con lenguajes distintos entre sí o incluso en grandes empresas con diferentes departamentos que usan diferentes lenguajes.

Esta misma problemática, siempre salvando las distancias, pero al utilizarlo como símil nos puede ayudar a explicar mejor lo que planteamos en este trabajo de tesis, ha ocurrido en otras ramas de estudio científico como la lingüística. En ella, la necesidad de encontrar una lengua de uso común que permita la comunicación entre los diferentes pueblos y naciones ha sido planteada desde hace siglos. Como en nuestro caso, un enfoque habitual ha sido el intentar hacer sobresalir una lengua frente al resto, con las dificultades que conlleva aprender una nueva lengua para

aquellos que no la tienen como lengua materna o primera, y la primacía no fácilmente justificable de aquellos que sí la tienen como lengua madre frente a los demás. A finales del siglo XIX se planificó el denominado Esperanto [Zamenhof, 1905], con el objetivo de que se convirtiera en una lengua base internacional. Se trataba de un idioma muy simple y de fácil aprendizaje, porque estaba basado en los aspectos más sencillos de todos los idiomas, con unas reglas gramaticales mínimas y características propias, y cuyo objetivo era que se convirtiera en una segunda lengua para todos los pueblos, que seguían manteniendo su propia lengua y usaban el esperanto como medio de comunicación entre naciones, dándole un carácter internacional.

Tomando la idea del esperanto como metáfora, el objetivo de este trabajo de tesis es el de ofrecer un marco de referencia para posibilitar, mejorar y agilizar la interoperabilidad y la mantenibilidad de los procesos de software, independientemente del lenguaje utilizado en su modelado. De esta forma, cada organización puede elegir el lenguaje que considere más adecuado en cada momento, con el fin de definir y documentar los procesos de software de la manera más apropiada para sistematizar su uso, sin que esa elección sea una opción irrevocable. Utilizando nuestro marco de referencia, la tarea de interoperar con procesos modelados en lenguajes diferentes y mantener o actualizar dichos procesos se convierte en una actividad menos ardua y más robusta y consistente, puesto que en dicho marco, como vamos a ver a lo largo de todo este trabajo de tesis, se especifica ese nuevo lenguaje, que se convierte en lenguaje base para el resto de los lenguajes, pero como tal es un lenguaje en sí mismo, y se disponen de todas la herramientas de soporte necesarias para que el uso de esta nueva lengua base sea fácilmente asumible por cualquier organización de software, sin la necesidad de disponer de experiencia y conocimiento específico.

De esta forma, una organización que desee migrar el lenguaje utilizado en el modelado de sus procesos de software podrá hacer uso de este marco de referencia por lo que nos facilita la mantenibilidad. Por otro lado, si en una misma empresa, departamentos diferentes, que utilizan diferentes lenguajes de modelado de procesos de software, quieren hacer interoperar sus procesos, o incluso, ampliando el contexto, en lugar de una única empresa se plantean empresas diferentes, donde la probabilidad de haber elegido el mismo lenguaje de modelado de procesos de software es todavía menor, deseen también esa interoperabilidad de los procesos en busca de una mayor competitividad conjunta, el marco de referencia que aquí se plantea es una solución adecuada y de fácil adopción para ellos.

Por último, nos gustaría destacar que este planteamiento, a diferencia de las propuestas de estandarización más comunes que se han ido sucediendo durante estos años, ofrece un enfoque diferente, en favor de la diversidad de lenguajes de modelado de procesos de software y de la libertad de elección de cada organización, facilitando la interoperabilidad y la mantenibilidad entre los procesos de software no enfocándose en la homogeneización del lenguaje de modelado de procesos de software utilizado, sino en la definición de un marco de referencia que lo soporte.

1.4. Estructura de la tesis

A continuación se presenta la manera en la que se estructuran los siguientes capítulos en esta memoria de tesis siguiendo la siguiente pauta de exposición: en primer lugar estudiaremos en profundidad cuál es la situación actual en cuanto a las propuestas de lenguajes de modelado de procesos de software se refiere para, a partir de ahí, establecer los principales objetivos que se desean alcanzar en el desarrollo de este trabajo de tesis. Seguidamente, se expone y profundiza la propuesta de solución, abarcando tanto los aspectos más teóricos de la misma como las particularidades más prácticas. Así, el resto de esta memoria se constituye de la siguiente manera:

El capítulo 2 ofrece una visión general del estado del arte en el dominio de los lenguajes de modelado de procesos de software. Debido a que se trata de un campo ampliamente estudiado por la comunidad científica, se ha limitado el estudio a la última década. Este estado del arte permite hacer un análisis sobre la cantidad de lenguajes existentes, las intenciones que han motivado su creación, así como la tendencia actual en la definición de nuevos lenguajes.

Con la visión alcanzada después del estudio del estado del arte, en el capítulo 3 se establece el problema a resolver, planteando los aspectos que han determinado el talante con el que se afronta, el contexto en el que se ha desarrollado, así como las principales influencias que han determinado la solución propuesta, ofreciendo una primera vista de la misma.

A partir de aquí, una vez establecidos los objetivos derivados del problema a resolver, en los siguientes tres capítulos se va a profundizar en cada uno de los elementos en los que se sustenta el marco de referencia que constituye la solución propuesta. Para ello, en primer lugar en el capítulo 4 se define un metamodelo que permite especificar el lenguaje de modelado de procesos de software propuesto como lenguaje base al que denominaremos INROMA (INteROperabilidad y MAntenibilidad), y que constituye una pieza clave de dicho marco. Esto es así porque, tal y como hemos comentado en secciones previas, en sí mismo es un lenguaje de modelado de procesos de software sencillo y de fácil aprendizaje, especialmente apto para organizaciones con escaso conocimiento de ingeniería de procesos, pero además constituye un soporte fundamental para alcanzar la interoperabilidad de procesos de software y la mantenibilidad de los mismos, bastiones ambos de nuestro marco de referencia.

En el capítulo 5 se establece, como segundo elemento clave, el método que permite el uso de INROMA como lenguaje común que fomenta la interoperabilidad y la mantenibilidad. Aquí se establecen las bases teóricas con las que se podría hacer uso del potencial del marco de referencia a partir de cualquier lenguaje de modelado de procesos de software existente, y se muestra, a modo de ejemplo, su uso para tres lenguajes concretos que hemos elegido por su representatividad en las empresas con las que trabajamos.

Y para terminar de completar el marco de referencia, en el capítulo 6 se define la tercera pieza que lo conforma: las transformaciones necesarias que constituyen el nexo de unión entre los diferentes lenguajes de modelado de procesos de software. Para ello, también a modo de ejemplo, se especifican algunas de las transformaciones para los lenguajes utilizados en el anterior capítulo y se establece el método a seguir para la definición de las mismas en el caso de incorporar nuevos

lenguajes en nuestro marco de referencia.

Todo este fundamento teórico se refrenda con la parte práctica de este trabajo de tesis, ya que todo el marco de referencia especificado y desgranado en estos tres últimos capítulos no alcanzaría el impacto deseado en las organizaciones de software si para ello no contáramos con una herramienta que diera soporte a su automatización. Dicha herramienta, a la que hemos denominado MONETA, se expone con detalle en el capítulo 7.

En el capítulo 8 se refieren dos casos de estudio en entornos empresariales que respaldan las necesidades que habían derivado el problema a resolver y se muestra como, a raíz de ellos, han surgido nuevos proyectos de transferencia a los que ya se está aplicando el marco de referencia resultado de este trabajo de tesis. Además, en el capítulo 9 se reseñan, a modo de conclusiones, las aportaciones obtenidas con este trabajo de tesis, la forma como se han ido alcanzando los objetivos planteados, y las principales líneas de trabajo futuras derivadas de los resultados logrados.

Para finalizar, en los anexos se amplían algunos de los contenidos previamente presentados. En los anexos A, B y C se incluyen la totalidad de las transformaciones de los lenguajes de modelado de procesos de software que se han utilizado como ejemplo en los capítulos 5 y 6. En el anexo D se presenta un breve manual de la herramienta MONETA desarrollada para dar soporte al marco teórico. Finalmente, en el anexo E se expone la trayectoria investigadora de la autora de este trabajo de tesis.

1.5. Conclusiones

En este capítulo se han introducido las definiciones relacionadas con los procesos de software más utilizadas desde diferentes puntos de vista, lo cual muestra su importancia como activo fundamental de una organización. También se ha expuesto la necesidad de definir y documentar los procesos de software, y cómo ésta queda reflejada en los diferentes estándares y modelos de referencia más utilizados por la industria. La evolución de los procesos y sus lenguajes de definición ha sido muy importante, existiendo actualmente una gran cantidad de ellos, no únicamente desde el punto de vista de ingeniería del software, sino también como elemento de negocio en una empresa que tiene como fin el desarrollo de aplicaciones. Elegir el lenguaje a utilizar para la representación de procesos es una tarea complicada y confusa ante la gran cantidad de opciones que se presentan. Pero, ¿por qué elegir? Sería muy interesante ser capaces de utilizar el lenguaje más adecuado en cada momento en función de las necesidades concretas de una organización en un determinado aspecto, y tener la posibilidad de usar varios lenguajes de modelado de procesos de software al mismo tiempo e indistintamente, permitiendo que los procesos de software modelados con diferentes lenguajes puedan interoperar. Ante esta situación, se ha presentado como objetivo de este trabajo de tesis el desarrollo de un marco de referencia para potenciar la interoperabilidad y mantenibilidad de los procesos de software. Por último se ha establecido una visión de cuál va a ser la estructura que a partir de este punto va a conformar esta memoria de esta tesis.

Capítulo 2

Estado del arte

Tal y como ha quedado explicitado en el capítulo anterior, la motivación de esta tesis consiste en definir y desarrollar un marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos de software, de forma que la elección de un lenguaje de modelado de procesos de software por parte de una organización no sea un vínculo estricto y rígido, sino que sea posible seleccionar aquél que más se adecue a las necesidades específicas de un momento determinado, pero que es posible evolucionar en el tiempo, e incluso interoperar con procesos de software que se encuentren modelados con otro lenguaje.

Para poder alcanzar dicho objetivo, el primer paso es conocer el estado de la situación actual en cuanto a los lenguajes de modelado de procesos de software se refiere, para lo que se ha llevado a cabo una revisión sistemática de la literatura. Es a raíz de los resultados y conclusiones obtenidas en este estudio de donde se extraen de forma más específica cuáles van a ser los objetivos de este trabajo de tesis.

Con este propósito, en este capítulo se aborda el estudio de la situación actual. Para ello, en la primera sección se describe el contexto del estudio, en el que se exponen otros estudios previos realizados y el método con el que se ha desarrollado la revisión que aquí se presenta. En la segunda sección se resumen los trabajos más relevantes encontrados. A continuación, en la tercera sección se realiza un análisis de la situación en cuanto a los lenguajes de modelado de procesos de software. Debido principalmente a que este estudio se llevó a cabo al comienzo de este trabajo de tesis, en la siguiente sección se realiza una actualización del estudio para los últimos años, mostrando que el análisis sigue siendo válido con los nuevos lenguajes encontrados. Y, por último, en la última sección de este capítulo se refieren las principales conclusiones del mismo.

2.1. Contexto del estudio de la situación actual

Después de la exposición de Osterweil [Osterweil, 1987] en la que decía que los procesos de software son software también, empezaron a aparecer propuestas de lenguajes de modelado de procesos de software. Ya desde finales de los años 80 la comunidad del software ha mostrado un creciente interés en encontrar la mejor forma de describir y modelar los procesos para que éstos sean entendidos e institucionalizados dentro de las organizaciones.

Antes de continuar vamos a establecer qué entendemos por modelo de procesos, ya que a lo largo de este capítulo utilizamos casi como sinónimos los términos modelado, descripción y representación de procesos de software. Un modelo de proceso es *una descripción abstracta de un proceso actual o propuesto, que representa elementos seleccionados del mismo que son considerados importantes para el propósito del modelo y pueden ser ejecutados por una persona o una máquina* [Curtis *et al.*, 1992]. Es decir, un modelo de proceso es una descripción del mismo a un determinado nivel. Podemos definir un lenguaje de modelado como aquél que se utiliza para expresar información o conocimiento con una estructura que está definida por un conjunto consistente de reglas, pudiendo ser gráfico o textual. El resultado del uso de un lenguaje de modelado de procesos es un modelo de proceso, y las reglas se utilizan para interpretar el significado de cada uno de los componentes de dicho modelo. A lo largo de esta memoria de tesis utilizaremos en muchas ocasiones como sinónimos los términos modelado, descripción y representación, si bien el vocablo más apropiado en el contexto que nos ocupa es el de modelado.

Una vez aclarada la terminología, nos centramos en los lenguajes de modelado de procesos de software, que constituyen en primera instancia nuestro objetivo de estudio para este estado del arte de la situación actual. Con el objetivo de enmarcar más claramente dicho estudio, a continuación por un lado se describen los estudios del estado del arte previos a este trabajo de tesis que se han encontrado en la literatura científica, lo que justifica la necesidad de realizar un nuevo estado del arte y, por otro lado, se describe el método con el que este estudio se ha llevado a cabo: una revisión sistemática de la literatura.

2.1.1. Antecedentes del estudio de la situación actual

Para poder comenzar este trabajo de tesis, se inició una búsqueda de los estudios del estado del arte que nos sirvieran de punto de partida. A pesar de que, tal y como se ha comentado anteriormente, la representación de los procesos de software ha sido un tema estudiado durante muchos años, existen muy pocos trabajos que expongan el estado del arte en este tema. A continuación vamos a resumir los más representativos.

Zamli [Zamli, 2001] realizó un estudio sobre el estado del arte de lo que denominó la segunda generación de Lenguajes de Modelado de Procesos o PMLs (Process Modeling Languages), entendiendo como tal aquellos lenguajes que se habían publicado después de 1996 [Sutton Jr. y Osterweil, 1997]. Este trabajo propone una simple clasificación en base al soporte para la promulgación de procesos. Clasifica un PML como no-promulgable, simulado o promulgable. Para entender mejor esta clasificación, debemos clarificar el concepto de promulgación en este

contexto. En el ámbito del Derecho, promulgar consiste en publicar formalmente una ley u otra disposición de la autoridad, a fin de que sea cumplida y hecha cumplir como obligatoria. En el dominio de la ingeniería de procesos podemos verlo como una analogía, puesto que consiste en una publicación de un proceso con un nivel de granularidad tal que permita un control exhaustivo del mismo, paso a paso, sin llegar necesariamente a la capacidad de ejecución del proceso. Se trata de un punto intermedio entre la mera publicación y la ejecución detallada de un proceso, con lo que no se dispone de la información detallada para ello. En concreto, en este trabajo los PMLs no-promulgables únicamente soportan el modelado de procesos, pero no su promulgación. Los PMLs simulados permiten una simulación de alto nivel de los procesos, pero no proponen un control de granularidad fina sobre los procesos de software, y se utilizan para realizar análisis estructurales de esos modelos. Finalmente, los PMLs promulgables son aquellos que permiten que un modelo de proceso sea promulgado para controlar el proceso de software, aunque no dispongan de un enlace explícito con la ejecución del mismo.

Algunos años después, el mismo autor [Zamli e Isa, 2004] estudió en profundidad la clasificación de los PMLs mediante el uso de una taxonomía y las áreas de soporte descritas en su trabajo anterior, listando las características que cada uno de los lenguajes estudiados cubrían.

En [Bendraou *et al.*, 2010] se realizó un estudio comparativo de seis lenguajes basados en UML para procesos de software, en el que los autores seleccionaban los principales requisitos identificados en la literatura para los lenguajes de modelado de procesos de software, y evaluaban esos seis lenguajes mediante estos requisitos y los comparaban entre ellos.

En la misma línea, en [Henderson-Sellers y Gonzalez-Perez, 2005] se examinan de forma crítica cuatro metamodelos que han sido construidos para sustentar y formalizar metodologías, entendiendo como tales el conjunto de procesos que se hace real, es decir, se instancia, en el contexto de una organización específica. A raíz de este análisis, los autores proponen un nuevo enfoque para el desarrollo del software y evaluación de capacidad que unos años más tarde quedará recogido en el estándar ISO 24744 [ISO/IEC, 2007a].

El hecho de que se encontraran tan pocos estudios previos y que aquellos que existen, o bien no son recientes o realizan una cobertura de trabajos muy pequeña, motivó la realización de un estudio propio del estado del arte como un punto de partida para conocer las diferentes formas de representación de procesos que han sido propuestas. En el siguiente apartado se describe tanto el método utilizado en el estudio de este trabajo de tesis como la ejecución del mismo.

2.1.2. Métodos y propuestas de mejora para una revisión sistemática de la literatura

Para llevar a cabo el estudio de la situación actual se optó por realizar una revisión sistemática de la literatura. Este método permite la identificación, evaluación e interpretación de todos los datos de investigación relevantes para una determinada pregunta de investigación (Reseach Question, RQ) en un área de investigación específica. La guía propuesta por Kitchenham se encuentra entre las más ampliamente aceptadas en el campo de la ingeniería del software [Kitchenham y Charters, 2007], aunque en los últimos años se han realizados diferentes críticas

y propuestas de mejoras a la misma. A continuación se nombran las aportaciones al método de Kitchenham más significativas.

La propia autora en [Kitchenham *et al.*, 2010] hace un análisis de cómo se están utilizando las revisiones sistemáticas en el ámbito del software tres años después de la publicación de su guía, concluyendo que el número de revisiones había aumentado significativamente, pero todavía no podía considerarse un método principal de investigación de ingeniería del software, puesto que, aunque cubre infinidad de temas, a menudo no evalúa la calidad de los estudios primarios obtenidos. Un trabajo muy similar fue el realizado en [Da Silva *et al.*, 2011], con una conclusión muy pareja: la mayoría de las revisiones sistemáticas no evalúan la calidad de los estudios primarios y no proporciona guías para los profesionales, lo que hace que disminuya su posible impacto en la práctica de la ingeniería del software.

En [Zhang *et al.*, 2011] el foco de atención se centra en la estrategia de búsqueda, puesto que se considera un paso crítico en la correcta aplicación del método de revisiones sistemáticas. Los autores defienden que es un paso que consume bastante tiempo y propenso a errores, por lo que debe ser cuidadosamente planificado y ejecutado. En el artículo se pretende mejorar este paso mediante la incorporación de los conceptos "quasi-gold standard" (QGS), que consiste en la recolección de estudios conocidos, y "quasi-sensitivity" para evaluar el rendimiento de la búsqueda. Estos mismos autores en [Zhang y Ali Babar, 2013] consideran la importancia de las revisiones sistemáticas desde el punto de vista empírico. En este artículo se pone de manifiesto que los investigadores están convencidos del valor de usar una metodología rigurosa y sistemática para las revisiones de la literatura. Sin embargo, consideran que es necesario establecer un equilibrio entre el rigor metodológico y el esfuerzo necesario.

En [Wohlin y Prikladniki, 2013] los autores argumentan la necesidad de realizar en las búsquedas de artículos un enfoque "bola de nieve", según el cual los estudios incluidos podrían ser ampliados si, además de los trabajos obtenidos con el método Kitchenham, se tienen en cuenta tanto las listas de referencias de estas publicaciones, lo que supone una vista hacia atrás, como las citas de estas publicaciones, que proporcionarían una vista hacia adelante.

Como resultado de todas las críticas recibidas, Kitchenham et al. investigan si las guías deberían ser modificadas [Kitchenham y Brereton, 2013]. En este artículo se exponen varias conclusiones en cuanto a la mejora del método: recomienda retirar el consejo de utilizar preguntas estructuradas para construir las cadenas de búsqueda e incluir el consejo de usar el concepto "quasi-gold standard", basado en una búsqueda limitada manual para ayudar a la construcción de las cadenas de búsqueda y la posterior evaluación del proceso de búsqueda. También comenta que las herramientas de análisis textual posiblemente podrían ser útiles para la decisión de incluir y excluir, así como para la definición de la cadena de búsqueda, pero debería realizarse una evaluación más rigurosa de las mismas. En cuanto al uso de herramientas para la gestión del proceso de revisión sistemática, considera que necesitan de una validación independiente de las que existen. Por último concluye que la evaluación de la calidad de los estudios que usan métodos empíricos sigue siendo un problema importante.

2.1.3. Estudio de la situación actual

Como se ha introducido en el apartado anterior, para el desarrollo del estudio de la situación actual se optó por realizar una revisión sistemática de la literatura, y el método seguido fue el original de Kitchenham, si bien es cierto que utilizamos el consejo de "quasi-gold standard" durante la ejecución de las búsquedas.

El método establece que una revisión sistemática debería incorporar tres fases: planificación, ejecución y presentación de resultados. La primera fase, la planificación, consiste en el desarrollo del protocolo de revisión, así como el desglose de cómo los investigadores que participan en la revisión deberían trabajar e interactuar para llevarla a cabo. Este protocolo prescribe un procedimiento controlado para la ejecución de la revisión e incluye preguntas de investigación (RQs), estrategias de búsqueda y evaluación, criterios de inclusión y exclusión, evaluación de la calidad de los trabajos, formulario de recogida de datos y métodos de análisis. La segunda de las fases se centra básicamente en la ejecución del protocolo definido en la fase anterior. Finalmente, la última fase describe cómo el informe final ha sido elaborado.

La revisión sistemática en su totalidad fue publicada en [García-Borgoñón *et al.*, 2014a]. En este artículo se expone en detalle el protocolo definido, que incorpora cada una de las tareas anteriormente enumeradas. La pregunta de investigación principal fue: "¿Cuál es el estado del arte de los lenguajes de modelado de procesos de software?", siendo posteriormente reformulada en las siguientes:

- RQ1. ¿Qué lenguajes de modelado de procesos de software se han definido? ¿Por qué?
- RQ2. ¿Cuál es la tendencia actual cuando se selecciona una tecnología base para definir un nuevo lenguaje de modelado de procesos de software (Software Process Modeling Language, SPML)?
- RQ3. ¿Cuáles son las limitaciones de la investigación actual?

Para poder resolver las cuestiones de investigación se realizó una búsqueda exhaustiva de trabajos y artículos de revistas y congresos en los principales bibliotecas digitales que trataran el tema de la descripción o modelado de procesos de software. En la búsqueda se utilizó la expresión Booleana "*A* ***AND*** *(B1* ***OR*** *B2* ***OR*** *B3)* ***AND*** *(C1* ***OR*** *C2)*" donde cada una de las componentes representa los conceptos recogidos en la tabla 2.1.

A. software process	B1. description	C1. metamodel
	B2. definition	C2. language
	B3. modeling	

Tabla 2.1: Expresiones de búsqueda

Tal y como se recomienda en el método de Kitchenham, el proceso de selección de los trabajos a incluir en la revisión sistemática se dividió en fases, en nuestro caso seis. Dichas fases, con el objetivo de ajustar la elección final de trabajos, disponían de criterios de inclusión y exclusión de los mismos, lo que nos permitía avanzar en el proceso. Para alcanzar una mayor objetividad, los investigadores participantes en cada una de estas fases eran diferentes en número, incorporando así, por defecto, una revisión por pares (peer-review) de la selección de trabajos que se iba obteniendo. En la tabla 2.2 se resumen los objetivos y los participantes de cada una de las fases.

Fase	**Descripción de fase**	**Participación**
F1	Selección de los estudios en base a búsqueda	Investigador líder
F2	Criba: exclusión basada en fechas	Investigador líder
F3	Criba: exclusión basada en títulos, resúmenes y palabras clave	Dos investigadores
F4	Reunión de consenso	Todos los investigadores
F5	Análisis de relevancia: exclusión basada en texto completo	Todos los investigadores
F6	Reunión de consenso	Todos los investigadores

Tabla 2.2: Fases de inclusión

Los criterios de inclusión y exclusión definidos para cada una de las fases anteriores se resumen en la tabla 2.3.

Fase	**Criterios de inclusión / exclusión**
F1	No duplicado Sólo trabajo publicado Contiene las cadenas de búsqueda
F2	Fechas de publicación después del 2000
F3	No editoriales, prefacios, discusiones
F4	No resúmenes de tutoriales, workshops o paneles
F5	Sólo en inglés Texto completo obtenido
F6	Nuevo lenguaje o una modificación propuesta del lenguaje Ni estudios ni revisiones

Tabla 2.3: Criterios de Inclusión y Exclusión por fase

En la tabla 2.4 se refleja cómo, mediante la aplicación de los criterios anteriores, fueron disminuyendo el número de trabajos a incluir en el estudio de la situación actual a medida que se fue avanzando en las fases del estudio. Las referencias de los 46 artículos finalmente incluidos se encuentran disponibles en la sección de Referencias.

Fase	Incluidos	Eliminados
Encontrados	1929	752
F1	1177	534
F2	643	230
F3	413	233
F4	180	48
F5	132	82
F6	50	4
Incluidos	46	0

Tabla 2.4: Estudios incluidos y excluidos por fase

Seleccionados los artículos del estudio, se definió un esquema de caracterización para la recogida de la información más relevante de los mismos y facilitar así el proceso de análisis de los datos. Dicho esquema se muestra en la tabla 2.5.

Info básica	Título y autor
Info de publicación	Libro, revista, conferencia o documento técnico en el que se publicó el estudio
Año	Año en que se publicó el estudio
Lenguaje definido	Nombre del lenguaje definido en el estudio, el problema abordado, su descripción y motivación
Basado en	Tecnología o el lenguaje de referencia en que se basa este nuevo lenguaje y las ventajas y desventajas de otras alternativas
Trabajos relacionados	Nombre de los estudios citados y si el estudio tiene una sección de estado del arte
Citado por	Se identifica el estudio y si se hace referencia explícitamente en otros estudios seleccionados
Trabajo futuro	El estudio proporciona trabajo futuro y retos relacionados con las preguntas de investigación

Tabla 2.5: Esquema de caracterización de datos

En el artículo anteriormente referenciado[García-Borgoñón *et al.*, 2014a] se dispone de una información más detallada de los resultados de la ejecución del protocolo de revisión. Una vez descrito el proceso para llevar a cabo la revisión sistemática de la literatura, en la siguiente sección se llevará a cabo el análisis de los resultados obtenidos.

2.2. Análisis de los resultados

Para llevar a cabo el análisis de los resultados de la revisión se van a considerar las preguntas de investigación anteriormente formuladas, que van a ser detalladas una a una en los siguientes apartados.

2.2.1. RQ1. ¿Qué lenguajes de modelado de procesos de software se han definido? ¿Por qué?

El número de lenguajes para modelar procesos de software que se han propuesto en el tiempo es muy elevado y hay que tener en cuenta que a la hora de hacer la selección, hemos elegido únicamente aquellos que fueron definidos con el propósito explícito de ser utilizados como lenguajes de modelado de procesos de software, es decir, quedarían fuera de esta selección lenguajes como los Diagramas de Actividad de UML o BPMN que, si bien somos conscientes de que se están utilizando en la práctica en ciertas empresas con este fin, no fueron creados con este objeto. De hecho, como veremos más adelante, algunas de estas propuestas están basadas en algún lenguaje de propósito más general al que se incluye cierta semántica específica necesaria para los procesos de software. Es por ello por lo que, a la hora de clasificarlos y analizarlos, registramos la base sobre la que se constituía la nueva propuesta.

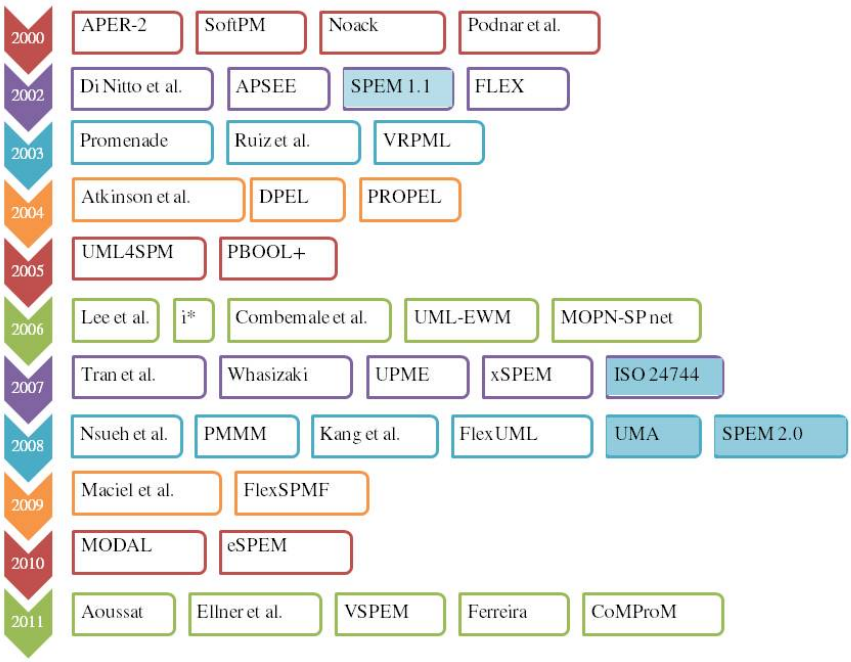


Figura 2.1: Resumen de los lenguajes de modelado de procesos de software

El primer paso para llevar a cabo este análisis era el de obtener una vista global de los lenguajes de modelado de procesos de software existentes. En esta vista se estableció como límite temporal las propuestas posteriores al año 2000 para poder acotar el trabajo, teniendo en cuenta que los únicos estados del arte de los que disponemos hacen referencia a los lenguajes anteriores y una vista de diez años hacia atrás es suficientemente representativa.

Al hacer la revisión de los 46 artículos finalmente seleccionados de acuerdo a los criterios establecidos, se observó que en muchos casos un artículo introducía una nueva propuesta de lenguaje, pero en otros casos varios artículos hacían referencia al mismo lenguaje.

Por otro lado, también nos encontramos con la situación de que los estándares de lenguajes de modelado de procesos de software desarrollados en este espacio de tiempo, no tenían una correspondencia directa y específica con ningún artículo, pero nos parecía importante que aparecieran de forma explícita en esta vista global de los lenguajes de modelado de procesos de software, por lo que fueron incorporados manualmente.

Con todo ello, en la figura 2.1 se presentan dicha vista global de los lenguajes de modelado de procesos de software existentes desde el año 2000, así como el año en el que se publicaron, marcando con un sombreado diferente aquellos lenguajes procedentes de estándares que se han incorporado.

Después de alcanzar esta primera vista de los lenguajes de modelado de procesos existentes desde el año 2000, el siguiente paso consistió en extraer de los 46 trabajos seleccionados una breve descripción del lenguaje que en cada uno de ellos se definía y la motivación por la que dichos lenguajes fueron creados. El resultado de esta tarea se muestra de forma detallada en las tablas 2.6 a 2.10. Los lenguajes incorporados manualmente en la vista global anterior no fueron objeto de esta tarea, puesto que las especificaciones no formaban parte de los trabajos seleccionados.

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2000	APER-2 [Chen *et al.*, 2000]	Lenguaje Programación	Es un lenguaje centrado en el desarrollador (centrado en las personas) para facilitar el cálculo de la carga de trabajo y modelar múltiples actividades concurrentes	Sólo existían lenguajes centrados en producto o centrados en la actividad
2000	SoftPM [Min *et al.*, 2000]	Redes de Petri	Propone un PSEE (Entorno de Ingeniería de Software Centrado en Procesos) en base al formalismo de Redes de Petri de alto nivel para modelar los procesos y explotar un mecanismo de modelado multinivel para la representación de procesos de software	No existían PSEEs que permitieran diferentes maneras de representar procesos (fáciles de realizar y no ambiguas) y que además preservaran la formalidad
2000	Noack [Noack, 2000]	UML	Se propone un lenguaje basado en UML para gestionar el conocimiento en una organización de desarrollo de software de gran tamaño	No existía la posibilidad de modelar un proceso integrando prácticas de ingeniería y de gestión de forma que se puedan conseguir tareas gestionables, componentes intercambiables y proyectos comparables
2000	Podnar et at [Podnar *et al.*, 2000]	SPML	Propone una especificación y un lenguaje descriptivo (formal y basado en otros orientados a objetos para comunicar sistemas) para modelar procesos de software	Apuesta por las similitudes existentes entre los procesos de software y los sistemas de tiempo real, lo que justifica usar un estándar ITU-T[1]
2002	FLEX [Chen *et al.*, 2002]	SPML	Propone un lenguaje basado en restricciones, reglas y objetos, que suporta descripción semántica, simplicidad de uso, flexibilidad, escalabilidad, reutilización y distribución. También permite su análisis, es ejecutable y evolutivo	Los lenguajes existentes eran de muy bajo nivel por lo que su capacidad de comprensión y reutilización eran difíciles
2002	Di Nitto et al [Di Nitto *et al.*, 2002]	UML	Propone un lenguaje que ofrece la posibilidad de aplicar un subconjunto de UML como un SPML ejecutable mediante la transformación de modelos UML en workflows ejecutables	UML fue pensado como un lenguaje semi-formal, no ejecutable, pero debido a su gran popularidad, los autores han intentado alcanzar la ejecución a partir del mismo
2002	APSEE [Lima Reis *et al.*, 2002] [Reis *et al.*, 2002]	SPML Gráfico	Se trata de un SPML gráfico basado en gramáticas que da soporte a la flexibilidad (entendida como cambios dinámicos) en la ejecución de los procesos	Los PSEE existentes hasta la fecha no ofrecían la flexibilidad para la ejecución del proceso ni un soporte integrado de herramientas para su análisis y mejora continua (como por ejemplo la simulación o la facilidad de reutilización)

Tabla 2.6: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2003	PROMENADE [Franch y Ribó, 2003]	UML	El lenguaje propuesto cubre la reutilización de procesos (la capacidad de construir nuevos procesos mediante la composición a partir de otros existentes) y la reutilización de proceso (la capacidad de construir procesos genéricos que pueden ser reutilizados más allá de los ya existentes)	Los lenguajes existentes tenían las siguientes limitaciones: expresividad, estandarización, modularidad y flexibilidad, motivos por los que los autores intentaron mejorarlos
2003	Ruiz et al [Ruiz *et al.*, 2003]	UML	Propone un lenguaje para el uso de MOF (Meta Object Facility) [OMG, 2014a] y XMI (XML Metadata Interchange) [OMG, 2014b] para representar los procesos de software	No había una propuesta tecnológica concreta aplicada a SPML
2003	VRPML [Zamli y Lee, 2003]	SPML Gráfico	Se trata de un SPML de control de flujo basado en la teoría de grafos centrado en el modelado visual	Los SPMLs existentes carecían de creación dinámica y asignación de tareas y recursos, así como del apoyo a los temas de sensibilización y visualización (tener SPML promulgables)
2004	Atkinson et al [Atkinson *et al.*, 2004] [Atkinson *et al.*, 2007]	SPML	Es un SPML basado en el control que trata de cubrir la simplicidad, flexibilidad, expresividad y promulgación	Los autores trataron de apoyar el desarrollo del modelo evolutivo y la verificación
2004	DPEL [Chou, 2004]	SPML	Propone un modelo promulgación de proceso descentralizado para construir un PSEE distribuido	Los PSEEs existentes eran centralizados, por lo que presentaban un único punto de fallo y un cuello de botella en relación con la promulgación de los procesos
2004	PROPEL [Hagen M., 2004] [Bendraou *et al.*, 2006]	UML	Propone un lenguaje para proporcionar conceptos que asisten a una descripción semiformal y a la relación entre los patrones de procesos	Los autores trataron de dar soporte a la flexibilidad de los procesos a través de patrones de procesos
2005	UML4SPM [Bendraou *et al.*, 2005] [Bendraou *et al.*, 2006] [Bendraou *et al.*, 2007b] [Bendraou *et al.*, 2009] [Bendraou *et al.*, 2012]	UML	Es un metamodelo MOF-compatible que tiene cuatro objetivos: expresividad, claridad, precisión y ejecutabilidad	Los autores trataron de cubrir las limitaciones de OMG SPEM 1.1 que aún no habían alcanzado el nivel requerido para la especificación de los modelos promulgables
2005	PBOOL+ [Thu *et al.*, 2005]	SPML Gráfico	Es un SPML gráfico centrado en la reutilización de procesos de software a través de componentes de procesos y patrones de procesos	Los autores estudiaron el problema de la caracterización de componentes de proceso reutilizables

Tabla 2.7: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes (cont)

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2006	Lee et al [Lee *et al.*, 2002]	UML	Propone un enfoque dirigido por metamodelos basados en UML para definición de políticas de asignación de tareas en procesos de software	No había ninguna propuesta orientada a políticas de asignación de tareas en el ámbito de los procesos de software
2006	i* [Cares *et al.*, 2006]	SPML	Propone un lenguaje basado en agentes para modelar procesos como basado en objetivos para diseñarlos	No había ninguna propuesta en relación con las tecnologías basadas en agentes aplicadas en el modelado de procesos de software
2006	Combemale et at [Combemale *et al.*, 2006]	SPEM 1.1	Propone una extensión de SPEM 1.1 especializada en contenidos semánticos	Los autores trataron de cubrir las limitaciones de OMG SPEM 1.1 que formaliza sólo en parte la semántica y no ayuda a la construcción de un modelo de proceso
2006	UML-EWM [Debnath *et al.*, 2006]	UML	Propone un lenguaje para cubrir el modelado de procesos como flujos de trabajo	No había ninguna propuesta que establezca una correspondencia entre una metodología de desarrollo y un proceso de flujo de trabajo
2006	MOPN-SP-net [Ge *et al.*, 2006] [Ge *et al.*, 2008]	Redes de Petri	Propone una multi-vista de modelo de base de procesos de software basada en múltiples objetos de Redes de Petri	Los autores sostienen que las Redes de Petri y las multi-vistas de un proceso de software parecían ser similares
2007	Tran et at [Tran *et al.*, 2007]	UML	El lenguaje permite una representación explícita de patrones en modelos de procesos	Los autores trataron de cubrir la construcción y mejora de modelos de procesos
2007	Washizaki [Washizaki, 2007]	SPEM 1.1	Propone un lenguaje que se ocupa de establecer claramente el interés común y la variabilidad en los flujos de trabajo de proceso cuando se modelan como diagramas de actividad UML	Las propuestas existentes no siempre se habían orientado hacia la optimización global y no dieron lugar a estructuras de modelos de procesos que fueran aplicables con carácter general
2007	UPME [Wu *et al.*, 2007]	UML	El lenguaje permite modelar procesos de software en tres pasos: creación del metamodelo, instanciación en un modelo y compilación del mismo para convertirlo en esqueletos de código orientado a objetos para su promulgación	Los autores trataron de cubrir la reducción de la complejidad a través de la reutilización de procesos mediante el uso de técnicas de metamodelado

Tabla 2.8: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes (cont)

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2007	xSPEM [Bendraou *et al.*, 2007a]	SPEM 2.0	Trata de extender SPEM 2.0 para permitir que los modelos de procesos puedan ser comprobados a través de un mapeo con Redes de Petri y controlados a través de una transformación en BPEL	Los autores proporcionaron una definición de un Executable SPEM 2.0
2008	Hsueh et at [Hsueh *et al.*, 2008]	UML	Propone un lenguaje para definir, verificar y validar los procesos de una organización	Los autores proponen un enfoque para validar los procesos en un entorno de simulación
2008	PMMM [Jablonski *et al.*, 2008]	Powertypes	Propone un lenguaje para la gestión de procesos específicos de dominio	Los autores trataron de separar los principios de modelado de procesos en general de lenguajes específicos de dominio
2008	Kang et at [Kang *et al.*, 2008]	XML	Propone un lenguaje para la descripción de componentes de procesos evolutivos	No existía un método sistemático para describir un componente de proceso de software evolutivo
2008	FlexUML [Martinho *et al.*, 2008]	UML	Propone un enfoque en dos etapas para modelar la flexibilidad controlada en los procesos de software	Había una necesidad de los participantes del proceso de poder expresar y controlar la cantidad de flexibilidad permitida en aquellos procesos
2009	Maciel et at [Maciel *et al.*, 2009]	SPEM 2.0	Propone un enfoque integrado para el modelado de procesos de MDA y la promulgación basado en algunos conceptos de especialización recogidos en SPEM 2.0	Había una falta de terminología y notación para abordar los aspectos de diseño de un proceso MDA estándar porque las herramientas sólo se centraban en la definición y ejecución de transformaciones de modelos
2009	FlexSPMF [Martinho *et al.*, 2009]	SPEM 2.0	Propone un marco para el modelado y la flexibilidad en el aprendizaje de los procesos de software	Los autores trataron de mejorar la flexibilidad de los procesos cuando se producen actividades dinámicas que deben evolucionar para adaptarse a los cambios
2010	MODAL [Koudri y Champeau, 2010] [Pillain *et al.*, 2011]	SPEM 2.0	Propone una extensión SPEM 2.0 para mejorar los modelos de procesos de co-diseño	Había una falta en los actuales SPML para lograr la integración de MBE (Model Based Engineering) en modelos de procesos de software y de sistemas
2010	eSPEM [Ellner *et al.*, 2010]	SPEM 2.0	Propone una extensión de SPEM 2.0 para modelar un comportamiento promulgable	Los autores defienden que SPEM 2.0 cubría una descripción muy gruesa de los procesos que tenían comportamiento

Tabla 2.9: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes (cont)

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2011	Aoussat [Aoussat *et al.*, 2011]	SPEM 2.0	Propone una extensión SPEM que define conectores explícitos del proceso de software que facilitan, se adaptan y controlan las interacciones de procesos de software	Los autores trataron de cubrir las limitaciones de OMG SPEM 2.0 acerca de los conceptos de arquitectura para el modelado de procesos de software basados en arquitecturas de software
2011	Ellner et at [Ellner *et al.*, 2011]	OMG's FUML	Propone una máquina de ejecución distribuida basada en FUML para la promulgación de los modelos de procesos de software	FUML era insuficiente para ejecutar modelos de procesos de software para impulsar proyectos realistas con los equipos grandes y geográficamente extendidos
2011	Ferreira [Ferreira *et al.*, 2011]	UML	Propone un enfoque de modelado para el diseño de procesos de software y su implementación orientado a procesos grandes y complejos	No había ninguna propuesta que cubriera el incremento y la complejidad de los procesos
2011	vSPEM [Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]	SPEM 2.0	Propone un lenguaje que modela la variabilidad en los procesos de software	No había un SPML que incorporara la riqueza en la variabilidad de forma que los procesos pudieran ser adaptados a los entornos y objetivos específicos de cada proyecto
2011	CoMProM [Golra y Dagnat, 2011]	UML	Propone un lenguaje basado en componentes para automatizar los procesos de desarrollo de software que permite la flexibilidad para tener procesos como componentes	Había muchos enfoques que ofrecían los procesos como componentes pero no proporcionaban la semántica de ejecución de estos componentes del proceso

Tabla 2.10: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes (cont)

Descripción del problema	**Referencia**
Documentar los procesos para mejorar la comunicación entre la gente en situaciones reales	[Min *et al.*, 2000]
Gestionar el conocimiento en una gran organización de desarrollo de software	[Noack, 2000]
Dar una representación a alto nivel para no expertos para facilitar su uso	[Chen *et al.*, 2002]
Ofrecer una representación de procesos que sea comprendida por la gente y formal para evitar ambigüedades	[Min *et al.*, 2000]
Definir, verificar y validar los procesos de una organización	[Hsueh *et al.*, 2008]
Crear un formalismo para representar e intercambiar procesos de software	[Bendraou *et al.*, 2012]
Extender el nivel de abstracción para mejorar la comprensión del modelo y reducir su esfuerzo en mantenerlo	[Hsueh *et al.*, 2008]
Permitir el análisis y simulación de procesos de software	[Podnar *et al.*, 2000]
Controlar la participación de personas en las actividades de desarrollo de software	[Lima Reis *et al.*, 2002] [Reis *et al.*, 2002]
Dar soporte al modelado con mayor nivel de detalle del comportamiento y cubrir el ciclo de vida del modelo	[Ellner *et al.*, 2010]
Modelar un proceso que integra prácticas de ingeniería y de gestión	[Noack, 2000]
Intercambiar y subcontratar resultados y componentes	[Noack, 2000] [Podnar *et al.*, 2000]
Reutilizar procesos de software	[Chen *et al.*, 2002] [Hagen M., 2004] [Thu *et al.*, 2005] [Tran *et al.*, 2007] [Wu *et al.*, 2007]
Validar un proceso antes de su promulgación	[Atkinson *et al.*, 2004] [Atkinson *et al.*, 2007]
Facilitar la automatización de la gestión de los procesos de software	[Lima Reis *et al.*, 2002] [Reis *et al.*, 2002]
Aplicar el concepto de líneas de producto software a los procesos de software	[Washizaki, 2007]
Incluir conceptos de arquitecturas software en el modelado de procesos de software	[Aoussat *et al.*, 2011]
Modelar la flexibilidad de los procesos de software	[Martinho *et al.*, 2008]
Gestionar el incremento en tamaño y complejidad de los grandes sistemas de procesos	[Ferreira *et al.*, 2011]
Apoyar la adaptación del proceso de software para cumplir con los objetivos y características específicas demandadas por las organizaciones y proyectos	[Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]
Incorporar otras actividades en un proceso de software más allá de la generación de código	[Maciel *et al.*, 2009]
Lograr la integración de MBE en modelos de procesos de sistemas y de software	[Koudri y Champeau, 2010]
Apoyar la promulgación explotando todo el potencial de MDE a través del uso de modelos	[Pillain *et al.*, 2011] [Golra y Dagnat, 2011]
Crear un estándar de facto para modelar y establecer procesos de software	[Zamli y Lee, 2003]
Definir una terminología y notación estándar para abordar aspectos de diseño de un proceso basado en MDA	[Maciel *et al.*, 2009]

Tabla 2.11: Descripción de los problemas tratados en los SPMLs existentes

Para finalizar la resolución de esta primera cuestión de investigación, la tabla 2.11 resume los principales problemas que los lenguajes intentan resolver, agrupados en cuatro grandes temáticas principales: soporte para la documentación estática del proceso, soporte para el análisis y gestión de los procesos previos a su promulgación, soporte para los procesos dinámicos y herramientas y entornos para procesos de software.

2.2.2. RQ2. ¿Cuál es la tendencia actual cuando se selecciona una tecnología base para definir un lenguaje de modelado de procesos de software (SPMLs)?

Para resolver esta cuestión hemos intentado descubrir cómo los SPMLs han sido clasificados en la literatura. Históricamente se han planteado diferentes generaciones de SPMLs como una posible manera de catalogarlos [Bendraou *et al.*, 2010]. La primera generación hacía referencia a aquellos lenguajes de modelado de procesos de software que estaban basados en redes de Petri, en reglas o definidos como un lenguaje de programación. Se centraban fundamentalmente en la ejecución del proceso y en su formalidad, lo que los convertía en algo complejo, inflexible y difícil de entender. La segunda generación coincide con el momento en el que UML llega a su madurez como un estándar en la industria del software. La idea que subyace en estos lenguajes es la de explorar la posibilidad de utilizar UML, en concreto los Diagramas de Actividad de UML, para modelar los procesos de software. Algunos lenguajes basados en UML aparecieron, caracterizados principalmente por la expresividad y falta de formalidad, lo que ha influido en la capacidad de ejecución de los modelos de procesos.

Sin embargo, de acuerdo con lo que se observa en las tablas 2.6 a 2.10, podemos concluir que hablar de generaciones no es particularmente apropiado, ya que, si bien la mayor parte de los lenguajes de la segunda generación han sido lanzados en fechas posteriores, algunas de las propuestas más recientes se pueden clasificar dentro de la primera generación. Por ello, se establece una nueva taxonomía de SPMLs, mediante la consideración de la tecnología base utilizada en el desarrollo del lenguaje. Hemos estructurado los SPMLs en tres grupos de acuerdo con los resultados de la revisión, tal y como se muestra en la figura 2.2. El primer grupo corresponde a los lenguajes basados en gramáticas, que incluye todos los estudios relacionados con lenguajes formales, matemáticos o de programación, ya sea mediante reglas o restricciones. Un segundo grupo contiene los SPMLs en las diferentes versiones de UML. Por último, en el tercer grupo se han considerado los lenguajes basados en modelos.

La figura 2.3 muestra la clasificación de todos los lenguajes obtenidos en la RQ1 de acuerdo a esta taxonomía. En esta figura los SPMLs basados en gramáticas se muestran con un recuadro más fino, los basados en UML se muestran con un color de relleno y los basados en modelos se muestran con un recuadro más grueso.

Un asunto muy relevante fue identificar la causa de la elección de la tecnología base en cada SPML por parte de los autores de las propuestas, justificando así la decisión de su adopción. De esta forma, en las tablas 2.12 y 2.13 se resumen las principales ventajas e inconvenientes que dichos autores han señalado en cuanto a la elección de la tecnología base de sus lenguajes.

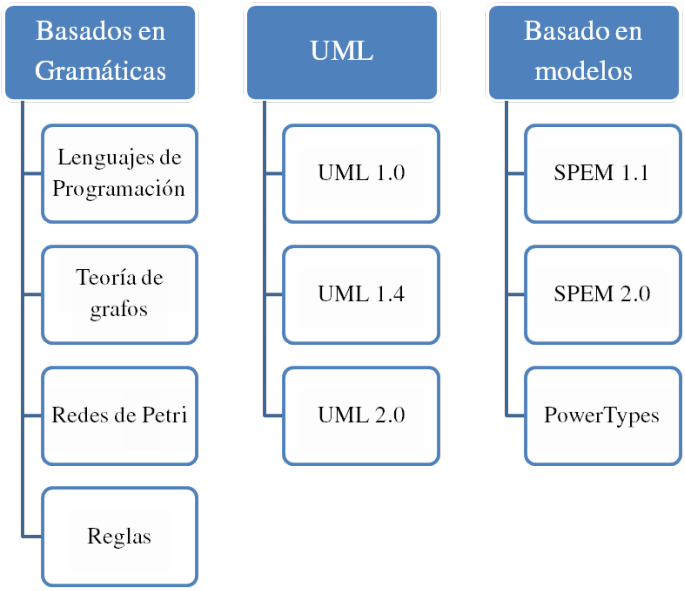


Figura 2.2: Clasificación de formalismos para SPMLs

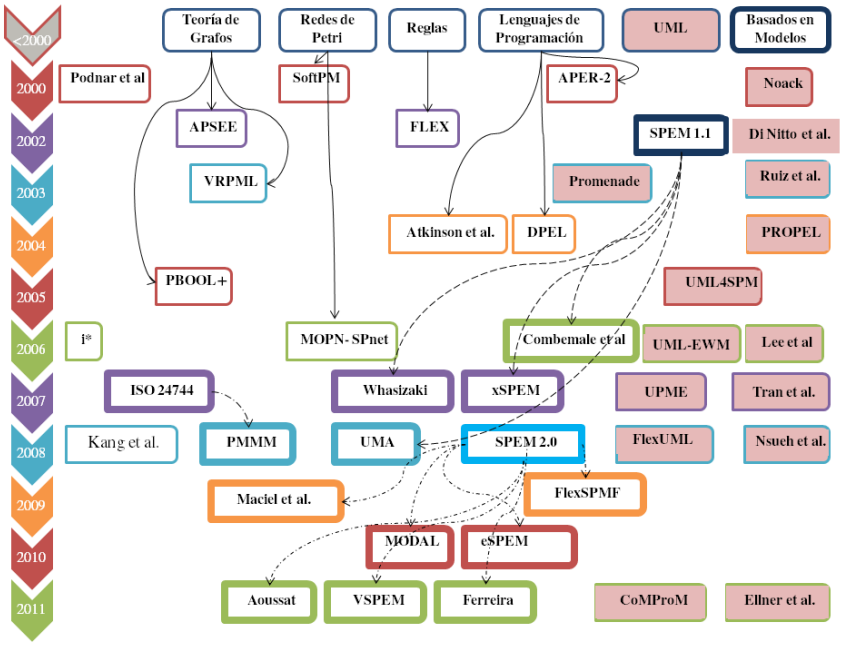


Figura 2.3: Relaciones y tecnologías base de los lenguajes de modelado de procesos de software existentes

Paradigma	Ventajas	Desventajas
SPML basado en gramática (en general)	▪ Sintaxis y semántica precisas [Podnar *et al.*, 2000]	▪ Dificultad para usarlo en el modelado de un proceso de software [Min *et al.*, 2000] ▪ Falta de semántica ejecutable [Chen *et al.*, 2002] ▪ No existe una vista global del modelo de proceso a nivel funcional y de comportamiento [Chen *et al.*, 2002] ▪ Dificultad para su uso por personas [Di Nitto *et al.*, 2002] ▪ Falta de soporte para procesos distribuidos [Di Nitto *et al.*, 2002]
SPML basado en gramática (Redes de Petri)	▪ Representación gráfica [Min *et al.*, 2000] [Podnar *et al.*, 2000] ▪ Capacidad para representar la estructura estática y las propiedades dinámicas de los procesos de software [Min *et al.*, 2000]	▪ Sólo es comprensible para procesos sencillos [Min *et al.*, 2000] [Podnar *et al.*, 2000] ▪ Es inapropiado para describir entidades del modelo [Podnar *et al.*, 2000] ▪ Dificultad para ser usado por personas [Di Nitto *et al.*, 2002] ▪ Falta de soporte para procesos distribuidos [Di Nitto *et al.*, 2002] ▪ Demasiado complejo para cumplir el rol general de promulgación de un proceso de software [Hsueh *et al.*, 2008]
SPML basado en gramática (Reglas)	▪ Sintaxis precisa [Podnar *et al.*, 2000]	▪ Dificultad para ser usado por personas [Di Nitto *et al.*, 2002] ▪ Falta de soporte para procesos distribuidos [Di Nitto *et al.*, 2002] ▪ El orden de las tareas dentro del proceso no puede ser controlado [Atkinson *et al.*, 2004]

Tabla 2.12: Ventajas y desventajas de las tecnologías base para crear SPMLs

Paradigma	Ventajas	Desventajas
UML	▪ La popularidad y el atractivo para los ingenieros de software (no hay que aprender otro formalismo) [Di Nitto *et al.*, 2002] [Franch y Ribó, 2003] [Wu *et al.*, 2007] ▪ Es el lenguaje de modelado más usado [Bendraou *et al.*, 2005] como un estándar [Di Nitto *et al.*, 2002] ▪ Representación gráfica [Di Nitto *et al.*, 2002] ▪ Capacidad de extensión [Di Nitto *et al.*, 2002] [Hsueh *et al.*, 2008] ▪ Soporte de herramientas [Di Nitto *et al.*, 2002] [Bendraou *et al.*, 2005] ▪ Facilidad de comprensión [Di Nitto *et al.*, 2002] [Wu *et al.*, 2007] ▪ Soporte de enfoques basados en modelos [Wu *et al.*, 2007] ▪ Diferentes vistas para presentar un proceso [Hsueh *et al.*, 2008] ▪ Verificación de un modelo mediante OCL [Hsueh *et al.*, 2008]	▪ Falta de promulgación [Di Nitto *et al.*, 2002] [Bendraou *et al.*, 2012] ▪ Falta de semántica formal [Di Nitto *et al.*, 2002] ▪ No es interpretable por máquinas (o por no ser lo suficientemente preciso o por falta de detalle)[Di Nitto *et al.*, 2002]
Basadas en metamodelos	▪ Separación entre los principios generales de modelado y los lenguajes específicos de dominio [Jablonski *et al.*, 2008] ▪ Patrones de modelado implementables (powertypes y conceptos de tipo/uso) [Jablonski *et al.*, 2008]	

Tabla 2.13: Ventajas y desventajas de las tecnologías base para crear SPMLs (cont)

En base a esto podemos concluir que UML, en concreto los Diagramas de Actividad de UML, se han considerado una posible tecnología base ya que constituye un estándar en ingeniería del software. Sin embargo, su principal debilidad es la falta de capacidad para ejecutar procesos, aunque algunos diagramas UML pueden ser ejecutados (o simulados), se les asigna una semántica de ejecución. Si consideramos la variante temporal en la vista de los SPMLs, se muestra que la tendencia es usar metamodelos a la hora de generar nuevos SPMLs.

2.2.3. RQ3. ¿Cuáles son las limitaciones de la investigación actual?

El objetivo de esta cuestión es reflejar el estado de investigación actual en el área de los lenguajes de modelado de procesos de software, lo que se ha revisado desde dos puntos de vista. Hemos considerado los datos recogidos en las secciones de "Trabajo futuro y Conclusiones" que aparecen en los artículos incluidos en el estudio. La tabla 2.14 los resume, agrupándolos por aquellos que son comunes en varios artículos. La tabla 2.15 muestra los aspectos pendientes para ser trabajados en el futuro. Hemos decidido agruparlos en temas relacionados, como aspectos a tener en cuenta cuando se consideren nuevas propuestas: integrar los modelos de procesos de software con herramienta de soporte a la toma de decisiones, ser útil en la práctica, mejorar la evolución de los procesos, verificar su utilidad en la práctica, alcanzar el consenso sobre los conceptos que se deben incluir al modelar los procesos de software y qué tecnologías podrían ser integradas en el modelado de procesos de software.

Una propuesta que pueda ofrecer tanto un entorno de modelado como de ejecución, siempre manteniendo un nivel apropiado de comprensibilidad, podría ser una alternativa interesante en este área, con la posibilidad de ser aplicada en organizaciones de software.

No obstante, el hecho de que existan tantos lenguajes de modelado de procesos de software nos indica que el objetivo en esta área no debería centrarse en buscar el lenguaje óptimo o ideal, porque no existe y es posible que nunca llegue a existir. Lo que se debería pretender es el ser capaces de utilizar el lenguaje más apropiado para un propósito concreto en un momento determinado, de modo que es posible que en una misma organización podamos encontrar procesos de software modelados con lenguajes diferentes. Si estos procesos no tienen ninguna relación entre sí, no hay ningún problema. El problema surgiría en el momento en que fuera necesario interoperar entre ellos. Así, disponer de las facilidades necesarias que nos proporcionen la interoperabilidad entre los procesos de software modelados con diferentes lenguajes, aislando del usuario las transformaciones necesarias para conseguirlo, sería un objetivo más adecuado en este ámbito. De esta forma los procesos de software no quedarían fuertemente vinculados al lenguaje elegido, pudiéndose optar por varios de ellos para representar diferentes aspectos en función de lo que se quiere conseguir.

Trabajo Futuro en SPML	Trabajo Futuro en el artículo	Referencia
Conseguir modelos UML promulgables	Usar ese enfoque para modelar otros procesos como RUP	[Di Nitto *et al.*, 2002]
Verificar la utilidad del modelo	Desarrollar un motor de programación	[Lee *et al.*, 2002]
Reutilizar procesos de software	Usarlo en promulgaciones heterogéneas y en el modelado de sistemas	[Reis *et al.*, 2002]
Desarrollar un SPML estándar	Desarrollar un modelo de promulgación basado en UML4SPM, así como incorporar transformaciones de modelo mediante ATL o QVT	[Bendraou *et al.*, 2005] [Bendraou *et al.*, 2006] [Bendraou *et al.*, 2007b] [Bendraou *et al.*, 2007a]
Promulgar un modelo de proceso descrito con SPEM	Describir la semántica operacional de los metamodelos	[Combemale *et al.*, 2006]
Construir modelos de proceso mediante la reutilización de patrones de proceso	Implementar el metamodelo propuesto como un perfil UML	[Tran *et al.*, 2007]
Crear aplicaciones de modelado de procesos en un entorno distribuido	Desarrollar reglas basadas en CMMI para verificar los procesos de una organización	[Hsueh *et al.*, 2008]
Dar soporte bidireccional a la evolución en la flexibilidad de los procesos	Integrar FlexEPFC con el motor de procesos Jazz	[Martinho *et al.*, 2008]
Permitir que la definición de proceso pueda ser modificada dinámicamente	Seguir utilizando Kermeta y técnicas de modelización orientadas a aspectos	[Bendraou *et al.*, 2009] [Bendraou *et al.*, 2012]
Lograr la integración con un enfoque de testeo basado en modelos	Desarrollar nuevos casos de estudio que permitan evaluar la propuesta de forma cuantitativa	[Maciel *et al.*, 2009]
Implementar componentes de proceso de forma eficiente	Permitir la personalización de los componentes de proceso existentes	[Koudri y Champeau, 2010]
Evaluar empíricamente el impacto de la promulgación de un proceso en proyectos reales	Implementar completamente la sintaxis concreta y las herramientas adicionales en eSPEM's	[Ellner *et al.*, 2010]
Dar soporte a la evolución del proceso	Desarrollar la semántica operacional en la extensión de FUML	[Ellner *et al.*, 2011]
Lograr la institucionalización de un proceso (adaptación antes de la representación)	Verificar la usabilidad y la aplicabilidad del proceso en casos reales	[Martinez-Ruiz *et al.*, 2011]
Hacer frente a la verificación de los logros e intenciones del proceso	Desarrollar un método para capturar y gestionar instantáneas de instancias de proceso para permitir el análisis y, posiblemente, la minería de datos asociados al proceso	[Pillain *et al.*, 2011]

Tabla 2.14: Trabajo futuro enunciado en los estudios

Descripción del problema	Referencia
SPM como una herramienta de apoyo a la toma de decisiones (integración con tecnologías de simulación)	[Podnar *et al.*, 2000]
Integración con herramientas de soporte	[Ellner *et al.*, 2010]
Mejorar el soporte para la trazabilidad de artefactos	[Ellner *et al.*, 2010]
Mejorar la flexibilidad de un proceso mediante patrones de proceso	[Hagen M., 2004]
Mejorar la evolución de un proceso	[Ellner *et al.*, 2010]
Dar soporte bidireccional a la evolución y flexibilidad de un proceso	[Martinho *et al.*, 2008]
Mejorar la flexibilidad mediante el uso de técnicas de reflejo (*reflection*)	[Bendraou *et al.*, 2005]
Verificar la utilidad midiendo la mejora en la calidad y en la productividad en la práctica	[Washizaki, 2007]
Desarrollar estudios cuantitativos y de análisis en el campo del modelado de procesos de software	[Podnar *et al.*, 2000]
Verificar la utilidad midiendo los mecanismos de variabilidad y su compresión en la práctica	[Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]
Evaluar de forma empírica el efecto de usar el formalismo	[Podnar *et al.*, 2000]
Desarrollar una validación empírica en un uso industrial	[Cares *et al.*, 2006]
Verificar la utilidad midiendo la comprensión de los diagramas de proceso en la práctica	[Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]
Evaluar de forma empírica el impacto de la promulgación en proyectos reales	[Ellner *et al.*, 2010]
Lograr la institucionalización del proceso (adaptación previa a su representación)	[Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]
Obtener una comprensión más profunda de los requisitos de la variabilidad del proceso en las organizaciones y su cumplimiento por parte de los mecanismos de la variabilidad del proceso	[Martinez-Ruiz *et al.*, 2011] [Martınez-Ruiz *et al.*, 2011]
Desarrollar un PSEE para procesos de software MDA	[Maciel *et al.*, 2009]
No existe un consenso general sobre qué temas deben estar cubiertos en SPM	[Noack, 2000]
No existe un consenso general sobre la forma de describir e integrar tecnologías en un modelo de proceso de software	[Noack, 2000]
No hay ningún enfoque concreto para disponer de un PML semánticamente rico	[Pillain *et al.*, 2011] [Golra y Dagnat, 2011]

Tabla 2.15: Descripción de aspectos abiertos para investigación futura

2.3. Actualización de la revisión sistemática

Dado que la revisión sistemática de la literatura fue realizada antes de la finalización de esta memoria de tesis, hemos querido realizar una actualización de la misma para buscar nuevos estudios que hayan podido surgir durante los tres últimos años, con el objetivo de completar nuestra visión general y validar que los trabajos futuros que apuntamos como conclusiones siguen siendo válidos.

Para ello, nos hemos basado en la misma metodología de Kitchenham siguiendo el consejo de "quasi-gold standard" durante la ejecución de las búsquedas, tal y como enunciamos en el apartado 2.1.3, utilizando las mismas palabras clave de búsqueda y en los mismos motores tal y como se detalla en [García-Borgoñón *et al.*, 2014a].

Sobre el método anterior, hemos modificado ligeramente las fases y los criterios de inclusión y exclusión. Por un lado, en el mismo momento de la búsqueda todas las publicaciones anteriores al año 2012 quedaron excluidas, ya que la anterior revisión incorporaba todas las publicaciones hasta finales de 2011. Por otro, en esta actualización consideramos interesante poder incluir estudios que en sí no propusieran ningún lenguaje de modelado de procesos de software, sino que fueran revisiones de la literatura, con el objetivo de detectar posibles lenguajes que quizás no hubieran sido incluidos en la primera revisión. En la tabla 2.16 se especifican las fases en las que se ejecutó esta actualización.

Fase	**Descripción de fase**	**Participación**
F1	Selección de los estudios en base a búsqueda y fechas	Investigador líder
F2	Criba: exclusión basada en títulos, resumenes y palabras clave	Dos investigadores
F3	Reunión de consenso	Todos los investigadores
F4	Análisis de relevancia: exclusión basada en texto completo	Todos los investigadores
F5	Reunión de consenso	Todos los investigadores

Tabla 2.16: Fases de inclusión

Al igual que se hizo en la revisión anterior, los trabajos fueron incluidos y excluidos en cada fase de acuerdo a unos criterios de selección detallados en la tabla 2.17.

Fase	Criterios de inclusión / exclusión
F1	No duplicado
	Sólo trabajo publicado
	Contiene las cadenas de búsqueda
	Fechas de publicación entre 2012 y 2014
F2	No editoriales, prefacios, discusiones
F3	No resúmenes de tutoriales, workshops o paneles
F4	Sólo en inglés
	Texto completo obtenido
F5	Nuevo lenguaje o una modificación propuesta del lenguaje
	Si que puede incluir estudios o revisiones

Tabla 2.17: Criterios de Inclusión y Exclusión por fase

En la tabla 2.18 se muestran los números de los trabajos que iban incluyéndose y excluyéndose en cada una de las fases. Asimismo, las referencias de los 13 artículos finalmente incluidos se encuentran disponibles en la sección de Referencias.

Fase	Incluidos	Eliminados
Encontrados	180	0
F1	180	67
F2	113	68
F3	45	2
F4	43	27
F5	16	3
Incluidos	13	0

Tabla 2.18: Estudios incluidos y excluidos por fase

Con los estudios seleccionados, se siguió el mismo esquema de caracterización que en la primera revisión mostrado anteriormente en la tabla 2.5. Seguidamente pasamos a mostrar los principales resultados obtenidos en esta nueva búsqueda atendiendo a las mismas tres preguntas que nos hicimos.

2.3.1. RQ1. ¿Qué lenguajes de modelado de procesos de software se han definido? ¿Por qué?

En esta actualización hemos podido comprobar que se mantiene la tendencia a seguir creando lenguajes de modelado de procesos de software en los últimos años, en los que se han definido más de 10 nuevas propuestas. En la figura 2.4 se presentan dichas propuestas así como el año en el que se publicaron. Cabe destacar que hemos añadido en el año 2011 cinco nuevas propuestas que han sido citadas en una revisión sistemática de la literatura.

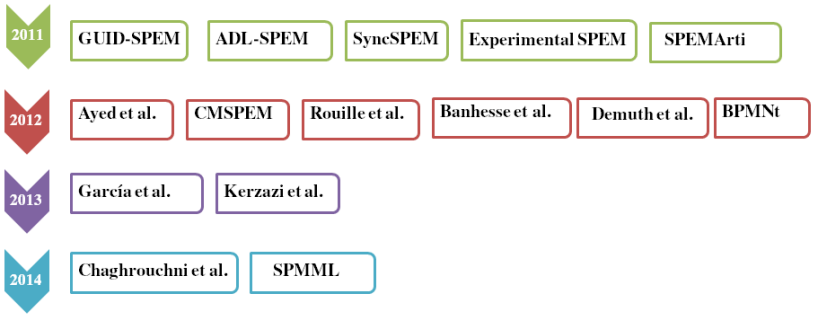


Figura 2.4: Resumen de los lenguajes de modelado de procesos de software

Al igual que hicimos en la primera revisión, se muestra en la tabla 2.19 una descripción de cada uno de ellos así como la motivación por la que fueron propuestos. De nuevo, al hacer el análisis de los 13 artículos, dos artículos hacían referencia al mismo lenguaje, de ahí que el número de lenguajes que aparecen en la vista sean únicamente 12.

Y, al igual que se ha hecho anteriormente, en la tabla 2.20 se resumen los principales problemas que estos lenguajes intentan resolver.

Año	SPMLs [Ref]	Basado en	Descripción	Motivación
2012	Ayed et al. [Ayed *et al.*, 2012]	SPEM 2.0	Lenguaje que pretende alinear OPF, SPEM y SEMDD (ISO 24744)	Conseguir *tailoring* de procesos en entornos ágiles
2012	Banhesse et al. [Banhesse *et al.*, 2012]	DSL Textual	Proponen un *Process Capability Profiles Metamodel* para conseguir la integración dinámica de los elementos incluidos en diversos modelos de referencia (como CMMI o SPICE) dentro de un ciclo de mejora de procesos	No existía una propuesta que estuviera enfocada a resolver el que una organización pueda querer cumplir diversos modelos o normas de referencia
2012	UML4SPM [Bendraou *et al.*, 2012]	UML 2.0	Evolución de la propuesta encontrada en la primera revisión en la que intenta dar soporte a la simulación y ejecución de los procesos	Dar soporte a la simulación y ejecución de los procesos definidos utilizando UML4SPM
2012	Demuth et al. [Demuth, 2012]	DSL textual	Proponen un lenguaje basado en restricciones para dar soporte a la evolución de los modelos	Dar soporte a la evolución de los lenguajes de modelado de forma independiente de las herramientas de soporte
2012	CMSPEM [Kedji *et al.*, 2014]	SPEM 2.0	Proponen una extensión de SPEM para dar soporte al desarrollo colaborativo (*Collaborative Model SPEM*)	Todas las propuestas existentes intentan que el proceso sea quien orqueste las herramientas de soporte, proponen hacerlo justo al revés
2012	Rouille et al. [Rouille *et al.*, 2012]	CVL y SPEM 2.0	Proponen incorporar a SPEM aspectos de CVL (Common Variability Language) para mejorar su variabilidad	Dar soporte a la variabilidad de procesos definidos en SPEM
2012	BPMNt [Pillat *et al.*, 2012]	BPMN	Proponen una extensión a BPMN (*BPMN tailoring*) para poderlo utilizar en el modelado de procesos de software a medida, utilizando conceptos de SPEM 2.0	Dar soporte al *tailoring* de procesos de software
2013	García et al. [García Guzmán *et al.*, 2013]	UML 2.0	Proponen utilizar patrones de procesos para gestionar el conocimiento en la mejora de procesos de software	Poder formalizar la gestión del conocimiento en el ámbito de los procesos de desarrollo de software en las organizaciones
2013	Kerzazi et al. [Kerzazi *et al.*, 2013]	SPEM 2.0	Proponen una extensión de SPEM para, mediante ontologías, identificar inconsistencias en la definición semántica de los conceptos modelados	Mejorar la semántica de los conceptos modelados mediante ontologías
2013	Kuhrmann et al. [Kuhrmann *et al.*, 2013]	SPEM 2.0	Se trata de una revisión sistemática de la literatura, que hemos incluido dado que nombre cinco extensiones de SPEM que no fueron identificadas en la primera fase ni hemos podido recuperar en esta segunda: GUID-SPEM, ADL-SPEM, SyncSPEM, experimental SPEM y SPEMArti	
2014	SPMML [Castro *et al.*, 2014]	DSL textual	Es un lenguaje (*Software Processes and Methodologies Modeling Language*) que pretende cubrir de forma ortogonal la relación entre la definición de los procesos y las metodologías	Crear un lenguaje que unifique los conceptos existentes en las múltiples metodologías y lenguajes de procesos exitentes
2014	Chaghrouchni et al. [Chaghrouchni *et al.*, 2014]	UML 2.0	Presenta cuatro modelos para describir un proceso, a nivel general, de ejecución, de desviaciones toleradas y de lo que ocurre en la realidad mediante una observación	Dar soporte a la evolución dinámica que puede tener la ejecución de un proceso respecto a cómo estaba definido

Tabla 2.19: Descripción de los lenguajes de modelado de procesos de software (SPMLs) existentes

Descripción del problema	**Referencia**
Tailoring de procesos	[Ayed *et al.*, 2012]
Dar soporte al cumplimiento de varios modelos de referencia	[Banhesse *et al.*, 2012]
Dar soporte a la simulación y ejecución de los procesos	[Bendraou *et al.*, 2012]
Independizar la evolución de un lenguaje de las herramientas de soporte	[Demuth, 2012]
Dar soporte al desarrollo colaborativo usando SPEM	[Kedji *et al.*, 2014]
Utilizar BPMN para describir procesos de software que puedan ser ajustados a medida	[Pillat *et al.*, 2012]
Dar soporte a la variabilidad de los procesos definidos en SPEM mediante CVL	[Rouille *et al.*, 2012] [Rouille *et al.*, 2013]
Gestionar el conocimiento dentro de una organización en el ámbito de los procesos de desarrollo de software	[García Guzmán *et al.*, 2013]
Reducir las inconsistencias semánticas entre los conceptos modelados con SPEM mediante el uso de ontologías	[Kerzazi *et al.*, 2013]
Unificar los términos existentes entre la gran multitud de lenguajes de definición de procesos y metodologías existentes	[Castro *et al.*, 2014]
Dar soporte a una evolución durante la ejecución de los procesos	[Chaghrouchni *et al.*, 2014]

Tabla 2.20: Descripción de los problemas tratados en los SPMLs existentes

2.3.2. RQ2. ¿Cuál es la tendencia actual cuando se selecciona una tecnología base para definir un lenguaje de modelado de procesos de software?

Manteniendo la misma taxonomía que hemos presentado en la figura 2.2, podemos observar en la figura 2.5 la clasificación de las nuevas propuestas de lenguajes obtenidos en la RQ1. Se mantiene la tendencia a extender SPEM para hacerlo más rico y cubrir mayores requisitos, aunque siguen existiendo propuestas derivadas de UML o basadas en gramáticas textuales.

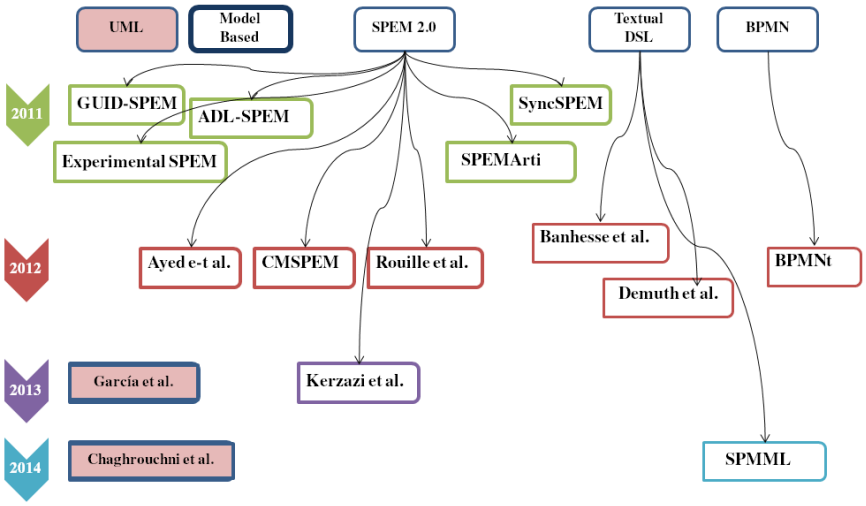


Figura 2.5: Relaciones y tecnologías base de los lenguajes de modelado de procesos de software existentes

2.3.3. RQ3. ¿Cuáles son las limitaciones de la investigación actual?

La tabla 2.21 muestra las principales líneas de trabajo futuro enunciadas en los estudios seleccionados. Destaca principalmente la gran cantidad de propuestas que fijan como principal reto mejorar la variabilidad, flexibilidad y evolución de los procesos definidos, permitiendo diferencias entre la definición inicial y su ejecución real. Igualmente son varios los trabajos en los que se cita explícitamente la necesidad de desarrollar estudios empíricos para utilizar las propuestas en la práctica.

Trabajo Futuro en SPML	Trabajo Futuro en el artículo	Referencia
Mejorar la capacidad para adaptar las necesidades de los modelos en cada entorno específico	Crear un catálogo de reglas que permita tomar mejores decisiones a nivel metodológico	[Ayed *et al.*, 2012]
Mejorar la relación entre el modelado de los procesos y la mejora de procesos en general	Crear un *Process Capability Best Practices* como modelo de referencia	[Banhesse *et al.*, 2012]
Mejorar la capacidad de simulación y ejecución de un proceso modelado	Realizar una evaluación en la práctica en el uso de UML4SPM en un entorno industrial	[Bendraou *et al.*, 2012]
Mejorar la adaptación a medida de los procesos definidos	Integrar capacidades de ajustes a la medida dentro de herramientas BPMN	[Pillat *et al.*, 2012]
Mejorar la variabilidad de los procesos definidos con SPEM	Desarrollar una herramienta de análisis para validar las restricciones sobre SPEM	[Rouille *et al.*, 2012] [Rouille *et al.*, 2013]
Desarrollar estudios empíricos en la industria	Mejorar aquellas amenazas detectadas para validar la propuesta	[García Guzmán *et al.*, 2013]
Estudiar cómo se produce la propagación del conocimiento entre todas las fases de desarrollo de software	Incorporar lenguajes como RDF y OWL en su propuesta	[Kerzazi *et al.*, 2013]
Poner en la práctica el uso de los lenguajes de modelado de procesos de software	Realizar un análisis en profundidad de los lenguajes de modelado de procesos de software para entender las fortalezas y debilidades de las propuestas	[Kuhrmann *et al.*, 2013]
Incorporar la unificación de términos dentro de SPEM	Aplicarlo a diversos proyectos de software y mejorar la propuesta actual	[Castro *et al.*, 2014]
Mejorar la evolución y el dinamismo en la ejecución de un proceso respecto a su definición	Permitir la coexistencia de dos modelos de proceso, el predefinido y el observado	[Chaghrouchni *et al.*, 2014]

Tabla 2.21: Trabajo futuro enunciado en los estudios

2.4. Conclusiones

En este capítulo se ha presentado el estudio del estado del arte de los lenguajes de modelado de procesos de software que se han definido desde el año 2000, llevado a cabo mediante una revisión sistemática de la literatura. Tal y como se ha podido observar a lo largo de todo el capítulo, existen multitud de lenguajes de modelado de procesos de software y una tendencia frecuentemente encontrada actualmente consiste en utilizar los metamodelos como la base sobre la que se realiza la definición de nuevos lenguajes. Como resultado de la revisión sistemática, se han observado diferentes aspectos como líneas de trabajo futuras, de las que nos gustaría destacar tres: la primera es la necesidad de disponer de mayor número de trabajos empíricos, que nos muestre las principales barreras que impiden que las organizaciones utilicen los lenguajes de modelado de procesos de software en la práctica para describir sus procesos. Otro aspecto a trabajar sería el estudiar cómo se relacionan los lenguajes de modelado de procesos de software con otros campos de la ingeniería de procesos, como son la simulación, mejora y la orquestación. Por último, plantear la existencia de entornos integrados para el modelado y la ejecución de procesos de software.

En este estudio también se ha visto que, lejos de tener una tendencia clara sobre un lenguaje de modelado de procesos de software que pudiéramos considerar como el lenguaje óptimo o el ideal, a pesar de varios intentos de estandarización que han ido ocurriendo, siguen surgiendo nuevas propuestas. Un planteamiento diferente al llevado a cabo en este momento es el que vamos a abordar a lo largo de este trabajo de tesis, en el cual el objetivo pasa de plantear un lenguaje estándar que cubra todas las necesidades de cualquier organización de software, al de utilizar el lenguaje de modelado de procesos de software más apropiado para un determinado momento, circunstancia o propósito, y establecer un mecanismo que facilite la interoperabilidad entre los procesos de software modelados con diferentes lenguajes. De esta forma los procesos de software no quedarían fuertemente vinculados o acoplados al lenguaje de modelado elegido inicialmente, siendo posible optar por varios de ellos para representar diferentes aspectos, en función de lo que se pretenda describir.

A la vista de los resultados obtenidos tras la realización de la revisión sistemática, podemos concluir que en los últimos años se mantiene como tendencia el crear nuevas propuestas de lenguajes de modelado de procesos de software, la mayoría como extensiones a SPEM, para hacerlo más rico y cubrir nuevos escenarios, especialmente orientados a mejorar la variabilidad, flexibilidad y evolución de los lenguajes y procesos. Encontramos propuestas que aúnan diversos lenguajes (ej. BPMN y SPEM, CVL y SPEM) pero siempre mediante la creación de un nuevo lenguaje, que en su estructura suma conceptos de los anteriores, lo que nos refuerza el carácter novedoso de nuestra propuesta. Por otro lado siguen faltando estudios empíricos de su uso y adopción en la práctica, algo que vuelve a ser remarcado en los estudios como un gran reto pendiente.

Capítulo 3

Planteamiento del Problema

En el capítulo 2 se ha presentado el estado del arte actual en cuanto a lenguajes de modelado de procesos de software se refiere, poniendo de manifiesto la cantidad existente y que la aparición de cada uno de ellos venía motivada para la resolución de un asunto que estaba pendiente de solventar. El análisis de la situación actual ha permitido vislumbrar el problema que se desea resolver y el enfoque que en este trabajo de tesis vamos a plantear, aspectos ambos en los que vamos a profundizar y justificar a lo largo de este capítulo. Para ello, en la primera sección se recopilan, a modo de resumen, los aspectos más relevantes que han determinado el problema que se pretende acometer con este trabajo de tesis, que será definido claramente en la siguiente sección. Una vez establecido el problema, se enuncia la solución que se propone en este trabajo de tesis y se plantean los objetivos que se pretenden alcanzar, un esbozo de la estructura de la solución y las influencias más destacadas que repercuten en el desarrollo de la misma. Finalmente, la última sección resume las principales conclusiones de este capítulo.

3.1. Aspectos relevantes que determinan el problema a resolver

Como paso previo a la manifestación del problema que se pretende resolver, a continuación exponemos los principales aspectos que lo especifican, los cuales vienen derivados de las conclusiones obtenidas en el capítulo anterior.

3.1.1. Las organizaciones de software tienen la necesidad de definir sus procesos

Una de las principales conclusiones del capítulo anterior es que no existen suficientes estudios empíricos que nos muestren la forma en la que los lenguajes de modelado de procesos de software se están utilizando en las organizaciones, si es que lo están haciendo, y si no es así, que expongan las principales barreras que impiden su uso para describir los procesos.

En el capítulo 1 se expuso brevemente cómo los diferentes estándares y modelos utilizados en la industria del software establecen la necesidad de tener los procesos definidos y mantenidos. Si las organizaciones quieren cumplir esos estándares deben obligatoriamente definir y documentar sus procesos, y mantenerlos. Las principales dificultades a las que se tienen que enfrentar las empresas que se plantean algún tipo de certificación van, desde la falta de conocimiento en determinadas áreas de procesos, hasta la falta de recursos humanos y herramientas adecuadas para poder definir e implantar los procesos dentro de la organización. En [García-Borgoñón *et al.*, 2014b] se recogen estas barreras o inhibidores de forma más detallada.

Uno de los elementos más relevantes es la manera en la que esos procesos van a ser definidos y documentados [Bendraou *et al.*, 2010]. Habitualmente las empresas optan por el desarrollo de un manual de calidad, en el que los procesos son recopilados a modo de documento, descritos en lenguaje natural. Este documento suele formar parte de la biblioteca de activos de las organizaciones.

Si bien es cierto que un lenguaje natural supone un esfuerzo relativamente pequeño a la hora de llevar a cabo la descripción de los procesos, debido principalmente a que la curva de aprendizaje es prácticamente nula, con la salvedad de los términos concretos relacionados con los procesos de software, la elección de dicho lenguaje suele ser el origen de determinados problemas a la hora de comunicar, mantener e incluso entender los procesos, ya que es susceptible de contener múltiples inconsistencias no precisamente fáciles de detectar.

3.1.2. Existen muchos lenguajes de modelado de procesos de software en la literatura

El hecho de que el lenguaje natural no sea el medio más adecuado para representar los procesos de software es la causa por la que, desde hace años, los ingenieros de procesos han visto en los lenguajes de modelado de procesos de software una oportunidad de formalizar dichos procesos y facilitar tanto la descripción como la comunicación y el entendimiento de los mismos.

Tal y como ha quedado reflejado en el capítulo 2, numerosos lenguajes de modelado de procesos de software han sido propuestos durante años. Se ha visto que con el paso del tiempo las tendencias seguidas en el desarrollo de estos lenguajes han ido variando, y lo que comenzó mediante la definición de nuevos lenguajes de programación, lenguajes basados en reglas o redes de Petri, ha seguido el mismo camino abierto por la ingeniería de software con la introducción del paradigma de ingeniería dirigida por modelos (MDE), algo lógico, por otra parte, puesto que la ingeniería de procesos (o ingeniería de métodos) no deja de ser un subconjunto de la ingeniería de software, cuyo objetivo es obtener el máximo provecho de la aplicación de la ingeniería dirigida por modelos en el mundo de los procesos [García-Molina *et al.*, 2013]. En la figura 3.1 se resume el marco de trabajo en el que se enmarca la ingeniería de procesos dentro de una organización de software. En ella se observa cómo, para poder definir una metodología de desarrollo de software, un ingeniero de procesos dispone de dos piezas fundamentales: en primer lugar, de un metamodelo de métodos o de procesos, que constituye el lenguaje de modelado de procesos de software elegido por la organización y que establece las reglas de construcción de un proceso de software; y, en segundo lugar, de un repositorio de componentes predefinidos y

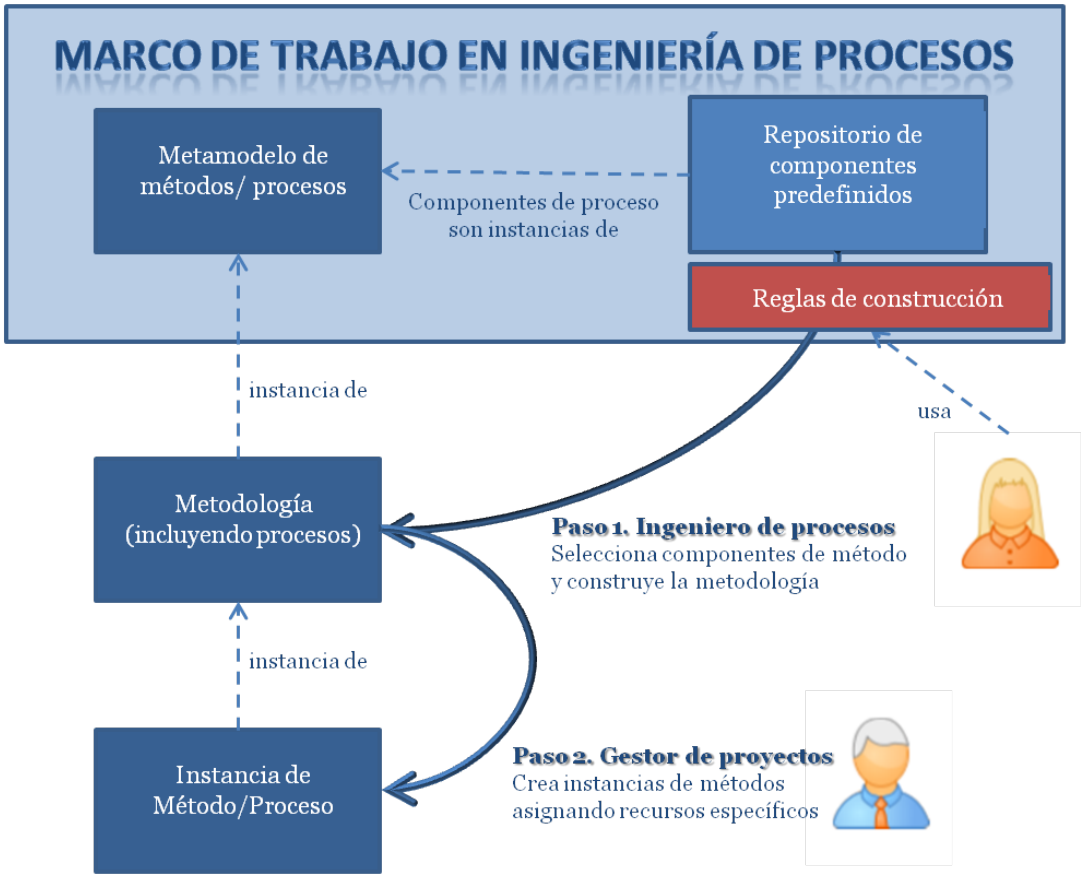


Figura 3.1: Marco de trabajo en Ingeniería de Procesos

reutilizables, modelados con el metamodelo en cuestión. De esta forma, cuando un ingeniero de procesos quiere incorporar un nuevo proceso de software a la metodología, o bien quiere definir una nueva metodología en el caso, por ejemplo, de que se trate de una organización de nueva creación, cuenta con ambos elementos que le facilitan la tarea, puesto que ese nuevo proceso de software se puede construir combinando las componentes reutilizables ya definidas, e incorporando nuevos aspectos utilizando siempre el metamodelo de procesos de software establecido. Por último, cuando un jefe de proyecto debe utilizar un proceso de software de la metodología para un proyecto concreto, realiza una instancia del mismo, donde los elementos genéricos pasan a ser específicos del proyecto en cuestión. La metodología se retroalimentaría de las lecciones aprendidas expuestas por los jefes de proyectos en la experiencia de su utilización en proyectos concretos, pudiéndose añadir, eliminar y modificar elementos incluidos en los procesos de software de la organización.

En el ámbito de la ingeniería de procesos de software, los lenguajes de modelado de procesos de software constituyen un elemento básico y fundamental para hacer factible toda la actividad relacionada con los procesos de software en una organización. Sin embargo, los numerosos lenguajes propuestos no han tenido una extensa aceptación por parte de las organizaciones de software. Muchos autores han apuntado posibles causas de su falta de uso más allá de pruebas de concepto académicas [Bendraou *et al.*, 2010], que van desde la falta de flexibilidad y necesidad

de conocimientos específicos para aquellos que estaban basados en lenguajes de programación y reglas, hasta la falta de semántica adecuada para hacer frente a los conceptos de procesos de software en el caso de que la opción sea la utilización de lenguajes estándares más generales como UML o BPMN. En todos los casos se alude a una importante curva de aprendizaje para su uso. Otro aspecto a destacar en esta falta de aceptación por parte de la industria es el hecho de la confusión que se produce a la hora de elegir ante todas las posibilidades existentes, más aún si cabe sabiendo que la elección de uno de ellos implica un vínculo difícil de romper entre los procesos de software y el lenguaje elegido.

Muchos han sido los intentos para definir un lenguaje de modelado de procesos de software que se convierta en un estándar en el ámbito de la ingeniería de procesos de software, de manera similar a como UML se ha convertido en un estándar de modelado en el mundo de la ingeniería del software. Sin embargo, este hecho no se ha llegado a producir ya que ningún lenguaje se ha erigido como estándar de facto por encima del resto. A pesar de que algunos de ellos están avalados por organizaciones como ISO con la ISO 24744:2007 [ISO/IEC, 2007a] u OMG con SPEM[OMG, 2008], siguen apareciendo nuevas propuestas que pretenden corregir las carencias detectadas en sus predecesores.

3.1.3. Los requerimientos de los lenguajes de modelado de procesos de software

En varias ocasiones se ha referido el hecho de que muchos lenguajes de modelado de procesos de software se han definido a lo largo del tiempo. Una forma que los diferentes autores han tenido para argumentar el desarrollo de un nuevo lenguaje, ha sido precisamente que los anteriores no han llegado a cumplir las expectativas que se planteaban a la hora de utilizar un lenguaje de modelado de procesos de software. Este razonamiento venía habitualmente soportado por la definición de un conjunto de requerimientos exigidos a estos lenguajes.

La cuestión a la que se alude en este apartado consiste en saber cuáles son los principales requerimientos demandados por los usuarios a los lenguajes de modelado de procesos de software. La gran mayoría de estos requerimientos son sugeridos por los autores en las propias propuestas, por lo que hemos utilizado los trabajos incluidos y los excluidos en la última fase (F6) de la revisión sistemática de la literatura para extraerlos e introducirlos brevemente.

Cabe señalar que, en lo que se refiere a los requerimientos, no todos los autores hacen referencia a los mismos conceptos, ni hay uniformidad a la hora de denominarlos, con lo que se ha realizado un trabajo de recapitulación y homogeneización de dichas nociones entre los diferentes autores que se refleja en las tablas 3.1 y 3.2. Dicha homogeneización se ha realizado en ingles, debido a las connotaciones de los términos a la hora de hacer la traducción al castellano. A partir de este momento, a lo largo de este trabajo de tesis utilizaremos la denominación de la columna situada a la izquierda para referirnos a un requerimiento en concreto. En las tablas hemos querido preservar la palabra original para evitar problemas de interpretación a la hora de establecer los términos de todos los autores, aunque en lo sucesivo lo utilizaremos traducido.

Requerimiento	[Min *et al.*, 2000]	[Noack, 2000]	[Podnar *et al.*, 2000]	[Zamli y Lee, 2001]	[Chen *et al.*, 2002]	[Di Nitto *et al.*, 2002]	[Lima Reis *et al.*, 2002]	[Reis *et al.*, 2002]	[Franch y Ribó, 2003]	[Zamli y Lee, 2003]	[Atkinson *et al.*, 2004]
Expressiveness	✓	Task manageable			Semantic Richness				✓	✓	✓
Executability					✓		Instantation	Instantation		✓	
Enactability	✓			✓			✓	✓		✓	✓
Simulability				✓			✓	✓			
Testability											
Modularity		Components exchangeable and reuse			Scalability and reuse				✓		
Formality			✓								
Tooling Support	Usability in practice					✓					
Standardization						✓			✓		
Graphical representation						✓					
Support Multiple Views										✓	
Understandability	Adoptability		✓		Easy of use	Popular Extensible				Easy to use	Simplicity
Abstraction											
Analyzability	✓	Projects comparable			✓						
Reflection					Evolutive					Dynamic	
Flexibility					✓				✓		✓
Scalability											

Tabla 3.1: Requerimientos de autores para lenguajes de procesos de software

Requerimiento	[Bendraou *et al.*, 2005]	[Bendraou *et al.*, 2006]	[Wu *et al.*, 2007]	[Jablonski *et al.*, 2008]	[Maciel *et al.*, 2009]	[Martinho *et al.*, 2009]	[Bendraou *et al.*, 2010]	[Koudri y Champeau, 2010]	[Martinez-Ruiz *et al.*, 2011]	[Martinez-Ruiz *et al.*, 2011]	[Zaki *et al.*, 2011]
Expressiveness	✓	Semantic Richness	Human comprehension				Semantic Richness				
Executability	✓	✓					✓	✓			✓
Enactability					✓						Automatic process enactment
Simulability					✓						
Testability											✓
Modularity	✓			Extensibility			✓				Decomposability
Formality	✓						✓	✓			
Tooling Support							✓				Integration and Electronic Process Guide
Standardization								Conformity to UML Standard			
Graphical representation		✓					✓				
Support Multiple Views	✓						✓				
Understandability	✓	✓	Popularity		✓						Easy to digest
Abstraction	✓										
Analyzability	✓										
Reflection	✓										Automatic audit trail
Flexibility						✓			Customizable	Customizable	Adaptability
Scalability											✓

Tabla 3.2: Requerimientos de autores para lenguajes de procesos de software (cont)

Una vez homogeneizados los términos y establecidos la denominación a utilizar, a continuación se traducen los términos al castellano y se define brevemente cada uno de ellos.

- **Expresividad** *(Expressiveness)*. Este requerimiento se refiere a la capacidad de expresar lo que realmente se realiza durante los procesos de desarrollo de software. Indica si todos los aspectos de un proceso se pueden modelar directamente a partir de los elementos definidos en el lenguaje de modelado de procesos de software. Este es un requerimiento que puede ser tan complejo como se desee, puesto que cuantos más elementos contemple un lenguaje de modelado de procesos de software mayor va a ser su expresividad. Sin embargo, en el contexto de este trabajo de tesis, vamos a referirnos a él como el hecho de que el lenguaje ofrezca los conceptos apropiados para cubrir la descripción de todos los elementos de proceso. Los elementos básicos comúnmente mencionados suelen ser: Actividad, Rol y Producto.
- **Ejecutable** *(Executability)*. Aquí nos referimos a la capacidad del lenguaje de proporcionar una semántica operacional que ofrezca el soporte para la ejecución de esas construcciones. El lenguaje puede ser interpretado directamente por una máquina o de alguna manera trazado con un lenguaje ejecutable. De esta forma se permitiría automatizar partes de los procesos.
- **Promulgable** *(Enactabiliy)*. Si un lenguaje cumple este requerimiento, es posible definir un modelo de proceso con un nivel de granularidad tal que permita un control exhaustivo del mismo, paso a paso, sin llegar a la capacidad de ejecución. Se trata de un punto intermedio entre la mera publicación y la ejecución detallada de un proceso.
- **Simulable** *(Simulability)*. Este requerimiento hace referencia a la capacidad de simulación de un modelo de proceso previo a su ejecución, lo que supone una ayuda a la hora de diseñar el proceso de software, o incluso de facilitar la toma de decisiones en su rediseño derivado de la mejora continua.
- **Comprobable** *(Testability)*. Con esto se hace referencia a la capacidad del lenguaje de realizar de forma automática pruebas para supervisar la forma en la que se promulgan, como una ayuda para posibles auditorías o revisiones de compatibilidad con estándares (como CMMI o SPICE).
- **Modularidad** *(Modularity)*. Esta condición trata sobre la capacidad de combinar diferentes componentes de procesos con el objetivo de construir uno nuevo.
- **Formalidad** *(Formality)*. Aquí se hace referencia al hecho de que la sintaxis y la semántica de un lenguaje de modelado de procesos de software se ha definido formalmente, con una alta granularidad.
- **Soportado por herramientas** *(Tooling Support)*. Este requerimiento hace referencia a la posibilidad de tener herramientas que nos permitan definir los procesos con el lenguaje, que más tarde se pueda convertir en algo compresible por los usuarios de los mismos, más allá de un simple editor de textos.
- **Estandarización** *(Standardization)*. Aquí se hace referencia a la cercanía del lenguaje de modelado de procesos de software a algún estándar existente y de uso generalizado (como podría ser UML), lo que facilitaría su despliegue en entornos reales.

- **Representación gráfica** *(Graphical representation)*. El lenguaje dispone de una semántica gráfica que facilite su descripción e intercambio entre los usuarios del mismo.
- **Soporte a múltiples vistas** *(Support Multiple Views)*. El lenguaje dispone de los elementos semánticos necesarios para la definición de vistas desde varias perspectivas del modelo de procesos.
- **Comprensible** *(Understandability)*. Este requerimiento hace referencia a la capacidad del lenguaje de ser entendido por los usuarios del mismo. Se trata de un requerimiento clave, puesto que si los usuarios no son capaces de comprender el lenguaje, dificilmente va a ser utilizado en la práctica.
- **Capacidad de abstracción** *(Abstraction)*. Aquí se hace referencia a la capacidad de un lenguaje de aislar ciertos aspectos de los modelos de procesos concretos, de forma que faciliten su comprensión.
- **Capacidad de análisis** *(Analyzability)*. Con este requerimiento, el lenguaje dispondría de los elementos necesarios para comparar diferentes modelos de procesos, tomar decisiones y actividades similares.
- **Reflexión** *(Reflection)*. Este requerimiento hace referencia a la capacidad del lenguaje de modelado de procesos de software a soportar directamente la evolución de los modelos de procesos.
- **Flexibilidad** *(Flexibility)*. Hace referencia a poder ser aplicado en una variedad de entornos de procesos, no únicamente en entornos de software, sino con procesos de negocio de otro ámbito.
- **Escalabilidad** *(Scalability)*. Este requerimiento hace referencia a la capacidad del lenguaje de modelado de procesos de software de describir procesos de diferentes tamaños, tanto grandes como pequeños.

Como se ha podido comprobar, los requerimientos que se han demandado a los lenguajes de modelado de procesos de software han sido muchos y muy variados. De hecho, alguno de ellos son incompatibles entre sí, como lo es, por ejemplo, el hecho de que un lenguaje que cumpla la condición de *Formalidad*, es decir, que disponga de un nivel de granularidad elevado, hace que difícilmente cumpla el requerimiento de *Comprensible* o de *Flexibilidad*, y en ocasiones se busca un compromiso entre ambos requisitos. En otros casos, un requerimiento puede ser un subconjunto de otro, por ejemplo, un lenguaje puede cumplir el requerimiento de *Promulgable* y no ser *Ejecutable*, mientras que si es *Ejecutable*, por definición, también es *Promulgable*.

Por último, y para concluir esta sección, queremos explicitar que todos los aspectos enunciados a lo largo de la misma son elementos condicionantes del problema que se desea resolver, el cual va a ser planteado de forma más detallada en la siguiente sección.

3.2. Planteamiento del problema a resolver

Los aspectos que han sido mencionados en la sección anterior nos hacen reflexionar sobre el hecho de que no existe un lenguaje *ideal* para modelar procesos de software. Cada lenguaje tiene sus ventajas e inconvenientes en función de unos requerimientos que se consideran para un momento específico, y ninguno es el mejor en todos ellos. La dificultad reside, principalmente, en saber elegir qué lenguaje seleccionar para modelar los procesos de software por parte de una organización, puesto que, una vez se ha elegido dicho lenguaje, se establece un vínculo con él tan estrecho que intentar utilizar otro lenguaje en otra circunstancia no es una tarea sencilla.

Así, con este vínculo, la organización asume y hace suyas las ventajas y desventajas del lenguaje en cuanto a la capacidad de modelado de sus procesos de software. No existe un mecanismo que permita poder trabajar al mismo tiempo con procesos de software modelados con diferentes lenguajes, cada uno de ellos elegidos para satisfacer un requerimiento concreto para el que se considera más adecuado. Por tanto, el problema que se pretende resolver es precisamente éste, la ausencia de un mecanismo que rompa este vínculo, de forma que se facilite el uso y la interacción entre procesos de software modelados con lenguajes diferentes, sin generar por ello inconsistencias o pérdidas de información.

Con el objetivo de clarificar el problema que estamos planteando, definimos tres situaciones prácticas que justifican la realización de este trabajo de tesis y que utilizaremos durante la validación de la solución propuesta. Son los siguientes:

- En primer lugar, planteamos la posibilidad de que dos departamentos diferentes dentro de una misma organización, que han modelado sus procesos de software con lenguajes diferentes, tienen que trabajar juntos, de manera que esos procesos deben combinarse y, en definitiva, interoperar.
- En segundo lugar, la evolución de procesos de software. Con el tiempo, una organización que dispone de unos procesos de software definidos con un lenguaje de modelado, decide migrarlos a otro lenguaje que aporta nuevas características, perdiendo la menor información posible y realizándose de la forma más automática y con menos dependencia de las personas, para evitar así problemas de inconsistencias entre las posibles interpretaciones que diera lugar. Esta capacidad de mantenibilidad de los procesos de software modelados con lenguajes diferentes, debería ser satisfecha con nuestra propuesta.
- En tercer lugar, pensemos en el hecho de que varias organizaciones diferentes, donde la probabilidad de haber elegido el mismo lenguaje de modelado de procesos de software es todavía más baja, deciden trabajar juntas para ser así más competitivas. Se trataría de un nuevo problema de interoperabilidad de procesos de software modelados con lenguajes diferentes, que debería también ser satisfecha con la propuesta que se plantea en este trabajo de tesis.

El problema a resolver, que de esta manera se plantea, está basado en dos conceptos: la interoperabilidad y la mantenibilidad. El IEEE[1] define interoperabilidad como la capacidad

[1] http://www.ieee.org

de dos o más sistemas o componentes de intercambiar información y utilizar la información que ha sido intercambiada [IEEE, 1990]. Si establecemos un marco de referencia que ofrezca la posibilidad de hacer que los procesos de software modelados con diferentes lenguajes interoperen, sería posible que cada lenguaje complementara sus puntos débiles con otras propuestas, y los procesos intercambiaran toda la información necesaria, llegando a cubrir la práctica totalidad de los requerimientos exigidos para un lenguaje de modelado de procesos de software, algo que consideramos de gran utilidad para un uso sistemático en las organizaciones de software. Por otro lado está el aspecto de la mantenibilidad. En el fondo el problema viene a ser el mismo, puesto que lo que se plantea es facilitar la natural evolución en el tiempo de los procesos de software, para lo cual podemos utilizar un lenguaje de modelado de procesos de software diferentes que incorpore nuevas características.

3.3. Objetivos del trabajo de tesis

Una vez planteado el problema a resolver y el contexto del mismo, ilustrado de forma más concreta con la presentación de las tres situaciones prácticas anteriores, el siguiente paso consiste en enunciar la propuesta de solución y los objetivos que con ella se pretenden alcanzar.

Ya hemos comentado anteriormente que en este trabajo de tesis vamos a utilizar como metáfora el esperanto para establecer la propuesta de solución, de forma que cada organización elija el lenguaje de modelado de procesos de software que considere oportuno según sus propios criterios y pueda contar con un marco de referencia que facilite la interoperabilidad y mantenibilidad de los modelos de procesos de software. Este marco está sustentado en tres pilares fundamentales: un lenguaje de modelado de procesos de software propuesto como lenguaje base, el método que permite su utilización y las transformaciones necesarias como nexo entre los lenguajes de modelado de procesos de software que se incorporan a nuestro marco de referencia.

A la vista de esta propuesta de solución, a continuación se presentan los objetivos a alcanzar en este trabajo de tesis. Son los siguientes:

1. Realizar un estudio del estado del arte de los diferentes lenguajes de modelado de procesos de software existentes, las causas por las que han sido creados y cuáles son los aspectos que llevan a las organizaciones a su utilización en entornos reales, si es que esto ocurre. Este objetivo ya se ha alcanzado con el estudio expuesto en el capítulo anterior.
2. Definir un lenguaje de modelado de procesos de software que sirva de lenguaje base en nuestro marco de referencia.
3. Definir un método para incorporar nuevos lenguajes de modelado de procesos de software al marco de referencia.
4. Definir las transformaciones que constituyen el punto de unión de todos los lenguajes de modelado de procesos de software usando el lenguaje base.
5. Implementar una herramienta de soporte para que el marco de referencia pueda ser utilizado en la práctica por organizaciones de software.

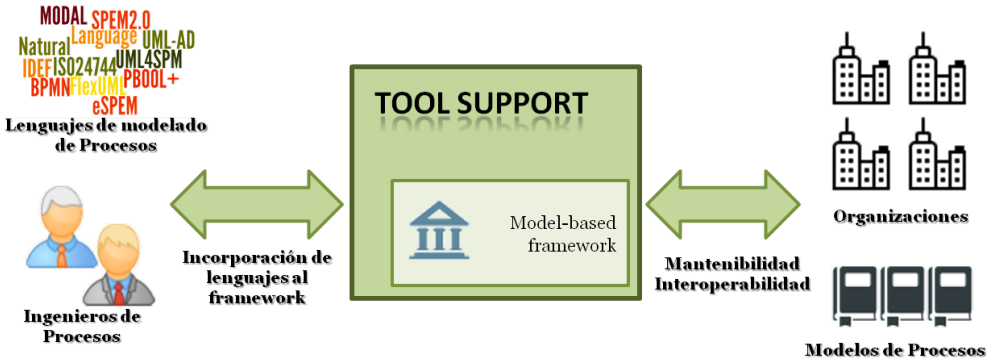


Figura 3.2: Vista del contexto del marco de referencia

6. Validar el marco de referencia con casos de estudio reales.

3.4. Presentación de la solución

En las secciones anteriores hemos planteado el problema a resolver y establecido los objetivos que se desean alcanzar. Antes de entrar en el detalle de cada una de las partes del trabajo de tesis, en esta sección se presenta un esquema que permite vislumbrar en qué va a consistir el marco de referencia para facilitar la interoperabilidad y mantenibilidad de modelos de procesos de software que aquí se está formulando.

En la figura 3.2 se muestra una primera vista del contexto en el que se va a desarrollar dicho marco de referencia. En ella se puede observar cómo, por un lado, es necesario incorporar los lenguajes de modelado de procesos de software que se deseen utilizar al marco de referencia. Por otro lado, los ingenieros de procesos pueden modelar sus procesos de software en el lenguaje de modelado de procesos de software que hayan elegido y que previamente haya sido incorporado al marco. Ambas acciones van a permitir y facilitar la interoperabilidad y mantenibilidad de los modelos de procesos definidos entre diferentes organizaciones, tal y como se establecía en la situaciones prácticas anteriores. Además, dicho marco estará soportado por una herramienta para su utilización por parte de organizaciones reales.

Haciendo mayor énfasis en el marco de referencia en sí, ya hemos comentado anteriormente que va a estar sustentado en tres pilares básicos: un lenguaje de modelado de procesos de software que va a ser propuesto como el lenguaje base en el marco, el método para incorporar nuevos lenguajes al marco de referencia y las transformaciones que hacen posible tanto la interoperabilidad como la mantenibilidad. Este planteamiento queda reflejado en la figura 3.3. Antes de continuar detallando la estructura de la solución queremos matizar el hecho de que cada pieza de la propuesta aquí planteada va a estar basada en estándares, y nos referiremos a ellos conforme se vaya pormenorizando en cada una de ellas.

Tal y como hemos podido comprobar en el estudio del estado actual, el número de lenguajes

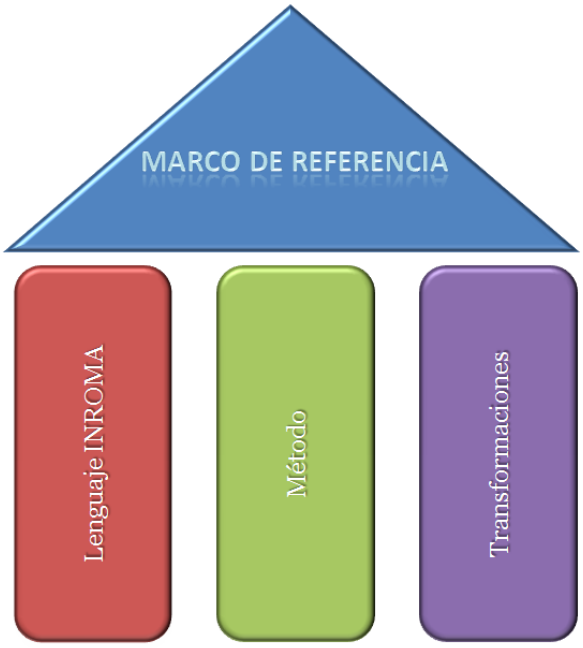


Figura 3.3: Los pilares del marco de referencia

de procesos de software definidos es considerable, y cada uno de ellos ha sido propuesto para dar respuesta a una determinada necesidad o requerimiento. El lenguaje base de modelado de procesos de software del marco de referencia no pretende ser un lenguaje más, sino que aspira a convertirse en ese lenguaje de uso común por todas las organizaciones, para permitir la interoperabilidad y mantenibilidad de los procesos de software. Dado que este es el fin con el que se crea, lo hemos denominado INROMA (INteROperabilidad y MAntenibilidad). En sí mismo es un lenguaje de modelado de procesos de software y como tal contiene los elementos necesarios para serlo. Pero se trata de un lenguaje sencillo y de fácil aprendizaje, que incorpora el conjunto mínimo común de elementos para poder modelar un proceso de software, y por tanto, puede ser utilizado como lenguaje de modelado especialmente en aquellas organizaciones con escaso conocimiento o experiencia en ingeniería de procesos, aunque éste no ha sido el objetivo con el que se ha definido. Siguiendo la tendencia actual de utilizar metamodelos en la definición de un nuevo lenguaje, INROMA va a estar basado en el estándar para el metametamodelado Meta-Object Facility (MOF) [OMG, 2014a] propuesto por el organismo de referencia en este ámbito, el Object Management Group (OMG) [2], para la definición de lenguajes y metamodelos, ya que se trata de la base a partir de la cual se desarrollan la mayor parte de los metamodelos que siguen el paradigma MDE.

La segunda pieza del marco de referencia es el método con el que nuevos lenguajes pueden ser incorporados al marco de referencia y aprovecharse de sus capacidades de interoperabilidad y mantenibilidad. Siguiendo dicho método se establecen las correspondencias entre los elementos presentes en INROMA y el concepto conveniente en el lenguaje que se pretende incluir.

[2] http://www.omg.org/

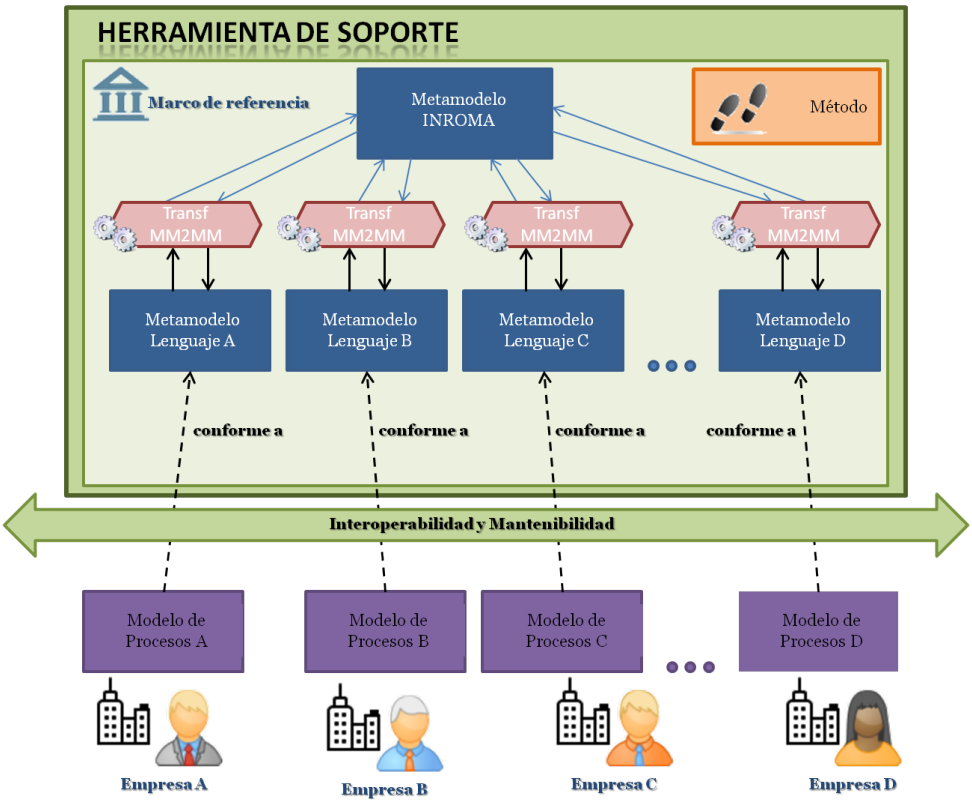


Figura 3.4: Vista detallada del marco de referencia

Por último se encuentra la tercera componente clave: las transformaciones. Con ellas se formalizan las correspondencias obtenidas con el método, y constituyen el punto de unión para la interoperabilidad de los lenguajes y, por tanto, de los procesos modelados con ellos. Al igual que en el caso anterior, para la definición de las transformaciones haremos uso de un estándar de referencia, en este caso Query/View/Transformation (QVT) [OMG, 2011a], definido por OMG para tales fines.

En la figura 3.4 se presenta una visión todavía más detallada del marco de referencia, incluyendo cada uno de sus pilares y las relaciones entre ellos. Así se observa cómo los diferentes lenguajes de modelado de procesos de software se encuentran incorporados en el marco mediante el método para poder hacer uso de ellos, y cómo se han definido las transformaciones bidireccionales entre los diferentes metamodelos de los lenguajes incorporados y el lenguaje INROMA. De esta forma se puede observar cómo el modelo de proceso A, que utiliza el lenguaje de modelado de procesos de software A, puede interoperar con cualquier otro modelo de procesos que utilicen alguno de los lenguajes de modelado que se han incorporado al marco de referencia a través de INROMA. Y, como ya hemos dicho anteriormente, el marco se implementa en una herramienta de soporte a la que hemos denominado MONETA, y que permite llevar a la práctica, con posibilidad de éxito, el planteamiento teórico de este marco de referencia, automatizando y haciendo lo más transparente posible a los usuarios la existencia de las transformaciones y el lenguaje INROMA.

3.5. Influencias conceptuales y tecnológicas

A la hora de definir cómo plantear el desarrollo de este trabajo de tesis ha existido un conjunto de aspectos, tecnologías y trabajos previos que han influido en la forma en la que se ha desarrollado. Estas influencias van a ser expuestas de forma detallada a continuación.

3.5.1. Las lecciones aprendidas de la revisión sistemática de la literatura

Cuando al comienzo del trabajo de tesis nos propusimos conocer cuál era el estado del arte en cuanto a lenguajes de modelado de procesos de software y nos planteamos realizar una revisión sistemática de la literatura, uno de nuestros primeros objetivos fue el conocer cuál era el lenguaje que pudiera considerarse como estándar, que cubriera todas las expectativas de los ingenieros de procesos. En el fondo queríamos encontrar cuál era el lenguaje *ideal*, y si no existía plantear una propuesta que sirviera como tal.

Fue durante el largo proceso de ejecución de la revisión y a través de las sugerencias recibidas de los revisores cuando vimos que, con toda seguridad, ese lenguaje *ideal* no existe tal y como esperábamos, es decir, no había ninguno mejor que otro, y cada lenguaje definido tenía ventajas y desventajas en función del ámbito en el que se aplicara. Por ello, el objetivo a partir de este momento era el buscar un mecanismo que nos permitiera obtener lo mejor de cada uno, utilizando siempre el más adecuado en cada momento.

3.5.2. Ingeniería dirigida por modelos

Por otro lado, tal y como se ha detallado en el capítulo 2, una de las conclusiones de la revisión sistemática es que la tendencia actual a la hora de definir nuevos lenguajes de modelado de procesos de software es utilizar enfoques basados en modelos. Así, parece adecuado que el planteamiento de solución tenga un enfoque basado en modelos. Por ello, el paradigma MDE va a constituir un elemento fundamental, puesto que va a convertirse en el sustento tecnológico sobre el que va a desarrollarse la solución planteada en este trabajo de tesis. A continuación se describen brevemente los fundamentos sobre los que este paradigma ha sido definido y los conceptos que en él se manejan.

La mente humana, continuamente y de forma inadvertida, está aplicando sus procesos cognitivos ante la realidad, algo que altera la forma subjetiva en la que percibe las cosas. Entre los diferentes procesos cognitivos que se aplican, la abstracción es uno de los más habituales [Brambilla *et al.*, 2012]. De forma sencilla, la abstracción consiste en la capacidad de encontrar elementos comunes en diferentes observaciones y, a partir de ellas, generar una representación mental de la realidad.

En la actividad científica, la abstracción ha sido y es ampliamente utilizada, y a menudo se refiere a ella como la actividad de modelar. Si definimos un *modelo* como una representación

parcial o simplificada de la realidad, que nos permite abordar una tarea compleja con un determinado fin, obtenemos que se puede considerar a un modelo como el resultado de la abstracción.

Teniendo en cuenta que la complejidad del software desarrollado ha ido en aumento en las últimas décadas, investigadores y desarrolladores vieron en la abstracción y, en consecuencia, en los modelos, una alternativa para abordar dicha complejidad [Schmidt, 2006]. El paradigma MDE plantea el uso de los modelos como un mecanismo para alcanzar lo concreto desde lo abstracto [Fondement y Silaghi, 2004]. De esta manera, en las primeras etapas de un desarrollo software se trabaja con modelos de alto nivel de abstracción, con un alto contenido en información del dominio, y a medida que se va avanzando por las siguientes etapas, este nivel de abstracción va disminuyendo en nuevos modelos, que contienen más detalles de implementación final. Es decir, unos modelos más abstractos se transforman en otros más concretos con el objetivo de producir software. Expresándolo como una ecuación [Brambilla *et al.*, 2012] podríamos decir que:

$$\text{Modelos} + \text{Transformaciones} = \text{Software}$$

De acuerdo a esta ecuación, los elementos básicos en MDE para el desarrollo de software son los modelos y las transformaciones. Para poder utilizar los modelos como una herramienta de trabajo, éstos deben generarse de acuerdo a las reglas de un determinado lenguaje de modelado (*Modeling Language*) o formalismo. Este lenguaje define la sintaxis del modelo (o notación) y su semántica (o significado). La figura 3.5 [Metzger, 2005] muestra gráficamente estas relaciones, manteniendo la terminología del autor, aunque más adelante se irá haciendo la equivalencia con los términos que hemos utilizado en este trabajo de tesis.

La sintaxis de un lenguaje de modelado está formada por una sintaxis concreta y una sintaxis abstracta. La sintaxis abstracta describe la estructura del lenguaje y la forma en la que se pueden combinar los distintos elementos, independientemente de su representación. La sintaxis concreta especifica la representación legible (gráfica o textual) de los elementos abstractos. Por otro lado, la semántica, que proporciona parte estática y dinámica, nos plantea restricciones y establece el significado de los elementos del lenguaje y de las diferentes formas de combinarlos.

La definición de un lenguaje de modelado en este contexto sigue los principios de MDE. La sintaxis abstracta se define en términos de un modelo y se denomina *Metamodelo*. Podemos definir un metamodelo como un tipo especial de modelo que especifica un lenguaje de modelado [Mellor, 2004]. El metamodelo define la estructura y restricciones para una familia de modelos.

En lo que se refiere a las transformaciones, éstas son el mecanismo que nos permite derivar unos modelos a partir de otros modelos ya existentes. Una transformación entre dos modelos representa una relación entre dos sintaxis abstractas y se define mediante un conjunto de relaciones entre los elementos de los correspondientes metamodelos [Thiry y Thirion, 2009]. Las transformaciones pueden ser verticales u horizontales [Metzger, 2005]. Una transformación es vertical si el modelo derivado tiene un menor nivel de abstracción que el modelo inicial. Por el contrario, una transformación es horizontal si el modelo derivado y el origen tienen el mismo nivel de abstracción.

Otro aspecto interesante de la filosofía que subyace en MDE, donde *"Everything is a model"*

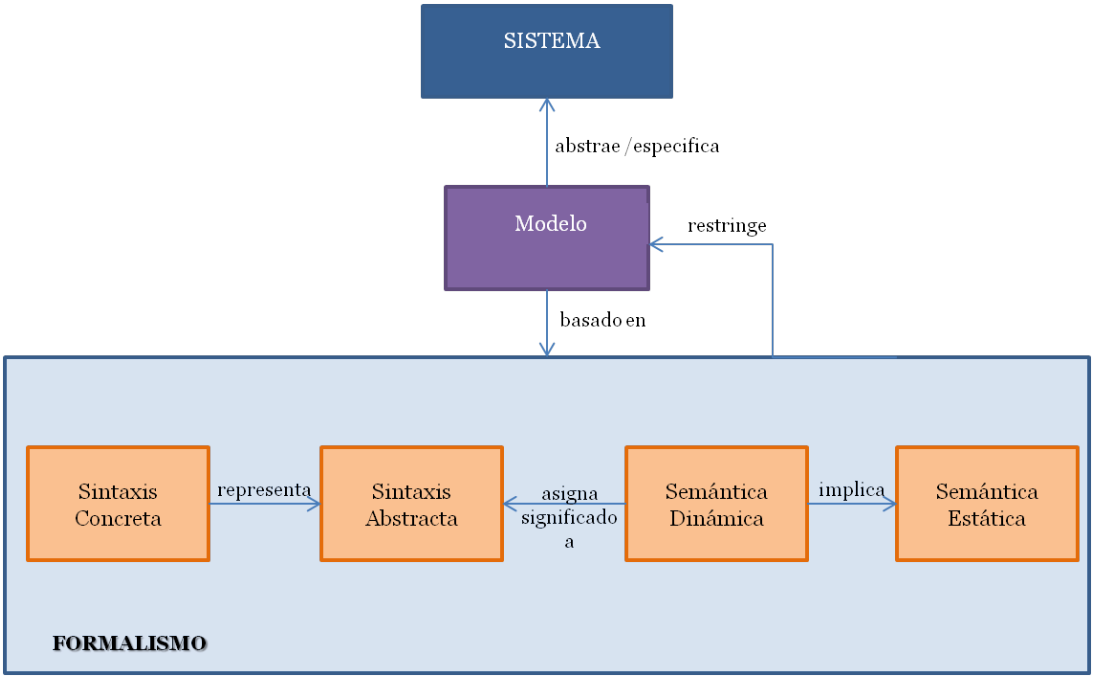


Figura 3.5: Relación entre sistema, modelo y formalismo

[Bézivin, 2005], es el hecho de que las transformaciones en sí mismas puedan ser vistas también como modelos, y gestionadas como tales, incluyendo la definición de su metamodelo [Brambilla *et al.*, 2012]. En la figura 3.6 se muestran las transformaciones entre modelos desde el punto de vista MDE: un programa de transformación de modelos toma como entrada un modelo que es conforme a un metamodelo origen y produce como salida un modelo conforme al metamodelo destino. El programa de transformación, compuesto por un conjunto de reglas, debería ser considerado en sí mismo un modelo y, en consecuencia, estar basado en su correspondiente metamodelo, que es una definición abstracta del lenguaje de transformación utilizado [Di Ruscio *et al.*, 2012].

Éstos son, de forma muy breve, los principales conceptos con los que se trabaja en los entornos basados en modelos o MDE, y que nos van a proporcionar el fundamento teórico sobre el que construir la solución. En los capítulos posteriores, conforme se vaya requiriendo por el contexto, profundizaremos en alguno de estos conceptos que aquí simplemente hemos presentado.

3.5.3. ISO/IEC TR 24744:2007

La norma ISO/IEC TR 24744:2007[ISO/IEC, 2007b] es un estándar ISO que, ante la proliferación de estándares, modelos y guías de buenas prácticas con nociones similares cuya descripción de proceso varía en formato, contenido y nivel de prescripción, establece cuáles son los conceptos fundamentales que un lenguaje de modelado de procesos debe contener, estableciendo unas guías para la definición de modelos de procesos y así mejorar la consistencia y uniformidad en la definición de estándares. Como el ámbito que nos ocupa es el de procesos de software, y se trata de un subconjunto concreto de procesos, estos lenguajes también deberían incluir al menos

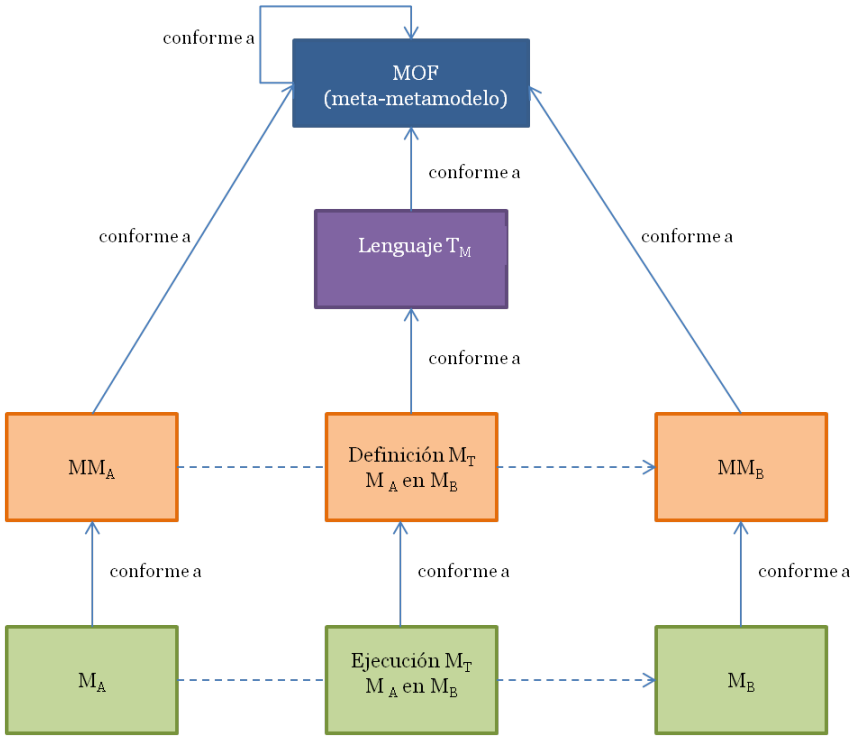


Figura 3.6: Definición de las transformaciones entre modelos

estos conceptos, que pasamos a detallar a continuación.

- **Título**. Resume el alcance del proceso, identificando su interés principal y distinguiéndolo de otros procesos dentro del contexto de modelos de procesos.
- **Propósito**. Describe el objetivo alcanzado si el proceso se lleva a cabo.
- **Resultados**. Son los resultados observables y evaluables de la consecución exitosa del propósito del proceso.
- **Actividades**. Definen un conjunto de acciones que se llevarían a cabo durante la ejecución del proceso.
- **Tareas**. Se refiere a las acciones especificas que se ejecutan para realizar una actividad y obtener sus resultados.
- **Elementos de información**. Determinan información utilizada por las personas que se producen y almacenan durante el ciclo de vida del software y del sistema.

3.5.4. La aplicación práctica en organizaciones de software y NDT

Uno de los aspectos que mayor influencia ha tenido en el desarrollo de este trabajo de tesis es la importancia que para nosotros tiene la aplicación práctica de la solución planteada. Cuando

hablamos de aplicación práctica nos referimos a la capacidad de ser llevada y transferida a organizaciones de software en un entorno empresarial, no únicamente en el ámbito académico.

En el grupo de investigación en el que se ha desarrollado este trabajo de tesis ya existía una propuesta metodológica ampliamente extendida en organizaciones de software y no software. Dicha propuesta, denominada NDT (Navigational Development Techniques) [Escalona, 2004], fue desarrollada inicialmente con el objetivo de acometer los requisitos de un sistema, incorporando un proceso y un conjunto de artefactos. Actualmente NDT ha evolucionado hasta convertirse en una metodología web guiada por modelos que ofrece un soporte completo, incluyendo herramientas, para todo el ciclo de vida de un sistema software: desarrollo, pruebas, aseguramiento de la calidad y gestión del proyecto [Escalona y Aragon, 2008].

NDT constituye una metodología completa con todos sus procesos definidos. Es más, ha sido implantada en varias organizaciones de software o utilizada como base para el desarrollo de otras metodologías. Dada la relevancia que NDT tiene en el grupo en el que se desarrolla este trabajo de tesis y su amplia aceptación en la industria, sus procesos y las necesidades que en su modelado se han tenido, nos han servido de guía para inferir las características que un lenguaje base de modelado de procesos de software debería disponer.

Una de las aspiraciones de este trabajo de tesis es que el lenguaje base de modelado de procesos de software que aquí se defina, es decir, INROMA, se convierta en el lenguaje utilizado en NDT para el modelado de procesos de software, puesto que de esta forma queda de manifiesto que cumple las necesidades que los procesos de software de las organizaciones están demandando, teniendo además como soporte teórico el estándar ISO/IEC TR 24744:2007. NDT, por tanto, no constituye únicamente una influencia, sino que su enriquecimiento se convierte en un fin en si mismo para este trabajo de tesis.

3.6. Conclusiones

En este capítulo se ha realizado el planteamiento del problema a resolver a raíz de los resultados obtenidos en la revisión sistemática de la literatura presentada en el capítulo anterior y de otros aspectos determinantes que aquí se han descrito. Derivado del problema a resolver, se ha presentado la propuesta de solución y los objetivos a alcanzar en este trabajo de tesis. Además, se ha presentado un esquema de la estructura de la solución que permite vislumbrar la propuesta planteada. Por último, las principales influencias que han tenido repercusión en el desarrollo de este trabajo han sido también expuestas.

En los siguientes capítulos se abordará en detalle la propuesta y el desarrollo de la solución que nos permita alcanzar los objetivos que aquí se han marcado.

Capítulo 4

El lenguaje base de modelado de procesos de software INROMA

Una vez establecido el alcance de este trabajo de tesis con la descripción del problema a resolver, los objetivos que se pretenden alcanzar y la propuesta de solución con su estructura, a partir de este punto vamos a ir tratando en profundidad una por una las piezas sobre las que se sustenta el marco de referencia para facilitar la interoperabilidad de procesos de software, de manera que se vayan cubriendo los objetivos planteados.

Para comenzar se profundiza en el lenguaje INROMA (INteROperabilidad y MAntenibilidad), el primero de los elementos que han aparecido en la propuesta de solución. INROMA ha sido desarrollado como un lenguaje base de modelado de procesos de software, con el objetivo de facilitar la interoperabilidad y mantenibilidad de procesos de software, pero constituye un lenguaje en sí mismo tal y como podremos constatar a lo largo de este capítulo. Teniendo en cuenta que se plantea bajo el paraguas del paradigma MDE, la definición del lenguaje se hace mediante un metamodelo.

Con el objetivo de exponer de forma clara el lenguaje INROMA, este capítulo se estructura de la siguiente manera. Comenzaremos con la definición de los objetivos de diseño y las capacidades que se espera tenga el lenguaje INROMA. A continuación se detalla el lenguaje en sí, exponiendo el metamodelo e incorporando el detalle de todos los elementos que lo constituyen, su estructura y relaciones. En la tercera sección se realiza una evaluación del lenguaje de acuerdo a las capacidades y objetivos previamente establecidos, de la que se extraen las aportaciones que INROMA proporciona. A continuación, en la siguiente sección se trata el aspecto de la extensibilidad de INROMA como aspecto inherente a su definición y las ventajas que nos aporta. Para finalizar, en la última sección se recopilan las principales conclusiones de este capítulo.

4.1. Objetivos de diseño y capacidades requeridas a INROMA

En los capítulos anteriores hemos referido en varias ocasiones que, de todos los lenguajes de modelado de procesos de software que se han estudiado en el estado del arte, ninguno se ha erigido por encima del resto como una opción prioritaria, a pesar de los varios intentos de estandarización auspiciados por los organismos más importantes en estas materias, como pueden ser OMG o ISO. También se ha comentado en diversas ocasiones que el uso de un lenguaje de modelado de procesos de software de forma sistemática en las organizaciones de software, más allá de simples pruebas de concepto, está lejos de lo que cabría esperar para iniciativas de estas características.

Frente a este planteamiento orientado a la estandarización, la propuesta del marco de referencia que se hace en este trabajo de tesis se basa en un enfoque en favor de la diversidad de lenguajes de modelado de procesos de software, de forma que cada organización puede elegir y mantener aquél que considere más adecuado por conocimientos o características, sin que para ello suponga una restricción posterior a la hora de trabajar con los procesos así modelados.

Estas reflexiones han condicionado seriamente los objetivos de diseño para el desarrollo del lenguaje de modelado de procesos de software incluido en el marco de referencia para la interoperabilidad y mantenibilidad de procesos de software, objetivos que pasamos a enumerar a continuación.

- Al tratarse de un lenguaje que podemos denominar base, es necesario que sea un lengu aje sencillo y fácil de aprender y utilizar. Salvando siempre las distancias, si se plantea INROMA como el esperanto de los lenguajes de modelado de procesos de software, es fundamental esa sencillez ya que en muchos casos va a ser considerado dentro de las organizaciones como un lenguaje adicional de modelado de procesos de software, puesto que ya se dispone de alguno, sino de varios. Muchos de los lenguajes estudiados son bastante complicados, por lo que definir un nuevo lenguaje de gran complejidad no nos garantizaría el propósito que andamos buscando. Por ello, un primer requerimiento del metamodelo que queremos proponer ha sido la comprensibilidad, entendido como la capacidad de ser inteligible por personas no expertas en este ámbito.
- Un segundo punto, fuertemente relacionado con el anterior, tiene que ver con la relación del metamodelo con los estándares existentes y ampliamente utilizados en el día a día dentro de las organizaciones de software, como puede ser UML. El objetivo consiste en que el lenguaje deberá estar basado en estándares de uso habitual, lo que se traduce en una mayor facilidad para su adopción, reduciendo la propia curva de aprendizaje que la incorporación de cualquier nuevo elemento conlleva, así como el mantenimiento posterior, puesto que no es necesario aprender algo completamente desde el principio.
- Debe contener los conceptos fundamentales que describan un proceso de software, pero ninguno adicional, ya que estamos buscando el mínimo común de los conceptos que aparecen en cualquiera de los lenguajes de modelado de procesos de software. De esta forma se garantiza que no se privilegia ningún lenguaje frente al resto, todos parten con las mismas oportunidades de participación en el marco de referencia para la interoperabilidad y mantenibilidad de procesos de software.

- En línea con lo anterior, el lenguaje debe contener, al menos, los conceptos mínimos que proporcionen la expresividad suficiente para poder modelar procesos de software. A este respecto se utilizan los elementos incluidos en la norma ISO/IEC TR 24744 como garantía de dicha expresividad, puesto que se trata de un estándar reconocido a nivel internacional.

Atendiendo a estos objetivos de diseño, establecemos las capacidades que INROMA debe disponer, y que resumimos en los dos puntos siguientes:

- **Un lenguaje de modelado de procesos de software**. Cumpliendo los objetivos de diseño planteados, INROMA es un lenguaje de modelado de procesos de software en sí mismo. Un lenguaje muy sencillo pero lenguaje al fin y al cabo. Esta capacidad viene garantizada por el hecho de que el lenguaje, según hemos establecido, recogerá todos los conceptos esenciales de procesos de acuerdo a la norma ISO/IEC TR 24744. Además, estará basado en estándares de uso común en la industria del software para su fácil adopción. De esta forma, el metamodelo que definamos constituirá la sintaxis abstracta del lenguaje de modelado de procesos de software y utilizará la sintaxis concreta de UML como propia. Además, al tratarse de un lenguaje de modelado de procesos de software sencillo, podrá ser utilizado como tal por personas no expertas en procesos, que quieran comenzar a definirlos de una manera más formalizada que el típico documento de manual de calidad, o por aquellas organizaciones que no tienen los recursos adecuados y quieran disponer de un lenguaje sencillo de modelado. Además, como lenguaje propiamente dicho, los requisitos principales que debe cumplir, de acuerdo con los establecidos en el capítulo anterior, son: expresividad, comprensibilidad y estandarización.
- **Es pieza clave en el marco de referencia para la interoperabilidad y mantenibilidad de procesos de software**. El metamodelo va a incluir el mínimo conjunto común de conceptos que son necesarios para modelar un proceso de software, ni más ni, por supuesto, menos. Esto nos garantiza por un lado que cualquier lenguaje de modelado de procesos de software se puede incorporar al marco de referencia con el objetivo de establecer la interoperabilidad y, por otro, que ningún lenguaje tenga privilegios frente al resto, en la interoperabilidad con INROMA, puesto que no incorpora más allá de los conceptos y elementos necesarios y suficientes.

Una vez se han establecido tanto sus objetivos de diseño, como las capacidades que se pretenden, en la siguiente sección se presenta el metamodelo que constituye la sintaxis abstracta de INROMA.

4.2. El metamodelo del lenguaje INROMA

En esta sección se presenta el metamodelo del lenguaje de modelado de procesos de software INROMA, que se incluye como una de las piezas fundamentales en el marco de referencia propuesto en este trabajo de tesis. La figura 4.1 muestra dicho metamodelo. A continuación ofrecemos una explicación detallada de cada una de las metaclases que lo constituyen. Para cada una de ellas se presenta su descripción, atributos, asociaciones, y sus generalizaciones directas y restricciones si existieran. Las metaclases se introducen de acuerdo a su importancia, siguiendo un orden lógico para una mayor comprensión del metamodelo.0 y no por orden alfabético.

4.2.1. Process

Descripción

La metaclase *Process* es la clase principal del metamodelo, alrededor de la cual se definen todos los demás elementos del metamodelo. De acuerdo a la definición de proceso de software introducida en el primer capítulo, representa un conjunto ordenado de acciones que son ejecutadas por alguien con el objetivo de producir una serie de resultados o productos. Permite representar cualquier proceso de software.

Generalización

Ninguna

Atributos

- *name:String* El nombre del proceso que constituye el identificador del mismo para los usuarios .
- *description:String* Ofrece una descripción detallada del proceso de forma textual, incluyendo aspectos tales como la finalidad del mismo, los objetivos que debe cumplir, etc.

Asociaciones

- *elements:ProcessElement[3..*]* Conjunto de los elementos que forman parte de la estructura de un proceso. Todo proceso tiene que tener un elemento inicial y, al menos, un elemento final y una actividad.
- *sequence:ProcessSequence[0..*]* Conjunto de conexiones que muestran el flujo entre los elementos del proceso, y que muestran el orden seguido por dichos elementos.
- *indicators:Indicator[0..*]* Conjunto de indicadores que definen los aspectos a medir de un proceso.

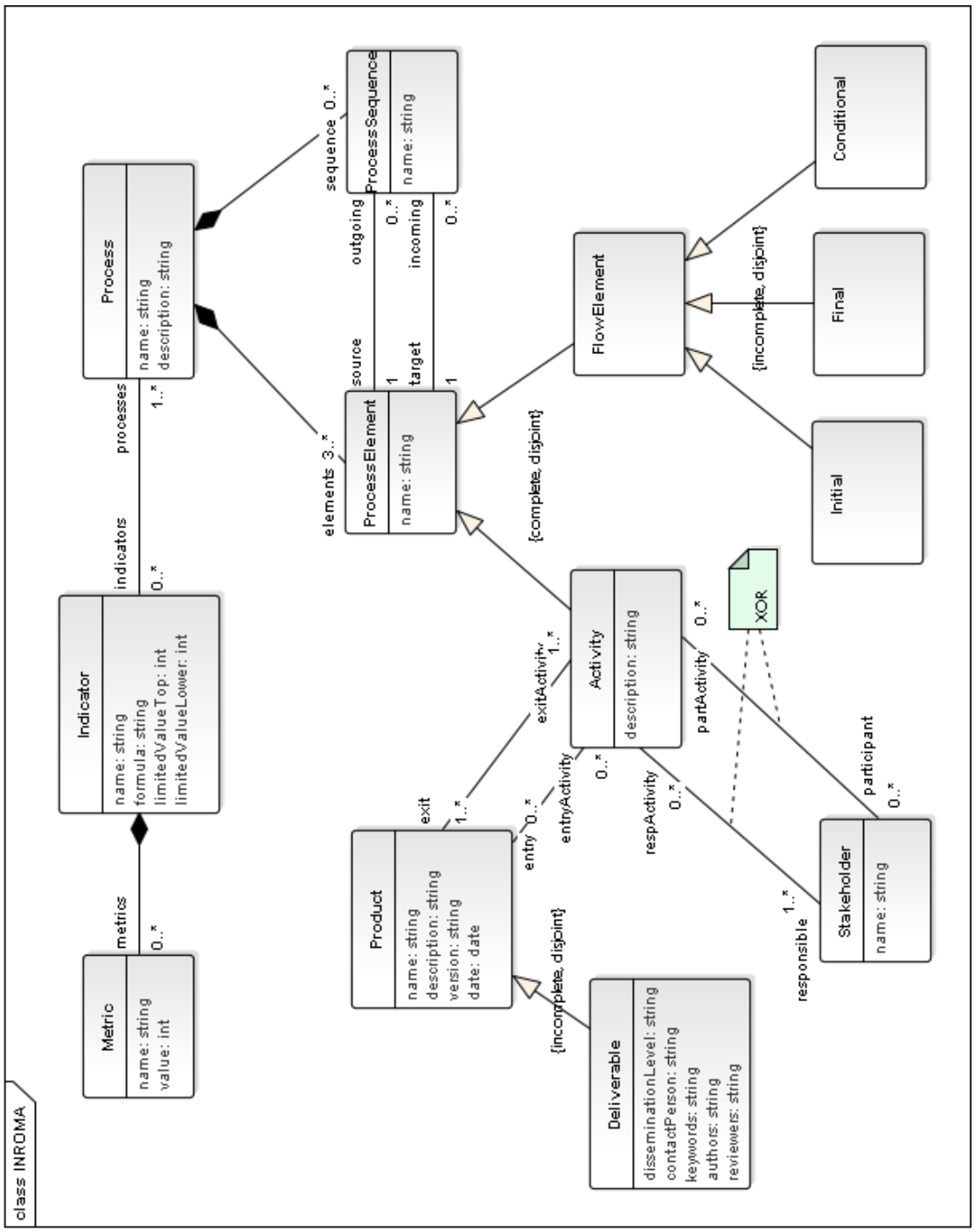


Figura 4.1: Metamodelo del lenguaje INROMA

Restricciones

Todo proceso debe respetar los siguientes requerimientos mínimos:

- Son procesos de entrada única - existe exactamente un único elemento de inicio en el proceso. El fragmento OCL 4.1 describe la restricción.

```
1 context Process
2 inv exactlyOneStart : (self.processElement → select(os|isTypeOf(Initial))) →
      size() = 1
```

Algoritmo 4.1: Restricción sobre la metaclase *Process*: El proceso tiene un elemento inicial

- Son procesos de múltiples salidas - al menos hay un elemento final en el proceso. El fragmento OCL 4.2 describe la restricción.

```
1 context Process
2 inv multipleEnds : (self.processElement → select(os|isTypeOf(Final))) → size
      () >= 1
```

Algoritmo 4.2: Restricción sobre la metaclase *Process*: El proceso tiene al menos un elemento final

- Son procesos que tienen al menos una actividad El fragmento OCL 4.3 describe la restricción.

```
1 context Process
2 inv atLeastOneActivity : (self.processElement → select(os|isTypeOf(Activity))
      ) → size() >= 1
```

Algoritmo 4.3: Restricción sobre la metaclase *Process*: El proceso tiene al menos una actividad

4.2.2. ProcessElement

Descripción

La metaclase *ProcessElement* representa cualquier elemento que pueda ser incorporado a un proceso. Un elemento de proceso se encuentra ordenado dentro del mismo a través de la secuencia definida para el proceso.

Generalización

Ninguna

Atributos

- *name:String* El nombre del elemento del proceso, que constituye su identificador unívoco de cara al usuario.

Asociaciones

- *process:Process[1]* Establece cuál es el proceso del que forma parte este elemento de proceso.
- *incoming:ProcessSequence[0..*]* Conjunto de enlaces de entrada al elemento de proceso, es decir, aquellas que lo tienen como destino.
- *outgoing:ProcessSequence[0..*]* Conjunto de enlaces de salida al elemento de proceso, es decir, aquellas que lo tienen como origen. Este atributo junto con el anterior permiten construir la estructura ordenada de elementos que conforman el flujo de proceso.

Restricciones

Ninguna

4.2.3. ProcessSequence

Descripción

La metaclase *ProcessSequence* establece las conexiones o enlaces entre dos elementos de proceso, por lo que posibilita la definición del flujo del proceso.

Generalización

Ninguna

Atributos

- *name:String* El nombre del elemento de secuencia, que constituye su identificador unívoco de cara al usuario.

Asociaciones

- *process:Process[1]* Representa el proceso del que forma parte esta conexión.
- *source:ProcessElement[1]* Constituye el elemento de proceso origen, es decir, desde el que parte dicho enlace.
- *target:ProcessElement[1]* Constituye el elemento de proceso destino, es decir, hacia el que se dirige la conexión.

Restricciones

Ninguna

4.2.4. Activity

Descripción

La metaclase *Activity* describe cualquier fragmento de trabajo que debe ser ejecutado durante el proceso de software.

Generalización

ProcessElement

Atributos

- *description:String* Ofrece una breve descripción de la actividad donde se resumen de forma textual los elementos relacionados con la misma.

Asociaciones

- *entry:Product[0..*]* Conjunto de productos de entrada que son necesarios para ejecutar la actividad.
- *exit:Product[1..*]* Conjunto de productos de salida que se obtienen como resultado de la ejecución de esta actividad.
- *participant:Stakeholder[0..*]* Roles que participan en la ejecución de una actividad pero no es el responsable de la misma.
- *responsible::Stakeholder[1..*]* Rol (o roles) responsable de la ejecución de una actividad, y que nos permite reflejar que para toda actividad siempre debe haber al menos un responsable.

Restricciones

Ninguna

4.2.5. FlowElement

Descripción

La metaclase *FlowElement* representa los elementos que forman parte de la estructura de un proceso pero no corresponden a la ejecución de ningún fragmento de trabajo. Son elementos puramente estructurales que nos permiten ajustar el flujo.

Generalización

ProcessElement

Atributos Ninguna

Asociaciones Ninguna

Restricciones

La metaclase *FlowElement* ha sido especializada en una serie de metaclases concretas. Con el propósito de no limitar la futura extensión del metamodelo, esta especialización se ha establecido como una relación de herencia UML incompleta y disjunta.

4.2.6. Initial

Descripción

La metaclase *Initial* representa el punto de entrada al proceso, a partir del cual se secuencian el resto de los elementos de proceso, pero no corresponde a la ejecución de ningún fragmento de trabajo.

Generalización

FlowElement

Atributos Ninguna

Asociaciones Ninguna

Restricciones El elemento inicial de un proceso no puede tener ningún enlace de entrada. El fragmento OCL 4.4 describe la restricción.

```
1 context Process
2 inv noIncoming : (self.processElement → select(os|isTypeOf(Initial))).incoming →
    size() = 0
```

Algoritmo 4.4: Restricción sobre la metaclase *Initial*: El elemento inicial no tiene enlace de entrada

4.2.7. Final

Descripción

La metaclase *Final* representa el punto de salida de un proceso, tras el cual podemos decir que se ha finalizado.

Generalización

FlowElement

Atributos Ninguna

Asociaciones Ninguna

Restricciones El elemento final de un proceso no puede tener ningún enlace de salida. El fragmento OCL 4.5 describe la restricción.

```
1 context Process
2 inv noOutgoing : (self.processElement → select(os|isTypeOf(Final))).outgoing →
    size() = 0
```

Algoritmo 4.5: Restricción sobre la metaclase *Final*: El elemento final no tiene enlace de salida

4.2.8. Conditional

Descripción

La metaclase *Conditional* permite establecer ramas excluyentes en la secuencia lógica de actividades.

Generalización

FlowElement

Atributos Ninguna

Asociaciones Ninguna

Restricciones

La restricción que se establece sobre un elemento Conditional es que debe tener al menos dos enlaces de salida y, al menos, uno de entrada. El fragmento OCL 4.6 describe la restricción.

```
1 context ProcessElement
2 inv: self.isTypeOf (Conditional) implies (self.outgoing → size()>=2 and self.
    incoming → size()>=1)
```

Algoritmo 4.6: Restricción sobre la metaclase *Conditional*: al menos, dos enlaces de salida y, al menos, uno de entrada

4.2.9. Product

Descripción

La metaclase *Product* representa cualquier pieza de información consumida (entrada), producida (salida) o modificada (entrada y salida) durante la ejecución de una actividad del proceso.

Generalización

Ninguna

Atributos

- *name:String* El nombre del producto que constituye su identificador unívoco para el usuario.
- *description:String* Una breve descripción del producto en la que se especifica, entre otras cosas, si es de entrada, salida o de ambos para una actividad de un proceso.
- *version:String* La versión del producto.
- *date:date* La fecha del producto que especifica la última vez que fue modificado.

Asociaciones

- *entryActivity:Activity[0..*]* Un producto puede ser un elemento de entrada para una o varias actividades, aunque no necesariamente es obligatorio disponer de un producto de entrada.
- *exitActivity:Activity[1..*]* Un producto es un elemento de salida para una o varias actividades. El hecho de que siempre deba existir un producto de salida para una actividad sirve como evidencia de su ejecución.

Restricciones

Ninguna

4.2.10. Deliverable

Descripción

La metaclase *Deliverable* representa un tipo concreto de producto que es entregable. Es importante distinguir entre los productos que son artefactos transitorios dentro de un proceso y los que son entregables, que representan el resultado tangible de una actividad. Un producto entregable requiere de mayor atención por parte de los roles participantes y, sobre todo, del responsable de la actividad.

Generalización

Product

Atributos

- *disseminationLevel:String* Establece si se trata de un entregable público o confidencial.
- *contactPerson:String* La persona de contacto en todo aquello que haga referencia al entregable.
- *keywords:String* Las palabras claves por las que el entregable es indexado para posibles búsquedas .
- *authors:String* Los autores del entregable.
- *reviewers:String* Los revisores del entregable.

Asociaciones

Ninguna

Restricciones

Ninguna

4.2.11. Stakeholder

Descripción

La metaclase *Stakeholder* representa cualquier actor, ya sea una persona o un sistema, que se encuentra involucrado en la ejecución de una actividad. En INROMA hemos querido establecer la diferenciación entre un actor responsable de una actividad y aquel que participa en la misma, pero que necesita la aprobación del responsable para considerar que dicha actividad ha sido ejecutada correctamente.

Generalización

Ninguna

Atributos

- *name:String* El nombre del actor involucrado en la realización de la actividad.

Asociaciones

- *partActivity:Activity[0..*]* Conjunto de actividades en el que el actor participa.
- *respActivity:Activity[0..*]* Conjunto de actividades en el que el actor es el responsable de su correcta ejecución.

Restricciones

Ninguna

4.2.12. Indicator

Descripción

La metaclase *Indicator* representa un indicador de proceso que se establecer a partir de una determinada fórmula y los valores límite objetivo para poder evaluar si el proceso ha cumplido o no dicho indicador. Es un elemento básico para cualquier cuadro de mando y es exigido por la mayor parte de los estándares de calidad de procesos.

Generalización

Ninguna

Atributos

- *name:String* El nombre del indicador que constituye su identificador para el usuario.
- *formula:String* Establece la formula con la que se va a obtener la métrica.
- *limitedValueTop:String* Establece el límite superior establecido para la evaluación del indicador.
- *limitedValueLower:String* Establece el límite inferior del valor para la evaluación del indicador.

Asociaciones

- *processes:Process[1..*]* El conjunto de procesos sobre los que actúa y se calcula un indicador.
- *metrics:Metric[0..*]* Conjunto de las métricas utilizadas para el cálculo de un indicador.

Restricciones

Ninguna

4.2.13. Metric

Descripción

La metaclase *Metric* representa una medida cuantitativa que nos va a permitir evaluar un determinado aspecto del proceso.

Generalización

Ninguna

Atributos

- *name:String* El nombre de la métrica, que constituye un identificador único de la misma.
- *value:int* El valor de la métrica actual.

Asociaciones

- *indicator:Indicator[1]* Establece el indicador para el que la métrica se registra.

Restricciones

Ninguna

4.3. Evaluación de las capacidades de INROMA

En la sección anterior se ha detallado el metamodelo que constituye la sintaxis abstracta del lenguaje de modelado de procesos de software INROMA, explicando minuciosamente cada uno de los elementos que lo componen. El propósito de esta sección es evaluar dicho lenguaje de acuerdo a las capacidades que han sido definidas previamente: es un lenguaje de modelado de procesos de software en sí mismo y es una de las tres piezas clave en el marco de referencia para facilitar la interoperabilidad y mantenibilidad de procesos de software.

4.3.1. Evaluación de INROMA como lenguaje de modelado de procesos de software

Vamos a evaluar la capacidad de INROMA como lenguaje de modelado de procesos de software desde dos perspectivas diferentes. En primer lugar, a partir de los requisitos genéricos de lenguajes de modelado de procesos de software que hemos detallado en el capítulo 3, se explicará de forma detallada qué requisitos y cómo son cubiertos por INROMA. Por otro lado, utilizaremos INROMA como lenguaje para modelar un proceso definido en la metodología NDT, como ejemplo de proceso de software real actualmente utilizado en diferentes organizaciones de software [Cutilla *et al.*, 2011] [Escalona y Aragon, 2008]. Hemos preferido utilizar un proceso de NDT frente al uso de un ejemplo teórico como podría ser ISPW-6[Kellner *et al.*, 1991], conocido por su utilización como marco de comparación y evaluación de propuestas para modelar procesos de software, puesto que uno de nuestros principales objetivos es mostrar la aplicabilidad práctica de lo que estamos proponiendo. No obstante, una evaluación teórica con un *(benckmark)* también podría haber sido factible.

Comenzamos con los requerimientos para lenguajes de modelado de procesos de software que, como hemos visto en las tablas 3.1 y 3.2, se han recopilado de los trabajos realizados por diferentes autores, y su utilización para la validación de INROMA como lenguaje. Para ello, a continuación se explica en detalle cómo INROMA cumple varios de estos requisitos.

- **Expresividad** *(Expressiveness).*
  Hemos visto en la definición de este requerimiento que se refiere a la capacidad de expresar lo que realmente ocurre durante los procesos de software. Gracias a esta característica podemos observar si los aspectos más relevantes de un proceso de software se pueden modelar a partir de los elementos definidos en el lenguaje, concretamente *Actividad*, *Rol* y *Producto*.
  La noción de *Actividad* viene dada a través de la metaclase *Activity*. En el metamodelo, *Activity* representa cualquier fragmento o pieza de trabajo que puede ser realizada durante la ejecución del proceso de software, de forma que un proceso es constituido por el conjunto de estos fragmentos junto con los elementos propios del flujo.
  La noción de *Rol* se ofrece a tavés de la metaclase *Stakeholder*. Para las actividades de un proceso de software, un Stakeholder puede ser el responsable o ejecutor principal de la actividad o, por otro lado, simplemente participar en ella de alguna forma. Esto queda reflejado a través de las asociaciones entre *Stakeholder* y *Activity*.

Por último, en lo que se refiere a *Producto*, la equivalencia en nuestro metamodelo es la metaclase *Product*, que puede ser de entrada o de salida dentro de una actividad del proceso. Además, se establece la especialización de un producto como resultado de una actividad, a través de la metaclase *Deliverable*.

- **Comprensible** *(Understandability).*
  La capacidad de ser comprensible es un requerimiento crucial, ya que es lo que induce a las personas a elegir un lenguaje frente a otro [Bendraou, 2007]. Hace referencia a la capacidad del lenguaje de modelado de procesos de software de ser entendido por sus usuarios y, como es obvio, si los usuarios de un lenguaje no son capaces de entenderlo, difícilmente van a llevarlo a la práctica.
  En el caso de INROMA, al estar basado en UML y utilizar sus notaciones como sintaxis concreta, el cumplimiento de este requerimiento va casi implícito frente a la definición de nuevos elementos y, sobre todo, nuevas notaciones para referirse a ellos. UML es un estándar gráfico, intuitivo y fácil de entender, y dispone de una amplia comunidad de desarrolladores de software como usuarios del mismo.

- **Estandarización** *(Standardization).*
  La importancia de la estandarización en un lenguaje de modelado de procesos de software viene muy ligado al requerimiento anterior. Utilizar estándares ya ampliamente extendidos garantiza, en cierta medida, su utilización y facilidad de despliegue en organizaciones de la industria del software. En el caso de INROMA, este requerimiento se cumple desde dos perspectivas diferentes: por un lado, ya hemos dicho que INROMA hace uso de UML como base de su representación; por otro lado, los conceptos incorporados en el metamodelo son los que se encuentran incluidos en la norma ISO/IEC TR 24744, tal y como lo hemos establecido en los objetivos de diseño del mismo. Se trata, por tanto, de otra perspectiva de la estandarización, en este caso a nivel conceptual.

Estos tres eran los requerimientos principales que nos habíamos planteado cumplir a la hora de establecer las capacidades y los objetivos de diseño. Sin embargo, no son los únicos de los definidos en el capítulo 3 que son observados por INROMA, los cuales pasamos a detallar a continuación.

1. **Formalidad** *(Formality).*
   Este requerimiento hacía referencia al hecho de que la sintaxis y la semántica del lenguaje de modelado de procesos de software se ha definido formalmente y con una alta granularidad. Al estar basado en UML, hemos confiado en la literatura [Briand *et al.*, 2005], [McUmber y Cheng, 2001], [Evans y Kent, 1999], para caracterizar este requerimiento. UML se suele considerar un lenguaje semi-formal y, aunque no en su completitud, podemos decir que este requerimiento está planteado en INROMA.

2. **Soportado por herramientas** *(Tooling Support).*
   Con este requerimiento se facilitaba la utilización del metamodelo en entornos reales. El hecho de tener herramientas de soporte para la definición y despliegue de los procesos de software de acuerdo a un lenguaje es básico para que éste sea realmente utilizado en las empresas. Es más que probable que, en este caso, el lenguaje de modelado de procesos de software sea conocido únicamente por parte de los ingenieros de procesos,

siendo transparente para el resto de la organización, que lo utilizarían de forma implícita. Obviamente, este tipo de soporte debería ir más alla de un simple editor de textos para la definición de un manual de calidad.
Tal y como se detalla en [García-Borgoñón *et al.*, 2013b] [Ponce *et al.*, 2013], INROMA dispone de un conjunto de herramientas que facilita su implantación y uso sistemático en entornos reales.

3. **Representación gráfica** *(Graphical representation).*
Este requerimiento hace referencia a la existencia de una notación gráfica para representar los procesos de software. Está muy relacionado con el requerimiento de comprensibilidad, puesto que se ha demostrado que el hecho de disponer de una representación gráfica facilita la descripción de cualquier elemento, así como el intercambio de información por parte de los usuarios del mismo.
En el caso que nos ocupa, de nuevo, al utilizar UML como soporte, este requerimiento está contemplado en INROMA.

4. **Capacidad de abstracción** *(Abstraction).*
Este requerimiento hacía referencia a la capacidad de un lenguaje de aislar ciertos aspectos para facilitar su comprensión. El hecho de utilizar estándares como UML y, sobre todo, MOF, con una orientación MDE en la definición de nuestro metamodelo, nos permite establecer diferentes niveles de abstracción, los denominados CIM (modelo independiente de la computación, en inglés, Computation Independent Model), PIM (modelo independiente de la plataforma, en inglés, Platform Independent Model) y PSM (modelo específico de la plataforma, en inglés, Platform Specific Model), en los que se establecen, con distintos niveles de abstracción, los diferentes aspectos a trabajar en cada momento, hace que este requerimiento haya sido cubierto.

5. **Capacidad de análisis** *(Analyzability).*
Cumpliendo este requerimiento, un lenguaje de modelado de procesos de software dispondría de los elementos necesarios para comparar diferentes modelos de procesos, tomar decisiones sobre los mismos con el objetivo de establecer una mejora continua en la implantación de todo proceso.
En INROMA, este requerimiento se cumple al incorporar las metaclases *Indicator* y *Metric*, cuyo objetivo es precisamente éste, el conocer mediciones dentro de los procesos y de su desarrollo en entornos reales que nos faciliten las actividades de mejora continua de los mismos.

6. **Flexibilidad** *(Flexibility).*
Este requerimiento es algo complejo, puesto que se basa en la posibilidad de usar un lenguaje de modelado de procesos de software en entornos no software. Hemos visto en capítulos anteriores que un proceso de software puede ser planteado desde dos perspectivas diferentes: la perspectiva de ingeniería de software y la de proceso de negocio. En un planteamiento de ingeniería del software, además de los elementos básicos que todo proceso de software debería contener, se incluyen aspectos muy concretos de este dominio de aplicación como podrían ser *Fase*, *Iteración*, *Ciclo de Vida*, etc. Por contra, si la opción planteada es una perspectiva de negocio, los conceptos a añadir serían del estilo de *Regla de Negocio*, *Unidad Organizacional*, *Transacción*, etc. [Bendraou y Gervais, 2007]
INROMA, al trabajar únicamente con los conceptos claves comunes de todo lenguaje de modelado de procesos de software, cumple este requerimiento puesto que es independiente

a la utilización de un enfoque de ingeniera del software o de negocio, pudiendo ser utilizado en ambos dominios.

7. **Escalabilidad** *(Scalability).*
   Este requerimiento hace referencia a la capacidad del lenguaje de modelar procesos de diferentes tamaños, tanto grandes como pequeños, algo que se cumple en INROMA, puesto que es independiente su uso en un proceso que únicamente contenga una actividad con una persona responsable de su ejecución y un único producto, o en un proceso de mayor complejidad en cuanto a actividades, participantes y productos obtenidos se refiere.

En la tabla 4.1 se observa de forma resumida los requisitos que cumple INROMA y cuáles no están cubiertos por el lenguaje aunque, como veremos más adelante, si que pueden llegar a estarlo gracias a la extensibilidad de INROMA derivada de estar basado en UML.

Requerimiento	**INROMA**
Expressiveness	✓
Executability	
Enactabiliy	
Simulability	
Testability	
Modularity	
Formality	✓
Tooling Support	✓
Standardization	✓
Graphical representation	✓
Support Multiple Views	
Understandability	✓
Abstraction	✓
Analyzability	✓
Reflection	
Flexibility	✓
Scalability	✓

Tabla 4.1: Requerimientos cubiertos por INROMA

Una vez evaluada la capacidad de INROMA como lenguaje de modelado de procesos de software desde el punto de vista más teórico, valorando el cumplimiento de los diferentes requerimientos establecidos por diferentes autores sobre los mismos, pasaremos a una perspectiva más práctica, observando su uso en entornos reales.

Para ello, tal y como se ha introducido anteriormente, y dada la influencia e importancia que la metodología NDT tiene en el desarrollo de este trabajo de tesis, haremos uso de un proceso perteneciente a dicha metodología. Como ya hemos avanzado en el capítulo 3, NDT surge en el año 2004 centrada en potenciar el aspecto navegacional en sistemas web, incorporando un proceso y un conjunto completo de artefactos para abordar gran parte del ciclo de vida de desarrollo de un sistema. En sus inicios, NDT se focalizaba en las fases de Requisitos y Análisis, haciendo especial énfasis en la definición de los requisitos, su trazabilidad con el análisis y la ejecución de transformaciones para dar soporte en las etapas de desarrollo. Con el paso del

tiempo y debido fundamentalmente a las experiencias obtenidas de su uso en organizaciones de software [García-Borgoñón *et al.*, 2013b] [Ponce *et al.*, 2013], NDT ha ampliado su alcance, incorporando un número considerablemente mayor de procesos, agrupados en seis categorías diferentes, que son:

1. Procesos de desarrollo de software, basados en el ciclo de vida del software que establece la propia metodología NDT.
2. Procesos de mantenimiento de software, basados en estándares como ITIL [ITIL, 2014] y CMMI [Chrissis *et al.*, 2011].
3. Procesos de pruebas de software, basados en la norma de reciente publicación ISO/IEC 29119 [ISO/IEC, 2014].
4. Procesos de calidad de software, basados en la norma ISO 9001:2008 [ISO/IEC, 2008] y las áreas de proceso definidas en el modelo CMMI.
5. Procesos de gestión de proyectos software, basados en la guía de dirección de proyectos PMBOK [PMI, 2008], y en el modelo CMMI, en sus diferentes constelaciones.
6. Procesos de seguridad, basados en la norma ISO de seguridad denominada ISO/IEC 27000:2009 [ISO/IEC, 2009].

Todos estos procesos se encuentran definidos de una forma más formal y completa en la herramienta NDTQ-Framework [Ponce *et al.*, 2013], un marco de procesos desarrollado mediante un perfil UML, de aplicación en entornos reales, sustentado en un metamodelo para la definición de todos sus elementos [García-Borgoñón *et al.*, 2013b]. NDT dispone, a su vez, de una suite de herramientas, denominadas NDT-Suite, que soportan el carácter MDE que dicha metodología promulga. Las herramientas incluidas en dicha suite se encuentran detalladas en [García-García *et al.*, 2012], [García-García *et al.*, 2013] y [García-García *et al.*, 2014].

Una vez introducido el ámbito de NDT y NDTQ-Framework, para ilustrar el uso de nuestro metamodelo en su capacidad de lenguaje de modelado de procesos de software, se ha elegido el proceso de Ingeniería de Requisitos.

Tal y como se concibe en NDT, el proceso de Ingeniería de Requisitos tiene como objetivo la definición y documentación de los requisitos en el desarrollo de un sistema. Para alcanzar dicho objetivo, este proceso se divide en un conjunto de actividades que son:

1. Obtener información del entorno y definir objetivos.
2. Aprobar el alcance.
3. Identificar y definir los requisitos de almacenamiento.
4. Identificar y definir actores.
5. Identificar y definir los requisitos funcionales.

6. Identificar y definir los requisitos de interacción.
7. Identificar y definir los requisitos no funcionales.
8. Generar el documento de requisitos.
9. Validar el documento de requisitos por los usuarios.
10. Aprobar el documento de requisitos.

En la ejecución de este proceso hay que tener en cuenta tres roles diferentes: el jefe de proyecto, que está encargado de su ejecución y es el responsable último de la elaboración del documento de requisitos, el responsable del área de usuarios, que nos va a validar si el documento recoge todos los requisitos tal y como ellos los entienden, y, por último, un comité de control que aprueba el documento de requisitos desde la perspectiva de la organización que desarrolla el software.

A continuación vamos a ver cómo este proceso se define utilizando INROMA como lenguaje de modelado de procesos de software. El proceso se divide en un conjunto de actividades que contienen un identificador único. Todas estas actividades son del tipo *Activity*. Definimos también tres *Stakeholders* diferentes: Project Manager, Monitoring Committee y User.

Por otro lado, varios productos son utilizados o generados en dicho proceso, algunos de entrada y otros de salida como resultado de sus actividades, para dejar evidencia de los mismos. En algunos casos, estos productos son entregables, con lo que serían del tipo *Deliverable* en INROMA. También existen indicadores, como el número de requisitos definidos, el número de cambios al alcance, etc., que forman parte de la definición del modelo de procesos de acuerdo a nuestro metamodelo.

En la figura 4.2 se observan los elementos que hemos ido comentando. Ya se ha dicho en varias ocasiones que INROMA utiliza como sintaxis concreta la misma de UML, por tanto, el modelo a primera vista es muy similar al que se obtendría utilizando los diagramas de actividades de UML, pero el metamodelo utilizado es el de INROMA.

Finalmente, queremos destacar la consideración que INROMA tiene hacia NDT, en esta ocasión en lo que se refiere a las restricciones del metamodelo. A la hora de plantear dichas restricciones, y partiendo del hecho de que no queríamos perder la flexibilidad que INROMA tiene por definición, nos inspiramos en la herramienta NDT-Quality [García-García *et al.*, 2014] para establecerlas. NDT-Quality es una herramienta de la suite de NDT cuyo objetivos es garantizar la calidad del uso de la metodología NDT en cada una de las fases del ciclo de vida del software, con lo que incorpora un amplio número de restricciones en las metaclases de los diferentes metamodelos con los que trabaja. El uso de dicha herramienta está ampliamente extendido en entornos empresariales, con lo que la experiencia empírica de su validez es muy elevada. Esta evidencia ha apoyado la decisión de inspirarnos en NDT-Quality a la hora de elegir y establecer las restricciones que se han incorporado en INROMA.

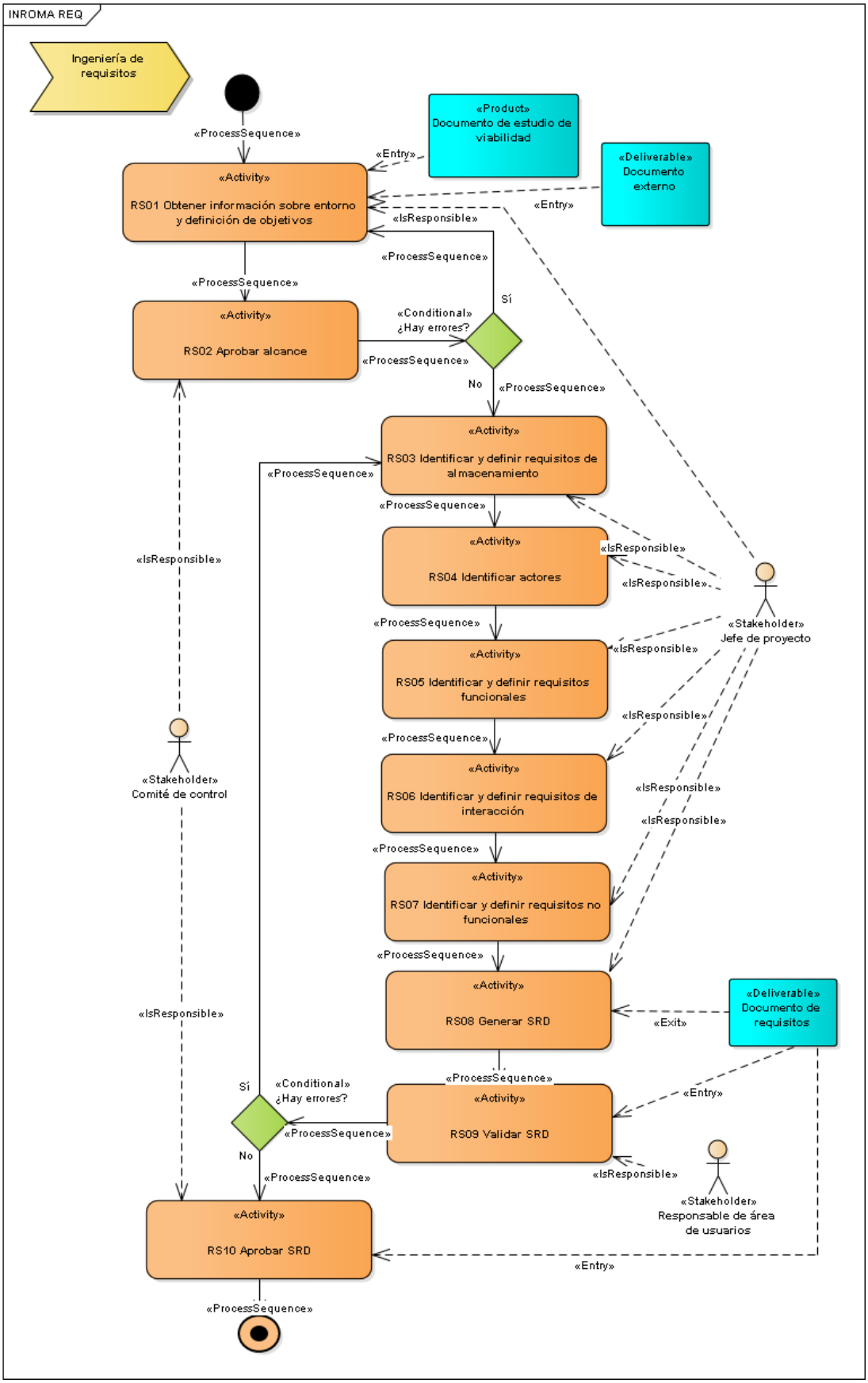


Figura 4.2: Modelo del proceso de Ingeniería de Requisitos de NDT utilizando INROMA

4.3.2. Evaluación de INROMA como pieza clave en el marco de referencia

Una vez evidenciada la capacidad de INROMA como lenguaje de modelado de procesos de software, vamos a proceder a la evaluación de la capacidad como pieza clave en el marco de referencia para facilitar la interoperabilidad y mantenibilidad de modelos de procesos de software que estamos desarrollando.

En el caso de INROMA no estamos planteando establecer un estándar para el modelado de procesos de software. De hecho, como ya hemos comentado en varias ocasiones, han sido numerosos los intentos de estandarización en este ámbito, y seguimos sin poder hablar realmente de un lenguaje estándar de seguimiento general. Es más, la opción más utilizada en la práctica no es tanto el uso de lenguajes definidos de forma específica para modelar procesos de software, sino lenguajes más generales, como pueden ser BPMN o los diagramas de actividad de UML, con los que las personas de las empresas se encuentran más cómodos y familiarizados, sin que éstos puedan ser obviamente considerados estándares para el modelado de procesos de software. Frente a esta tendencia estandarizante, la definición de INROMA pretende favorecer la diversidad de lenguajes y la libertad de elección por parte de las organizaciones.

El planteamiento de INROMA para esta capacidad viene determinado por el hecho de que se plantea como un lenguaje base o, dicho de otra forma, como un segundo lenguaje o lenguaje adicional. Las organizaciones de software que ya utilizan un lenguaje de modelado de procesos de software no deberían tener que cambiar su elección, ni tampoco ser obligadas a aprender un nuevo lenguaje muy amplio y complicado por el mero hecho de que quiera interoperar con otras empresas. Por ello, INROMA se define como un lenguaje que contiene el mínimo común de los conceptos para el modelado de procesos de software que pueden aparecer en cualquier lenguaje definido para este fin.

Es más, una de las primeras características que se plantearon, además de la sencillez, fue el hecho de que ningún lenguaje pudiera tener privilegios frente a los demás. Si el enfoque de INROMA estuviera más ligado a BPMN, podríamos haber incorporado elementos como los Eventos, pero en ese momento la definición de un nuevo lenguaje perdería, desde nuestro punto de vista, su sentido, puesto que para ello habríamos optado por BPMN o un subconjunto del mismo. Para refrendar nuestro planteamiento, sirva como dato anecdótico el hecho de que el nivel de errores en el modelado con BPMN es todavía muy elevado [Recker, 2012] o estudios empíricos en los que se demuestra que de los 100 elementos de los que dispone BPMN, únicamente cuatro son comunes en más del 50 % de los diagramas [Istoan, 2014] [Recker, 2010]. Esta justificación es válida también para cualquier otro lenguaje, no es algo específico ni concreto de BPMN.

Una de las opciones que se nos planteó fue la elección de un subconjunto del lenguaje BPMN o de los diagramas de actividad de UML, algo perfectamente válido, de no ser porque, de esta forma, estaríamos privilegiando un lenguaje frente al resto. Para considerar el lenguaje base del marco de referencia lo más objetivo posible frente a los diferentes lenguajes de modelado de procesos de software que habían ido apareciendo en la última década, optamos por la definición de un nuevo lenguaje que incorporara los conceptos detallados en la norma ISO/IEC TR 24744:2007 que, como hemos visto en capítulos anteriores, es un estándar internacional que establece cuáles son los elementos fundamentales que un proceso debería incorporar, y un proceso de software

no deja de ser un tipo específico de procesos. Así, la ISO/IEC TR 24744:2007 se convierte en principal soporte teórico de la definición del lenguaje INROMA, justificando la influencia que dicho estándar ha tenido en este trabajo de tesis.

En la tabla 4.2 se puede observar cómo los elementos de la ISO/IEC TR 24744:2007 se encuentran incluidos en INROMA.

Elemento ISO	**Descripción del elemento**	**Elemento en INROMA**
Título	Resume el alcance del proceso, identificando su interés principal y distinguiéndolo de otros procesos dentro del contexto de modelos de procesos	La metaclase *Process* y su atributo *name*
Propósito	Describe el objetivo alcanzado si el proceso se lleva a cabo	La metaclase *Process* y su atributo *shortDescription*
Resultados	Son los resultados observables y evaluables de la consecución exitosa del propósito del proceso	La metaclase *Deliverable*, y también las metaclases *Metric* e *Indicator*
Actividades	Definen un conjunto de acciones que se llevarían a cabo durante la ejecución del proceso	Las metaclases *ProcessElement* y *ProcessSequence*
Tareas	Se refiere a las acciones especificas que se ejecutan para realizar una actividad y obtener sus resultados	La metaclase *Activity* y la metaclase *Stakeholder*
Elementos de información	Determinan información utilizada por las personas que se producen y almacenan durante el ciclo de vida del software y del sistema	La metaclase *Product*

Tabla 4.2: La ISO/IEC TR 24744:2007 contemplada en INROMA

A la vista de estas correspondencias entre los elementos de INROMA y los que se encuentran en la ISO/IEC TR 24744:2007 se observa que podemos cumplir el estándar con INROMA y que además contiene los elementos que todo proceso de software debe disponer, por lo que queda de manifiesto su capacidad como lenguaje de modelado de procesos de software para formar parte del marco de referencia, marco que iremos completando conforme vayamos desarrollando en los siguientes capítulos los otros dos elementos que lo sustentan.

4.4. La extensibilidad de INROMA

Hemos comentado en varias ocasiones que una de las características principales de INROMA es la de incluir el mínimo conjunto común de conceptos que son necesarios para modelar un proceso de software, ni más ni, por supuesto, menos y, por tanto, estarán incluidos en cualquier lenguaje de modelado de procesos que funcione como tal. Esto es así para garantizar que cualquier

lenguaje de modelado de procesos de software puede ser incorporado al marco de referencia, sin potenciar o privilegiar el uso de unos lenguajes frente a otros. Sin embargo, el hecho de que esté basado en UML hace que, de forma inherente, INROMA tenga una propiedad añadida: la extensibilidad o, dicho de otra forma, la capacidad de ser fácilmente extensible para incorporar nuevos conceptos o ideas requeridas en otros entornos.

En el ámbito de un marco de referencia para facilitar la interoperabilidad y la mantenibilidad de procesos de software como el que estamos proponiendo, esta característica podría no ser de gran valor, e incluso plantear un contrapunto, puesto que no se debería permitir incorporar nuevos elementos en INROMA que, aunque aportaran más información, supusieran un problema para la interoperabilidad entre procesos de software modelados con diferentes lenguajes, puesto que nos movemos en el ámbito general de todos los lenguajes. No obstante, si en lugar de encontrarnos en el ámbito del marco de referencia genérico del que estamos hablando en todo momento, quisiéramos utilizar INROMA en otro entorno, más acotado y específico, donde el número de lenguajes a incorporar no fuera ilimitado sino muy concreto, y estos lenguajes coincidieran en unos elementos que hasta este momento no se encuentran recogidos en INROMA, podríamos hacer uso de esta capacidad y extender el lenguaje para incorporarlos. Es decir, la extensibilidad es una propiedad inherente de INROMA por estar basada en UML que, lejos de ser un problema, puede facilitar su uso en otros entornos más allá de nuestro marco de referencia genérico.

Con esta característica, INROMA, que como lenguaje no cumple todos los requerimientos de los lenguajes de modelado de procesos de software recogidos en el capítulo 3, podrían ser abordados mediante las extensiones correspondientes al lenguaje.

Como un ejemplo a todo esto a continuación mostramos cómo extender INROMA para cumplir los requerimientos *Simulable* y *Comprobable*. Dichas extensiones han sido abordados en sendos trabajos dentro del grupo IWT2 [García *et al.*, 2014], [García-Borgoñón *et al.*, 2014c]. El caso de que un lenguaje de modelado de procesos de software sea *Simulable* hace referencia a la capacidad de dicho lenguaje de incluir conceptos que nos permitan simular el modelo de proceso previo a su promulgación y ejecución en entornos reales. Esto nos ofrece una gran ayuda para el diseño de procesos, especialmente aquellos que contemplan multitud de opciones, como una herramienta básica de toma de decisiones.

En la figura 4.3 podemos observar cómo se ha extendido INROMA para incorporar elementos de simulación. Esta extensión se describe de forma detallada en [García *et al.*, 2014], y aquí haremos una introducción breve de la misma.

Entrando en detalle en los elementos representados en la figura 4.3, un Modelo consiste en un conjunto de parámetros (*Parameter*), variables (*Variable*), eventos (*Event*), entidades (*Entity*), transiciones (*Transition*) y, el elemento que constituye el nexo con INROMA puesto que se comparte, actividades (*Activity*).

Las entidades están relacionadas con los eventos, que pueden provocar un cambio de estado en las mismas según una condición de activación, y con las actividades, que actuarán sobre ellas realizando una determinada acción (crear, añadir a la cola, destruir,..) expresada mediante el atributo de enlace *Action*. El evento puede producirse al darse una determinada condición y un

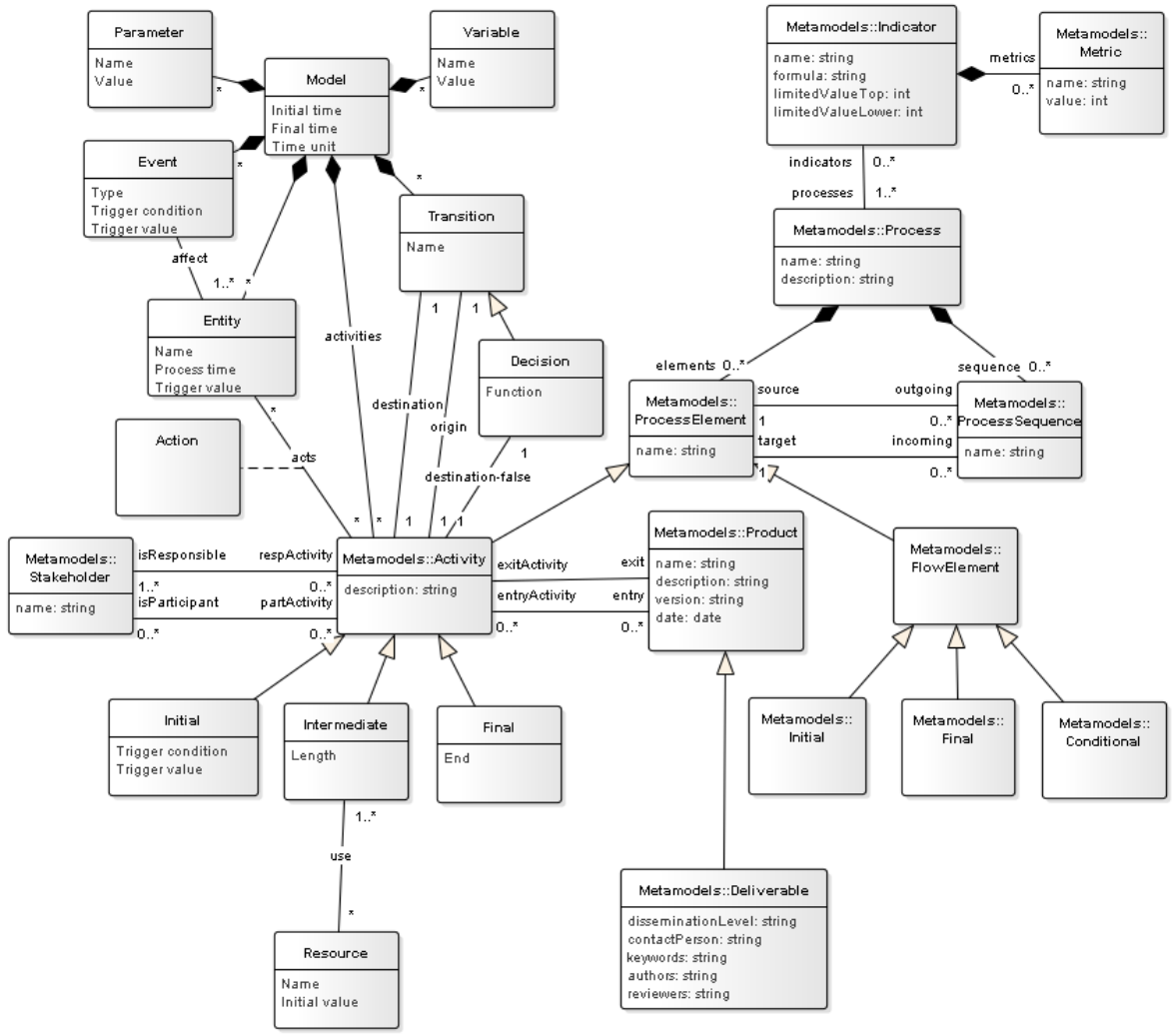


Figura 4.3: Extensión de INROMA para la simulación de eventos discretos

valor de activación, representados en sus atributos *TriggerCondition* y *TriggerValue*.

Por otro lado, las actividades que actúan sobre las entidades pueden ser de tres tipos: inicial (*Initial*), final (*Final*) e intermedia (*Intermediate*). Una actividad inicial es la que da comienzo al proceso de simulación mediante una función de activación y un valor, la final contiene un atributo fin que recoge la información para finalizar la simulación y las actividades intermedias son las que permiten que el proceso avance y guardan su duración en dicho atributo.

La metaclase *Transition* permite el paso de una actividad a otra, y la metaclase *Decision* representa la alternativa entre dos actividades, que depende de la evaluación de una condición. Toda transición tiene una actividad de origen y otra de destino.

En cuanto a incorporar a INROMA la propiedad de ser *Comprobable*, podemos ver una extensión de nuestro metamodelo en la figura 4.4, extensión que se explica en profundidad en [García-Borgoñón *et al.*, 2014c].

El objetivo de esta extensión era la de dotar a INROMA de los elementos necesario para hacerlo comprobable, es decir, para poder evaluar si un proceso se ejecuta en la realidad tal y como se ha definido. Para ello se incorporan nuevos elementos que permiten representar determinados conceptos. La metaclase *Action* se constituye en el elemento fundamental del metamodelo para su extensión comprobable. Definimos una acción para cada actividad (*Activity*) que prentendemos comprobar. Es posible que no comprobemos todas las actividades de un proceso y, para garantizarlo, establecemos la separación entre acción y actividad. Una acción tiene una fecha de inicio, una fecha de fin, el estado en un momento concreto y el resultado del test o prueba, que sirve a modo de registro.

Una acción tiene un conjunto de precondiciones y postcondiciones. La metaclase *Precondition* representa las acciones previas que necesitamos chequear para ejecutarla. La metaclase *Postcondition* muestra la siguiente acción y si ha sido ejecutada de la forma apropiada.

La metaclase *WorkProduct* representa aquel elemento de *Product* que es desarrollado en una acción, por tanto, un producto puede ser considerado como un conjunto de varios *WorkProducts*. Finalmente, la metaclase *Stakeholder* representa a alguien (herramienta o persona) que está ejecutando una determinada acción.

En definitiva, y como resultado de la posibilidad de extender INROMA, la gran mayoría de los requerimientos que como lenguaje de modelado de procesos de software INROMA actualmente no contempla, podrían alcanzarse. Entre ellos destacamos el requerimiento de *Ejecutable*, objeto de una tesis desarrollada en el grupo IWT2, que comentaremos con más detalle en el capítulo 9.

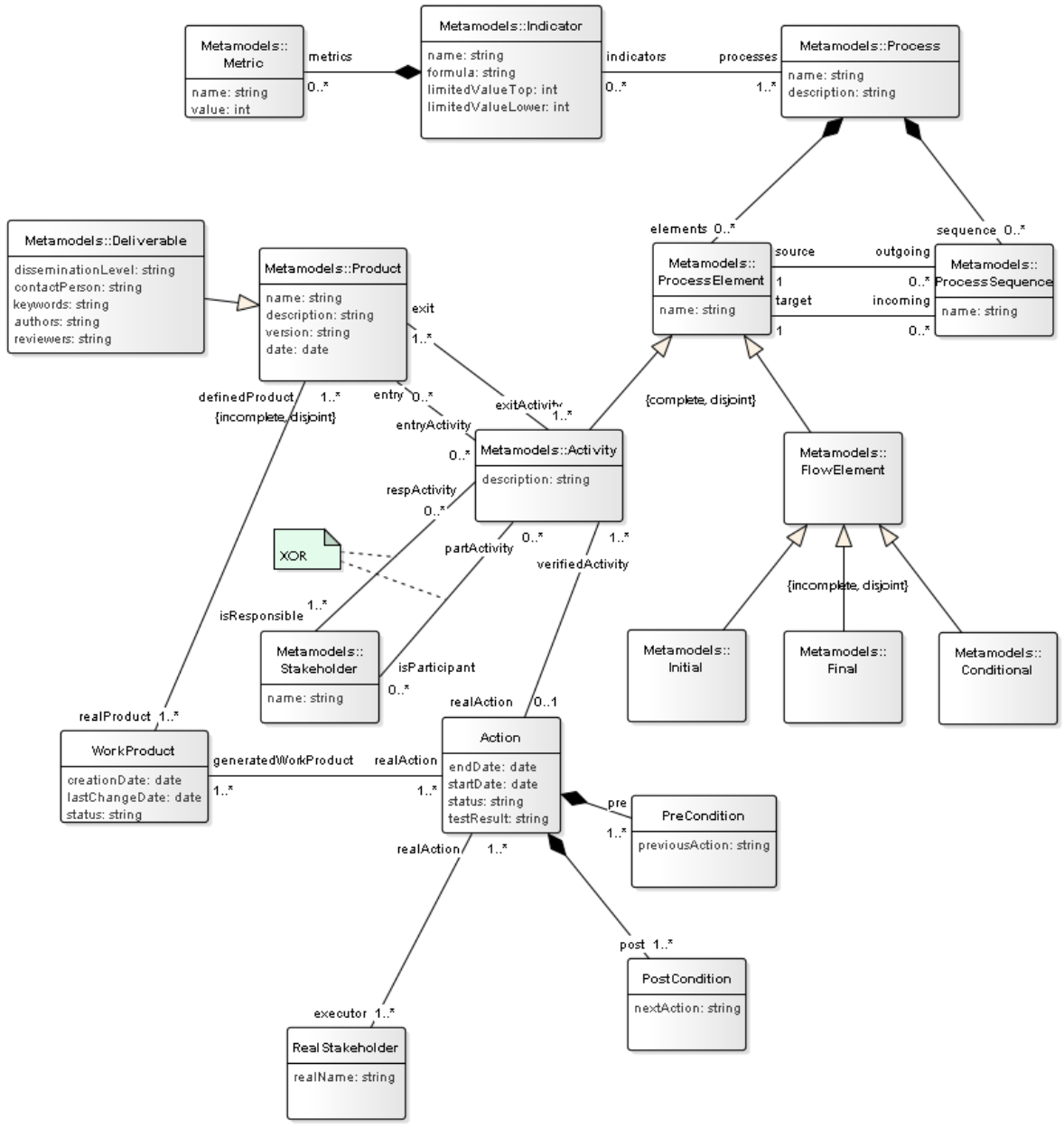


Figura 4.4: Extensión de INROMA para incorporar la propiedad de comprobable

En la figura 4.5 se puede observar a modo de resumen el cumplimiento actual y potencial de los requerimientos de lenguajes de modelado de procesos de software por parte de INROMA. Se trata de un gráfico radial que presenta tres valores discretos: 0, para representar que INROMA no soporta el requerimiento, 10 en el caso en el que sí que lo soporte y 5 para aquellos que potencialmente podrían soportarse en un futuro.

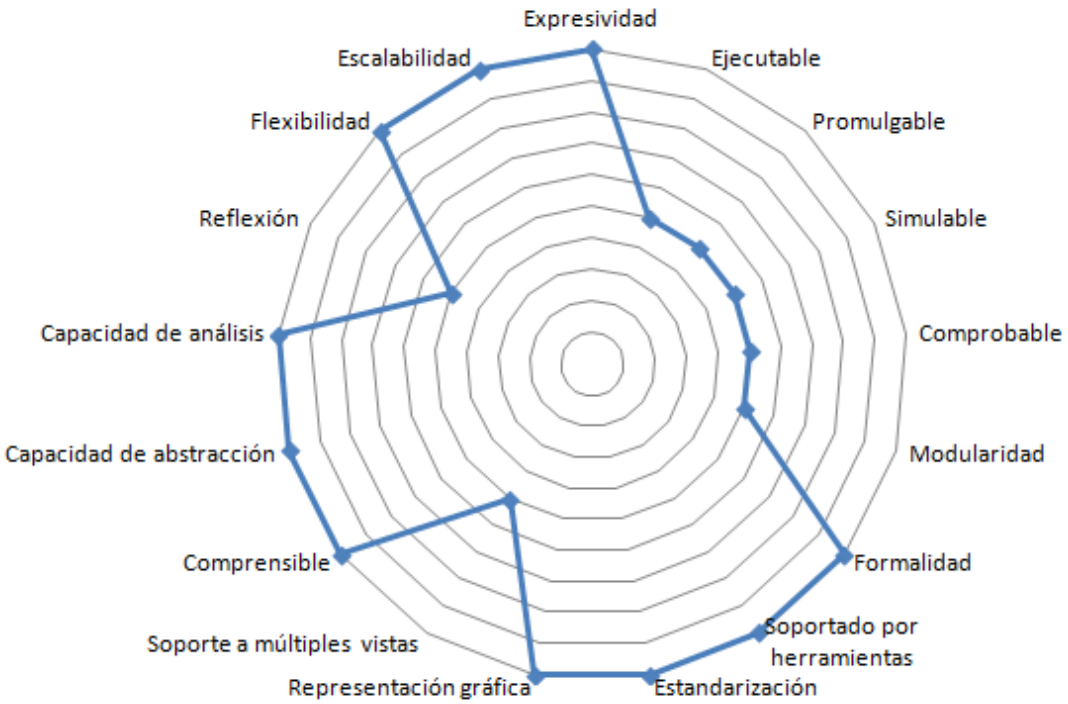


Figura 4.5: Cumplimiento actual y potencial de los requerimientos de lenguajes de modelado de procesos de software por parte de INROMA

4.5. Conclusiones

En este capítulo se ha presentado uno de los tres pilares de nuestra propuesta, el lenguaje de modelado de procesos de software INROMA. A lo largo de este capítulo se ha presentado el metamodelo que constituye la sintaxis abstracta del lenguaje, describiendo con detalle cada uno de los elementos y sus relaciones. Además, se han especificado y evaluados los objetivos de diseño y las capacidades aportadas por INROMA, que son dos: es un lenguaje de modelado de procesos de software en sí mismo y, a su vez, se trata de un lenguaje sencillo que se va a utilizar en el marco de referencia para la interoperabilidad y mantenibilidad de los modelos de procesos de software. Por último, se ha profundizado en una propiedad inherente en INROMA al estar basado en UML, la extensibilidad, exponiendo cuáles son las ventajas que dicha característica nos aporta e ilustrando mediante ejemplos dicha propiedad.

Capítulo 5

El método para incorporar lenguajes de modelado de procesos de software en el marco de referencia

En el capítulo anterior se ha presentado el lenguaje INROMA como la primera de las tres piezas sobre las que se sustenta el marco de referencia para facilitar la interoperabilidad de modelos de procesos de software. En este capítulo vamos a profundizar en la segunda de las piezas, el método mediante el cual un lenguaje de modelado de procesos de software puede ser incorporado en el marco de referencia, tanto desde un punto de vista teórico, con la exposición del método propiamente dicho, como desde el punto de vista más práctico, utilizando como ejemplos lenguajes de modelado de procesos de software específicos. Para ello, la estructura del capítulo es la siguiente. En la primera sección plantearemos el contexto de la interoperabilidad en el ámbito de MDE y cómo se ha reflejado en la arquitectura de nuestra propuesta de solución. A continuación se definirá el método de incorporación de un lenguaje de modelado de procesos de software al marco de referencia y, seguidamente, se expondrá a modo de ejemplo el uso de dicho método con algunos lenguajes de modelado de procesos de software utilizados en la industria de los que disponemos de ejemplos reales de uso. Para finalizar, se resumirán las conclusiones más relevantes del capítulo.

5.1. Una arquitectura para facilitar la interoperabilidad

En capítulos anteriores se ha introducido que IEEE define *Interoperabilidad* como *la capacidad de dos o más sistemas o componentes de intercambiar información y de usar dicha información intercambiada* [Thomas y Nejmeh, 1992]. La interoperabilidad es un problema ampliamente estudiado en la ingeniería del software que requiere ser trabajado desde el punto de vista sintáctico y semántico, estableciendo los vínculos o puentes oportunos entre los diferentes sistemas. Estos vínculos, habitualmente, se generan de forma ad-hoc para cada dupla de sistemas. Esta solución manual constituye una fuente importante de errores, consume mucho tiempo

y es difícilmente reutilizable, incluso cuando tratamos con herramientas similares [Bruneliere *et al.*, 2010].

Con la aparición de MDE se ha contemplado la posibilidad de plantear la interoperabilidad desde un nivel de abstracción superior, aplicando técnicas de ingeniería dirigida por modelos. Este enfoque, que se ha denominado Interoperabilidad Dirigida por Modelos (Model-Driven Interoperability, MDI) [Brambilla *et al.*, 2012] establece los vínculos entre los conceptos relacionados definidos en los metamodelos de los sistemas que se plantean interoperar, y define transformaciones modelo a modelo (Model-to-Model) y modelo a texto (Model-to-Text) como una forma de abordar la interoperabilidad.

En [Pastor *et al.*, 2013] se realiza una revisión de los principales trabajos realizados en MDI y, de acuerdo a las evidencias en ellos presentadas, estos trabajos se categorizan y clasifican de acuerdo a las siguientes características:

- **Entramado de modelo** *Model Weaving (MW).* Considera la definición de enlaces entre los metamodelos de los modelos involucrados, que son definidos habitualmente mediante un modelo *entramado.*
- **Meta-extensiones** *Meta-Extensions (ME).* Corresponde a la definición de nueva información (extensiones) en los enfoques de modelado involucrados para proporcionar características de modelado adicionales que son necesarias para ejecutar las operaciones de interoperabilidad.
- **Verificación de interoperabilidad** *Interoperability Verification (IV).* Los modelos y las operaciones de interoperabilidad pueden tener incidencias que provocan un intercambio incompleto de información. Además, es importante analizar los mecanismos de verificación para asegurar la correcta especificación de los artefactos que están involucrados en los escenarios de interoperabilidad.
- **Artefacto de pivote** *Pivot Artifact (PA).* Esta característica está relacionada con el uso de un artefacto intermediario (metamodelo u ontología) para gestionar las diferencias estructurales e identificar equivalencias conceptuales.
- **Dominio de aplicación** *Application Domain (AD).* Indica el dominio con el que está relacionado el enfoque analizado, de forma que las contribuciones analizadas puedan ser generalizadas a otros dominios.

En nuestra propuesta de desarrollar un marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los procesos de software, es obvio pensar que la interoperabilidad es uno de los principales focos de atención. Teniendo en cuenta que estamos enmarcando la solución en el ámbito de MDE, se ha optado por utilizar un enfoque similar al empleado en alguno de los trabajos de MDI [Bruneliere *et al.*, 2010], [Berger *et al.*, 2010], [Vallecillo, 2010], [Moreno y Vallecillo, 2008], caracterizados por la definición de un metamodelo pivote, que corresponde a la categoría de **Artefacto de Pivote *(PA)*** anteriormente citada y que utiliza un metamodelo como artefacto. En el marco de referencia ese metamodelo pivote es INROMA, tal y como queda reflejado en la figura 5.1, en la que se refleja la arquitectura del mismo. En ella se observa

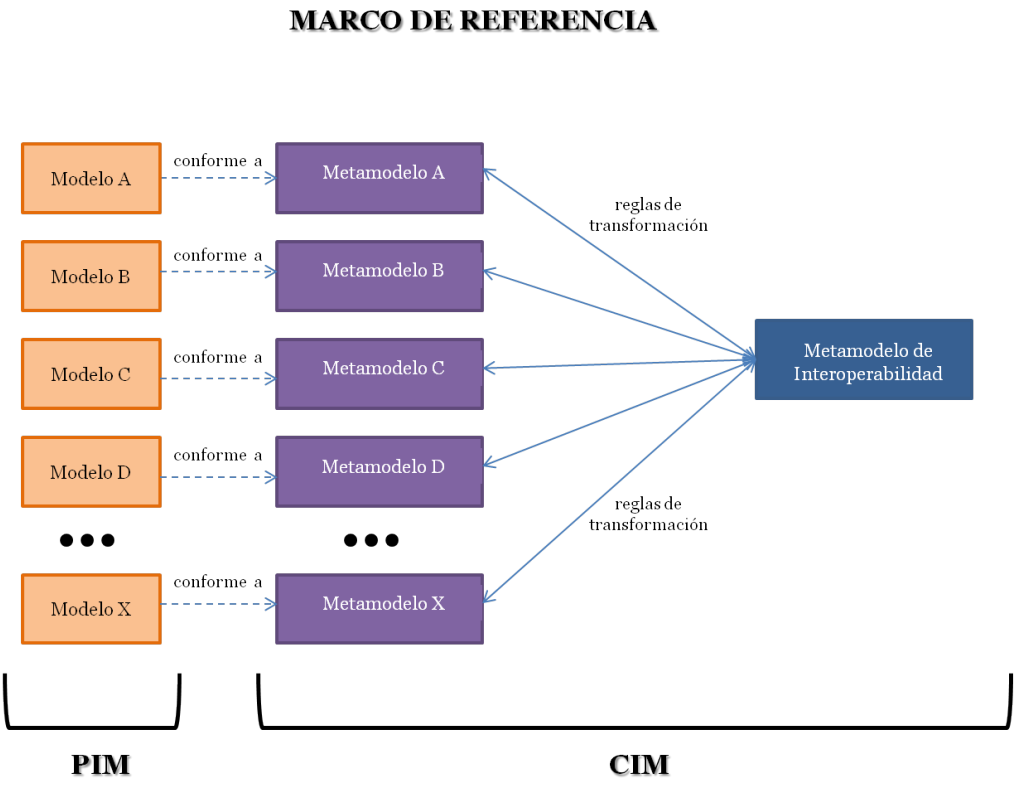


Figura 5.1: Arquitectura del marco de referencia para facilitar la interoperabilidad y mantenibilidad

que se establecen trazas bidireccionales entre los conceptos de los metamodelos de cada uno de los lenguajes de modelado de procesos de software incluidos en el marco de referencia y el metamodelo del lenguaje INROMA, el cual actúa de puente conceptual entre ellos. Estas trazas son planteadas a modo de transformaciones modelo a modelo horizontales, es decir, entre modelos que se encuentran en un mismo nivel de abstracción, en este caso un nivel CIM (Computation Independent Model).

De esta forma, si un modelo A se ha modelado con un lenguaje LA, es decir, se ha definido conforme a un Metamodelo A, y queremos conseguir un modelo B modelado en un lenguaje LB, es decir, que sea conforme a un Metamodelo B, los enlaces se establecen a través del metamodelo de INROMA, y no directamente entre los modelos específicos. De esta forma subimos un nivel de abstracción en la interoperabilidad, y el número de metamodelos entre los que se puede interoperar es tan amplio como se desee sin que la complejidad aumente de forma exponencial sino de forma lineal. La interoperabilidad la estamos estableciendo siempre desde un punto de vista semántico. La sintaxis concreta de cada uno de los modelos participantes sigue siendo la específica del lenguaje de modelado de procesos de software elegido. Lo mismo ocurre con la mantenibilidad, ya sea utilizando dos lenguajes diferentes o el mismo lenguaje evolucionado, siempre se va a realizar mediante INROMA como metamodelo pivote.

Una vez establecida la arquitectura del marco de referencia, en la siguiente sección aborda-

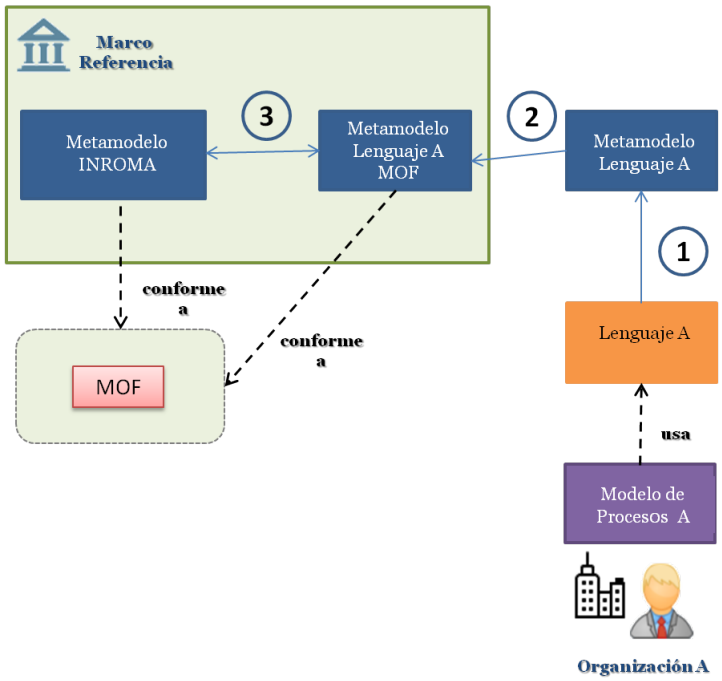


Figura 5.2: Vista del método sobre la arquitectura del marco de referencia

remos en detalle el segundo de los pilares del mismo: el método para incorporar lenguajes al marco.

5.2. El método del marco de referencia

Una vez expuesto cómo la arquitectura del marco de referencia da cabida a la interoperabilidad y la mantenibilidad, el objetivo del método es establecer una sistemática para incorporar nuevos lenguajes de modelado de procesos de software al marco de referencia. En la figura 5.2 se presenta la utilización del método para la incorporación del lenguaje L sobre una vista de la arquitectura del marco de referencia, en la que se observan los tres pasos principales del método representados con un círculo numerado. Estos pasos son los siguientes:

1. **Descubrimiento del metamodelo de L**. Para poder realizar una nueva incorporación de un lenguaje de modelado de procesos de software al marco de referencia es necesario disponer de la sintaxis abstracta de dicho lenguaje. Sin embargo, dicha sintaxis no siempre está disponible de forma explícita. En tales casos es necesario *descubrir* el metamodelo, entendiendo como tal una representación más abstracta, a partir de la especificación del lenguaje, ejemplos de uso, etc. Aunque la sintaxis concreta del lenguaje es muy importante a la hora de modelar un proceso de software, en este paso no resulta, sin embargo, relevante.

2. **Transcripción**. Este paso consiste en expresar la sintaxis abstracta del lenguaje L en

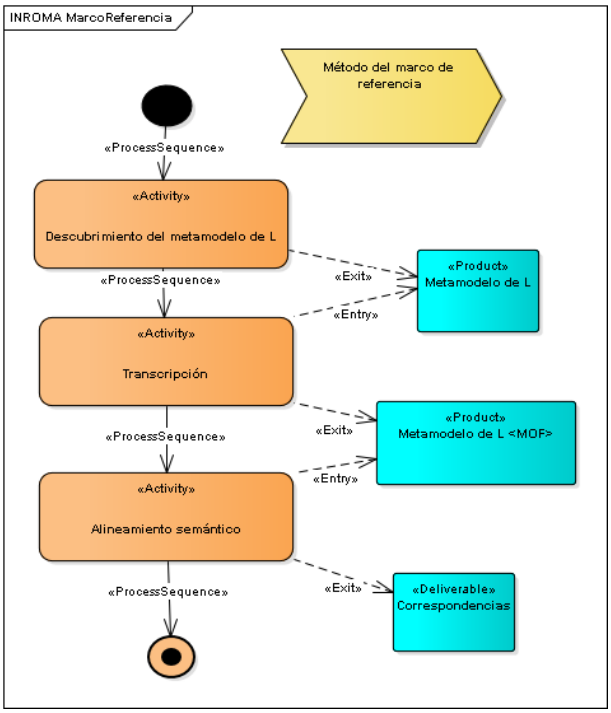


Figura 5.3: El método del marco de referencia modelado con INROMA

términos del metametamodelo MOF, obteniendo el metamodelo de L (*ML*) para poder ser utilizado dentro del entorno del marco de referencia.

3. **Alineamiento semántico**. En este paso se van a realizar las correspondencias conceptuales existentes entre el metamodelo del lenguaje L (*ML*) e INROMA. Como resultado se obtiene la tabla de correspondencias entre L e INROMA que, como veremos en el capítulo posterior, constituye la fuente de información para poder desarrollar las transformaciones.

Como se ha podido observar en la definición del mismo, este método se caracteriza por ser genérico, ya que puede ser aplicado para cualquier lenguaje, independientemente de la forma en la que se encuentra especificado. En la figura 5.3 se observa el método del marco de referencia modelado utilizando INROMA como lenguaje. En él se observan los tres pasos anteriormente explicados como tres actividades del método: *Descubrimiento del metamodelo de L*, *Transcripción* y *Alineamiento Semántico*. En la primera actividad se obtiene como resultado el producto *Metamodelo de L* que es refinado en el siguiente paso del método, para convertirse en el producto de entrada de la última de las actividades. Como producto entregable de este método se obtiene la tabla de correspondencias entre el lenguaje L e INROMA, que será utilizada para desarrollar las transformaciones.

A continuación, en la siguiente sección presentamos, a modo de ejemplo, el uso de este método para la incorporación de varios lenguajes de modelado de procesos de software utilizados en la industria.

5.3. El método en la práctica: Ejemplos de aplicación

Para ilustrar la aplicación del método definido en la sección anterior, a continuación presentamos su uso con tres lenguajes de modelado de procesos diferentes. Los lenguajes elegidos para tal fin son los diagramas de actividad de UML2.0, que se ha convertido en el estándar de facto en ingeniería del software y es la base sobre la que se definen muchos de los lenguajes de modelado de procesos de software, tal y como hemos podido comprobar en el capítulo 2, BPMN, utilizado por multitud de empresas e instituciones de todos los sectores, ofreciendo un enfoque de negocio del software, e IDEF (Integration DEFinition) [IDEF, 1995], otro lenguaje de modelado de procesos de software de uso extendido en el sector aeronáutico.

5.3.1. Los lenguajes elegidos para los ejemplos

Una cuestión significativa en este punto es la correcta elección de los lenguajes de modelado de procesos de software a utilizar en estos ejemplos. Lo es por varios aspectos: por un lado se pretende buscar ejemplos que sean representativos de la realidad en las empresas de software, por lo tanto, los lenguajes aquí elegidos deberían tener presencia real en organizaciones del ámbito del software. Otro criterio a utilizar a la hora de elegir esta muestra de lenguajes es la diferenciación entre ellos, tanto a nivel conceptual, incorporando diferentes aspectos o formas de abordar alguna cuestión, como en su sintaxis concreta. Por último, y teniendo en cuenta que la solución propuesta está incluida en el paradigma MDE, los lenguajes de modelado de procesos de software elegidos deben incorporar un metamodelo para definir su sintaxis abstracta o, en el caso de que no lo tengan, ser posible extraerlo de la propia especificación del lenguaje.

En este sentido, nuestra elección fue la siguiente: el primer lenguaje elegido es UML-DA, los diagramas de actividad de UML, una de las alternativas más frecuentemente utilizadas en las organizaciones de software debido, principalmente, al conocimiento que los ingenieros del software tienen del mismo. El segundo lenguaje elegido es BPMN, el cual está siendo promovido por las más importantes organizaciones multinacionales como estándar de facto para la representación de los procesos de negocio, y los procesos de software pueden ser vistos como procesos de negocio de empresas cuya finalidad es el desarrollo de software, y por ello está siendo elegido por un considerable número de empresas. Y, por último, y también con la aspiración de contar con un lenguaje bien diferenciado a los otros, es decir, que no tenga como base ninguno de los anteriores, elegimos IDEF3, el cual, si bien no es muy utilizado en empresas de software, sí tiene una amplia aceptación en la industria aeronáutica, en la que, aunque no tiene por objetivo la venta directa al mercado de productos software, también se desarrolla como parte de su producción. Los tres son lenguajes muy diferentes entre sí, fundamentalmente porque los objetivos para los que fueron creados también eran radicalmente distintos, porque disponen de sintaxis concretas muy desiguales, porque en su semántica también aparecen formas heterogéneas de afrontar cuestiones similares, y de todos ellos podemos tener empresas software que nos faciliten ejemplos prácticos reales.

Cabe destacar que ninguno de estos lenguajes aparece en el capítulo 2 explícitamente como lenguajes de modelado de procesos de software. Únicamente se hace referencia a UML como la

base sobre la cual algunas de las propuestas más actuales se han definido, pero UML-DA no aparece directamente como tal. Esto es debido a que, si bien los tres lenguajes elegidos pueden ser utilizados como lenguajes de modelado de procesos de software, no fueron definidos o propuestos con este fin ya que su propósito es más general. Sin embargo, nos hemos decantado por ellos frente al resto por la disponibilidad de casos de estudio reales en empresas en los que estos lenguajes se utilizan, que nos permiten obtener la información necesaria para el desarrollo de nuestro marco de referencia, así como la validación del mismo. No obstante, queremos recalcar que cualquier otro lenguaje de modelado de procesos de software de los que hemos visto en el capítulo 2 podría haber sido perfectamente válido para nuestros ejemplos, y ésta habría sido nuestra elección en el caso de que hubiéramos dispuesto de casos reales de estudio.

Una vez argumentada la elección de estos tres lenguajes, comenzaremos con la utilización del método definido en la sección anterior. El uso de estos tres lenguajes como ejemplos se propaga a los siguientes capítulos: en las transformaciones definidas en el capítulo 6 o en la presentación de la herramienta en el capítulo 7 y en los casos de estudio que son el objeto del capítulo 8.

En los tres ejemplos seguiremos un mismo esquema: comenzaremos con una pequeña introducción del lenguaje de modelado de procesos de software elegido, aplicaremos el método descrito en la sección anterior y mostraremos el resultado de dicha aplicación mediante la tabla de correspondencias obtenida.

5.3.2. Unified Modeling Language, UML-Diagramas de Actividad

UML (Unified Modeling Language) [OMG, 2011b] está considerado como el estándar de facto para el diseño y modelado de software. Se trata de un conjunto de formalismos para capturar aspectos detallados de diseño de los sistemas software, muy extendida y ampliamente utilizada por los ingenieros del software. Es este hecho, su uso extendido, lo que lo convierte en una de las primeras opciones para un ingeniero de software a la hora de representar los procesos a través de sus Diagramas de Actividad (Activity Diagrams, AD). Como se ha podido comprobar en el capítulo 2, la opción de utilizar los diagramas de actividad de UML como un lenguaje de modelado de procesos de software ha sido utilizada por varios autores en la última década.

A partir de la versión 2.0 (actualmente existe la versión 2.5), UML va más allá de la representación gráfica, ofreciendo un importante potencial en la representación de una gran variedad de procesos [Bendraou *et al.*, 2010]. Entre los conceptos soportados por UML cabe destacar los conceptos de actividad (*Activity*), usado para representar el comportamiento global de un sistema, acciones (*Actions*), que encapsulan el comportamiento de negocio y son usados para representar actividades dentro de un proceso, y los nodos de control (*Control Nodes*) utilizados para describir los típicos patrones de control de flujo.

Para mostrar su capacidad de representación de procesos, son varios los autores que los han evaluado utilizando los *Workflow Patterns* [Van Der Aalst *et al.*, 2003a], una taxonomía de construcciones genéricas que originalmente se utilizaban para evaluar los sistemas de flujos de trabajo, y que en los últimos años se ha utilizado para evaluar los lenguajes de modelado de procesos de negocio. Diferentes trabajos han mostrado, con evaluaciones respecto a los Workflows,

las fortalezas [Russell *et al.*, 2006] [Wohed *et al.*, 2005] así como las debilidades [Russell *et al.*, 2006] de los diagramas de actividad de UML para el modelado de procesos de software.

El hecho de que OMG adoptara BPMN para la representación de procesos (de negocio en general, y de software en particular), podría hacer pensar que la capacidad de representación de procesos de BPMN respecto a los diagramas de actividad de UML es mucho mayor, algo que no resulta tan evidente a la vista de algunos trabajos empíricos con ambos [Birkmeier y Overhage, 2010]. Son muchos los que, conociendo ya la semántica y notación gráfica de UML, optan por los diagramas de actividad para el modelado de procesos de software.

Una vez introducido el contexto de los diagramas de actividad de UML, procederemos a la realización del método definido en la sección anterior. Definimos UML-DA como el lenguaje, que dispone de una sintaxis abstracta, que es la parte del metamodelo que hace referencia a las actividades y las acciones, y una sintaxis concreta, que es su notación gráfica.

Los dos primeros pasos del método consisten en descubrir el metamodelo de UML-DA y disponer de él en términos de MOF. Ambos pasos son relativamente sencillos en el caso de este lenguaje puesto que en su especificación así se definen. Para facilitar su uso en el siguiente paso, se han reagrupado los conceptos de UML-DA que se van a utilizar en la realización del método, como se puede observar en la figura 5.4 [1].

El tercer paso del método consiste en la alineación semántica, esto es, en investigar las correspondencias entre el metamodelo de UML 2.5, en lo referente a Diagramas de Actividad, e INROMA en busca de las equivalencias conceptuales entre ambos. En la tabla 5.1 aparecen tales correspondencias, lo que viene a ser el resultado de la ejecución del método.

INROMA	UML-DA 2.5
Activity	Action
Process	Activity
ProcessSequence	ControlFlow
Initial	InitialNode
Final	FinalNode
Conditional	DecisionNode
Product	DataStoreNode

Tabla 5.1: Correspondencias entre INROMA y UML-Diagramas de Actividad

[1] El metamodelo aquí reflejado es un subconjunto del metamodelo completo de UML Superstructure. Para consultar el metamodelo completo se sugiere acudir a la especificación [OMG, 2011b]

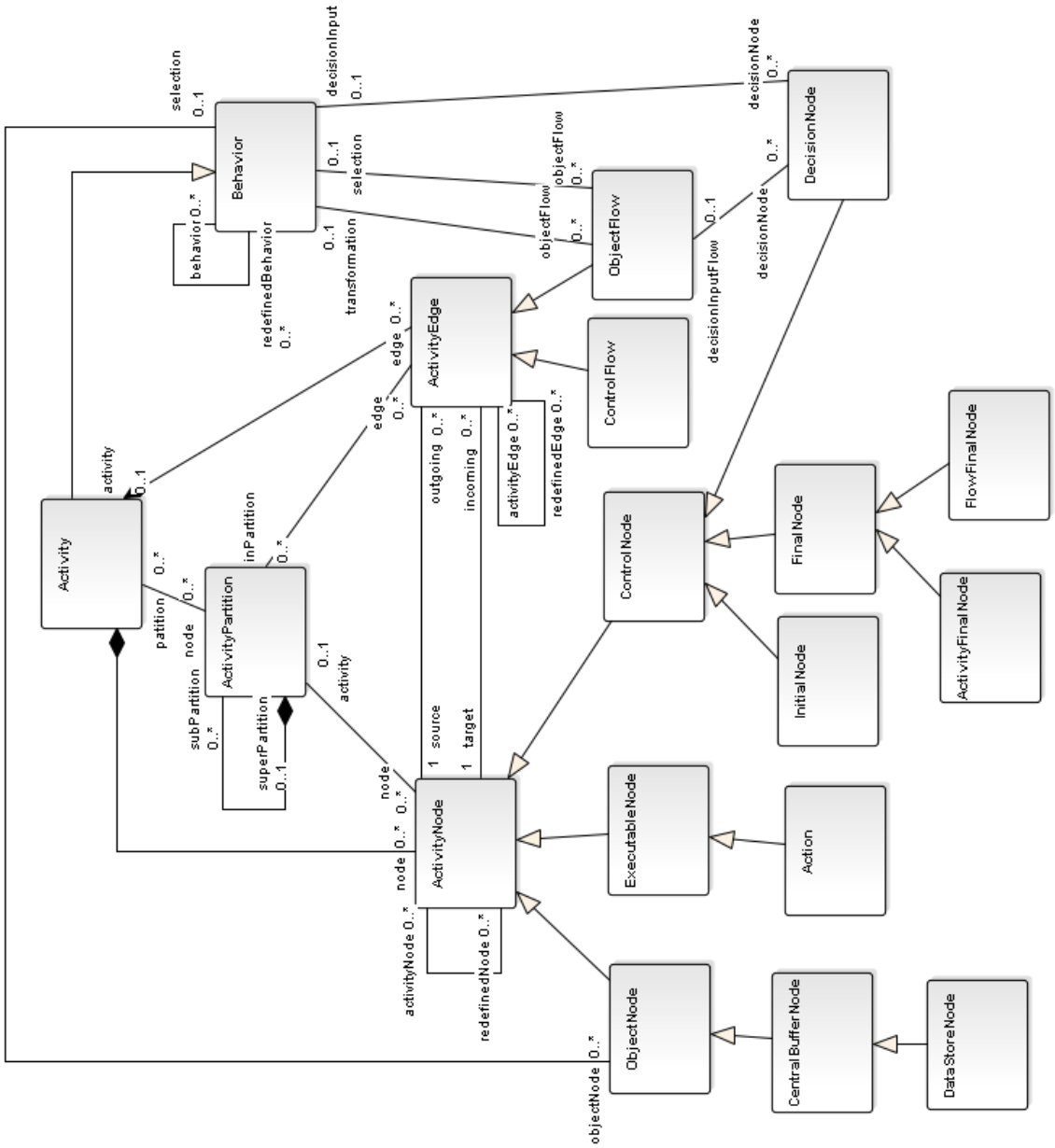


Figura 5.4: Vista parcial del metamodelo de los diagramas de actividad de UML

5.3.3. Business Process Model and Notation, BPMN

Bussiness Process Model and Notation (BPMN) [OMG, 2010] es una notación gráfica para la representación de modelos de procesos de negocio, inicialmente desarrollada por la Business Process Management Initiative [BPMI.org y OMG, 2006], adoptada por el Object Management Group (OMG) en 2006 para su estandarización, y actualmente es mantenida y evolucionada por dicho grupo. Se ha convertido, por tanto, en la apuesta de OMG para la representación de procesos de negocio frente a otras opciones como podrían ser los Diagramas de Actividad de UML [Birkmeier y Overhage, 2010], de forma que se establece el diferente campo de acción de ambos estándares: mientras BPMN se enfoca hacia los procesos de negocio, los diagramas de actividad de UML lo hace hacía el diseño de software, con lo que no son competencia entre sí, sino que ofrecen vistas diferentes de los sistemas [Cibran, 2009].

BPMN fue desarrollado con el objetivo de disponer de una notación que fuera entendible por todos los usuarios de procesos de negocio, desde los analistas que crean los primeros borradores de los procesos hasta los desarrolladores responsables de la implementación de la tecnología sobre la cual esos procesos son ejecutados, y, finalmente, las personas de negocios que gestionan y monitorizan dichos procesos. Es decir, con el desarrollo de BPMN se buscaba disminuir la brecha existente entre los departamentos de tecnologías de la información (Information Technologies, IT) y de negocio [Weske, 2007].

Actualmente se encuentra en su versión 2.0, y su aplicación en la industria y en la academia está ampliamente probada [Recker, 2010], llegando a convertirse en un estándar de facto para la representación de los procesos de negocio. Entre otras, una de las aportaciones de esta versión respecto a la anterior es la definición de su metamodelo de manera formal, lo que facilita enormemente su adopción en un contexto MDE y, por tanto, su utilización en este trabajo de tesis. Son estos factores, su amplio uso por parte de la industria del software y la existencia de un metamodelo formal, los que han influido en su elección como uno de los tres lenguajes de modelado de procesos de software elegidos en este apartado.

Los modelos de BPMN representan, típicamente, el flujo de control de actividades, las organizaciones y los roles involucrados en ellas, así como los mensajes que se intercambian. Los elementos de BPMN se encuentran agrupados en cinco categorías básicas que son Objetos de Flujo (*Flow Objects*), Datos (*Data*), Objetos de Conexion (*Connecting Objects*), Calles (*Swimlanes*) y Artefactos (*Artifacts*). Los Objetos de Flujo son los más básicos elementos usados en la creación de diagramas de procesos de negocio. Los Datos se utilizan para mostrar la información que se utiliza a lo largo del proceso de negocio. Los Objetos de Conexión son los utilizados para interconectar Objetos de Flujo mediante diferentes tipos de flechas. Las Calles son usadas para agrupar actividades en categorías separadas por diferentes responsabilidades funcionales, y, finalmente los Artefactos, pueden ser añadidos para ofrecer una mayor información a lo que se procesa o incluso comentarios. Está fuera del alcance de este trabajo de tesis referirnos a cada uno de los elementos de la especificación de BPMN de forma más detallada, aunque sí trataremos alguno de estos elementos de manera más minuciosa.

Una vez introducido el contexto de BPMN procederemos a la realización del método definido en la sección anterior. Se establece que el lenguaje es BPMN, el cual dispone de una sintaxis

abstracta que es su propio metamodelo, y una sintaxis concreta, que es la notación gráfica con la que se representa cada elemento y que puede consultarse en la propia especificación [OMG, 2010].

Al igual que en el caso anterior, los dos primeros pasos del método consisten en descubrir el metamodelo de BPMN y disponer de él en términos de MOF. De nuevo, ambos pasos son relativamente sencillos en el caso de este lenguaje puesto que en su especificación así se definen. En la figura 5.5 se han reagrupado el subconjunto de los conceptos del metamodelo BPMN que van a ser utilizados durante la realización del método[2].

A continuación, el tercer paso del método consiste en la alineación semántica, esto es, en investigar las correspondencias entre el metamodelo de BPMN 2.0 y el de INROMA en busca de dichas equivalencias conceptuales. En la tabla 5.2 aparecen tales correspondencias, lo que viene a ser el resultado de la ejecución del método.

INROMA	BPMN 2.0
Activity	Task
Process	Process
ProcessSequence	SequenceFlow
Initial	StartEvent
Final	EndEvent
Conditional	ExclusiveGateway
Stakeholder	ResourceRole
Product	DataInput, DataOutput

Tabla 5.2: Correspondencias entre INROMA y BPMN 2.0

[2]El metamodelo aquí reflejado es un subconjunto del metamodelo completo de BPMN. Para consultar el metamodelo completo se sugiere acudir a la especificación [OMG, 2010]

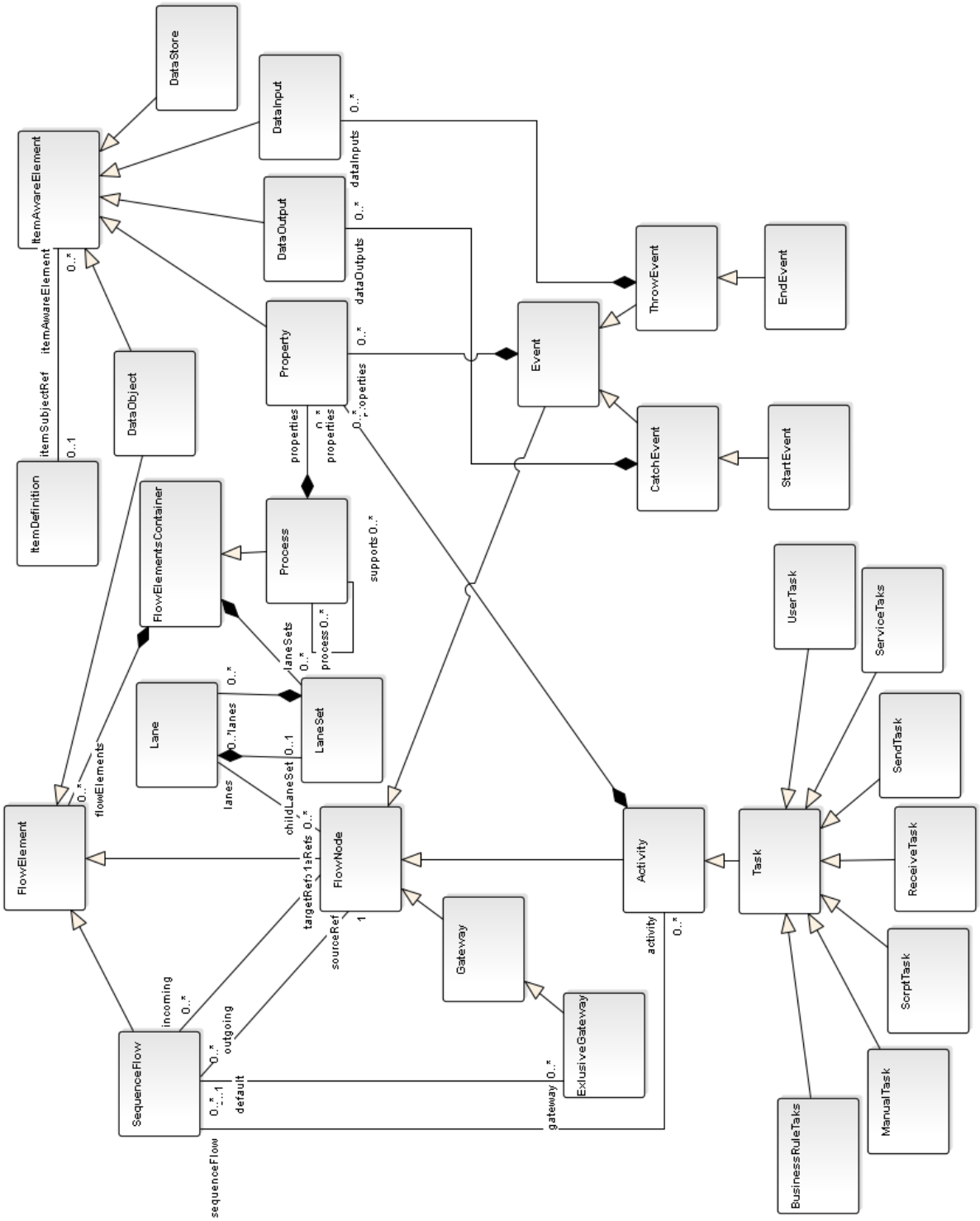


Figura 5.5: Vista parcial del metamodelo BPMN 2.0

5.3.4. Integration DEFinition, IDEF3

El conjunto de lenguajes de modelado IDEF [IDEF, 1995] surgió en la década de los 70, auspiciado por el programa U.S. Air Force Integrated Computer Aided Manufacturing (ICAM). El objetivo de ICAM era aprovechar la tecnología informática para incrementar la productividad de fabricación. Una suposición fundamental del programa fue la necesidad de disponer de un método de modelado para soportar el diseño y análisis de los sistemas poderoso, pero al mismo tiempo fácilmente utilizable. Por ello el programa emprendió el desarrollo de los métodos IDEF que, inicialmente incluyen: un método de modelado de función (IDEF0), un método de modelado de datos o conceptos (IDEF1), un método de especificación del modelo de simulación (IDEF2). Adicionalmente, más adelante se desarrolló el método de modelado de procesos (IDEF3), que es el que va a ocupar nuestra atención. Asimismo existen dos métodos adicionales IDEF4, que es un método de diseño de software orientado a objetos, e IDEF5, que constituye un método de ingeniería y adquisición de conocimiento [Menzel y Mayer, 1998] [Kim *et al.*, 2003] [Noran, 2000].

El IDEF3 Process Description Capture Method fue creado específicamente para capturar descripciones de secuencias ordenadas de actividades o eventos. El principal objetivo era proporcionar un método estructurado mediante el cual un experto de dominio puede expresar conocimiento sobre la operación de un particular sistema u organización y que les permita reflejar las relaciones de precedencia y causalidad entre las diferentes actividades, algo que era imposible de expresar en los métodos IDEF previos (IDEF0, IDEF1, IDEF1X e IDEF2).

Una vez introducido el contexto de IDEF3, procederemos a la realización del método definido en la sección anterior. En este caso definimos IDEF3 como lenguaje el cual, como los anteriores, dispone de una sintaxis abstracta y una sintaxis concreta.

A diferencia de los dos casos anteriores, en la especificación de IDEF3 no está explícitamente disponible el metamodelo del lenguaje, por lo que se ha tenido que extraer a partir de dicha especificación y de los diferentes ejemplos disponibles de procesos que lo utilizan para su modelado. Este paso para este lenguaje en concreto, dadas las características específicas de IDEF3, consistió en la recopilación de todos los conceptos existentes y las relaciones entre ellos, a la que denominamos metamodelo, por ser una representación asbtracta de dicho lenguaje. Una vez extraído este metamodelo, en un segundo paso se adecua y se describe de acuerdo a los establecido para que pueda ser considerado conforme al metamodelo MOF. El resultado se puede observar en la figura 5.6.

Una vez disponible el metamodelo del lenguaje, de nuevo el tercer paso consiste en investigar las correspondencias entre IDEF3 e INROMA en busca de sus equivalencias semánticas. En la tabla 5.3 aparecen tales correspondencias, lo que viene a ser el resultado de la ejecución del método.

INROMA	IDEF3
Activity	UOB
Process	Scenario
ProcessSequence	ArrowFlow
Conditional	Junction
Product	Object

Tabla 5.3: Correspondencias entre INROMA e IDEF3

5.4. Conclusiones

En este capítulo se ha presentado de forma detallada el segundo de los elementos clave que conforman el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los procesos de software: el método de incorporación de lenguajes a dicho marco. Además de establecer el contexto de interoperabilidad y MDE en el que se ha definido la arquitectura del marco de referencia y que influye en gran medida en la definición del método, se ha presentado de forma detallada dicho método, modelándolo con INROMA y mostrando su aplicación con tres lenguajes de modelado de procesos de software, obteniendo como resultado un conjunto de correspondencias. Como ya hemos comentado anteriormente, estos lenguajes han sido elegidos por la disponibilidad que tenemos de ejemplos de procesos reales de empresa modelados con ellos, lo que nos va a permitir ir utilizándolos a partir de este momento, especialmente en la validación del marco una vez desarrollado.

Hasta este momento, en el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los procesos de software disponemos de INROMA como lenguaje base y del método para incorporar nuevos lenguajes al mismo. Mediante el método somos capaces de establecer las correspondencias existentes entre INROMA y cualquier lenguaje que se incluya en el marco. Sin embargo, todavía falta darle formalidad a dichas correspondencias. Por ello, para completar definitivamente el marco de referencia que estamos proponiendo en este trabajo de tesis falta por definir la tercera de las piezas claves del mismo: las transformaciones.

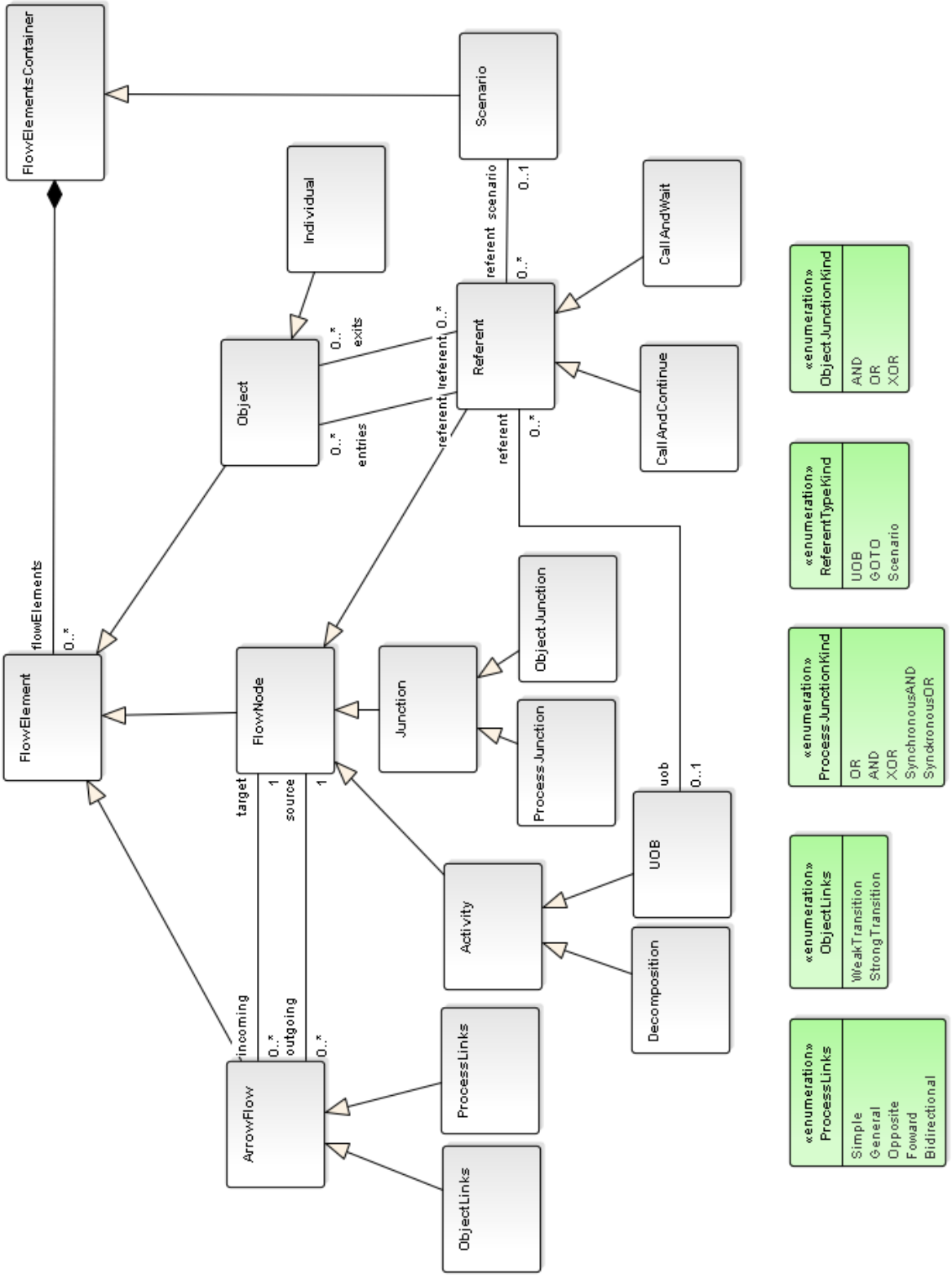


Figura 5.6: Vista parcial del metamodelo IDEF3

Capítulo 6

Las transformaciones en el marco de referencia

Hasta este punto se han descrito de forma detallada dos de las tres piezas claves que conforman el mecanismo de referencia para facilitar la interoperabilidad de los procesos de software, el lenguaje de modelado de procesos de software perteneciente al marco y el método para incorporar nuevos lenguajes de modelado de procesos de software a dicho marco. Para finalizar con la descripción de nuestra propuesta de solución faltaría desarrollar la tercera de las piezas: las transformaciones o reglas de derivación entre modelos que van a ser presentadas a continuación.

Con este fin el capítulo se estructura como sigue. En la primera sección se realizará una breve introducción sobre el concepto de transformaciones en MDE, sus principales características y alternativas. A continuación se presentará la incorporación de las transformaciones al marco de referencia propuesto, siguiendo las pautas establecidas en el paradigma MDE. En la tercera sección se desarrollarán las transformaciones para los lenguajes de modelado de procesos de software utilizados como ejemplos en el capítulo anterior, incorporando únicamente la explicación para alguno de los elementos, quedando recogidos en los anexos A, B y C la totalidad de estas transformaciones. Por último, para finalizar este capítulo, se resumirán las conclusiones más relevantes obtenidas en su desarrollo.

6.1. Transformaciones Modelo-a-Modelo en MDE

Las transformaciones, como ya hemos introducido en capítulos anteriores, son pieza clave en MDE al igual que lo son los metamodelos. Sin embargo, a diferencia de éstos, que disponen de tecnologías base bien establecidas a partir de las cuales construir metamodelos como MOF [OMG, 2014a], todavía no existe una base bien establecida sobre la que describir cómo a partir de un modelo origen lo transformamos para producir uno destino [Di Ruscio *et al.*, 2012], siendo muchas las propuestas para llevarlas a cabo y diferentes los criterios para la elección de una de

estas propuestas frente a otra.

En [Kleppe *et al.*, 2003] se define una *transformación Modelo-a-Modelo como la generación automática de un modelo destino a partir de un modelo origen de acuerdo a un conjunto de reglas de transformación.* Una *regla de transformación es una descripción de cómo una o mas contrucciones de un lenguaje origen puede ser convertida en una o más construcciones en el lenguaje destino.* Ya hemos comentado anteriormente que el conjunto de reglas de transformación debería ser considerado en sí como un modelo y, por tanto, estar basado en su correspondiente metamodelo o definición abstracta del lenguaje de transformación utilizado. Muchas son las propuestas que, durante la última década, se han planteado desde la industria o la academia, diferenciándose principalmente en sus paradigmas, construcciones, enfoques de modelado y soporte de herramientas, además de su idoneidad para resolver determinadas problemas, pudiendo clasificarlas en [Czarnecki y Helsen, 2006]:

- **Enfoques de manipulación directa**. Ofrecen una representación de modelos interna y algunas APIs para manipularlas. Normalmente ofrecen un marco orientado a objetos y los usuarios implementan las reglas de transformación mediante un lenguaje de programación.
- **Enfoque operacional**. Muy similar al anterior pero ofrece un soporte más dedicado para las transformaciones de modelo. Una solución típica en esta categoría es extender el formalismo de metamodelado utilizado con facilidades para expresar elementos de programación. En esta categoría estarían incluidos el QVT Operational mappings [OMG, 2011a], XMF [1] y Kermeta [Muller *et al.*, 2005].
- **Enfoque relacional**. Este enfoque agrupa las propuestas declarativas en las que el concepto principal son las relaciones matemáticas y la resolución de restricciones. Todas estas propuestas están libres de efectos laterales, a diferencia de lo que ocurre con las operacionales, además de proporcionar de forma natural soporte para la multidireccionalidad de las reglas. A este grupo pertenece QVT Relations [OMG, 2011a] o AMW [Del Fabro *et al.*, 2005].
- **Enfoque híbrido**. Las propuestas de este tipo combinan diferentes técnicas de ambas categorías. A este grupo pertenecen ATL [Jouault *et al.*, 2008] [Jouault y Kurtev, 2006] y ETL [Kolovos *et al.*, 2008], los cuales envuelven cuerpos imperativos en sentencias declarativas.
- **Enfoque basado en transformación de grafos**. Describen la transformación de un modelo origen a uno destino a través de grafos, con toda la teoría sobre los mismos de trasfondo. A esta categoría pertenecen AGG [Taentzer, 2004], TGG [Schürr, 1995] o VIATRA2 [Varró *et al.*, 2002].
- **Enfoques basados en reglas**. En ellos es posible describir reglas en la forma *guarda - action*. Durante su ejecución, las reglas son activadas de acuerdo a sus guardas y no mediante invocación, y se basan en los lenguajes de reglas tradicionales.

De esta clasificación se desprende que el número de propuestas planteadas es muy elevado y la elección de una por delante de otra resulta bastante ardua. En nuestro caso, nos hemos

[1] http://www.xactium.com/

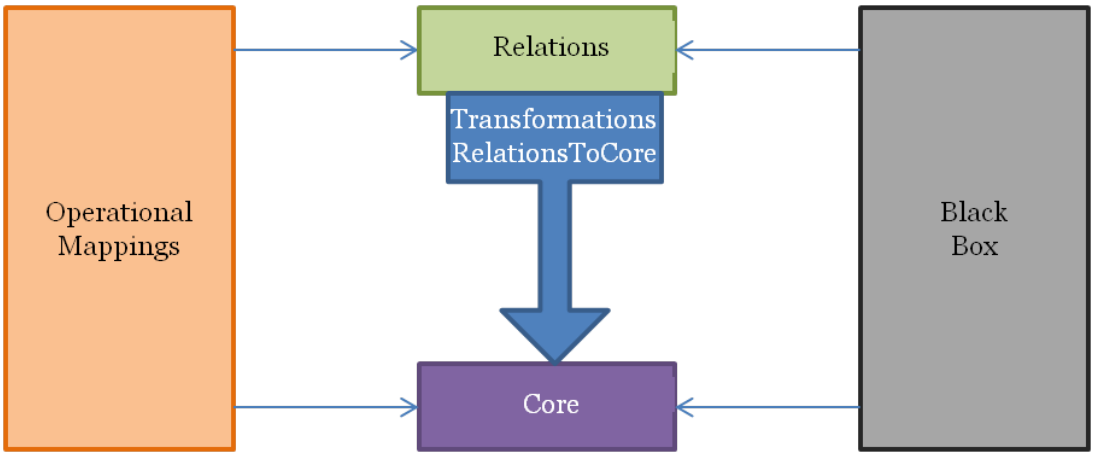


Figura 6.1: Los lenguajes de QVT: relaciones y arquitectura

decidido por QVT como lenguaje de transformación, al ser el estándar propuesto por OMG que, si bien en su versión actual dispone de ciertos aspectos sin resolver, confiamos tenga una mayor proyección tanto en la parte conceptual como de soporte en herramientas respaldado por este organismo de estandarización.

El estándar Query/View/Transformation, QVT, no es un único lenguaje sino que en su especificación se definen tres lenguajes diferentes, estableciendo un híbrido declarativo/imperativo, con la parte declarativa dividida en una arquitectura de dos niveles, tal y como se muestra en la figura 6.1.

Son muchos los autores que sugieren el uso de diferentes enfoques para distintos problemas de transformación entre modelos [Romeikat *et al.*, 2008]: los enfoques declarativos son mejores para especificar simples transformaciones y relaciones entre los elementos de los modelos origen y destino, mientras que las propuestas imperativas nos permiten resolver problemas de transformación más complejos. En QVT asumen dicha creencia, estableciendo dos lenguajes declarativos, QVT *Relations* y QVT *Core*, y uno imperativo, QVT *Operational Mappings, OM*.

Relations soporta un complejo mecanismo de coincidencia de patrones (*pattern matching*) que de forma implícita crea instancias para registrar lo que ocurre en una transformación. Ofrece, por naturaleza, soporte a la bidireccionalidad de las transformaciones, reduciendo el esfuerzo en la sincronización de los escenarios. Además, este lenguaje dispone de una sintaxis gráfica. El lenguaje *Core* es más simple, estableciendo la coincidencia de patrones únicamente para un conjunto de variables. Es igual de potente que *Relations* pero resultaría más farragoso, por lo que se establece como referencia para la semántica de *Relations*, el cual, mediante un lenguaje de transformación propio, puede ser enlazado con *Core*.

Sin embargo, en muchas ocasiones para la definición de transformaciones complejas no es posible el uso de un enfoque puramente declarativo [Gardner *et al.*, 2003]. QVT extiende *Relations* y *Core* mediante un tercer lenguaje, *Operational Mappings*, un enfoque imperativo que proporciona extensiones de OCL [OMG, 2012] con un estilo meramente procedimental. Además,

deja una puerta abierta al uso de un lenguaje externo de programación, para la implementación de algoritmos complejos y la reutilización de librerías externas, mediante el mecanismo de *Black-Box*. Este mecanismo, aunque existente, no es muy recomendable por las posibles inconsistencias que puedan surgir durante su uso.

QVT Relations nos ofrece bidireccionalidad, algo que en el marco de referencia que estamos desarrollando es fundamental al reducir el esfuerzo de sincronización que toda transformación requiere. También debemos apuntar que el hecho de contar con una representación gráfica facilita su comprensión y diseño a las personas ajenas a la programación. Como hemos apuntado anteriormente, el enfoque declarativo de *Relations* no podría ser utilizado en el caso de disponer de transformaciones muy complejas, pero en nuestro caso las transformaciones no son tan complicadas como para no hacer uso de toda la potencialidad que nos ofrece. Otro de los aspectos importante a tener en cuenta es la experiencia del grupo IWT2, en el que se ha desarrollado este trabajo de tesis, en el uso de QVT frente al resto de las propuestas. Todos estos elementos han propiciado la elección de utilizar QVT Relations para las transformaciones del marco de referencia.

Una vez justificada su elección, vamos a profundizar en el lenguaje QVT Relations, en su sintaxis textual y gráfica, para dejar sentadas las bases sobre las que definiremos las transformaciones en nuestro marco de referencia.

QVT Relations define una sintaxis concreta textual mediante una gramática cuya especificación completa se encuentra en [OMG, 2011a] y de forma muy resumida presentamos en el algoritmo 6.1.

```
1  transformation ... {
2     top relation R {
3        checkonly|enforce domain a...
4        checkonly|enforce domain b...
5        when {...}
6        where {...}
7     }
8
9     top relation S ...
10    relation ...
11    relation ...
12    ...
13 }
```

Algoritmo 6.1: Versión simple de la sintaxis concreta textual de QVT Relations

Una **transformation** establece las relaciones bidireccionales que existen entre los elementos de dos modelos. Las **top relation** son aquellas que deben ejecutarse siempre, mientras que el resto son invocadas directamente dentro de alguna cláusula de otra relación. Una relación puede estar condicionada por dos predicados, las cláusulas **when** y **where**. La primera especifica las pre-condiciones que deben cumplirse por todos los elementos del modelo que participan en la relación y puede requerir el cumplimiento de ciertos valores en las variables y sus dominios. La segunda especifica las condiciones que deben mantenerse en la relación. Cuando una transformación se ejecuta en la dirección de un dominio **checkonly**, implica que tan sólo se comprueba

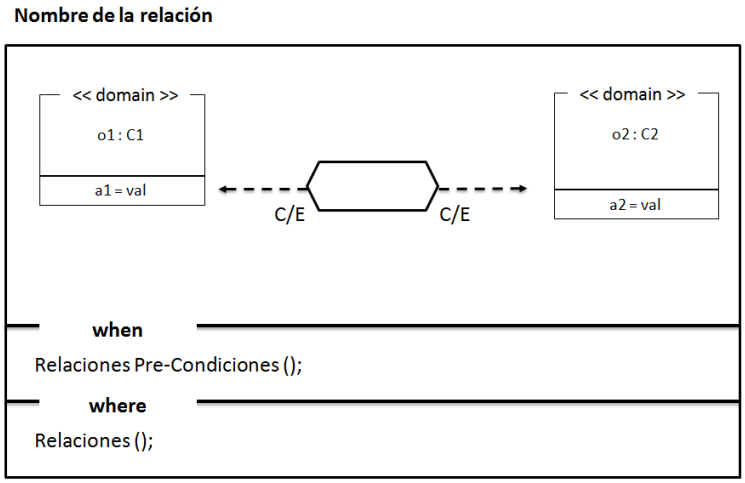


Figura 6.2: Ejemplo básico de la sintaxis concreta gráfica de QVT Relations

si existe un dominio válido en el modelo que satisface la relación. En un dominio **enforce**, en el caso de que no existiera (lo que implica que ese chequeo de que en ambos modelos existan los elementos que cumplen la relación falla), el modelo destino es modificado para poder satisfacer la relación. Las relaciones incluyen expresiones en las que se establecen los mapeos entre los elementos del modelo origen y los declarados dentro del dominio que, a su vez, puede incluir variables libres sin asignar o pueden estar ya con un valor como resultado de la ejecución de otras evaluaciones o relaciones.

QVT Relations también incluye una sintaxis gráfica que complementa a la textual anteriormente comentada y que se materializa en los diagramas de transformación como una notación visual para representar transformaciones, dominios y patrones. Cabe destacar que la notación gráfica también permite el uso de los diagramas de clase UML para representar transformaciones, si bien se conciben como un subconjunto de los diagramas de transformación.

En la figura 6.2 se presenta gráficamente una relación con la sintaxis gráfica de QVT Relations. En ella se representa de forma gráfica que una relación se establece entro dos o más patrones. Cada patrón es un conjunto de objetos, referencias y valores. La estructura de un patrón puede expresarse mediante diagramas de objetos UML, aunque la especificación sugiere su extensión, con los elementos añadidos en los diagramas de transformación, como: la definición de dominios (clases, objetos y valores), las relaciones entre dominios (anotadas como C en el caso de **checkonly** o E en para **endforce**), elementos que aparecen en el primer recuadro de la transformación, así como las relaciones (***non-top***) a ejecutar en las cláusulas **when**, reflejadas en el segundo recuadro, y **where**, que son visibles en el tercer recuadro.

Teniendo en cuenta que lo que pretende el marco de referencia es facilitar la interoperabilidad de modelos de procesos, se ha decidido que la dirección hacia INROMA sea de tipo **enforce**, para que en el caso de que fuera la primera vez que se realizara la transformación para ese proceso y no existiera, este proceso fuera creado. En la dirección opuesta, desde INROMA hacia cualquiera de los lenguajes de modelado de procesos de software con el que se estén constitu-

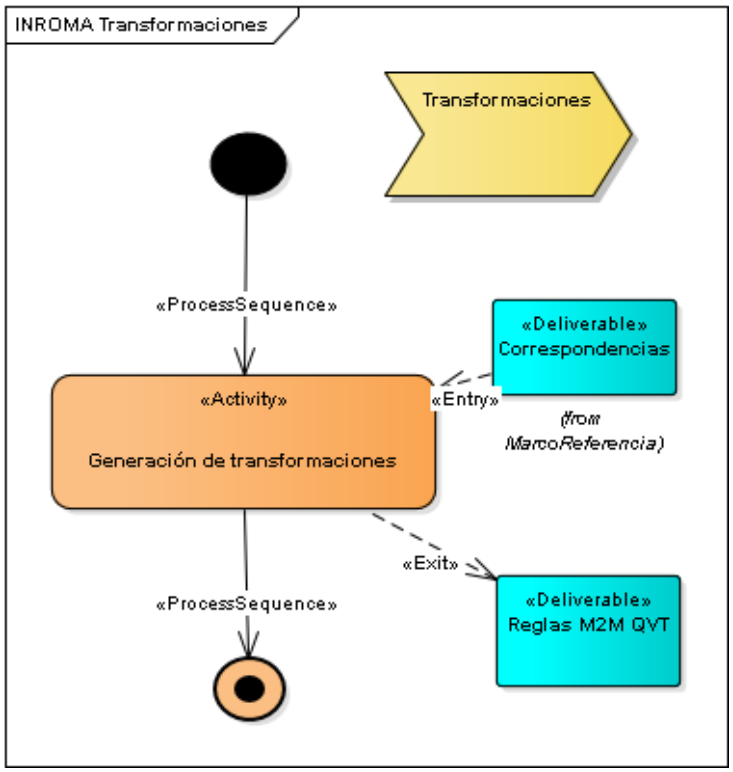


Figura 6.3: Las tranformaciones en el marco de referencia

yendo las transformaciones, se establece una relación anotada como **checkonly** para no realizar modificaciones en el proceso de software modelado en el lenguaje original pero sí anotar las diferencias. De esta forma, los procesos siguen estando modelados en el lenguaje de modelado de procesos de software definido por la organización y su reflejo o traducción a INROMA únicamente se utiliza para el momento en el que se quiera interoperar con otros procesos modelados con lenguajes diferentes al del proceso en cuestión.

Una vez establecida la base teórica, a continuación se va a desarrollar las transformaciones y la forma en la que con ellas se va a trabajar en el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los modelos de procesos de software.

6.2. Las transformaciones en el marco de referencia

Una vez introducidas las transformaciones, a continuación se expone la manera en la que pasan a convertirse en la tercera de las piezas clave del marco de referencia. Al igual que hemos hecho con el método, definimos mediante un modelo INROMA en el que se recoge la generación de las tranformaciones.

Así, en la figura 6.3 se observa cómo a partir de las correspondencias obtenidas con el método

del marco de referencia se definen las transformaciones oportunas en QVT Relations mediante la actividad *Generación de transformaciones*. Para ello, para cada lenguaje incorporado en el marco de referencia se establecerá una *top relation* en la que se establece la relación entre el concepto *Process* de INROMA y su correspondencia en la tabla de correspondencias. En ella se establece las relaciones *Activity*, *Initial*, *Final*, *Conditional* e *Indicator* de INROMA con su elemento correspondiente. Y, dentro de la relación de *Activity*, se hace lo propio con *Product*, *ProcessSequence* y *Stakeholder*.

Para ilustrar claramente la forma en la que se definen las transformaciones, en la siguiente sección se generan las transformaciones para los lenguajes de modelado de procesos de software utilizados en el capítulo 5, mostrando alguna de ellas, las más representativas, como ejemplos de uso. La totalidad de las transformaciones definidas para dichos lenguajes se encuentran recogidas en los anexos A , B y C.

6.3. Las transformaciones en la práctica: Ejemplos de aplicación

Seguidamente vamos a exponer en mayor detalle cómo se van llevar a cabo las transformaciones entre los lenguajes de modelado de procesos de software incorporados al marco de referencia mediante el método definido para tal fin y el lenguaje INROMA. Para ello, utilizaremos como ejemplos los lenguajes seleccionados en el capítulo anterior, es decir, los diagramas de actividad de UML2.0, BPMN e IDEF, y atendiendo a las tablas de correspondencia igualmente presentadas, haremos uso del lenguaje QVT Relations para establecer las transformaciones modelo a modelo bidireccionales entre cada uno de los lenguajes e INROMA, utilizando tanto la sintaxis textual como la gráfica.

6.3.1. Transformaciones INROMA - Diagramas de Actividad UML

Seguidamente vamos a exponer las principales relaciones necesarias para establecer la transformación entre INROMA y los Diagramas de Actividad de UML. El listado completo de las mismas se encuentra detallado en el anexo A.

La transformación tiene una única relación principal obligatoria que establece la correspondencia entre las actividades de UML y los procesos de INROMA, cuya vista gráfica se observa en la figura 6.4. En ella, atendiendo al vínculo entre las metaclases *Activity* de UML y *Process* de INROMA, realizamos el mapeo a través del nombre, lo que se repetirá para todos los lenguajes a incorporar en el marco de referencia. Se mantiene el criterio establecido en la definición de transformaciones dentro del marco de referencia en lo relativo a las anotaciones **checkonly** y **enforce** en las direcciones hacia UML e INROMA respectivamente. De nuevo esta relación principal es la encargada de invocar a las que llenan las actividades (*MapActivities*), el estado inicial (*MapInitial*), los estados finales (*MapFinal*) y los condicionales (*MapConditional*). Cabe destacar que la ausencia de indicadores en los diagramas de actividad de UML hace inviable establecer la correspondencia con las metaclases *Metric* e *Indicator* de INROMA.

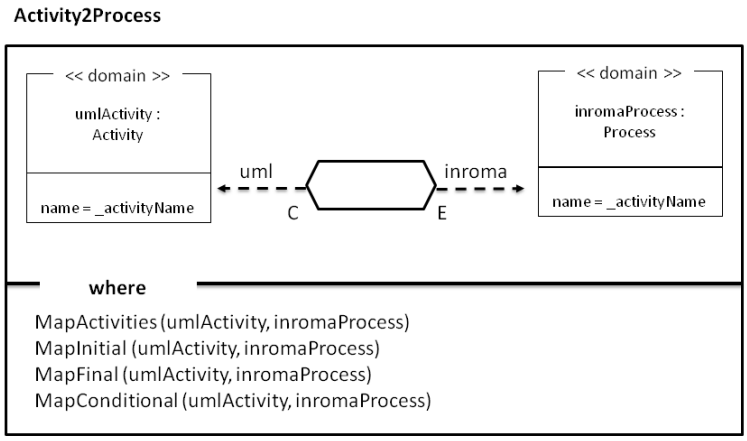


Figura 6.4: Vista QVT Relations de la relación Activity2Process en la transformación UML2INROMA

La notación textual de esta relación se detalla en el algoritmo 6.2.

```
1   top relation Activity2Process
2   {
3       _activityName : String;
4       checkonly domain umlmodel umlActivity : uml::Activity
5       {
6           name = _activityName
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          name = _activityName
11      };
12      where {
13          MapActivities (umlActivity, inromaProcess);
14          MapInitial (umlActivity, inromaProcess);
15          MapFinal (umlActivity, inromaProcess);
16          MapConditional (umlActivity, inromaProcess);
17          -- No existen Indicators en UML, no es posible realizar el mapeo
18                  -- MapIndicators (umlActivity, inromaProcess);
19      }
20  }
```

Algoritmo 6.2: Relación principal para la transformación UML2INROMA

A modo de ejemplo, en la figura 6.5 se observa la relación encargada de mapear las actividades (*MapActivities*) en la transformación UML2INROMA. Dado que viene invocada por la relación principal, los dominios de la actividad en UML y el proceso en INROMA ya están establecidos.

En este caso vamos a establecer una correspondencia entre las metaclases *Action* de los diagramas de actividad de UML y *Activity* de INROMA. Para ello, desde la *Activity* de UML iremos a través de la relación *node* para encontrar las *Action*, y vincularemos su nombre. En el caso de INROMA, utilizaremos la relación *elements* para llegar a las *Activity*. Fijados la actividad (en

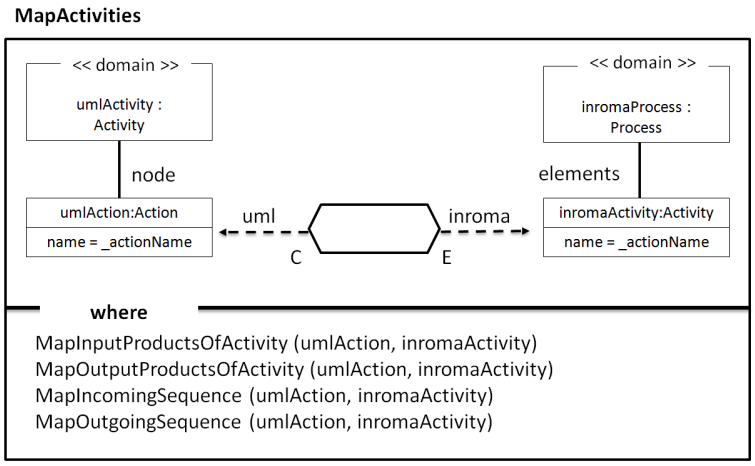


Figura 6.5: Vista QVT Relations de la relación MapActivities en la transformación UML2INROMA

UML) y el proceso (en INROMA), así como la acción (en UML) y la actividad (en INROMA), esta relación invoca a las encargadas de vincular los productos de entrada (*MapInputProductsOfActivity*) o salida (*MapOutputProductsOfActivity*), así como de establecer la secuencia de las mismas (*MapIncomingSequence* y *MapOutgoingSequence*). Debido a que en los diagramas de actividad de UML no existe un vínculo con los actores, no es posible llevar a cabo el mapeo de la metaclase *Stakeholder* de INROMA.

La notación textual queda detallada en el algoritmo 6.3.

```
1   relation MapActivities
2   {
3       _actionName : String;
4       checkonly domain umlmodel umlActivity : uml::Activity
5       {
6           node = umlAction : uml::Action {name = _actionName}
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          elements = inromaActivity : inroma::Activity { name = _actionName }
11      };
12      where {
13          MapInputProductsOfActivity (umlAction, inromaActivity);
14          MapOutputProductsOfActivity (umlAction, inromaActivity);
15          -- No existe relación entre un Actor y los diagramas de actividad en UML,
                no es posible realizar el mapeo
16          --MapStakeholdersOfActivity (umlAction, inromaActivity);
17          MapIncomingSequence (umlAction, inromaActivity);
18          MapOutgoingSequence (umlAction, inromaActivity);
19      }
20  }
```

Algoritmo 6.3: Relación MapActivities para la transformación UML2INROMA

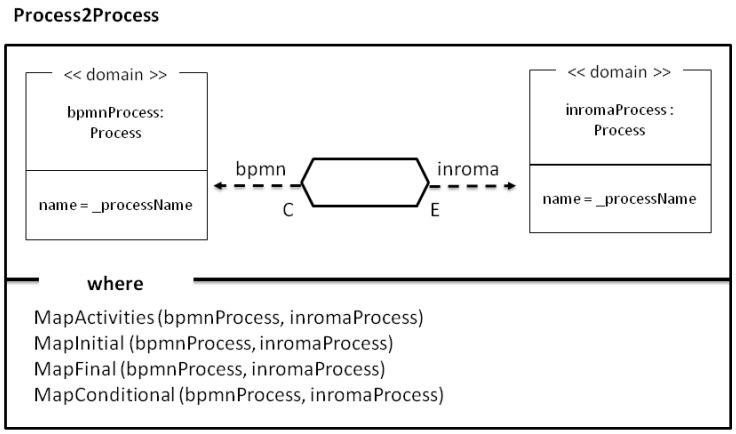


Figura 6.6: Vista QVT Relations de la relación Process2Process en la transformación BPMN2INROMA

6.3.2. Transformaciones INROMA - Business Process Model and Notation, BPMN

En esta sección vamos a presentar algunas de las relaciones establecidas para realizar la transformación bidireccional entre INROMA y BPMN, atendiendo a la tabla de correspondencias presentada en el capítulo anterior. El listado completo de las mismas puede verse en el anexo B.

La transformación tiene una única relación principal, obligatoria (**top relation**), que establece las correspondencias entre los procesos. En el algoritmo 6.4 se detalla su notación textual y su representación gráfica puede observarse en la figura 6.6.

```
1   top relation Process2Process
2   {
3       _processName : String;
4       checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
5       {
6           name = _processName
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          name = _processName
11      };
12      where {
13          MapActivities (bpmnProcess, inromaProcess);
14          MapInitial (bpmnProcess, inromaProcess);
15          MapFinal (bpmnProcess, inromaProcess);
16          MapConditional (bpmnProcess, inromaProcess);
17          -- No existen Indicators en BPMN, no es posible realizar el mapeo
18          -- MapIndicators (bpmnProcess, inromaProcess);
19      }
20  }
```

Algoritmo 6.4: Relación principal para la transformación BPMN2INROMA

En ella de nuevo, atendiendo al vínculo entre las metaclases *Process* de BPMN y de INROMA, realizamos el mapeo a través del nombre. En estas transformaciones nuevamente se mantiene el criterio en lo relativo a las anotaciones **checkonly** y **enforce** en las direcciones hacia BPMN e INROMA respectivamente.

Esta relación principal hace que se ejecuten otra serie de relaciones, que tendrán prefijado el dominio de los procesos y que se explicitan bajo la cláusula **where**. En particular, la que se encarga del mapeo de las actividades (*MapActivities*), el estado inicial (*MapInitial*), los estados finales (*MapFinal*) y los condicionales (*MapConditional*). Cabe destacar que debido a la ausencia de indicadores o métricas dentro del metamodelo de BPMN, no es posible realizar un mapeo a las metaclases *Metric* e *Indicator* de INROMA.

A modo de ejemplo, describimos la relación *MapActivities*, la encargada de mapear las actividades dentro de la transformación BPMN2INROMA, cuya notación textual de dicha relación queda detallada en el algoritmo 6.5.

```
1   relation MapActivities
2   {
3       _taskName : String;
4       checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
5       {
6           flowElements = bpmnTask : bpmn2::Task {name = _taskName}
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          elements = inromaActivity : inroma::Activity { name = _taskName }
11      };
12      where {
13          MapInputProductsOfActivity (bpmnTask, inromaActivity);
14          MapOutputProductsOfActivity (bpmnTask, inromaActivity);
15          MapStakeholdersOfActivity (bpmnTask, inromaActivity);
16          MapIncomingSequence (bpmnTask, inromaActivity);
17          MapOutgoingSequence (bpmnTask, inromaActivity);
18      }
19  }
```

Algoritmo 6.5: Relación MapActivities para la transformación BPMN2INROMA

Dado que viene invocada por la relación principal, los dominios del proceso ya están establecidos. En este caso vamos a establecer una correspondencia entre las metaclases *Task* de BPMN y *Activity* de INROMA. Para ello, desde el proceso iremos a través de la relación *flowElements* para encontrar las *Task*, y vincularemos su nombre. En el caso de INROMA utilizaremos la relación *elements* para llegar a las *Activity*. Fijados el proceso (en BPMN e INROMA), la tarea (en BPMN) y la actividad (en INROMA), esta relación invoca a otras que serán las encargadas de vincular los productos de entrada (*MapInputProductsOfActivity*) o salida (*MapOutputProductsOfActivity*), los stakeholders (*MapStakeholdersOfActivity*), así como de establecer la secuencia de las mismas (*MapIncomingSequence* y *MapOutgoingSequence*). Debido a que en BPMN no se establece la responsabilidad de una tarea, hemos establecido la correspondencia como participante, sin establecer ningún responsable. En la figura 6.7 se puede observar la representación gráfica de la relación *MapActivities* de la transformación BPMN2INROMA.

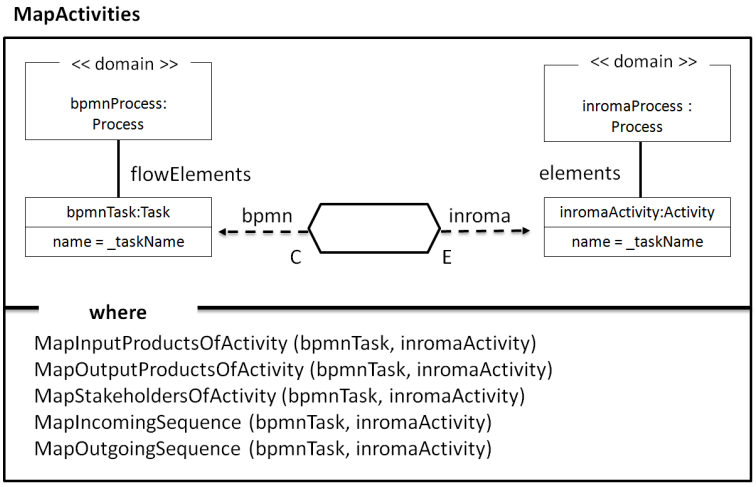


Figura 6.7: Vista QVT Relations de la relación MapActivities en la transformación BPMN2INROMA

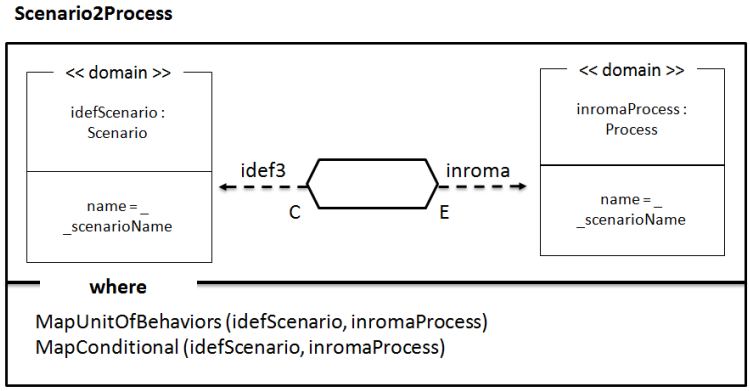


Figura 6.8: Vista QVT Relations de la relación Scenario2Process en la transformación IDEF2INROMA

6.3.3. Transformaciones INROMA - Integration DEFinition, IDEF

Por último presentamos algunas relaciones dentro de la transformación entre INROMA e Integration DEFinition, cuyo detalle se encuentra reflejado en el anexo C.

La transformación tiene una única relación principal obligatoria que establece la correspondencia entre un escenario de IDEF3 y los procesos de INROMA, cuya vista gráfica de la relación *Scenario2Process* puede verse en la figura 6.8. De nuevo, en estas transformaciones se mantiene el criterio en lo relativo a las anotaciones **checkonly** y **enforce** en las direcciones hacia IDEF3 e INROMA respectivamente. Esta relación principal será la encargada de, a través de la cláusula **where** de QVT, lanzar el resto de relaciones como la que mapea las *Unit of Behaviors* y los *Conditional*. En esta ocasión no sólo no podremos establecer correspondencias con los indicadores, sino que tampoco estableceremos las mismas en relación a los estados iniciales y finales, al

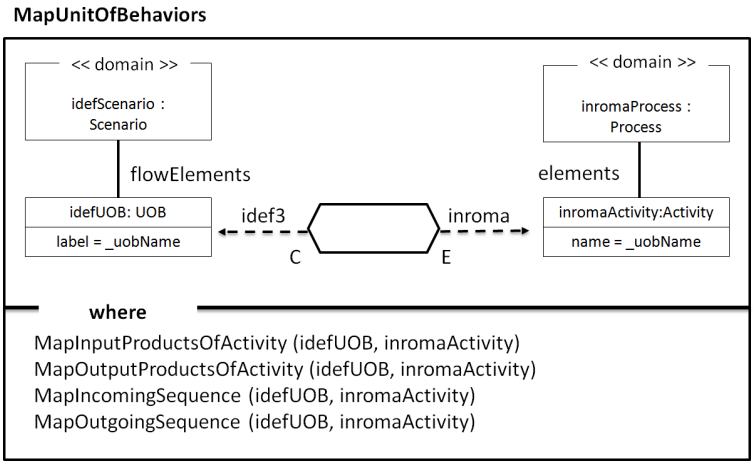


Figura 6.9: Vista QVT Relations de la relación MapUnitOfBehaviors en la transformación UML2INROMA

estar ausentes en IDEF3. La notación textual de esta relación se detalla en el algoritmo 6.6.

```
1   top relation Scenario2Process
2   {
3       _scenarioName : String;
4       checkonly domain idefmodel idefScenario : idef3::Scenario
5       {
6           name = _scenarioName
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          name = _scenarioName
11      };
12      where {
13          MapUnitOfBehaviors (idefScenario, inromaProcess);
14          -- No existe Inicial ni Final en IDEF3, no es posible realizar el mapeo
15          -- MapInitial (idefScenario, inromaProcess);
16          -- MapFinal (idefScenario, inromaProcess);
17          MapConditional (idefScenario, inromaProcess);
18          -- No existen Indicators en IDEF3, no es posible realizar el mapeo
19          -- MapIndicators (idefScenario, inromaProcess);
20      }
21  }
```

Algoritmo 6.6: Relación principal para la transformación IDEF2INROMA

A modo de ejemplo, describiremos la relación encargada de mapear las **Unit of Behaviours** con las actividades en la transformación IDEF2INROMA, denominada *MapUnitOfBehaviors*, y cuya representación gráfica puede observarse en la figura 6.9.

Dado que viene invocada por la relación principal, los dominios del escenario en IDEF3 y el proceso en INROMA ya están establecidos. En este caso vamos a establecer una correspondencia entre las metaclases *Unit of Behavior* de IDEF3 y *Activity* de INROMA. Para ello, desde el

Scenario de IDEF3 iremos a través de la relación *flowElements* para encontrar las *UOB*, y vincularemos su nombre a través del atributo *label*. En el caso de INROMA, seguiremos la relación *elements* para llegar a las *Activity*. Establecidos el estereotipo (en IDEF3) y el proceso (en INROMA), así como la UOB (en IDEF) y la actividad (en INROMA), esta relación a través de la cláusula **where** invoca a las relaciones encargadas de establecer las correspondencia entre los productos de entrada (*MapInputProductsOfActivity*) y de salida (*MapOutputProductsOfActivity*), así como de establecer la secuencia de las mismas (*MapIncomingSequence* y *MapOutgoingSequence*). Al igual que ocurría en los diagramas de actividad de UML, en IDEF3 no se establece el concepto de *Stakeholders* por lo que no es posible llevar a cabo el mapeo de la metaclase *Stakeholder* de INROMA. Al igual que en ejemplos anteriores, detallamos la notación textual de esta relación en el algoritmo 6.7.

```
1   relation MapUnitOfBehaviors
2   {
3       _uobName : String;
4       checkonly domain idefmodel idefScenario : idef3::Scenario
5       {
6           flowElements = idefUOB : idef3::UOB {label = _uobName}
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          elements = inromaActivity : inroma::Activity { name = _uobName }
11      };
12      where {
13          MapInputProductsOfActivity (idefUOB, inromaActivity);
14          MapOutputProductsOfActivity (idefUOB, inromaActivity);
15          -- No existen Stakeholders en IDEF3, no es posible realizar el mapeo
16          -- MapStakeholdersOfActivity (idefUOB, inromaActivity);
17          MapIncomingSequence (idefUOB, inromaActivity);
18          MapOutgoingSequence (idefUOB, inromaActivity);
19      }
20  }
```

Algoritmo 6.7: Relación MapUnitOfBehaviors para la transformación IDEF2INROMA

6.4. Conclusiones

En este capítulo hemos visto de forma detallada cómo se generan las transformaciones para ser incorporadas en nuestro marco de referencia. Dichas transformaciones constituyen el conjunto de reglas de derivación entre INROMA y cualquier lenguaje de modelado de procesos de software que se desee incorporar al marco de referencia, y será necesaria su generación para cada nuevo lenguaje. Además, para ilustrar la forma en la que se establecen dichas transformaciones, a modo de ejemplo hemos mostrado las más representativas correspondientes a los lenguajes de modelado de procesos de software utilizados en el capítulo 5. La totalidad de las transformaciones definidas para dichos lenguajes se encuentran recogidas en los anexos A , B y C.

En este punto las tres bases sobre las que se sustenta el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de modelos de procesos de software ya han sido definidas,

por lo tanto, la propuesta de solución ya se ha desarrollado desde el punto de vista teórico. Sin embargo, si nuestra pretensión es la utilización práctica del mismo, con el objetivo de hacer llegar los beneficios de dicho marco a las organizaciones de software, es necesario definir una herramienta de soporte que lo implemente. Este será el objetivo a tratar en el siguiente capítulo. Los lenguajes de modelado de procesos de software utilizados como ejemplo en este capítulo y el anterior van a ser utilizados con el marco de referencia, tanto en la exposición de la herramienta como en los ejemplos de los casos de estudio, de ahí, como ya hemos comentado en varias ocasiones, su elección.

Capítulo 7

MONETA: Herramienta de soporte para el marco de referencia

En los tres capítulos anteriores se ha profundizado en el desarrollo de la propuesta de solución de este trabajo de tesis con el objetivo de conformar un marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los modelos de procesos de software. Para ello se ha ido trabajando en los tres fundamentos o piezas claves sobre los que se sustenta dicho marco: un lenguaje de modelado de procesos de software para el marco, al que hemos denominado INROMA, un método para la incorporación de lenguajes de modelado de procesos de software y las transformaciones para poder operar en dicho marco.

Una vez finalizada la exposición teórica sobre la que se desarrolla nuestra propuesta, todavía no hemos cumplido con la totalidad de los objetivos que nos marcamos en su comienzo. Una de nuestras pretensiones principales era la de que este marco pudiera ser llevado a la práctica y de él se beneficiaran en su utilización diferentes organizaciones de software, ya fueran empresas dedicadas al desarrollo y mantenimiento de software o departamentos de software dentro de empresas más grandes. Para ello se nos plantea la necesidad de desarrollar una herramienta de soporte que implemente el marco de referencia que tan minuciosamente hemos definido. A esta herramienta la hemos denominado MONETA, y el objetivo de este capítulo es tratar de mostrar en profundidad cómo dicha herramienta da soporte al marco de referencia.

Para conseguir este objetivo el presente capítulo se estructura como sigue. En la primera sección se detalla cuál va a ser el entorno de desarrollo y ejecución de MONETA y las causas de su elección, exponiendo claramente cómo dicho entorno va a dar soporte a la herramienta que se desea desarrollar. A continuación se presenta de forma detallada cómo la herramienta MONETA da soporte a los tres elementos claves que estamos mencionando en todo momento. En la tercera sección se realizará la validación con las situaciones reales que han justificado la realización de este trabajo de tesis y que fueron descritas en los capítulos iniciales, utilizando MONETA como implementación de la solución propuesta para ello. Por último, en la última sección se mencionan las conclusiones más relevantes del capítulo.

7.1. Entorno de trabajo para MONETA: Enterprise Architect

A la hora de hacer el planteamiento para el desarrollo de la herramienta MONETA se disponía, como suele ser habitual en la elaboración de cualquier aplicación software, de dos alternativas sobre las que fundamentarnos:

- Por un lado, la posibilidad de diseñar e implementar una herramienta desde cero (*from scratch*), que disponga de las capacidades de modelado de procesos, a ser posible con una interfaz gráfica, y ejecución de las transformaciones;
- La otra opción consiste en tomar como base una herramienta ya existente en el mercado que nos ofrezca los mecanismos básicos de modelado y que admita la inclusión de complementos o extensiones, lo que nos permitiría a nosotros como desarrolladores externos ampliar las funciones básicas que ofrece.

Ambas opciones tienen sus ventajas e inconvenientes. En la primera opción, la principal ventaja consiste en controlar la herramienta desarrollada en su totalidad, incluyendo las futuras evoluciones, a modo de versiones o adaptaciones de la misma, algo de lo que no se dispone en el caso de optar por una herramienta externa como base, puesto que estamos sujetos a la forma de evolucionar comercialmente el producto que el propietario de la misma elija. Sin embargo, el desarrollar una herramienta de estas características desde la nada supone un esfuerzo demasiado grande para llegar a un nivel similar al de las herramientas de este tipo que existen en el mercado.

Hace ya unos años, el grupo IWT2, en el que se encuadra el trabajo de esta tesis, optó por utilizar Enterprise Architect [SparxSystems, 2014] como base para sus desarrollos e investigaciones en el ámbito de MDE. Esta decisión se fundamentó en un estudio llevado a cabo por el grupo para la Consejería de Cultura y Deporte de la Junta de Andalucía [IWT2 y Consejeria de Cultura de la Junta de Andalucia, 2013], donde se realizaba una comparativa de nueve herramientas de modelado de acuerdo a un esquema de caracterización en el que se tenían en cuenta aspectos tales como la compatibilidad con UML, el soporte a mecanismos de extensión de UML para la definición de nuevos perfiles, la generación sistemática de modelos y código a partir de otros modelos, la generación de documentación de forma automática o el soporte para la gestión del ciclo de vida del software en grandes proyectos.

En dicho estudio se concluía que Enterprise Architect era la herramienta de mejor relación calidad/precio que soportaba todas las características del marco comparativo, por lo que fue elegida como aquella a utilizar en los desarrollos del grupo. Así, con la elección de una herramienta comercial de estas características, IWT2 disponía de una base sólida y de garantías para el desarrollo de los proyectos de transferencia con empresas y organismos públicos y privados, con la posibilidad de incluir las extensiones necesarias derivadas de los resultados de sus proyectos de investigación.

En este contexto se fundamenta la elección de Enterprise Architect como el entorno de trabajo para el desarrollo de MONETA puesto que, tal y como ha quedado patente a lo largo de toda la memoria, una de nuestras aspiraciones es la aplicación y transferencia de los resultados

de este trabajo de tesis a empresas. No obstante, nos gustaría reseñar que, aunque únicamente se haya utilizado como una validación de las pruebas de concepto de lo que aquí estamos proponiendo, también hemos hecho uso del entorno Eclipse [The Eclipse Foundation, 2014b] y sus diferentes extensiones para el modelado, metamodelado y definición de transformaciones (Eugenia [The Eclipse Foundation, 2014c], QVTMedini [IKV++, 2011], BPMN Modeler [The Eclipse Foundation, 2014a], Papyrus [The Eclipse Foundation, 2015], etc.) comprobando la viabilidad de su posible elección como herramienta base.

7.1.1. Añadir un nuevo lenguaje en Enterprise Architect

Como más adelante detallaremos, la posibilidad de poder ir incorporando nuevos lenguajes en MONETA y, por tanto, que este requerimiento estuviera soportado por Enterprise Architect, consistía en otro de los elementos clave para su elección. Enterprise Architect dispone de un conjunto de lenguajes que ya vienen integrados en su distribución comercial, entre los que se encuentran tanto BPMN como los diferentes diagramas de UML, y además dispone de una funcionalidad para la definición de perfiles UML. Veamos a continuación cómo dicha funcionalidad satisface nuestra necesidad de incorporar nuevos lenguajes de modelado de procesos de software en MONETA.

Ya hemos referido en capítulos anteriores que un lenguaje de modelado de procesos de software debería tener una sintaxis concreta y una sintaxis abstracta, siendo esta última su metamodelo. Para la definición de sintaxis concretas [Brambilla *et al.*, 2012] existen dos alternativas claramente diferenciadas:

- Definir un nuevo lenguaje específico de dominio, desde la nada, al que podemos aportar toda la expresividad y semántica propia del mismo; o bien,
- Extender UML con nuevos conceptos o modificando alguno de ellos, aprovechando de esta forma su semántica. El mecanismo de extensión que el estándar ofrece se denomina *perfil UML* [Fontoura *et al.*, 2000], y esta opción nos ofrece una doble ventaja: por un lado, UML se ha convertido en un estándar de facto en los últimos años y se encuentra ampliamente extendido en la industria del software, lo que facilitaría su adopción, y por otro, se dispondrían de las herramientas compatibles con UML, lo que facilitaría su aceptación en el mundo empresarial.

Tal y como se establece en la especificación de UML, la enunciación de un perfil pasa por la definición personalizada de tres componentes:

1. Estereotipos (*Stereotype*), que permiten definir elementos de dominio específico, es decir, los elementos del metamodelo. Una vez estereotipados es necesario establecer las metaclases UML de las que extienden.
2. Restricciones (*Constraints*), mecanismo que permite establecer condiciones sobre los elementos estereotipados del metamodelo que nos permitan verificar un modelo bien formado.

3. Valores etiquetados (*Tagged value*), que nos permiten añadir propiedades adicionales y asociarlas a un elemento estereotipado del metamodelo.

En nuestro caso, dado que MONETA se desarrolla sobre Enterprise Architect, y ésta dispone de la funcionalidad de definir nuevos perfiles UML, vamos a ver a continuación cómo la definición de esos nuevos perfiles son claves en nuestra herramienta.

7.2. MONETA: la herramienta de soporte para el marco de referencia

Una vez determinado el entorno de trabajo en el que se desarrolla MONETA, que como ya hemos comentado está basado en Enterprise Architect, y su funcionalidad de definición de perfiles, en esta sección se va a entrar en detalle en dicha herramienta, considerando sus capacidades y detallando cómo cada una de las tres piezas claves del marco de referencia se encuentran contenidas en MONETA.

7.2.1. El lenguaje de modelado de procesos de software INROMA en MONETA

Un primer paso en el desarrollo de MONETA como herramienta de soporte del marco de referencia era la introducción de la primera de las piezas clave sobre las que nuestro marco se sustenta: el lenguaje INROMA. Tal y como hemos visto en la sección anterior, la incorporación de INROMA en MONETA se realiza por medio de la definición de un perfil de UML, el cuál se observa en la figura 7.1.

En la tabla 7.1 se muestran los estereotipos del perfil UML de INROMA, así como la metaclase que se ha utilizado para realizarlo, y se justifica la elección de la metaclase correspondiente a cada estereotipo.

Con la definición del perfil de INROMA tal y como aquí se ha descrito, podemos utilizar INROMA como lenguaje de modelado de procesos de software en MONETA. Aunque el objetivo para el que fue elaborado era el de servir como lenguaje base en el marco de referencia, ya hemos dicho anteriormente que, también es un lenguaje en sí mismo y como tal puede ser usado. Veamos en la figura 7.2 una captura de pantalla donde se puede observar cómo modelar un proceso de software con INROMA, donde se puede observar la sintaxis concreta del lenguaje en el editor de la propia herramienta.

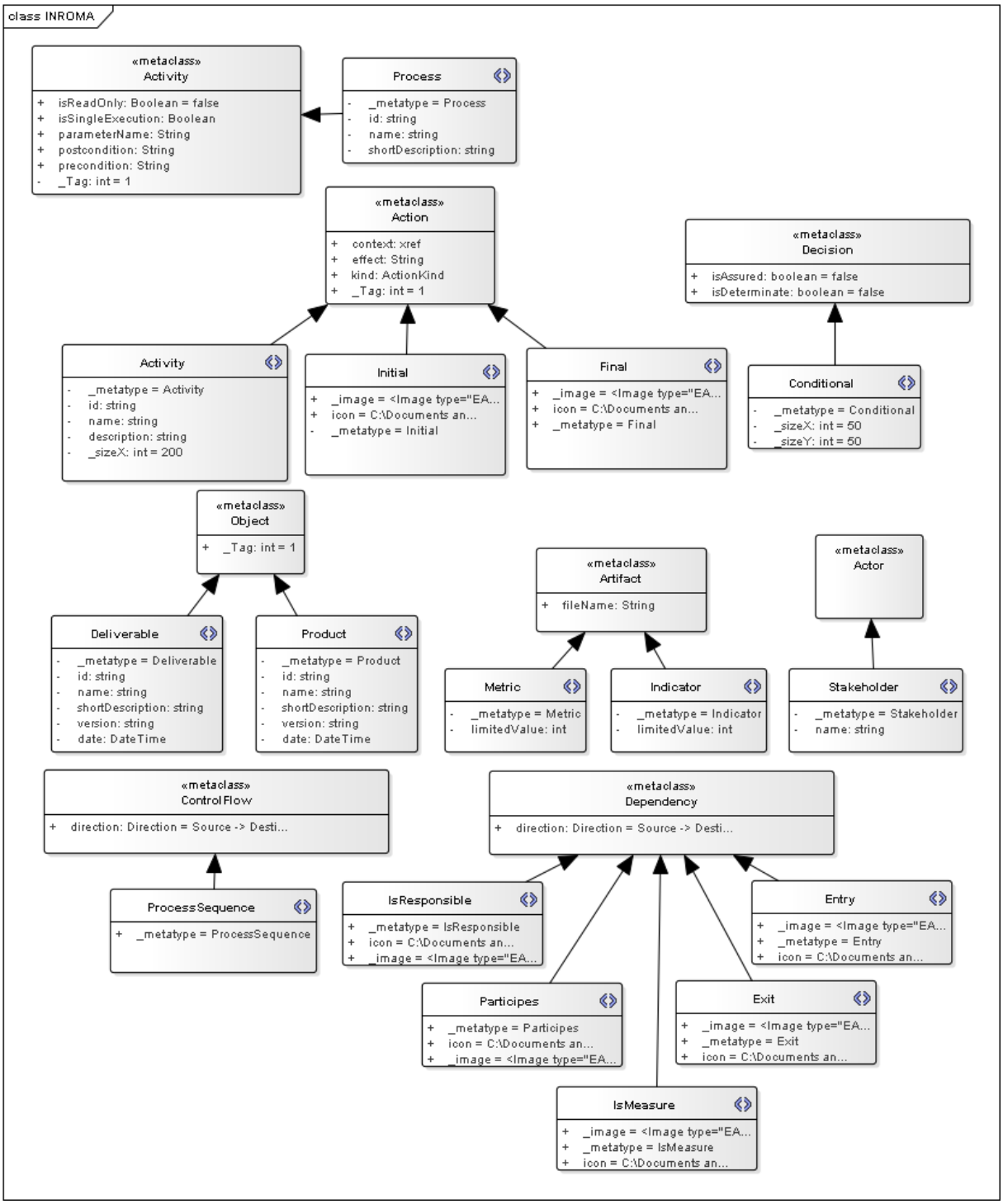


Figura 7.1: Perfil UML para INROMA

Metaclase UML	Estereotipo	Justificación
Activity	Process	La metaclase *Activity* representa la secuencia coordinada de acciones, y por ellos ha sido elegida para ser extendida con Process.
Action	Activity	La metaclase *Action* representa un simple paso dentro de una *Activity*, lo que en INROMA se ha denominado *Activity*, aunque puede llevar a confusión.
	Initial	
	Final	
Decision	Conditional	Este elemento se corresponde a un *ControlNode* de UML, más concretamente los que hacen referencia a *DecisionNode*, que en EA está definido como *Decision*.
Object	Deliverable	Esta meclase de EA hace referencia a la metaclase *ObjectNode* de UML, que se define como un *ActivityNode* que indica una instancia de un estado de un clasificador concreto, y por eso aquí se ha elegido para representar los *Products* de INROMA.
	Product	
Artifact	Metric	La metaclase *Artifact* es la especificación de una pieza física de información que se usa o produce en el desarrollo de software, y se define como una especialización de *Classifer*, por lo que se ha elegido como elemento del que extender estos estereotipos.
	Indicator	
Actor	Stakeholder	La metaclase *Actor* de UML especifica cualquier ente (humano o no) que interactua con el sistema, por lo que representa fielmente lo que pretendemos con *Stakeholder*.
ControlFlow	ProcessSequence	UML modela el flujo de ejecución mediante *ActivityEdges* y, por tanto, sus especializaciones. Una de ellas es precisamente *ControlFlow*, que es la que se define para una secuencia de *Actions*, por lo tanto es la metaclase de la que extienden los elementos de secuencia de INROMA.
Dependency	isResponsible	Se ha optado por la metaclase *Dependency* para extender estos esterotipos, como una forma de establecer la relación semántica que hay entre ellos.
	participes	
	entry	
	exit	
	isMeasure	

Tabla 7.1: Justificación de las metaclases en el perfil UML de INROMA

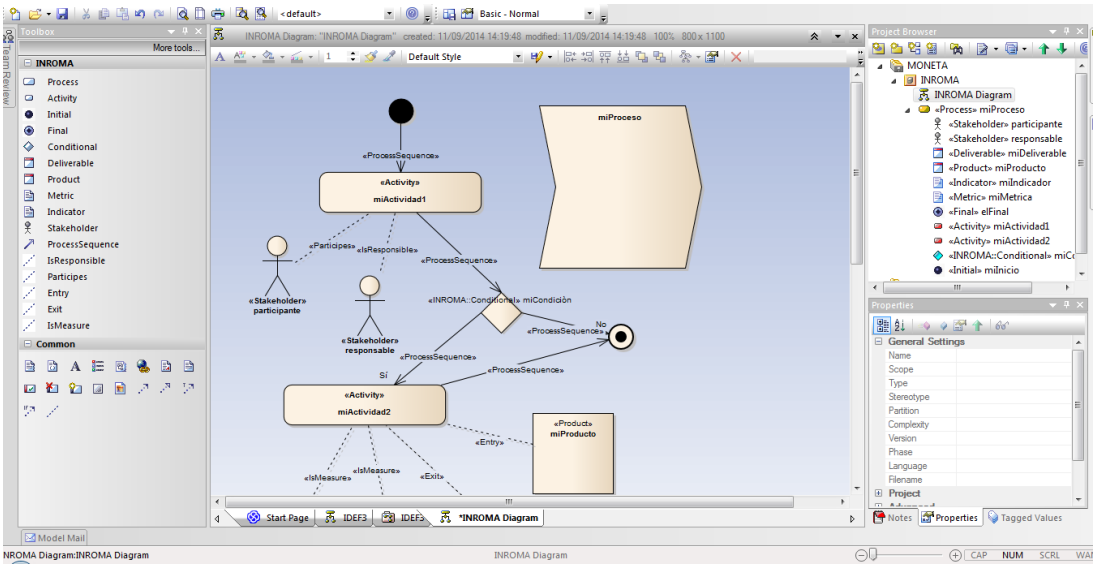


Figura 7.2: Editor de INROMA en MONETA

7.2.2. El método del marco de referencia en MONETA

Con la incorporación de INROMA en MONETA ya tenemos el primero de los fundamentos del marco de referencia incluido en la herramienta. Pasemos, pues, a incorporar la siguiente pieza clave, el método para incluir nuevos lenguajes de modelado de procesos de software.

Ya hemos mencionado en el capítulo 5 que para poder incorporar un lenguaje de modelado de procesos de software es necesario poder configurar ese lenguaje con una morfología compatible, es decir, debería disponer, o ser posible definir, el metamodelo que constituye la sintaxis abstracta de dicho lenguaje. También se había comentado que la sintaxis concreta es importante pero en nuestro contexto teórico no era absolutamente necesaria. Sin embargo a la hora de desarrollar MONETA se ha visto la necesidad de disponer para todo lenguaje de modelado de procesos de software que queramos incorporar tanto de su sintaxis abstracta, es decir, el metamodelo, como de su sintaxis concreta.

En la implementación de MONETA se estableció la necesidad de incorporar al menos unos ejemplos de lenguajes de modelado de procesos de software para poder demostrar el soporte al marco de referencia. Como no podría haber sido de otra forma, dado que durante el desarrollo de la parte teórica del marco hemos utilizado como ejemplos de lenguajes modelado de procesos de software UML-DA 2.5, BPMN 2.0 e IDEF3, éstos fueron los lenguajes que se decidió incluir en MONETA. Al tomar esta decisión comprobamos que tanto BPMN como UML, en sus diferentes técnicas y diagramas, ya se encontraban incluidos en la distribución comercial de Enterprise Architect, con lo que estos lenguajes venían por defecto accesibles desde MONETA. No era éste el caso de IDEF3, para el que tuvo que desarrollarse un perfil UML de forma similar a cómo se desarrolló el de INROMA. En la figura 7.3 se observa dicho perfil.

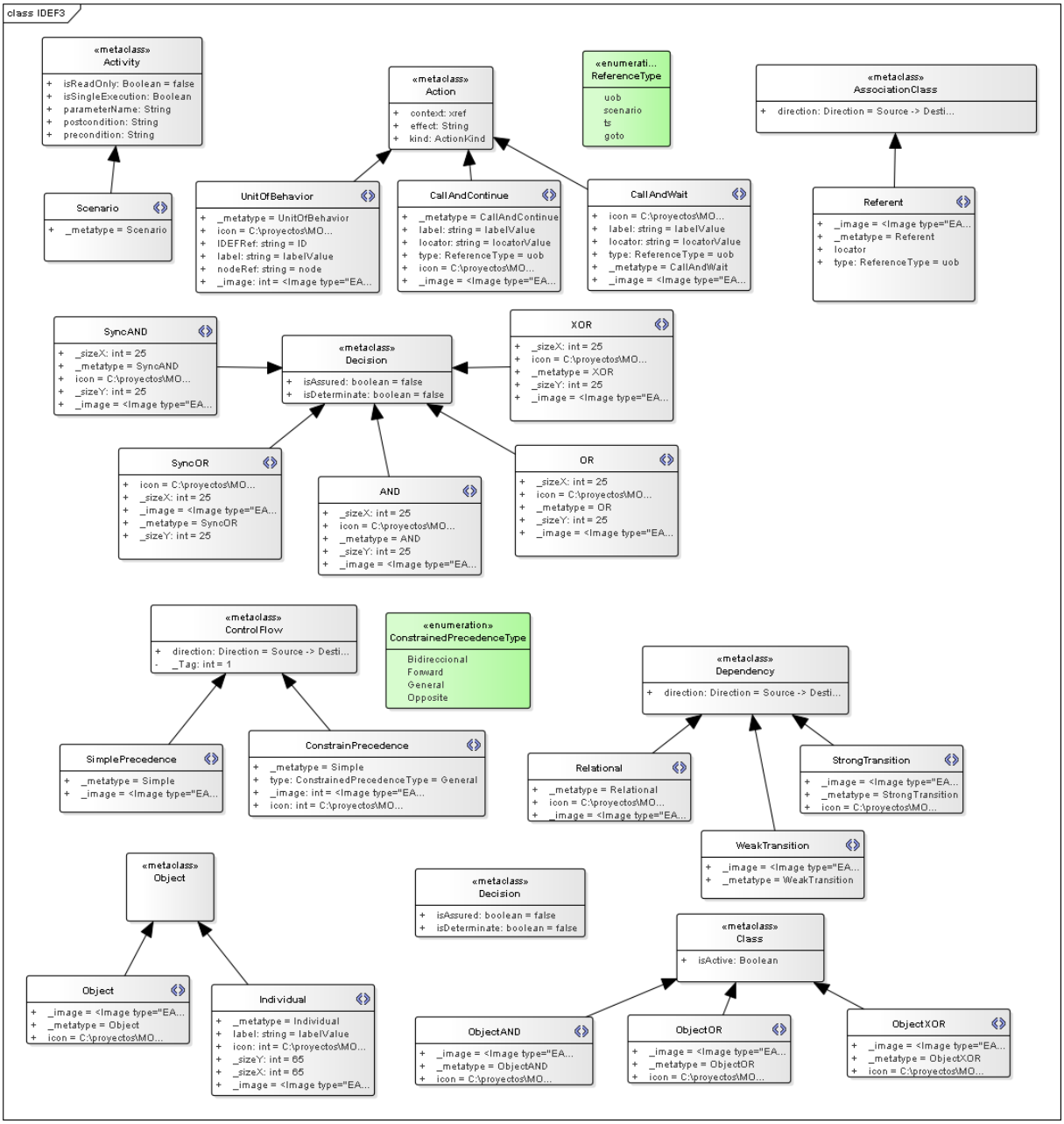


Figura 7.3: Perfil UML de IDEF3

Como en el caso de INROMA, es necesario justificar la elección de la metaclase correspondiente a cada uno de los estereotipos del perfil UML definidos para IDEF3. En las tablas 7.2 y 7.3 se muestran de forma resumida los estereotipos, las metaclases correspondientes y la justificación de su elección.

Metaclase UML	**Estereotipo**	**Justificación**
Activity	Scenario	*Scenario* corresponde en IDEF3 al concepto de *Process*, por eso se ha optado por extenderlo de *Activity* ya que se especifica como una secuencia de acciones y el control de su flujo.
Action	UnitOfBehavior	Estos elementos corresponden a los pasos individuales que componen un *Scenario*. Por ello, para este estereotipo utilizamos la metaclase *Action* como subclase de *ActivityNode*.
	CallAndContinue	
	CallAndWait	
Decision	SyncAND	Todos estos elementos se corresponden a un *ControlNode* de UML, mas concretamente los que hacen referencia a *DecisionNode*, que en EA está definido como *Decision*.
	SyncOR	
	AND	
	OR	
	XOR	
AssociationClass	Referent	La metaclase *AssociationClass* es una clase que es parte de la relación de asociación que existen entre otras dos clases, algo que ocurre con *Referent*, puesto que representa la llamada a otra clase desde la transición de un objeto a otro.
ControlFlow	SimplePrecedence	UML modela el flujo de ejecución mediante *ActivityEdges* y, por tanto, sus especializaciones. Una de ellas es precisamente *ControlFlow*, que es la que se define para una secuencia de *Actions*, por lo tanto es la metaclase de la que extienden los elementos de secuencia de IDEF3.
	ConstraintPrecedence	
Dependency	Relational	La metaclase *Dependency* establece una relación semántica entre los objetos enlazados de tal forma que el origen no está completo sin el destino, y esta es la propiedad con la que hemos querido dotar a estos elementos.
	WeakTransition	
	StrongTransition	

Tabla 7.2: Justificación de las metaclases en el perfil UML de IDEF3

Por último, fue necesario introducir también en MONETA la sintaxis concreta correspon-

Metaclase UML	Estereotipo	Justificación
Object	Object Individual	Esta meclase de EA hace referencia a la metaclase *ObjectNode* de UML, que se define como un *ActivityNode* que indica una instancia de un estado de un clasificador concreto, y por eso aquí se ha elegido para representar los *Objects* de IDEF3.
Decision	ObjectAND ObjectOR ObjectXOR	De nuevo, esta metaclase corresponde a *DecisionNode*, y la justificación es la misma que en el caso anterior, aunque hemos querido destacar que se encuentran en diagramas diferentes, en este caso en los diagramas de *Objects*, por eso los hemos separado de los anteriores.

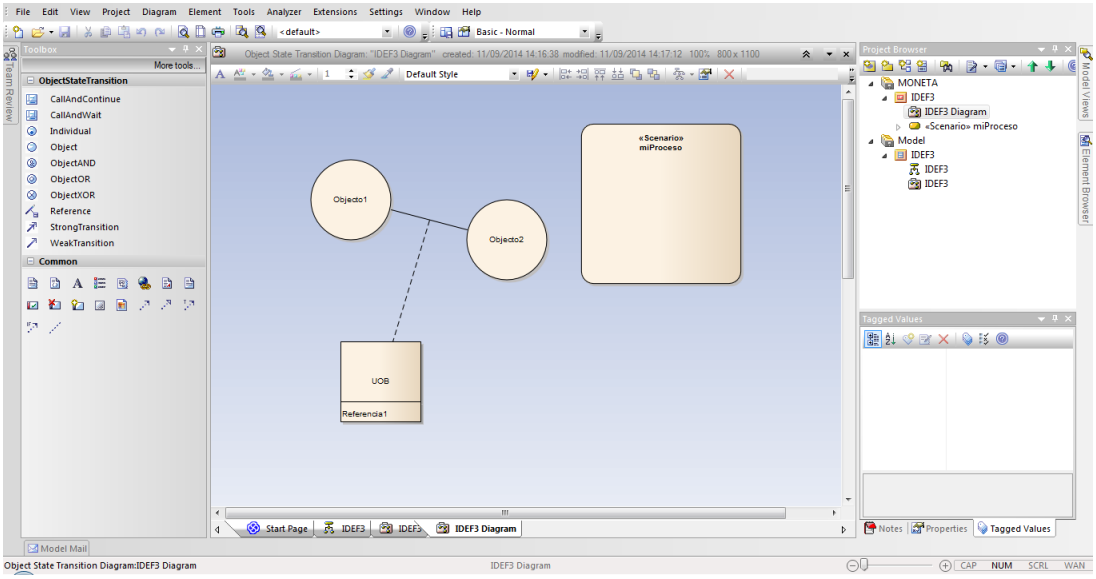
Tabla 7.3: Justificación de las metaclases en el perfil UML de IDEF3

diente al lenguaje IDEF3 para poder utilizarla a la hora de modelar procesos de software en ese lenguaje. En la figura 7.4 se muestra cómo se pueden utilizar IDEF3 dentro de MONETA, y se puede observar cómo la sintaxis concreta IDEF3 ha sido incorporada dentro de la herramienta y es accesible desde el editor de componentes para los modelos.

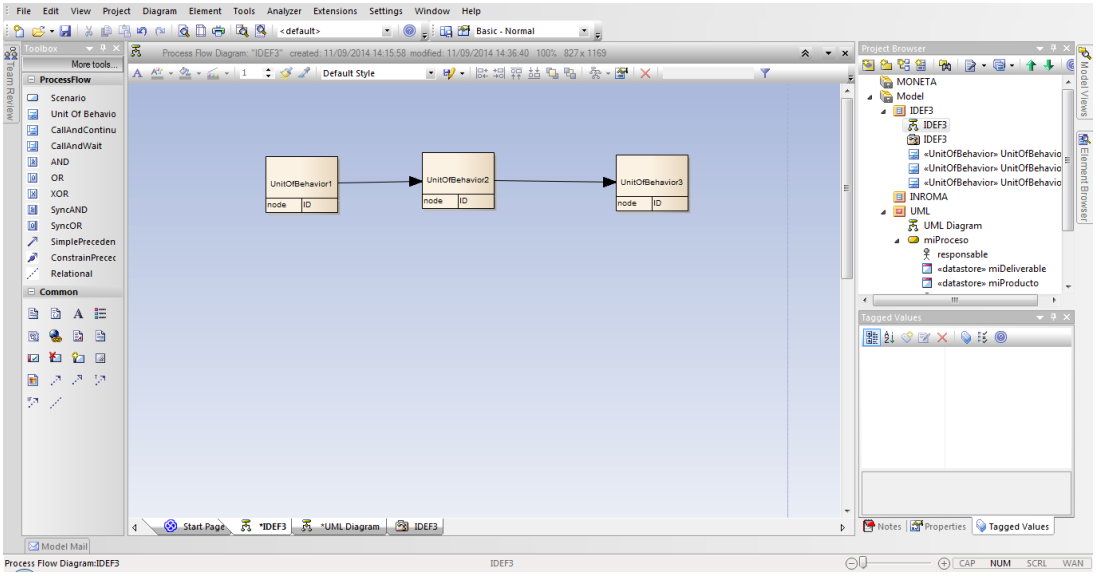
Una vez disponibles los lenguajes de modelado de procesos de software en MONETA, el siguiente paso del método consistía en establecer las correspondencias. Hoy por hoy, dichas correspondencias se realizan de forma manual, con lo que no es posible ver su reflejo en la herramienta. Sin embargo, no hay que olvidar que es un paso necesario más para la posibilidad de utilizar MONETA como soporte al marco de referencia, por lo que aunque no se disponga de automatización es algo absolutamente necesario. Como las correspondencias ya las vimos en el capítulo 5 no las repetiremos de nuevo y nos remitimos a dicho capítulo en el caso de que deseemos su revisión.

7.2.3. Las transformaciones en MONETA

Para completar MONETA como herramienta de soporte al marco de referencia, el último elemento clave que nos queda por incorporar son las transformaciones. Con ellas la tercera de las piezas clave habrá sido implementada. Como se ha visto en el capítulo 6 las transformaciones se han definido en QVT Relations. Actualmente este estándar no se encuentra soportado por Enterprise Architect, ni para el modelado de las transformaciones ni, obviamente, como motor de ejecución de las mismas.



(a) Vista Objetos



(b) Vista Procesos

Figura 7.4: Editor para IDEF3 en MONETA

Con el objetivo de facilitar a los usuarios el modelado de las transformaciones con QVT Relations se desarrolló, tal y como habíamos hecho con INROMA e IDEF3, un perfil UML para este estándar, que se encuentra recogido en la figura 7.5.

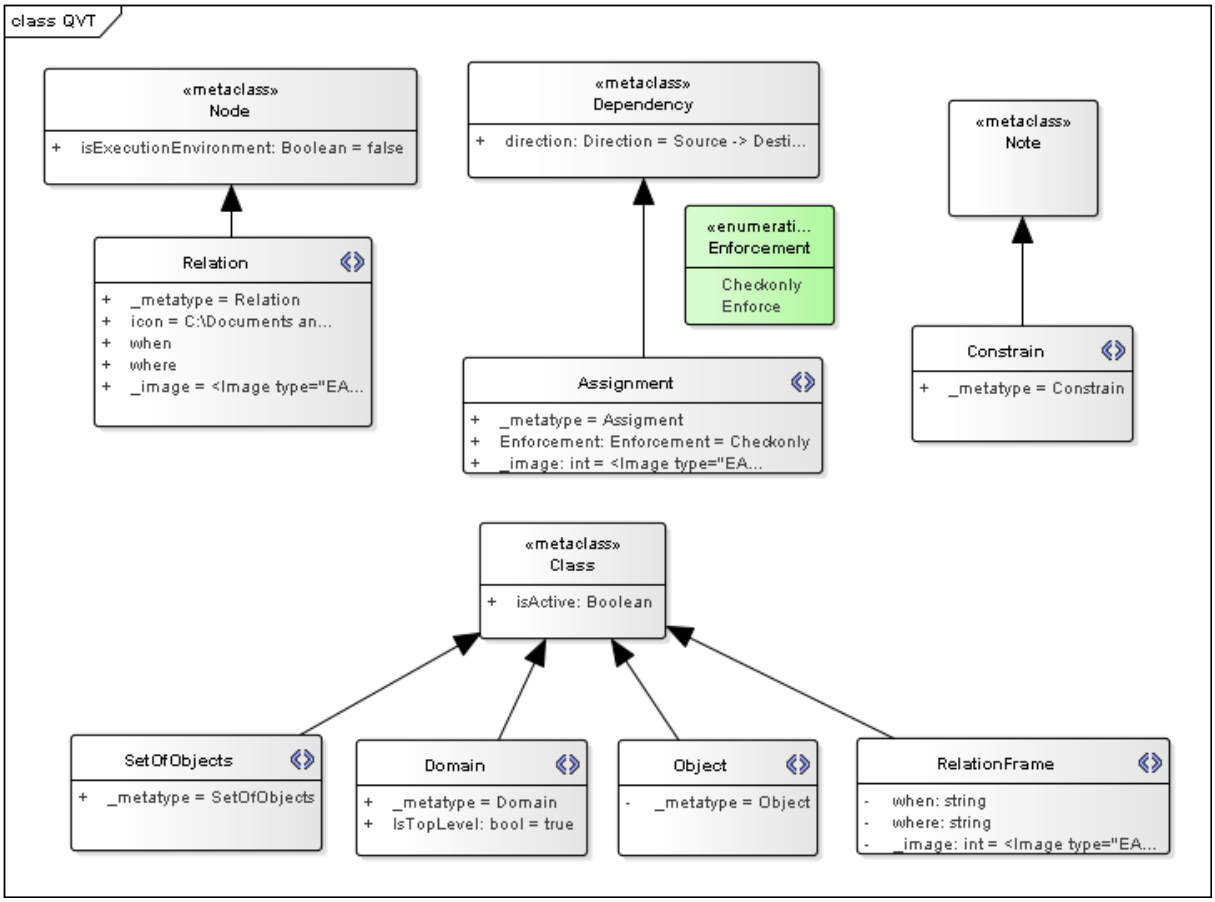


Figura 7.5: Perfil UML de QVT Relations

Como en los casos anteriores, justificamos la elección de cada una de las metaclases correspondientes con los estereotipos del perfil. En la tabla 7.4 se muestran de forma resumida los estereotipos, las metaclase correspondientes y la justificación de su elección.

Por último, fue necesario introducir también en MONETA la sintaxis concreta correspondiente a QVT Relations para poder utilizarla a la hora de modelar las transformaciones. En la figura 7.6 se muestra cómo se pueden utilizar QVT Relations dentro de MONETA al haber sido incorporada dicha sintaxis en el editor de componentes.

Metaclase UML	Estereotipo	Justificación
Node	Relation	La metaclase *Node* hace referencia a un entorno de ejecución y permite establecer el contexto de la tansformación entre los diferentes participantes de la misma.
Class	SetOfObjects Domain Object RelationFrame	Cada uno de los elementos que participan en una transformación se han considerado un estereotipo de la metaclase *Class*, puesto que es la más general para poder representar meras entidades que participan en la relación.
Note	Constraint	La metaclase *Note* nos permite dejar constancia de las restricciones que aparecen en una relación QVT Relations y, por tanto, también en su representación gráfica.
Dependency	Assignment	La metaclase *Dependency* establece una relación semántica entre los objetos enlazados. En este caso permite establecer la relación entre cada uno de los elementos de la transformación y el contexto de la misma.

Tabla 7.4: Justificación de las metaclases en el perfil UML de QVT-Relations

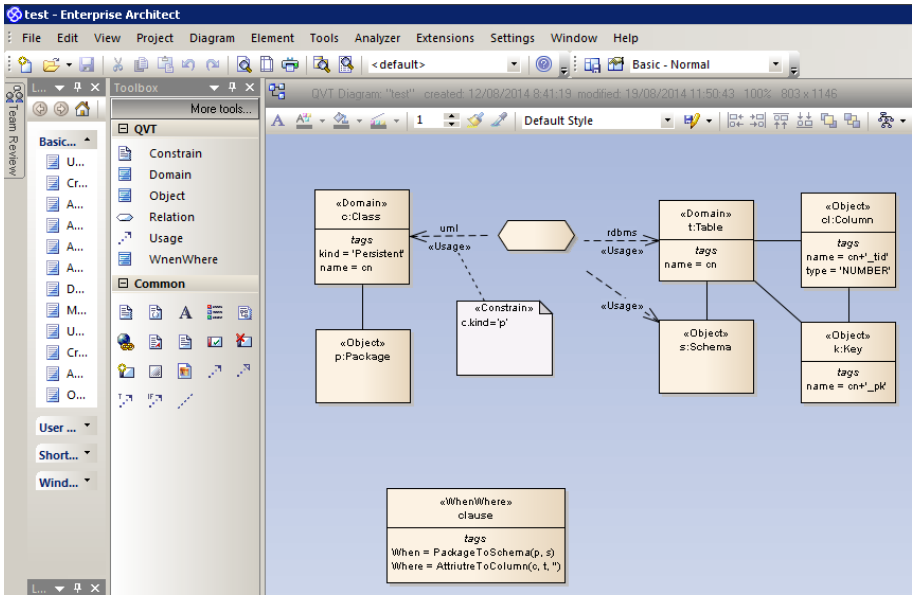


Figura 7.6: Caja de herramientas para el perfil de QVT Relations

Una vez incorporada a MONETA la capacidad de modelar las transformaciones, lo que restaba por implementar era la ejecución de las mismas.

Hemos considerado fuera del alcance de este trabajo de tesis la implementación completa de un motor de ejecución de transformaciones de QVT Relations, así que se optó para la utilización del mecanismo de extensión de componentes que ofrece el propio Enterprise Architect, denominado *Add-In*, para desarrollar en el lenguaje C# la implementación de las transformaciones.

Así, una vez modelada una transformación con la sintaxis concreta de QVT Relations, será necesario incorporar con *Add-Ins* el código fuente equivalente. En los algoritmos 7.1 y 7.2 se observa un ejemplo de esta equivalencia de las transformaciones definidas en QVT e implementadas en MONETA.

```
1   top relation Process2Process
2   {
3       _processName : String;
4       checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
5       {
6           name = _processName
7       };
8       enforce domain inromamodel inromaProcess : inroma::Process
9       {
10          name = _processName
11      };
12      where {
13          MapActivities (bpmnProcess, inromaProcess);
14          MapInitial (bpmnProcess, inromaProcess);
15          MapFinal (bpmnProcess, inromaProcess);
16          MapConditional (bpmnProcess, inromaProcess);
17          -- No existen Indicators en BPMN, no es posible realizar el mapeo
18          -- MapIndicators (bpmnProcess, inromaProcess);
19      }
20  }
```

Algoritmo 7.1: Relación principal para la transformación BPMN2INROMA

```
1       void Process2Process(EA.Element bpmnProcess, EA.Element inromaProcess)
2       {
3           if (inromaProcess == null)
4               inromaProcess = addNewProcess(bpmnProcess.Name);
5
6           if (inromaProcess == null)
7               return;
8
9           MapActivities(bpmnProcess, inromaProcess);
10          MapInitial(bpmnProcess, inromaProcess);
11          MapFinal(bpmnProcess, inromaProcess);
12          MapConditional(bpmnProcess, inromaProcess);
13      }
```

Algoritmo 7.2: Implementación de la transformación principal BPMN2INROMA en C#

7.3. MONETA en uso: la validación de las situaciones reales

Una vez expuesto en profundidad la forma en la que en MONETA se implementan las tres piezas claves que conforman el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los modelos de procesos de software, es el momento de abordar la validación de la herramienta, para lo que utilizaremos las situaciones reales que han justificado el desarrollo de este trabajo de tesis y que fueron descritas en el capítulo 3. Para ello, en los siguientes apartados haremos uso de MONETA para resolver una por una las situaciones reales planteadas. En esta sección no se va a entrar en el detalle del uso de MONETA, para lo cual en el anexo D se encuentra recogido el manual de usuario de la herramienta.

7.3.1. Migración de un lenguaje A a un lenguaje B

La primera de las situaciones reales que se planteó fue la migración de procesos de software. Pensemos en una organización que tiene los procesos de software modelados con un lenguaje La. Con el tiempo, por diversas razones, la organización decide migrar esos modelos de procesos de software de ese lenguaje La a otro lenguaje Lb. El objetivo a la hora de abordar este cambio sería conseguir tener una mínima pérdida de información derivada de la migración entre los lenguajes, y que la tarea en sí fuera lo más independiente posible de las personas que la llevaran a cabo, evitando las posibles inconsistencias derivadas de interpretaciones diferentes. Veamos entonces cómo el marco de referencia y MONETA dan soporte a este caso de uso.

Como ya hemos dicho en varias ocasiones, para poder trabajar con MONETA en un lenguaje de modelado de procesos de software concreto, éste debe haber sido incorporado a la herramienta. Por ello y dado que, como se ha visto anteriormente, los lenguajes BPMN, diagramas de actividad de UML e IDEF3 se encuentran ya incluidos, en lugar de utilizar lenguajes genéricos en la visualización de las situaciones reales que constituyen nuestros ejemplos, haremos uso de estos lenguajes.

En este primer caso, vamos a considerar que el lenguaje La es BPMN y el lenguaje Lb son los diagramas de actividad de UML, UML-DA. Inicialmente, el proceso de software se encuentra modelado en BPMN, tal y como se muestra en la figura 7.7.

MONETA, como herramienta de soporte del marco de referencia, nos permite realizar las transformaciones necesarias para obtener un modelo de proceso de software equivalente en otro lenguaje que también este disponible. En este caso, hemos seleccionado el lenguaje UML-DA. En esta migración se ha procedido a la transformación de BPMN a INROMA, para posteriormente transformar de INROMA a UML-DA, algo que ocurre de forma completamente transparente al usuario. El resultado se encuentra reflejado en la figura 7.8.

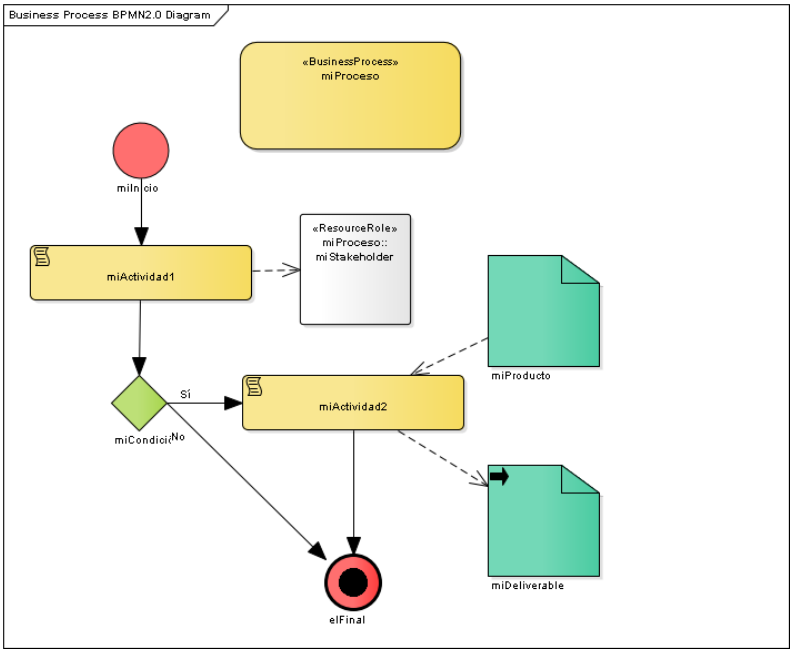


Figura 7.7: Situación real 1: Modelo de proceso de software en BPMN

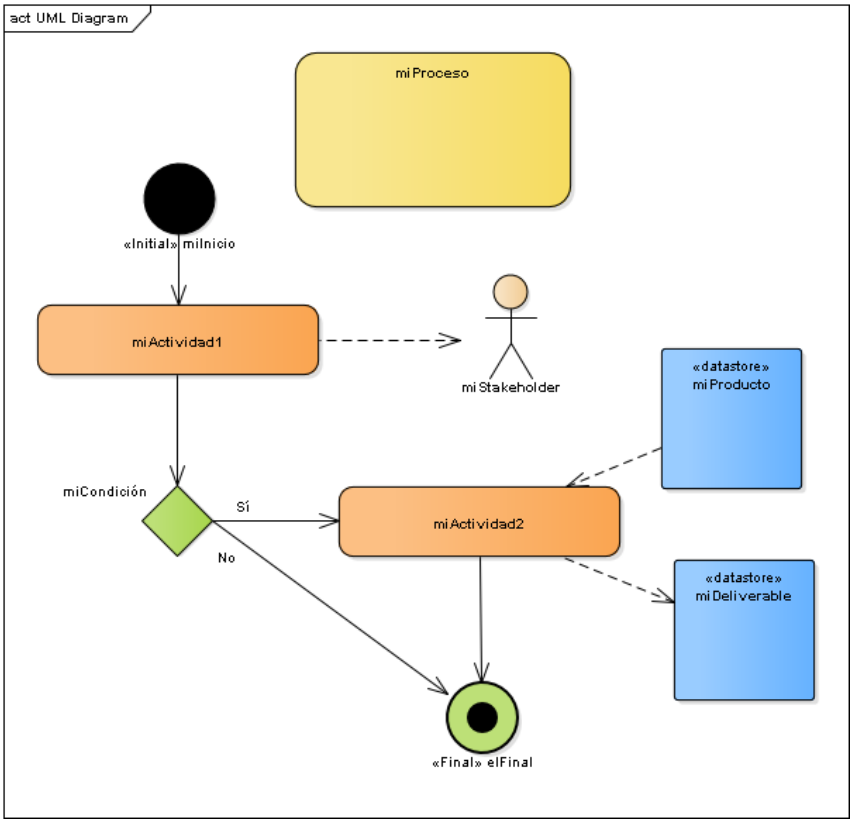


Figura 7.8: Situación real 1: Modelo de proceso de software en UML-DA obtenido con MONETA

A pesar de que el paso a través de INROMA es transparente al usuario, hemos considerado mostrar el proceso intermedio obtenido en INROMA para ilustrar en mayor medida el funcionamiento de MONETA. Dicho proceso se muestra en la figura 7.9.

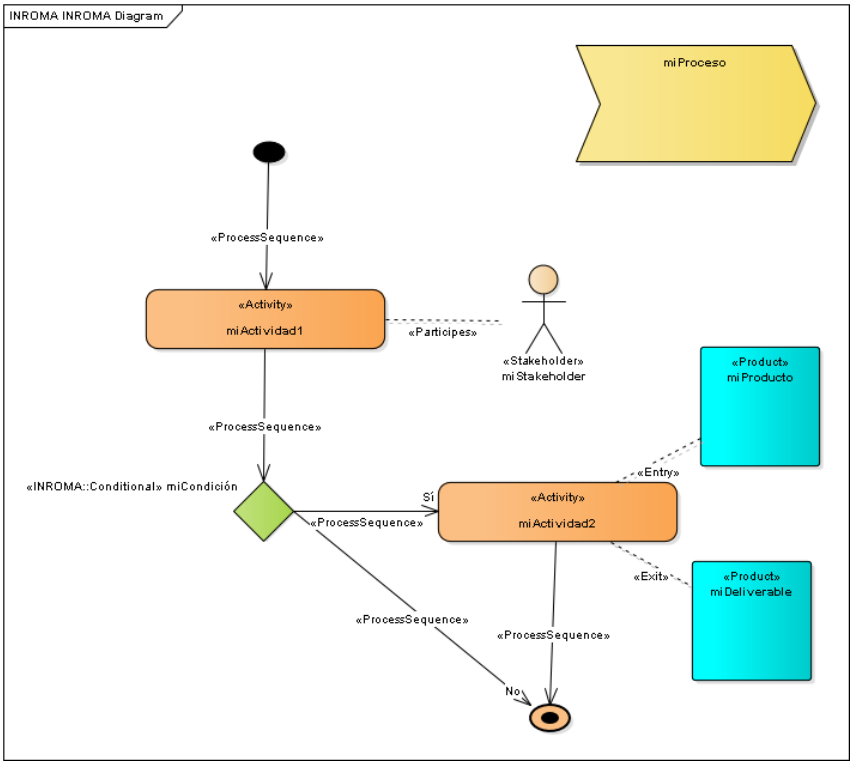


Figura 7.9: Situación real 1: Modelo de proceso de software en INROMA obtenido con MONETA. Se trata de un modelo intermedio y transparente al usuario

En resumen, como se ha podido observar esta migración se ha realizado de forma prácticamente automática, puesto que lo único que se necesita de partida es el modelo de proceso de software inicial. Realizando con MONETA la migración se garantiza la ausencia de inconsistencias entre ambos modelos, así como la falta de pérdida de información en dicho cambio, objetivos ambos planteados en el enunciado de la primera de las situaciones reales que estamos utilizando para la validación de MONETA.

7.3.2. Interoperabilidad de procesos de software modelados en lenguajes diferentes

La segunda de las situaciones reales planteadas constituye un problema típico de interoperabilidad de modelos de procesos de software. Supongamos que una organización dispone de dos departamentos en los que se trabaja con lenguajes de modelado de procesos de software diferentes y, en algún momento estos procesos necesitan interoperar. Como hemos referido en varias ocasiones anteriormente, es necesario que los lenguajes con los que vamos a trabajar en MONETA hayan sido previamente incorporados a la herramienta.

Consideraremos, pues, que en este caso el departamento A utiliza UML-DA para modelar sus procesos de software, tal y como se muestra en la figura 7.10.

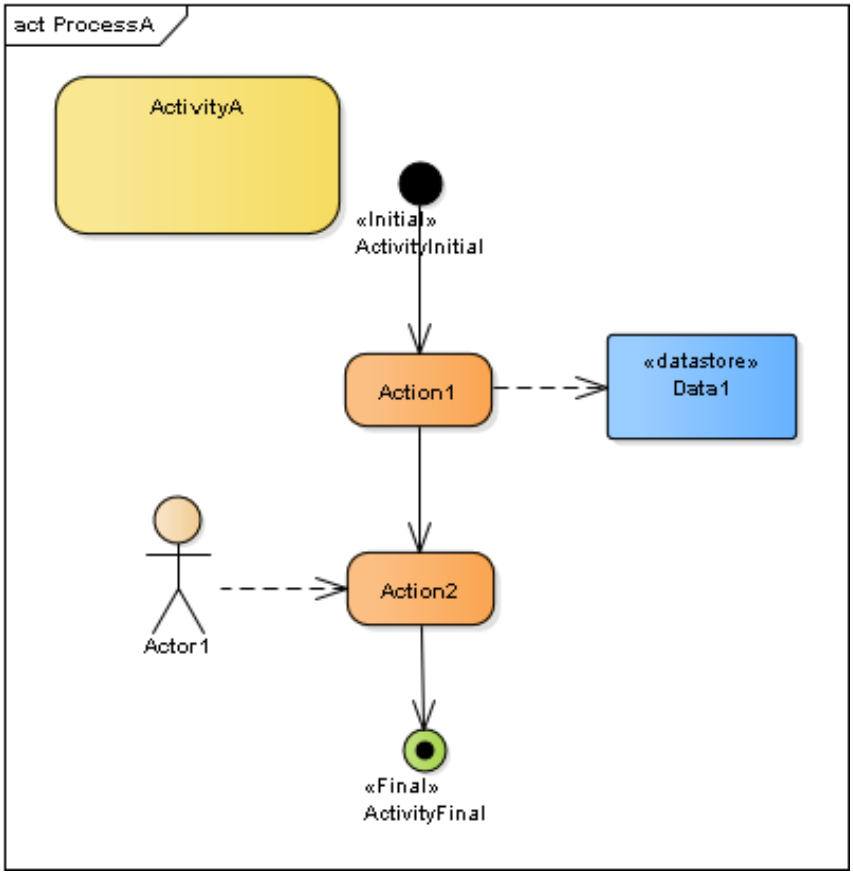


Figura 7.10: Situación real 2: Modelo de proceso de software en UML-DA del departamento A

Por otro lado, vamos a considerar que el departamento B, como se observa en la figura 7.11, utiliza BPMN para modelar sus procesos de software.

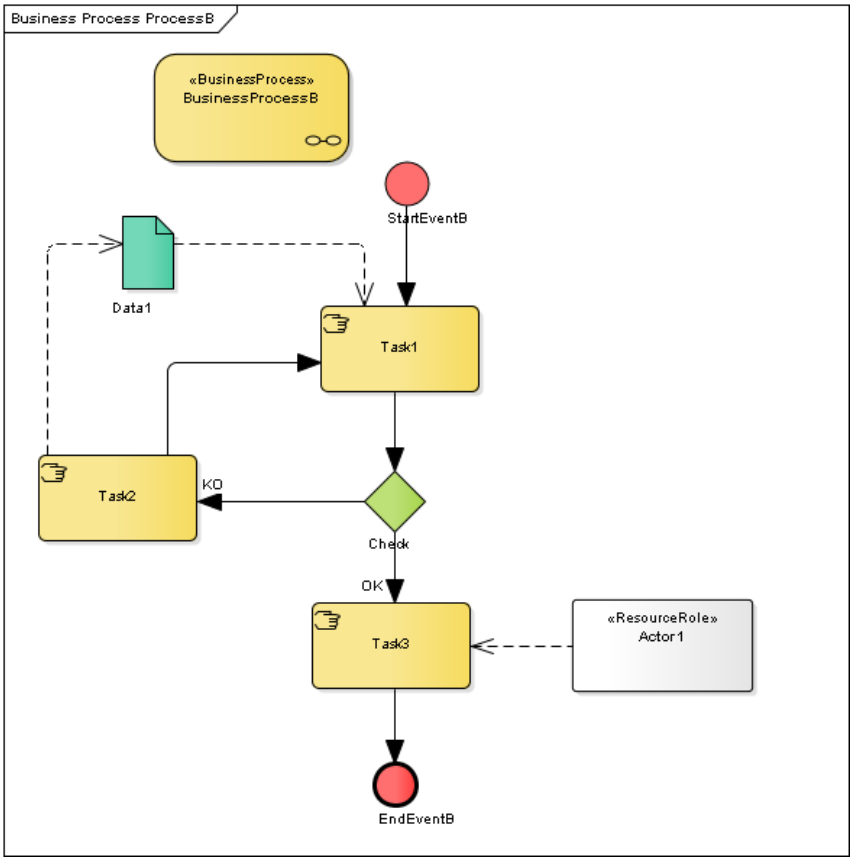


Figura 7.11: Situación real 2: Modelo de proceso de software en BPMN del departamento B

Son, por tanto, dos procesos de software diferentes, en los que resulta que la *Task2* del proceso del departamento B es el proceso del departamento A, es decir, ambos procesos interoperan entre ellos. Sin embargo, ninguno de los departamentos se plantea cambiar de lenguaje, ni hay prevalencia de ninguno de ellos frente al otro. MONETA resuelve esta situación utilizando INROMA como lenguaje común. Así, de forma transparente al usuario, convierte ambos modelos de proceso a INROMA, permitiendo trabajar sobre ellos e incorporando la información necesaria.

Como hemos hecho en el ejemplo anterior, mostramos en las figuras 7.12 y 7.13 los modelos de proceso de ambos departamentos generados por MONETA y que utiliza como lenguaje INROMA.

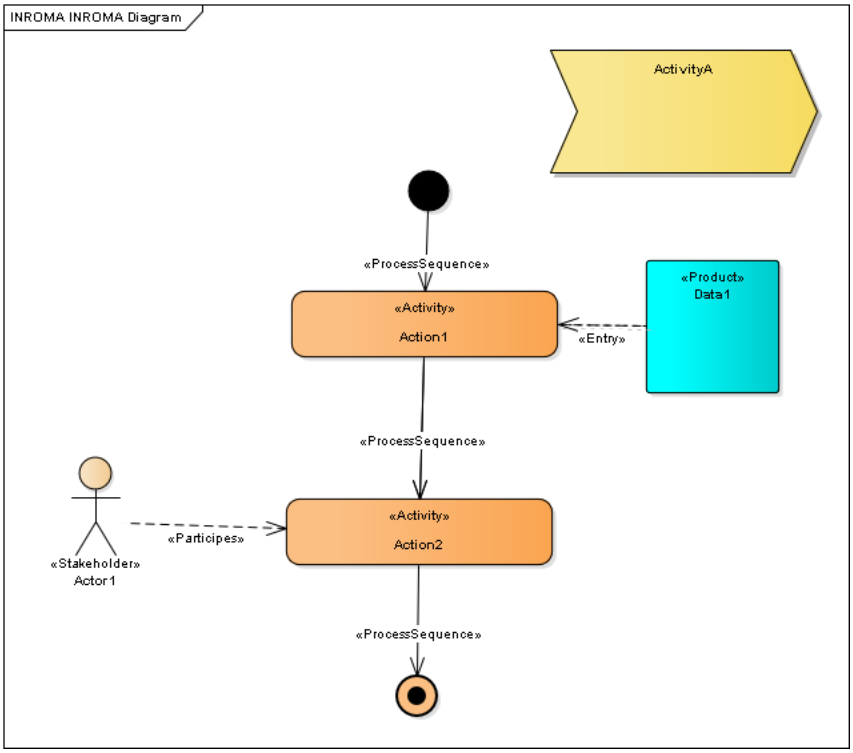


Figura 7.12: Situación real 2: Modelo de proceso de software del departamento A generado en INROMA por MONETA

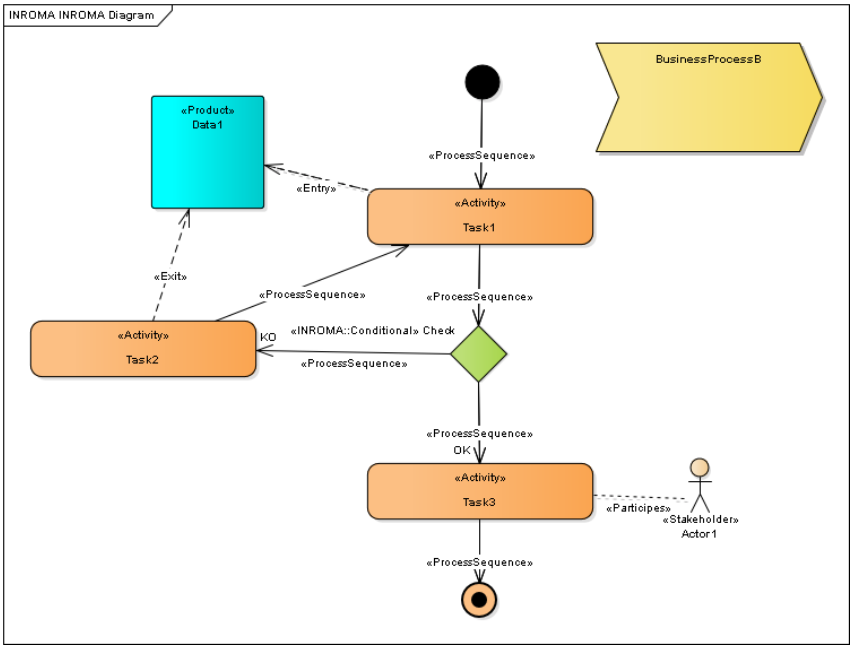


Figura 7.13: Situación real 2: Modelo de proceso de software del departamento B generado en INROMA por MONETA

El modelo de proceso resultante de la unión de ambos procesos y trabajado por parte de los ingenieros de procesos de ambos departamentos se muestra en la figura 7.14.

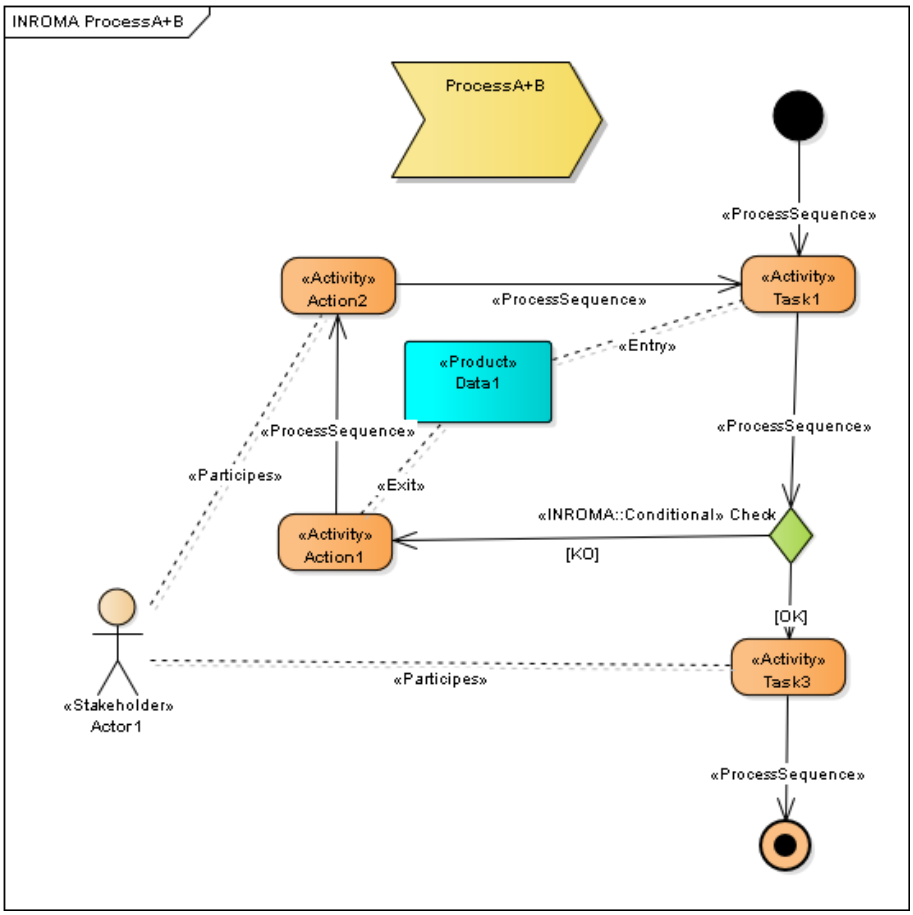


Figura 7.14: Situación real 2: Modelo de proceso de software resultante de interoperar ambos procesos en INROMA

No obstante, debido a que al enunciar esta segunda situación habíamos comentado que ninguno de los dos departamentos plantea cambiar de lenguaje, deberíamos obtener este mismo proceso en BPMN y en UML-DA, algo, de nuevo, que se obtiene automáticamente con la herramienta MONETA. Los modelos de proceso resultantes de esta acción se muestran en las figuras 7.15 y 7.16

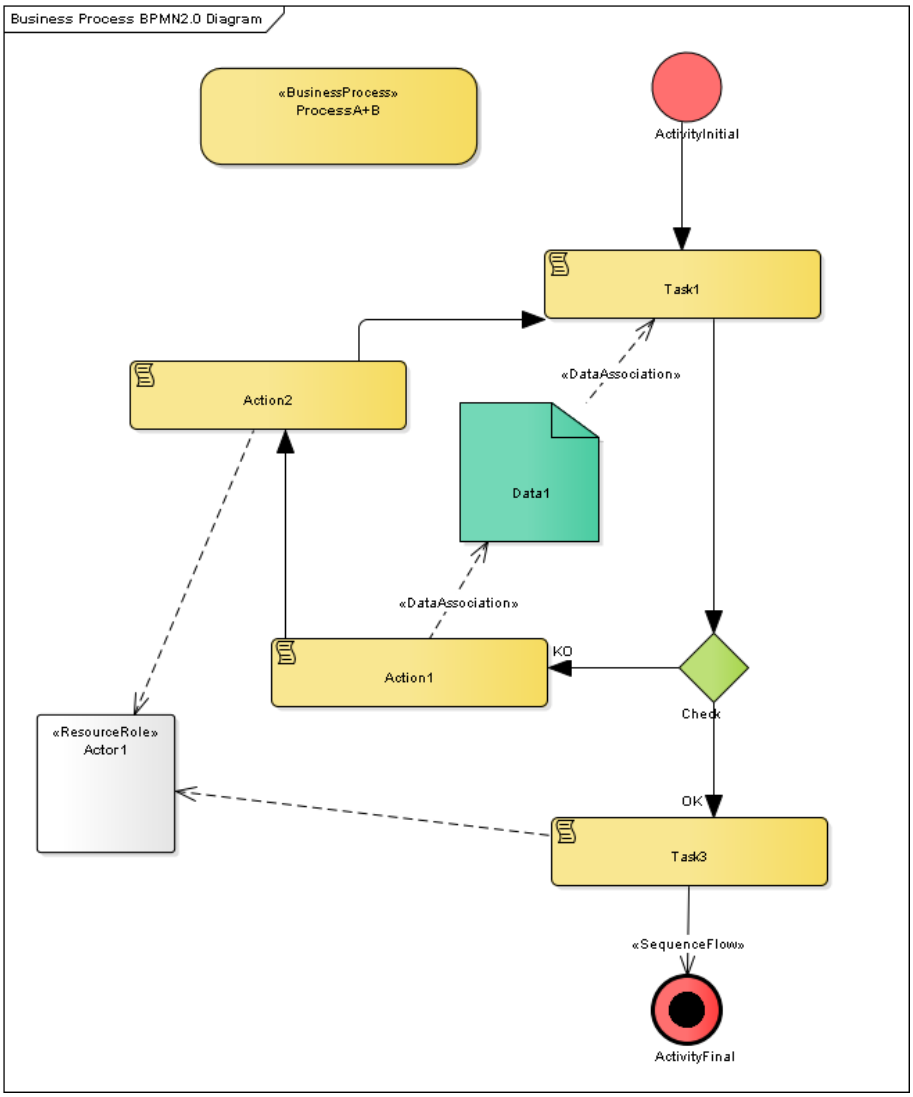


Figura 7.15: Situación real 2: Modelo de proceso de software resultante de interoperar ambos procesos en BPMN

Resumiendo el resultado de esta segunda situación real que justifica la realización de este trabajo de tesis, de nuevo se cumple el objetivo inicialmente planteado validando la herramienta ya que, gracias a MONETA, podemos interoperar entre procesos de software que se encuentran modelados con lenguajes diferentes.

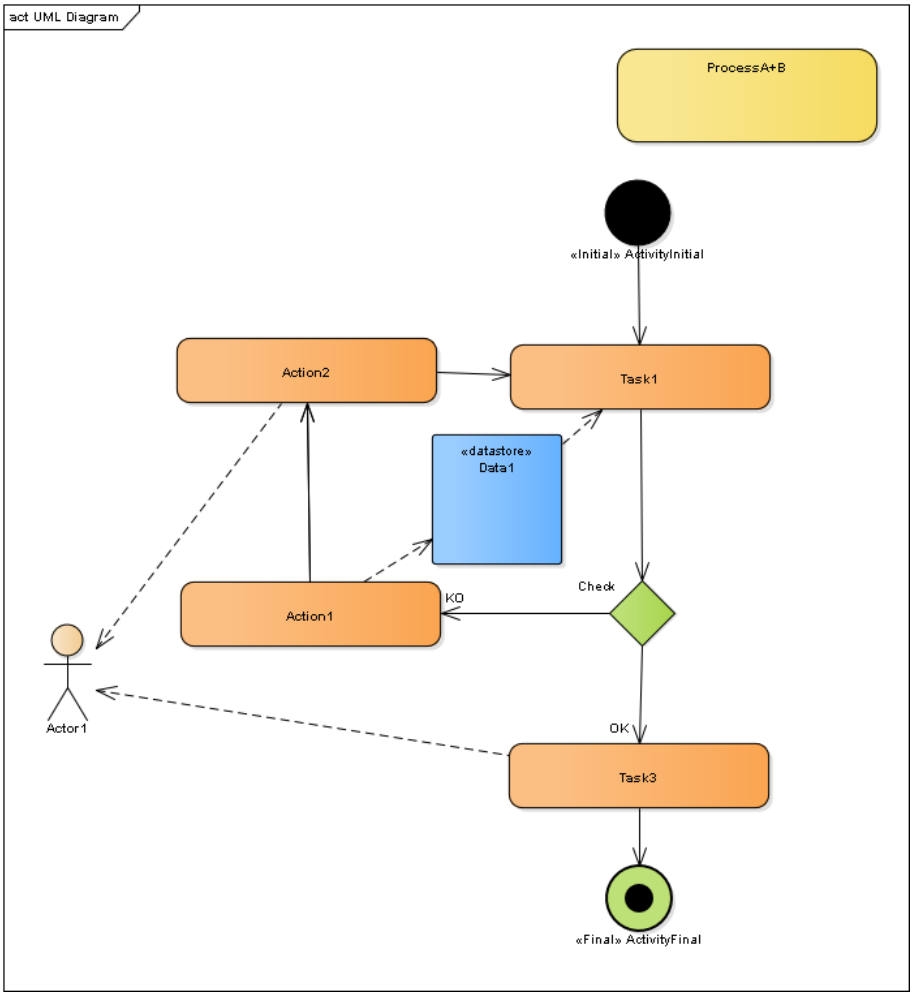


Figura 7.16: Situación real 2: Modelo de proceso de software resultante de interoperar ambos procesos en UML-DA

7.3.3. Otro caso de interoperabilidad de procesos de software

La tercera y última de las situaciones reales que justificaban la realización de este trabajo de tesis introducidas en el capítulo 3 planteaba un caso de interoperabilidad algo diferente al anterior. Supongamos, por ejemplo, tres organizaciones que deciden empezar a trabajar juntas, y se plantean modelar el proceso a seguir. Cada una de ellos utiliza un lenguaje de modelado de procesos de software diferente. Imaginemos que la empresa A utiliza UML-DA, la empresa B utiliza BPMN y la empresa C utiliza IDEF3. De nuevo queremos explicar que la elección de estos mensajes para la realización de la validación viene determinada por la necesidad de que en MONETA se hayan incorporado los lenguajes con los que trabajar.

Para poder solucionar la situación propuesta, en lugar de elegir un lenguaje de estos tres para modelar el nuevo proceso, el marco de referencia, y por ende MONETA, propone utilizar su propio lenguaje, INROMA, muy sencillo de aprender para aquellos que ya utilizan otro lenguaje y también para los que es la primera vez que se sitúan ante un lenguaje de modelado de procesos, puesto que recoge únicamente aquello que es necesario para todos los procesos de software. De esta forma, ninguna de las empresas tiene privilegios inicialmente frente al resto. El resultado es la generación de un nuevo proceso común en INROMA, proceso mostrado en la figura 7.17.

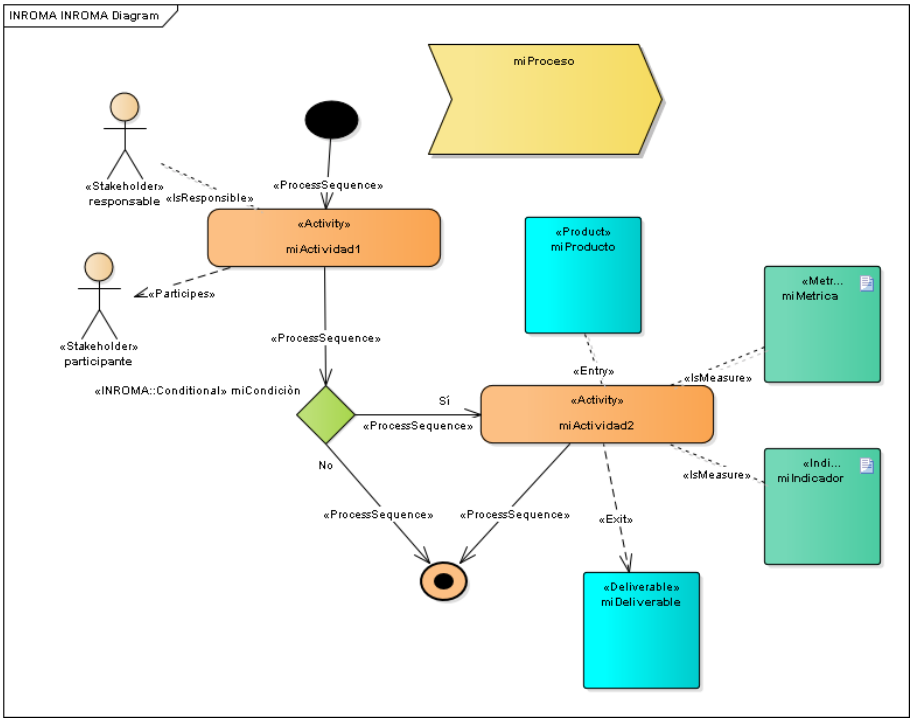


Figura 7.17: Situación real 3: Modelo de proceso de software común para las empresas A, B y C

Una vez establecido este proceso común, cada empresa en sus sedes siguen trabajando en el lenguaje elegido, por lo que se deberá realizar la transformación de este modelo INROMA a cada uno de los lenguajes de las empresas. Como ya hemos visto en las situaciones anteriores, esta funcionalidad nos la facilita MONETA. Así, una vez definido el proceso, incorporarlo al compendio de procesos de cada organización es una actividad prácticamente automática. En las figuras 7.18, 7.19 y 7.20, se observa cómo quedaría el proceso común en cada uno de los lenguajes de las organizaciones.

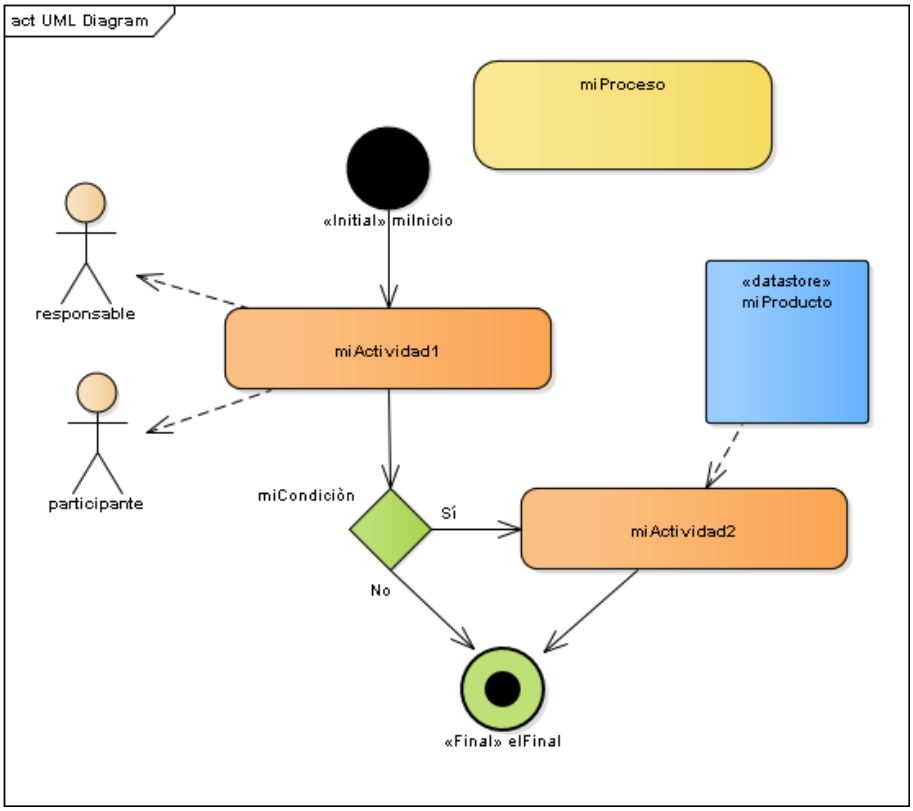


Figura 7.18: Situación real 3: Modelo de proceso de software común en UML-DA para la empresa A

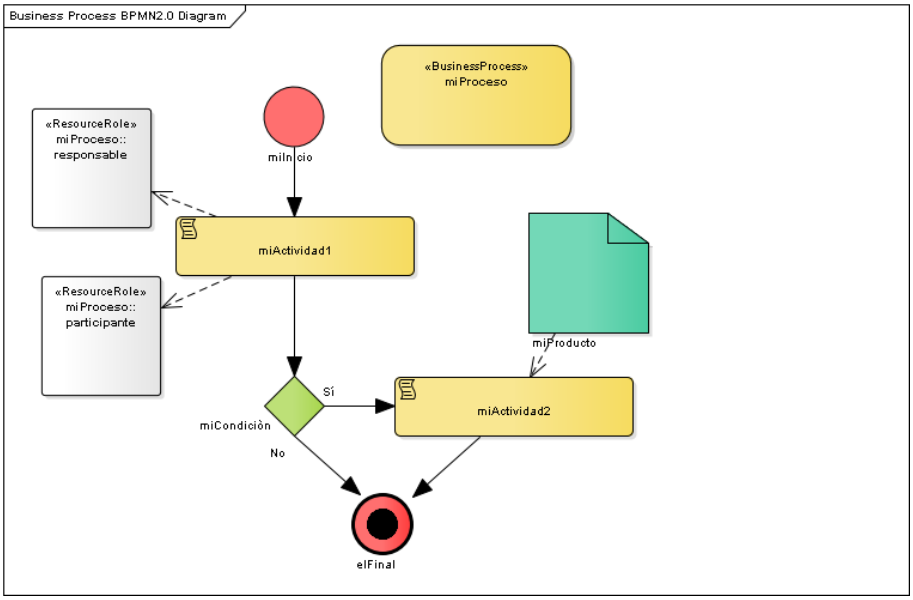
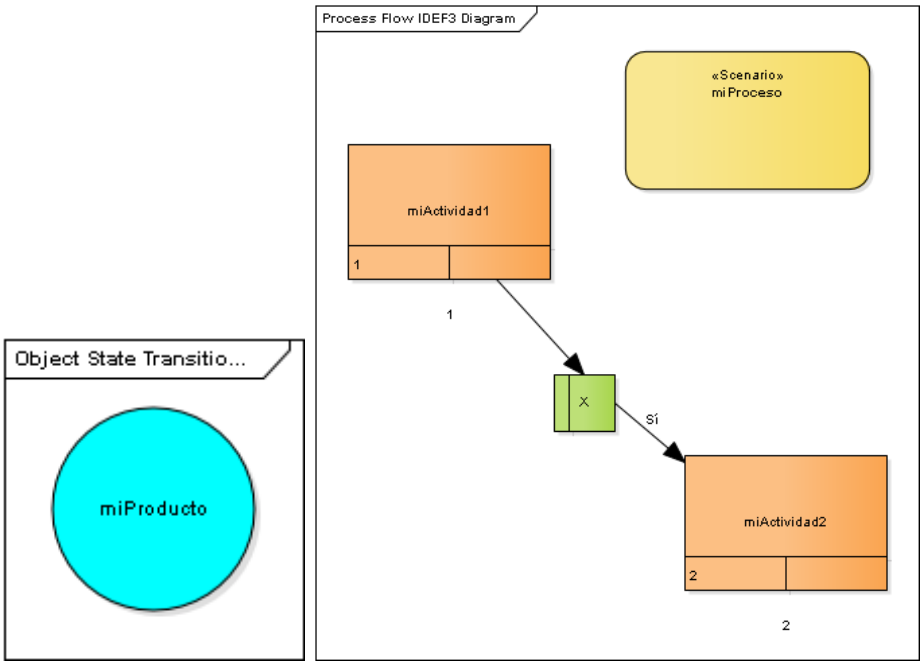


Figura 7.19: Situación real 3: Modelo de proceso de software común en BPMN para la empresa B



(a) Vista Objetos (b) Vista Procesos

Figura 7.20: Situación real 3: Modelo de proceso de software común en IDEF3 para la empresa C

Para concluir, simplemente destacar que en esta sección hemos visto en detalle cómo el marco de referencia y su herramienta de soporte MONETA pueden resolver los problemas que se plantearon en el capítulo 3 como situaciones reales que justifican el desarrollo de dicho marco de referencia. Para ello se han validado dichas situaciones mostrando fielmente las capacidades que el marco de referencia desarrollado proporciona.

7.4. Conclusiones

En este capítulo se ha presentado MONETA como la herramienta de soporte del marco de referencia que hemos ido definiendo y desarrollando a lo largo de este trabajo de tesis. Se ha mostrado cómo se han incorporado a MONETA las tres piezas claves del marco de referencia, de forma que se garantiza que dicho marco está siendo soportado por la herramienta, y se ha validado MONETA utilizando las tres situaciones reales que justificaban la realización de este trabajo de tesis descritas en el capítulo 3 a la hora de determinar el problema a resolver. Estas situaciones reales nos han permitido validar la herramienta frente a los requisitos que habíamos planteado. Sin embargo, será en el siguiente capítulo dónde se expondrán los casos de estudio reales que nos han permitido validar el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los procesos de software.

Capítulo 8

Evaluación del marco de referencia: Casos de estudio reales

Llegados a este punto, después de establecer el problema a resolver, de haber ido profundizando una por una en las tres piezas claves de las que consta el marco de referencia para facilitar la interoperabilidad de los modelos de procesos de software, para finalmente haber expuesto la herramienta que da soporte a dicho marco, MONETA, validada con las situaciones reales que justificaban la realización de este trabajo de tesis, es el momento de evaluar la solución propuesta en casos de estudio reales.

El objetivo de este capítulo consiste en corroborar, con casos específicos extraídos de proyectos reales con la industria, que el problema que en este trabajo de tesis se ha planteado existe, es una realidad en situaciones más o menos cotidianas y que, aunque en el momento en el que se dieron se llevó a cabo un planteamiento completamente ad-hoc para resolver la situación, el resultado de este trabajo de tesis habría facilitado en gran medida las soluciones implementadas, debido principalmente a la magnitud del trabajo manual que originó. De hecho, dichos proyectos fueron el germen de la inspiración para la realización de este trabajo de tesis, y en los proyectos que les siguen, continuación de aquellos, vamos a tener la oportunidad de utilizar los resultados aquí obtenidos, lo que facilitará en gran medida su desarrollo.

Para ello, este capítulo se estructura de la siguiente manera. En primer lugar se establece el contexto de los proyectos reales elegidos, en los que se evidencia la existencia de la problemática aquí planteada en la industria. A continuación, en las dos secciones siguientes se describen de forma detallada los dos proyectos elegidos y la forma en la que se habrían abordado si el marco de referencia desarrollado en este trabajo de tesis hubiera existido entonces. De hecho, éste será el "modus operandi" en los proyectos posteriores. Para finalizar, en la última sección se resumen las principales conclusiones obtenidas a lo largo del capítulo.

8.1. Elección de los casos de estudio

A lo largo de la exposición realizada en el transcurso de este trabajo de tesis se han establecido los procesos de software como el contexto en el que se ha trabajado en el desarrollo del marco de referencia, puesto que dicho marco está definido para la interoperabilidad y mantenibilidad de los procesos de software y, en todo momento, se ha hecho alusión a los lenguajes de modelado de procesos de software como parte integrante del mismo. Como consecuencia de la búsqueda insistente del mínimo común de los conceptos recogidos en los lenguajes de modelado de procesos de software, se ha alcanzado una solución que resulta de fácil adopción en empresas que disponen de procesos de negocio en general, más allá del software. Este hecho, lejos de ser un hándicap a nuestra solución, supone una ventaja añadida al marco de referencia, puesto que amplía en mucho las posibilidades de aplicación en entornos industriales.

Uno de los objetivos planteados al enunciar nuestra propuesta de solución consistía precisamente en poder validarla con ejemplos reales concretos, con la intención de mostrar su aplicabilidad en la práctica. El hecho de que este trabajo se haya desarrollado dentro del grupo de investigación Ingeniería Web y Testing Temprano (IWT2) de la Universidad de Sevilla, el cual posee colaboraciones establecidas con entidades, tanto públicas como privadas, en dominios muy diversos mediante diferentes proyectos de I+D de ámbito autonómico, nacional e internacional, ha facilitado en gran medida la consecución de dicho objetivo, ya que nos ha proporcionado casos y situaciones en empresas en los que la problemática aquí enunciada era una realidad. Esto nos ha permitido validar el marco de referencia aquí desarrollado, además de asegurar la posibilidad de transferencia inmediata de los resultados de la tesis, en concreto dicho marco, a entornos empresariales, algo que ha sido una aspiración subyacente en todo momento durante su desarrollo.

De entre los posibles casos de estudio a elegir para presentar tanto la problemática contextualizada como la manera con la que, utilizando nuestra propuesta, se habría abordado la solución, se han seleccionado dos. Dicha elección viene motivada por la envergadura, la representatividad, la importancia y la actualidad de los proyectos elegidos y de las organizaciones presentes en los mismos, sin condicionar dicha elección al hecho de que estas empresas pertenezcan o no al ámbito del software. Además, se da el hecho de que ambos proyectos han sido el punto de partida de sendos marcos de colaboración más amplios con las organizaciones involucradas, marcos que continúan vigentes a través del desarrollo de nuevos proyectos en los cuales los resultados obtenidos, en concreto el marco de referencia, podrán ser ya utilizados.

Los proyectos seleccionados son los siguientes:

- SAS, proyecto para el desarrollo de la adaptación de la plataforma eSalud a una arquitectura orientada a servicios (Service Oriented Architecture, SOA), de forma que permita una mayor modularidad, independencia, mantenibilidad y la usabilidad del desarrollo de servicios clínicos en dicha plataforma. Este proyecto se ha llevado a cabo para la Consejería de Salud y Bienestar Social de la Junta de Andalucía en colaboración con el grupo GiT (Grupo de Investigación Tecnológica) del Hospital Universitario Virgen del Rocío de Sevilla.

- CALIPSOneo, advanCed Aeronautical soLutIons using Plm proceSses and tOols, proyecto en el que entre sus objetivos se encontraba la definición de una metodología de trabajo para permitir a los ingenieros definir, simular, optimizar y validar los procesos de montaje aeronáuticos en un entorno virtual 3D antes de su ejecución real en una línea de montaje. Todo ello a través de un proceso integral de recolección de requisitos, personalización y uso de software PLM existente en el mercado. Este proyecto fue llevado a cabo en colaboración con Airbus Defense & Space.

En las tablas 8.1 y 8.2 se puede observar la información más relevante relativa a dichos proyectos.

Plataforma de E-Salud: Adaptación de la Plataforma a una Arquitectura de Procesos y Pilotaje del Registro para Estudios Epidemiológicos de Lesión de Médula	
Investigador Principal:	María José Escalona Cuaresma
Referencia:	P040-13/E09
Fecha de inicio:	03/05/2013
Fecha de fin:	30/09/2013
Empresa / Organismo financiador:	Fisevi
Presupuesto:	14.995€+ 9.700€(ampliación)

Tabla 8.1: Información detallada del proyecto Plataforma E-Salud

CalipsoNEO: Soluciones Aeronáuticas Avanzadas usando Procesos y Herramientas PLM	
Investigador Principal:	Carmelo del Valle
Referencia:	P051-12/E08
Fecha de inicio:	24/01/2012
Fecha de fin:	31/12/2013
Empresa / Organismo financiador:	Airbus Defense & Space
Presupuesto:	139.026€

Tabla 8.2: Información detallada del proyecto CalipsoNEO

Una vez contextualizados los proyectos elegidos como objeto de estudio para validar el marco de referencia en entornos reales, en las siguientes secciones se profundizará en ambos proyectos. Para cada uno de estos proyectos se describirá en detalle el objetivo del mismo y la necesidad que han evidenciado entorno al problema planteado en este trabajo de tesis, que es en definitiva por lo que han sido elegidos como casos de estudio. Además, se expondrá cómo fueron abordados en su desarrollo y, finalmente, cómo lo habrían sido si hubieran contado con el marco de referencia y la herramienta MONETA en su ejecución.

8.2. Caso de estudio: SAS

eSalud es una plataforma desarrollada por la Junta de Andalucía para la tramitación electrónica de procesos clínicos y el intercambio de información utilizando sistemas software de información. El trabajo planteado era de gran envergadura y tenía como objetivo adaptar dicha plataforma a una arquitectura orientada a servicios de forma que se alcanzara una modularidad, independencia, mantenibilidad y usabilidad en el desarrollo de módulos funcionales que proporcionan soporte a los servicios clínicos del hospital. Para acometerlo se definió un proyecto inicial, a modo de prueba de concepto, en el que se pretendía verificar la conveniencia de utilizar un enfoque MDE para desarrollar el trabajo completo de adaptación. Este proyecto se llevó a cabo mediante una colaboración entre los grupos GiT e IWT2 durante los años 2013-2014 y se encuentra descrito en [García-García *et al.*, 2015].

Durante las primeras fases del proyecto, uno de los puntos primordiales era el de adoptar una decisión de común acuerdo entre ambos grupos de investigación para representar los procesos clínicos que se encontraban incorporados en la plataforma eSalud. Por un lado, el grupo GiT disponía de todos los procesos clínicos descritos de forma gráfica en el lenguaje BPMN, y su principal argumento de uso era que BPMN es, a día de hoy, el lenguaje de modelado de procesos de negocio con mayor expresividad, aunque esto implique una mayor complejidad. Por otro lado, el enfoque MDE que el grupo IWT2 pretendía darle al proyecto implicaba transformar los procesos clínicos de BPMN a diagramas de actividad de UML para permitir, por medio de las herramientas desarrolladas por el grupo, dar soporte a su ejecución. Obviamente, la elección de lenguajes de modelado de procesos en ambos entornos era diferente y era necesario llegar a un consenso y seleccionar un lenguaje común para este proyecto, aunque esto suponía una serie de connotaciones que vamos a ir desglosando a continuación.

Para alcanzar dicho consenso fue necesario demostrar lo que se argumenta en [Birkmeier y Overhage, 2010], en el que se demuestra de forma empírica que en la práctica no existe una diferencia fundamental en el uso de BPMN frente a los diagramas de actividad de UML, puesto que los elementos de BPMN que se utilizan realmente a la hora de describir un proceso ronda las tres cuartas partes de la totalidad del estándar. Esta comprobación se hizo utilizando el proceso clínico modelado en BPMN más complejo del que se disponía y realizando una traducción a un diagrama de actividad de UML, obteniendo como resultado que los modelos de procesos eran completamente equivalentes.

El acuerdo al que se llegó fue el de traducir los procesos clínicos modelados en BPMN a diagramas de actividad de UML de forma manual. Este trabajo de traducción conllevó dos importantes inconvenientes que, aunque fueron asumidos en la realización del proyecto, nos gustaría recalcar. El primero de ellos es obvio, y tiene que ver con el importante coste que supone esta traducción, derivado del esfuerzo en horas de los recursos del proyecto y de la complejidad de las tareas de análisis de los ejemplos del hospital, tareas que por otro lado ya habían sido realizadas para obtener el modelo de proceso en BPMN y que ahora se volvían a ejecutar. El segundo de ellos quizá no resulta tan evidente y se trata de las inconsistencias que aparecen al trabajar los modelos de forma manual. Está comprobado que una misma persona transformando manualmente un modelo de un lenguaje a otro en tiempos distintos produce modelos diferentes, debido principalmente a que siempre hay decisiones contextuales específicas

del momento. Si en lugar de ser una única persona se incorporan a esta labor más participantes, estas diferencias se acrecientan, lo que constituye una importante fuente de errores. Esto, que supone un problema en un entorno reducido, se amplifica extraordinariamente en ámbitos de trabajo grandes, como el que podría ser un hospital, que es el caso que nos ocupa. Además, aunque no es un problema en sí mismo, el hecho de elegir los diagramas de actividad de UML frente a BPMN supone que, de alguna forma, estamos dando prevalencia al primero frente al segundo, demandando un importante cambio en el modo de trabajo en el entorno del hospital Virgen del Rocío.

Una vez descrita la necesidad demandada en la ejecución de este proyecto y la forma en la que se abordó, veamos lo que habría pasado en el caso de que el marco de referencia y la herramienta MONETA hubieran existido, y cómo el planteamiento habría cambiado. En primer lugar, la traducción tal cual se hizo, es decir, pasar de BPMN a los diagramas de actividad de UML, no habría sido necesaria llevarla a cabo. El grupo GiT podría haber seguido trabajando con el lenguaje de modelado de procesos BPMN y el grupo IWT2 haber mantenido su elección de los diagramas de actividad de UML para hacer uso de las herramientas desarrolladas por el grupo en la ejecución de este proyecto, sin imponer un lenguaje por encima del otro. Ese es uno de los principales objetivos del marco de referencia y de MONETA.

Así, los procesos hospitalarios se podrían haber mantenido y actualizado siempre que fuera necesario con BPMN y, mediante las transformaciones incorporadas en MONETA, obtener de forma automática el modelo de proceso del diagrama de actividad de UML equivalente, para que pudiera ser tratado con las herramientas del grupo IWT2, y si en algún caso se detectara la necesidad de incluir alguna nueva actividad o se observara algún tipo de inconsistencia o fallo en el proceso modelado con los diagramas de actividad de UML, el cambio realizado para solventarlo podría haber sido automáticamente transformado al modelo BPMN, evitando así las inconsistencias, pero manteniendo la forma de trabajo de ambos grupos.

Veamos un ejemplo que nos permita visualizar todas estas ventajas teóricas del uso de MONETA en este contexto que hemos expuesto. Las figuras que vamos a mostrar a continuación son capturas de pantalla de un fragmento de uno de los procesos utilizados y obtenidos con MONETA, puesto que el diagrama completo es de tal envergadura que sería ilegible en un documento como esta memoria de tesis. No obstante, en la web del grupo IWT2[1] se encuentra disponible para su descarga el proceso completo tanto de este caso de estudio real como el siguiente. También queremos mencionar que, de nuevo considerando la legibilidad de los ejemplos en esta memoria, hemos eliminado el marco de la caja de herramientas y del editor en las figuras.

[1] http://iwt2.org/

El fragmento de proceso clínico seleccionado, disponible en BPMN, se ha modelado con MONETA tal y como se puede observar en la figura 8.1.

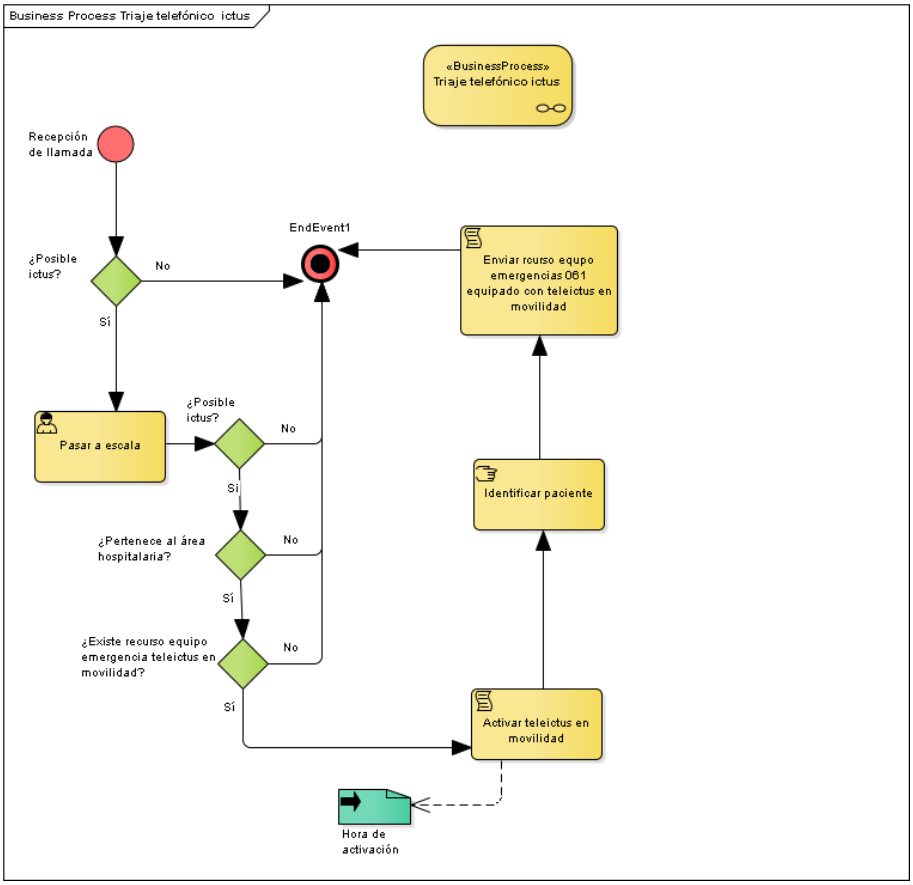


Figura 8.1: Fragmento del diagrama BPMN del proceso de triaje de un ictus en MONETA

De acuerdo a lo que hemos ido detallando en esta sección, el objetivo del uso de MONETA era conseguir el modelo en diagramas de actividad de UML equivalente para este proceso. Siguiendo los pasos descritos en el manual de usuario que se encuentra en el anexo D, se obtiene el modelo representado en la figura 8.2, consiguiendo de forma automática lo que en el proyecto se hizo manualmente.

Al igual que hemos hecho en el capítulo 7, aunque en el uso real por parte del usuario, la obtención del modelo en INROMA es completamente transparente, nos ha parecido adecuado presentarlo para ilustrar de forma conveniente el ejemplo que aquí queremos mostrar. De esta forma, el modelo INROMA intermedio para este fragmento de proceso del triaje es el que se puede observar en la figura 8.3.

Todo esto se encuentra almacenado dentro de los modelos de MONETA. Si en un momento en el transcurso del proyecto se detectara la necesidad de actualizar el proceso en BPMN con cambios derivados de su tratamiento en UML-DA, llevaríamos a cabo la transformación inversa gracias al carácter bidireccional del marco de referencia. Es indiscutible el ahorro de costes y la eliminación de errores que se obtiene gracias a MONETA, frente a un funcionamiento puramente

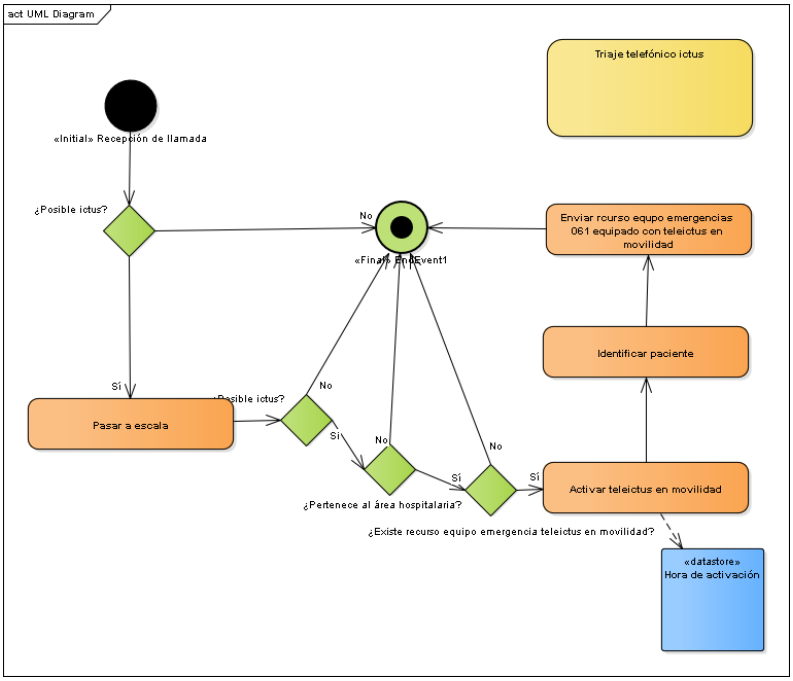


Figura 8.2: Diagrama UML-DA del proceso de triaje de un ictus obtenida con MONETA

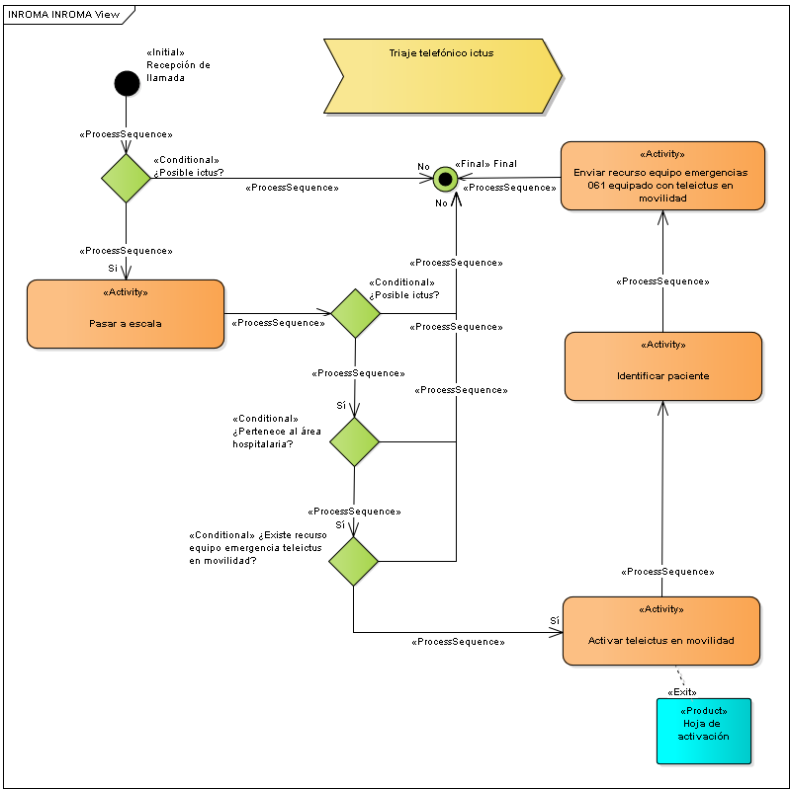


Figura 8.3: Diagrama intermedio en INROMA del proceso de triaje de un ictus obtenida con MONETA

manual.

Este primer proyecto ha abierto el camino para establecer un marco de colaboración y poder abordar el trabajo global que inicialmente se planteaba, acometiendo nuevos proyectos en los que poder utilizar los resultados de este trabajo de tesis. De hecho, actualmente se está desarrollando uno de ellos en los que se encuentran involucrados el grupo IWT2, el hospital Virgen del Rocío y la empresa Fujitsu, con lo que, al incluir nuevos participantes que hagan uso de MONETA, el efecto multiplicador de utilización en entornos reales se amplía considerablemente, y también el uso del resto de las funcionalidades que el marco de referencia y MONETA ofrecen.

8.3. Caso de estudio: CALIPSOneo

El proyecto CALIPSOneo finalizó a finales del 2013 y, como ya hemos introducido en una sección anterior, tenía como objetivo conseguir la máxima integración de información, procesos de negocio, personas y sistemas en la realización del diseño industrial de un producto. Fue el primero al que están siguiendo otros proyectos, algunos ya en marcha como EOLO o GEOLIA, dentro del marco de colaboración establecido entre IWT2 y Airbus Defense & Space.

Como hemos comentado previamente, uno de los objetivos consistía en la definición de una metodología de trabajo para permitir a los ingenieros definir, simular, optimizar y validar los procesos de montaje aeronáuticos en un entorno virtual 3D antes de su ejecución real en una línea de montaje. Todo ello a través de un proceso integral de recolección de requisitos, personalización y uso de software PLM existente en el mercado.

A través de los años, se realizaron varios análisis que concluyeron en un conjunto de beneficios para Airbus Military [Mas *et al.*, 2013]. A partir de este conocimiento, el equipo del proyecto CALIPSOneo llevó a cabo una exhaustiva fase de captura de requisitos del proyecto, el plan de gestión de requisitos, y la matriz de trazabilidad de requisitos para satisfacer las necesidades PLM del negocio. En última instancia, CALIPSOneo contempló el diseño de una nueva metodología PLM para ajustarse a una solución PLM colaborativa y el desarrollo de la solución software que proporcionase soporte a ese concepto [Salido *et al.*, 2013].

El contexto en el que nos encontramos a la hora de desarrollar este proyecto fue el siguiente. Por un lado, los procesos estaban definidos desde un punto de vista estratégico, por un departamento que tenía entre sus funciones la conformación de dicha estrategia. Para ello se utilizaba IDEF como lenguaje de modelado de procesos, muy habitual en la industria aeronáutica, y con gran capacidad semántica para el fin con el que estaba elegido. Sin embargo, dichos procesos debían ser implementados sobre herramientas de soporte por un departamento diferente, en un plano más operativo, para lo cual IDEF no resulta el lenguaje más adecuado.

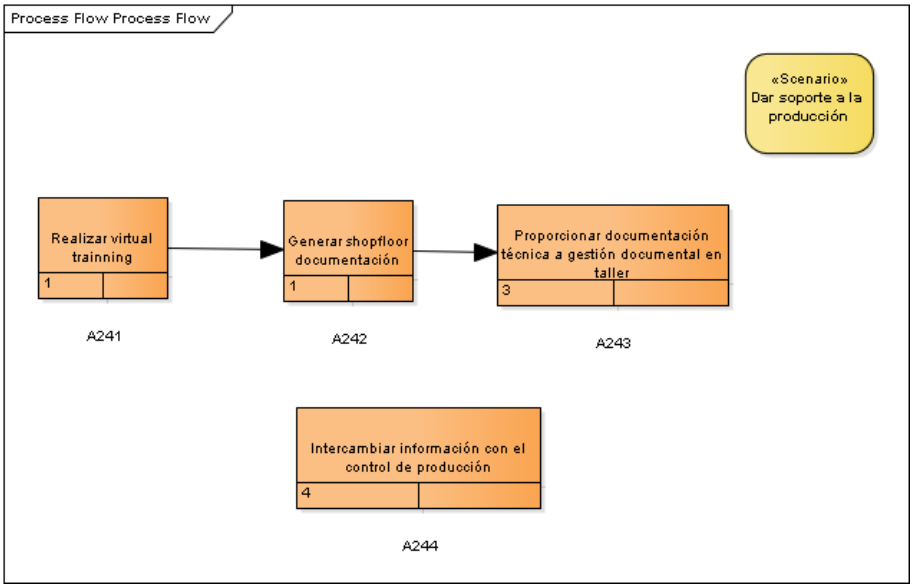
En los inicios del proyecto se intentó hacer uso de IDEF, e incluso hacer una extensión al mismo de forma que se enriqueciera y sirviera de lenguaje vehicular entre ambos departamentos. Sin embargo se demostró que no era la opción más adecuada, máxime incluso teniendo en cuenta la falta de herramientas de soporte adecuadas para esta lenguaje a nivel comercial, y

el grupo IWT2 planteó dar un giro y utilizar un enfoque MDE, usar UML-DA como lenguaje de modelado de los procesos de Airbus y utilizar asimismo las herramientas de soporte que el grupo ya tiene definidas. Nos volvemos a encontrar una situación similar a la anterior, en la que un departamento modela procesos en IDEF y otro departamento tiene en UML-DA su lenguaje para las herramientas de soporte.

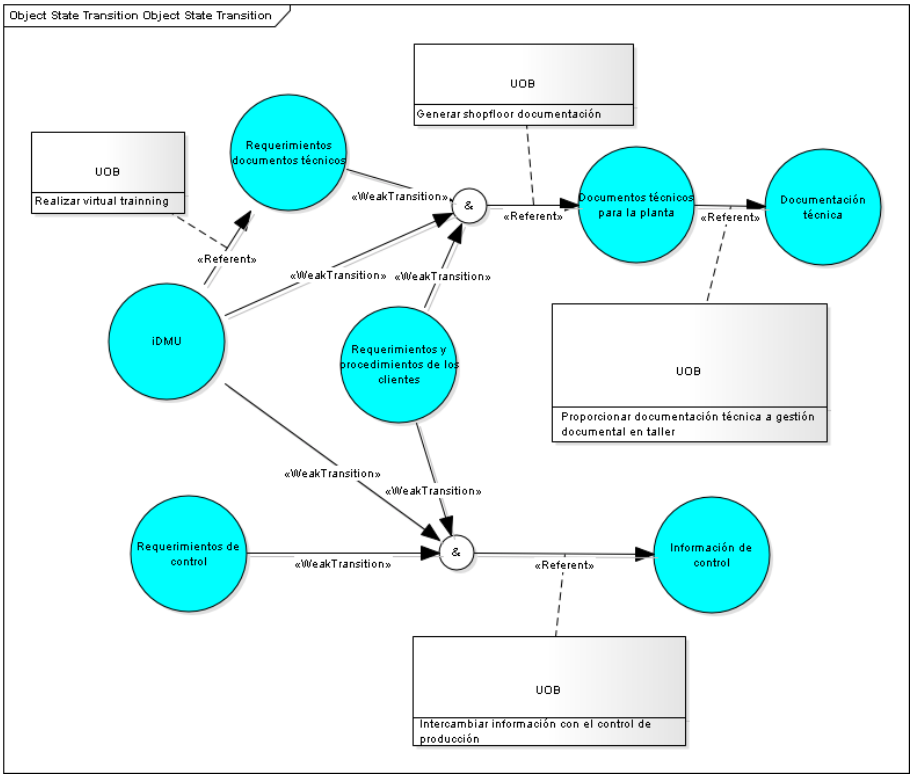
De nuevo la traducción de los procesos se hizo de forma manual, con problemas similares a los referidos en el caso de estudio anterior, pero en este caso dichos problemas se acrecentaban. El paso de un enfoque estratégico a un enfoque operativo obligaba al segundo departamento a, además de implementar los procesos recibidos, evolucionarlos y alimentarlos con nuevas fuentes de información. De esta forma, era necesario mantener manualmente una consistencia en la información muy complicada de mantener, puesto que no siempre se conseguía la retroalimentación continua, teniendo en cuenta que de un mismo proceso estratégico se podían obtener un buen número de procesos operativos.

En el caso de poder haber utilizado el marco de referencia e INROMA desde el comienzo, se habría facilitado en gran medida estas tareas de mantenibilidad e interoperabildad de los procesos de Airbus, ya que la automatización de las mismas, no solamente las agilizan, sino que también las independizan de las personas que las ejecutan y el momento o circunstancia en la que ocurren.

Debido a las gran cantidad de procesos existentes en el proyecto, y dado que el objetivo de este capítulo es demostrar la validez del marco de referencia de acuerdo a lo que nos habíamos planteado, hemos decidido mostrar en este trabajo de tesis un único proceso, el denominado PROTEUS, definido en el departamento PLM con IDEF. Este proceso ha sido transcrito en MONETA, para lo que se ha utilizado el lenguaje IDEF3, en el que es necesario modelar tanto los procesos como los objetos que en ellos intervienen, y podemos observarlo en la figura 8.4. De nuevo, las figuras que aparecen son capturas de pantallas obtenidas de la utilización de MONETA, si bien se han eliminado los aspectos visuales del editor para mejorar su legibilidad.



(a) Vista diagrama de Procesos



(b) Vista diagrama de Objetos

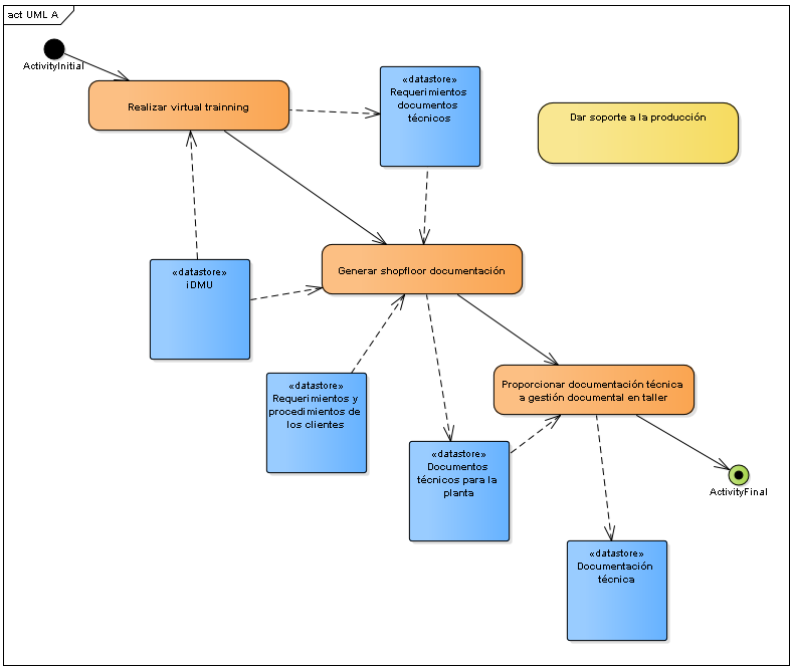
Figura 8.4: Diagrama del proceso PROTEUS modelado con IDEF3 en MONETA

A la hora de desarrollar la herramienta de soporte para este proceso, fue necesario su transformación a UML-DA. Ya hemos dicho que esto se trata de una tarea completamente automática a través de MONETA, pudiendo observarse el resultado en la figura 8.5.

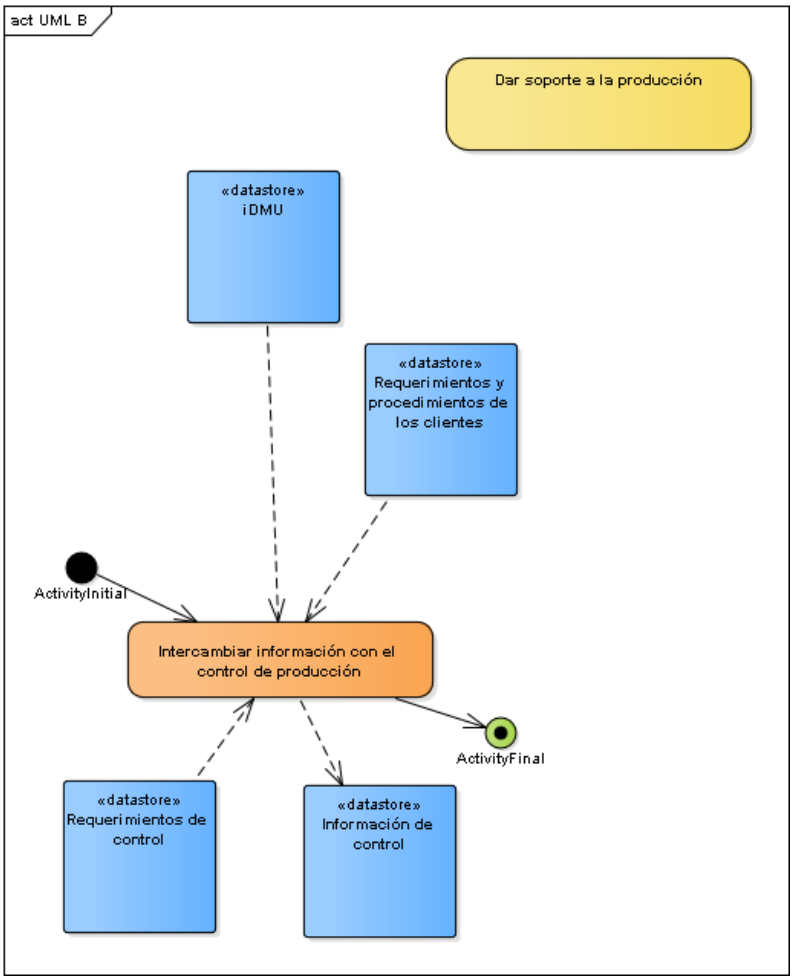
No es difícil observar la posibilidad de enriquecer y alimentar este proceso, en definitiva, evolucionarlo. Para mantener su consistencia con aquellos definidos en IDEF no habría más que hacer uso de MONETA para que dichas aportaciones quedaran también recogidas en el modelo de procesos de este lenguaje.

Al igual que hemos hecho con el ejemplo anterior, aunque el modelo en INROMA es completamente transparente al usuario, hemos querido destacarlo para ilustrar en mayor medida el ejemplo que estamos planteando. Dicho modelo se encuentra reflejado en la figura 8.6.

Finalmente, sólo nos queda destacar que las ventajas y funcionalidades que MONETA nos ofrece van a ser utilizadas también en los proyectos continuación dentro del marco de colaboración con Airbus Defense & Space, pero no sólo con ellos. Dada la gran cantidad de posibilidades que MONETA nos proporciona a la hora de trabajar con modelos de procesos y diferentes lenguajes de modelado de los mismos, ya son varias las empresas que se han interesado por la misma para la utilización en ámbitos muy diferentes.

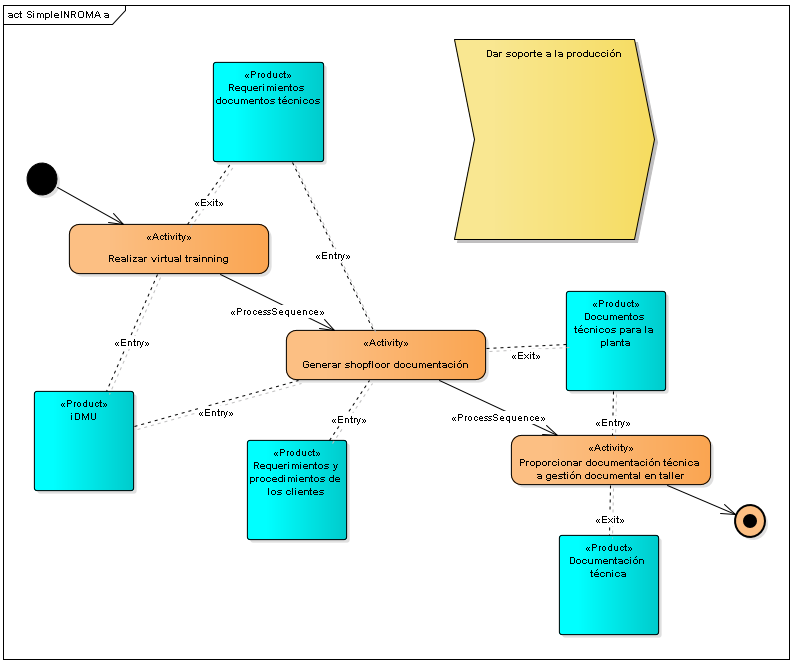


(a)

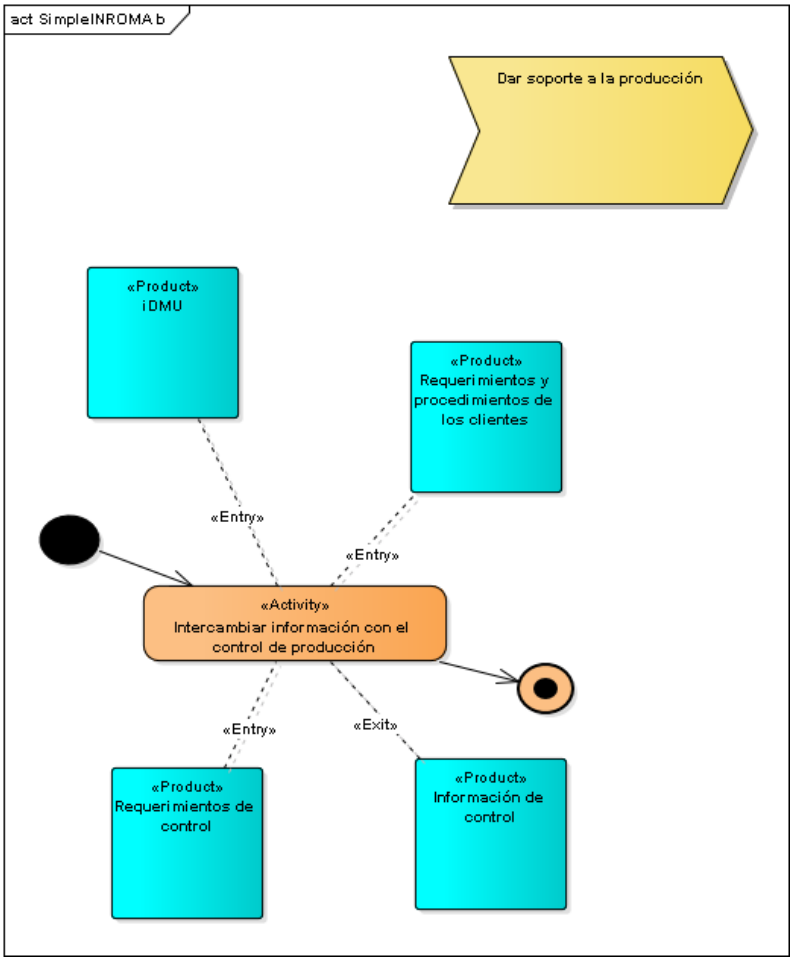


(b)

Figura 8.5: Diagrama de PROTEUS modelado con UML-DA



(a)



(b)

Figura 8.6: Diagramas intermedios en INROMA del proceso PROTEUS obtenido con MONETA

8.4. Conclusiones

Para concluir, destacar que en este capítulo se han presentado dos casos de estudio con empresas reales en los que se ha podido constatar que la necesidad que ha inspirado el desarrollo de este trabajo de tesis existe en la industria. Las situaciones aquí referidas nos han mostrado que, hasta ahora, se ha trabajado en su resolución de forma completamente manual y ad-hoc para cada caso concreto. También se ha comentado que esta manera de abordar las transformaciones de los modelos de procesos entre diferentes lenguajes implica, no sólo el coste tanto en esfuerzo como en complejidad de las tareas asociadas, sino que constituye también una importante fuente de errores e inconsistencias a la hora de trabajar con los modelos de procesos así obtenidos. La disponibilidad del marco de referencia desarrollado en este trabajo de tesis así como, obviamente, la herramienta MONETA que le da soporte y que facilita en gran medida su aplicación práctica en entornos industriales, va a suponer un antes y un después en la realización de estas tareas relacionadas con modelos de procesos y lenguajes de modelado que forman parte de muchos proyectos asociados con mejora metodológica de las organizaciones, especialmente cuando el número de participantes y de localizaciones se va incrementando.

Capítulo 9

Conclusiones y trabajos futuros

Durante este trabajo de tesis hemos ido abordando en profundidad uno por uno todos los elementos que conforman el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los procesos de software. Para poder llevarlo a cabo, se ha realizado un estudio exhaustivo de la situación actual y, una vez analizados los resultados obtenidos, se han concretado las motivaciones que han cimentado el desarrollo de dicho marco.

Teniendo en cuenta que para resolver el problema planteado se ha optado por una perspectiva de ingeniería dirigida por modelos, nuestra propuesta de marco de referencia está soportado por tres elementos claves, que han constituido el grueso del trabajo aquí presentado: un lenguaje de modelado de procesos de software que cumple los requisitos necesarios para ser utilizado en dicho marco, al que hemos denominado INROMA, un método que facilita la incorporación de los diferentes lenguajes de modelado de procesos de software existentes al marco, con el objetivo de posibilitar la interoperabilidad y la mantenibilidad de los procesos de software con ellos modelados, y, por último, las transformaciones que hacen posible dicho marco.

Además, para hacer viable el uso de esta solución teórica en entornos empresariales, se ha desarrollado la herramienta MONETA, que da soporte práctico a todo este trabajo. Para ello, MONETA ha sido doblemente validada. Por un lado, se ha utilizado para implementar las tres situaciones reales inicialmente descritas en el planteamiento del problema que justificaban la realización de este trabajo de tesis y, por otro lado, se ha evaluado mediante la validación con dos casos de estudio reales, elegidos de proyectos llevados a cabo con empresas.

Esto es, en definitiva, lo que ha constituido el desarrollo de este trabajo de tesis. Sin embargo, es necesario dejar constancia de cuáles han sido las principales aportaciones de este trabajo para poder darlo por concluido, y éste va a ser el objetivo del presente capítulo. Para lograrlo, en la primera sección se detalla el marco de investigación en el que se ha desarrollado esta tesis doctoral, que ha constituido una influencia importante en los resultados obtenidos. En este marco hablaremos tanto del contexto en el que se ha desarrollado, como del conjunto de grupos de investigación con los que se ha colaborado y también han influido en el resultado de la misma. A continuación se detallan precisamente estas aportaciones haciendo referencia a los objetivos

planteados al comienzo del trabajo. Y, para finalizar, se proponen un conjunto de trabajos y líneas de investigación que siguen la senda iniciada por ésta y otras tesis doctorales vinculadas. La última sección resume las principales conclusiones obtenidas a lo largo del capítulo.

9.1. Marco de investigación en el que se desarrolla este trabajo

El contexto en el que se ha enmarcado este trabajo de tesis ha influenciado en gran medida en su desarrollo. Dicho contexto engloba tanto la idiosincrasia propia del grupo IWT2 como las relaciones de dicho grupo con otros grupos de investigación y empresas. Ambos aspectos son descritos en detalle a continuación.

9.1.1. Línea de investigación en el grupo IWT2

En varias ocasiones a lo largo de este trabajo de tesis se ha comentado la importancia de haber sido desarrollado en el grupo IWT2, puesto que no sólo nos aportaba un soporte teórico, sino que su capacidad de ejecución de proyectos de transferencia de I+D a empresas y organismos de toda índole, nos facilitaba la identificación de los problemas y necesidades que en entornos empresariales se detectan, así como las expectativas que dichos problemas generaban dentro de las propias organizaciones. Más adelante comentaremos la importancia de esto último, pero ahora nos centramos en el soporte teórico.

El grupo IWT2 posee desde hace unos años una línea estratégica de investigación en la que se trabaja en cómo combinar de forma satisfactoria el paradigma de ingeniería dirigida por modelos con la gestión de procesos de negocio en múltiples ámbitos, con la finalidad de resolver diferentes necesidades con las que se ha ido encontrando. El hecho de contar con una línea sobre procesos permite armonizar propuestas y planteamientos, ya que los resultados de un trabajo de investigación apoyan las hipótesis de los siguientes, abriendo camino y avanzando conjuntamente.

Dentro de los objetivos de esta línea de investigación se contempla la necesidad de mejorar la competitividad de las organizaciones mediante la mejora continua de sus procesos de negocio, facilitando una gestión eficaz y eficiente de los mismos, utilizando para ello el paradigma MDE. Para lograrlo, actualmente se encuentran en estado de desarrollo muy cercano a su finalización tres trabajos de tesis estrechamente relacionados, en la que se abordan aspectos clave que constituyen la base a partir de la cuál están surgiendo nuevos trabajos, tanto de transferencia a empresas como de aplicación a otros ámbitos. Los aspectos estudiados y que constituyen la base sobre la que se conforma la línea de procesos de negocio son los siguientes:

1. Proporcionar a las organizaciones un entorno basado en modelos para orquestar y ejecutar los procesos de negocio a partir de su modelado, en el que se consiga reducir, e incluso eliminar, la brecha existente entre los procesos definidos en las organizaciones y lo que realmente ocurre en el día a día de las mismas, es decir, las diferencias entre los procesos

definidos y los procesos actuales. Para alcanzar dicho objetivo se contempla la definición de una serie de protocolos con los que se transforma el modelo de definición del proceso a su modelo de ejecución y orquestación para, en este punto, obtener una versión textual, desplegable y ejecutable en un motor de ejecución de procesos.

2. Aportar un marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los procesos de software, aspecto que está siendo abordado en este trabajo de tesis tal y cómo hemos descrito en detalle. De esta forma se garantiza la independencia del proceso modelado frente al lenguaje de modelado seleccionado a la hora de abordar proyectos interorganizacionales.

3. Proporcionar a las organizaciones la capacidad de modernización a través de la extracción dinámica de la vista global de sus procesos de negocio a partir de la información inferida en bases de datos relacionales, estructuras de tablas, restricciones y activadores PL/SQL de reglas de negocio. De esta forma se obtienen los elementos esenciales que se encuentran involucrados en una organización en el desarrollo de proyectos: composición de procesos y métricas de los mismos, documentos, estructuras de equipos de trabajo, etc.

A partir de estos tres trabajos que, como hemos comentado, constituyen la base de conocimiento para posteriores investigaciones, en el grupo IWT2 se han iniciado un par de trabajos de tesis dentro del ámbito de la gestión de procesos clínicos, motivados por la incipiente trayectoria del grupo en la realización de proyectos de transferencia e I+D en el sector sociosanitario, y otro relacionado con la gestión de los procesos involucrados en la cadena de suministro, fruto de la estrecha relación del grupo IWT2 con la división de Logística del Instituto Tecnológico de Aragón. Todos estos trabajos están siendo llevados a cabo por doctorandos miembros del grupo IWT2.

9.1.2. Relaciones con otros organismos que han influido en el desarrollo de este trabajo de tesis

Una vez conocida la importancia de la línea de procesos de negocio en el grupo IWT2, a continuación se nombran algunos de los grupos de investigación, organismos y empresas, que sin ser los únicos, sí los podemos considerar como los más relevantes, con los que el grupo IWT2 se encuentra actualmente trabajando entorno a esta línea de procesos de negocio, y que han sido y siguen siendo relevantes tanto en el desarrollo de este trabajo de tesis como a la hora de abordar nuevas líneas de trabajo ya en marcha o planteadas para un futuro.

- **El Grupo de Innovación Tecnológica de la fundación FISEVI**. Como se ha comentado en el capítulo 8, uno de los casos de estudio utilizados para validar el marco de referencia ha sido precisamente el proyecto de adaptación de la plataforma eSalud del hospital universitario Virgen del Rocío de Sevilla a una arquitectura basada en procesos SOA para permitir una mayor modularidad, independencia, mantenibilidad y usabilidad durante el desarrollo de módulos funcionales que proporcionan soporte a los servicios clínicos del hospital. En este proyecto se colaboró estrechamente con el Grupo de Innovación

Tecnológica (GiT), un grupo de investigación relacionado con la I+D+i y las nuevas tecnologías que se encuentra bajo la coordinación del gerente de la Fundación Pública Andaluza para la Gestión de la Investigación en Salud de Sevilla (FISEVI)[1], y en colaboración con las Unidades de Gestión Clínica y las no Asistenciales de los hospitales universitarios Virgen Macarena y Virgen del Rocío, en concreto con el Servicio de Tecnologías de la Información. Anteriormente se ha comentado el hecho de que esta relación tan continuada ha conllevado el hecho de que actualmente se están realizando dos trabajos de tesis en el ámbito de procesos clínicos.

- **El Grupo de Investigación en Ingeniería del Software de la Universidad de Oviedo**. Aunque algo más adelante será destacado como una futura línea de trabajo común entre ambos grupos de investigación, ya se han realizado las primeras propuestas para incorporar técnicas de pruebas de software con el objetivo de asegurar la conformidad de los procesos ejecutados conforme a su definición. Esto es posible gracias a que el Grupo de Investigación de Ingeniería del Software (GIIS) de la Universidad de Oviedo tiene como línea de trabajo la definición de metodologías y herramientas para la calidad del software, especialmente en el ámbito de las pruebas de software.

- **El Grupo de Investigación en Mejora del Proceso Software y Métodos Formales de la Universidad de Cádiz**. También se comentará más adelante, pero otra línea de trabajo paralela a la anterior es la que se lleva a cabo con este grupo de Mejora del Proceso Software y Métodos Formales (SPI&FM) de la Universidad de Cádiz. La línea de trabajo con este grupo versa sobre la simulación de procesos de software, y también se han realizado los primeros trabajos al respecto.

- **El Laboratorio de Investigación y Formación en Informática Avanzada de la Universidad Nacional de La Plata, Argentina**. Conjuntamente con el Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA) de la Universidad Nacional de La Plata (Argentina) se vienen realizando desde hace tiempo trabajos de investigación para mejorar los procesos de la metodología NDT, concretamente en el proceso de ingeniería de requisitos en lo que tiene que ver con la conciliación de los requerimientos. Estos trabajos conjuntos han propiciado el establecimiento de convenios de colaboración para la realización de estancias y proyectos internacionales, así como tesis doctorales conjuntas.

- **Le Laboratoire Cognitions Humaine et Artificielle, Universite Paris 8**. Este grupo de investigación está formado por un equipo multidisciplinar que investiga sistemas cognitivos naturales y artificiales a través de diferentes estudios pragmáticos y semánticos, con el propósito de modelar procesos cognitivos implicados en el aprendizaje durante toda la vida del individuo. Enmarcado en el proyecto Merimée, que tiene por objetivo la colaboración entre estudiantes de doctorados franceses, se han establecido trabajo de investigación con el objetivo de entrelazar esta línea con los procesos de negocio.

- **La División de Logística del Instituto Tecnológico de Aragón (ITAINNOVA)** Los trabajos relacionados con los procesos de negocio dentro del grupo IWT2 han motivado

[1]La Fundación FISEVI surge con el objetivo de dar servicio a los investigadores de los centros del Sistema Sanitario Público de Andalucía en la provincia de Sevilla, aglutinando las actividades que anteriormente eran desarrolladas por otras entidades de gestión. Su cartera de servicios se compone de todo un conjunto de actividades de apoyo al desarrollo de la investigación en Salud en el seno del Sistema Sanitario Público de Andalucía, así como la gestión de ayudas y contratos que permiten las actividades de investigación de los diferentes grupos de los centros de dicho Sistema Sanitario ubicados en la provincia de Sevilla.

un marco de colaboración entre éste y la división de Logística de ITAINNOVA, en concreto con su línea de investigación MDE, puesto que uno de los trabajos en esta línea, apoyada en el desarrollo de una tesis dentro del grupo, trata sobre los procesos de una cadena de suministro, modelando las colaboraciones entre los diferentes agentes de la cadena con un lenguaje de modelado de procesos que incorpore la capacidad semántica para expresar de forma adecuada las necesidades que ahí se plantean.

- **Southampton Solent University**. En los últimos años se viene trabajando en estrecha colaboración con la Doctora emérita Margaret Ross en el ámbito de la calidad del software, lo que ha propiciado la realización de estancias en dicha universidad y la revisión por su parte tanto de los trabajos de investigación, y así como de los artículos desarrollados.

Todos estos son ejemplos de colaboración con otros grupos de investigación, organismos, entidades públicas y empresas en los que la línea estratégica de procesos de negocio del grupo IWT2 tiene especial protagonismo y con los que, de alguna u otra forma, se ha trabajado en el desarrollo de esta tesis, tal y como queda de manifiesto en el anexo E donde se recopilan los principales trabajos, artículos, proyectos y estancias de la doctoranda.

El contexto dentro del grupo IWT2 y las relaciones con otros grupos han conformado el marco estratégico en el que se ha desarrollado este trabajo de tesis, que ha constituido una importante influencia a la hora de afrontar las soluciones planteadas y la manera en la que se han llevado a cabo, de ahí el hecho de que lo hayamos destacado como un aspecto relevante. Una vez precisado este marco contextual, en la siguiente sección se exponen las principales aportaciones obtenidas en el desarrollo de este trabajo de tesis.

9.2. Aportaciones de esta tesis

Una vez descrito y documentado el trabajo realizado durante la realización de esta tesis doctoral y expuesto el marco de trabajo en el que se ha llevado a cabo, es momento para recapitular las principales aportaciones obtenidas como resultado de la misma. En cada una de estas aportaciones haremos referencia al objetivo que al comienzo del trabajo se planteó para observar que existe al menos una correspondencia para todos los objetivos, con lo que se puede concluir que todos ellos han sido cubiertos.

9.2.1. Un estudio del estado del arte de los lenguajes de modelado de procesos de software

Conocer cuál era la situación actual en cuanto a los lenguajes de modelado de procesos de software se refiere, era algo fundamental en el desarrollo de esta investigación, puesto que el planteamiento seguramente habría sido diferente en el caso de existir un número más reducido de propuestas. Para conocer el estado del arte se realizó una revisión sistemática de la literatura sobre los trabajos relacionados con los lenguajes de modelado de procesos de software existentes

en la última década. En este estudio se realizó una clasificación de los mismos de acuerdo a una nueva taxonomía y se detectó la tendencia actual en el desarrollo de nuevos lenguajes de modelado de procesos de software: hacer uso de la potencia ofrecida por los modelos, ya que ofrecen una capacidad de abstracción mayor que la de los históricos enfoques como los lenguajes de programación, las redes de Petri o las reglas. Además, se recogieron las limitaciones del estudio y principales líneas abiertas en cada uno de estos trabajos, algo muy destacable porque la revisión sistemática de la literatura ofrece un punto de partida para continuar la investigación en este campo. Los resultados de dicho estudio se publicaron en [García-Borgoñón *et al.*, 2014a].

Con esta aportación se cubre el primero de los objetivos planteados en el capítulo 3 que, tal y como estaba formulado, consistía en *Realizar un estudio del estado del arte de los diferentes lenguajes de modelado de procesos de software existentes, las causas por las que han sido creados y cuáles son los aspectos que llevan a las organizaciones a su utilización en entornos reales, si es que esto ocurre.*

Además de conocer la situación actual en cuanto a los lenguajes de modelado de procesos de software se refiere, gracias a este estudio se ha llevado a cabo una recopilación de los principales requerimientos exigidos a estos lenguajes en la literatura. Al hacer la revisión sistemática nos percatamos de que unos de los elementos más repetidos en los diferentes trabajos encontrados eran, precisamente, los requerimientos que se exigen a un lenguaje de modelado de procesos de software. Esto se convertía en un argumento de peso para la creación o propuesta de un nuevo lenguaje o, simplemente, la extensión de alguno de los existentes. Sin embargo, aunque el elemento referido era común, el conjunto de requisitos que cada trabajo proporcionaba difería entre ellos y tampoco tenían una homogeneidad en cuanto a la denominación aportada. Por ello, consideramos adecuado realizar una compilación de los requerimientos que fuimos detectando en el transcurso del desarrollo de la revisión sistemática de la literatura teniendo en cuenta tanto los trabajos finalmente incluidos en la misma como aquellos que fueron excluidos en sus últimas fases. Una vez realizada dicha recopilación, establecimos los términos que permitían armonizar los mismos aspectos introducidos por diferentes autores, con distintos nombres, de forma que se obtuvo como resultado un marco (*Framework*) para la evaluación de las capacidades de un lenguaje de modelado de procesos de software. Además de dar soporte a esta evaluación, este marco puede ser utilizado para realizar comparativas entre diferentes lenguajes.

9.2.2. Un marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los modelos de procesos de software

La principal aportación realizada con el desarrollo de esta tesis doctoral es, sin duda, el marco de referencia para facilitar la interoperabilidad y la mantenibilidad de los modelos de procesos de software, ya que constituye el resultado principal de la misma. Este marco, utilizando la metáfora del esperanto tal y como se ha comentado en el capítulo 3, hace posible que cada organización elija y trabaje en el lenguaje de modelado de procesos de software que más satisfaga sus necesidades, sin tener por ello que convertir esta elección en algo irrevocable e indisoluble. Hay que destacar la gran influencia que en el desarrollo de dicho marco ha tenido la ingeniería dirigida por modelos, puesto que ha condicionado en gran medida las soluciones adoptadas.

Para conseguir dicho objetivo, hemos comentado en múltiples ocasiones que el marco se sustenta en tres pilas fundamentales: un lenguaje de modelado de procesos de software propuesto como lenguaje base al que hemos denominado INROMA, un método que permite su utilización como lenguaje de referencia y las transformaciones necesarias para convertir INROMA en el nexo entre los lenguajes de modelado de procesos de software que participan en el marco de referencia.

1. **El lenguaje de modelado de procesos de software INROMA**. Este lenguaje, como su propio nombre indica, ha sido definido para dar soporte al marco de referencia en su misión de facilitar la interoperabilidad y mantenibilidad (INROMA, INteROperabilidad y MAntenibilidad). Teniendo en cuenta el enfoque basado en modelos que se había establecido a la hora de plantear el marco, INROMA se desarrolló utilizando ese mismo paradigma, obteniendo como resultado una sintaxis abstracta, el metamodelo, y una sintaxis concreta, gráfica y muy similar a los diagramas de actividad de UML.

   INROMA tiene como propiedad fundamental el ser el lenguaje de modelado de procesos de software que soporta el marco de referencia y, por tanto, incluye los mínimos elementos comunes a todos los lenguajes de modelado de procesos de software existentes, con el objetivo de garantizar la interoperabilidad y no privilegiar ningún lenguaje frente al resto. Para conseguirlo nos hemos basado en los conceptos definidos en la ISO/IEC TR 24744:2007, convirtiéndose en un lenguaje sencillo y de fácil adopción, muy cercano a la filosofía *esperantista* que se buscaba en la definición del marco. Un lenguaje al fin y al cabo, que hemos evaluado utilizando el marco para la evaluación de las capacidades de los lenguajes de modelado de procesos de software. Otra característica intrínseca en INROMA, al estar basado en UML, es su capacidad de extensión que, en entornos diferentes al de la interoperabilidad, puede ser explotado en toda su magnitud, incorporando capacidades que actualmente el lenguaje no contempla. Los resultados de la definición y evolución de INROMA ha sido publicado en varias revistas y conferencias [García-Borgoñón *et al.*, 2013b] [Ponce *et al.*, 2013] [García-Borgoñón *et al.*, 2013a], sujetas a revisiones de expertos externos, que nos ha permitido aprender e incorporar muchas de sus recomendaciones para la generación de la versión final del mismo.

   Para finalizar lo que hace referencia a INROMA, destacar que con su desarrollo se ha cubierto el segundo de los objetivos planteados, que consistía en *Definir un lenguaje de modelado de procesos de software que sirva de lenguaje base en nuestro marco de referencia.*

2. **Un método que permite la utilización de INROMA como lenguaje base**. Con la definición de dicho método se garantiza la posibilidad de incorporar nuevos lenguajes de modelado de procesos de software en el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos. El principal aspecto a tener en cuenta en la utilización del método son las correspondencias entre los elementos que conforman el metamodelo de INROMA y los del lenguaje de modelado de procesos de software que se pretende incorporar al marco. A modo de guía para el uso de dicho método se han realizado tres ejemplos con lenguajes de modelado de procesos: dos muy habituales en la industria hoy en día, como son BPMN y UML-DA, de los que ya teníamos disponible y accesible el metamodelo, y otro, IDEF3, más específico del campo de la aeronáutica, del que hemos definido su metamodelo para poder utilizarlo en el marco de referencia.

Con la definición de este método, el objetivo *Definir un método para incorporar nuevos lenguajes de modelado de procesos de software al marco de referencia* se ha alcanzando.

3. **Las transformaciones dentro del marco de referencia**. Además del lenguaje y del método, para completar el marco de referencia era necesario establecer la manera en la que todos estos elementos se enlazan para realmente facilitar la interoperabilidad y la mantenibilidad de los procesos de software. Para ello se sugiere la forma en la que estas transformaciones deben ser definidas, dado el carácter bidireccional de las mismas, utilizando el estándar QVT Relations para ello. Al igual que sucede en el caso del método, a modo de guía se han utilizado los tres lenguajes previamente elegidos como ejemplo para la definición de las transformaciones.

   Así, con el desarrollo de este último pilar del marco de referencia, el objetivo *Definir las transformaciones que constituyen el punto de unión de todos los lenguajes de modelado de procesos de software* se ha logrado.

Una vez definidos y desarrollados cada uno de los pilares del marco de referencia, el objetivo que engloba a todos ellos consistente en *Plantear el marco de referencia que se sustenta sobre estos tres elementos previamente defindos* también ha sido alcanzado.

9.2.3. Una herramienta que da soporte al marco de referencia: MONETA

También hemos comentado en repetidas ocasiones a lo largo de este documento que una de nuestras pretensiones principales era la de que este marco pudiera ser llevado a la práctica y de él se beneficiaran en su utilización diferentes organizaciones de software, ya fueran empresas dedicadas al desarrollo y mantenimiento de software o departamentos de software dentro de empresas más grandes. Para ello se planteó la necesidad de desarrollar una herramienta de soporte que implementara el marco de referencia y que abarcara los tres pilares fundamentales que lo conforman. La herramienta resultante, denominada MONETA, fue desarrollada tomando como base el entorno de desarrollo de Enterprise Architect, dadas las prestaciones que nos proporcionaba.

Enterprise Architect incorpora los lenguajes BPMN y UML-DA para el modelado de procesos, lo que nos permitía poder utilizarlos directamente en MONETA, pero no es el caso común entre los lenguajes de modelado de procesos. Por ejemplo, el otro lenguaje utilizado a lo largo de este documento, IDEF3, no está incluido por defecto en la versión comercial. Para la implementación en MONETA se han desarrollado los perfiles UML de IDEF3 e INROMA, con el objetivo de hacer uso de la misma en la validación de las situaciones reales planteadas a la hora es proponer la solución. Además de estos perfiles, que podríamos considerarlos como la implementación de las sintaxis abstracta de estos lenguajes, también ha sido necesario definir sus sintaxis concretas. Con ello se conseguía tener soporte para los dos primeros pilares del marco de referencia.

En cuanto a las transformaciones, también se ha definido un perfil UML para el lenguaje QVT Relations para las mismas, así como el desarrollo concreto para su ejecución mediante el

uso de las capacidades que nos proporciona Enterprise Architect a través de *Add-ins*, lo que erige a MONETA en la herramienta de soporte para el marco de referencia.

Una vez finalizado el desarrollo de MONETA, se ha utilizado para la validación del marco de referencia con los tres casos de uso teóricos planteados al inicio de este trabajo de tesis. De esta forma se cubre el objetivo de *Implementar una herramienta de soporte para que el marco de referencia pueda ser utilizado en la práctica por organizaciones de software.*

9.2.4. Validación del marco de referencia con casos reales

Una vez desarrollados el marco de referencia y la herramienta de soporte, el último objetivo a cumplir era *Validar el marco de referencia con casos de estudio reales*, como una forma de corroborar las hipótesis que se habían planteado a la hora de formular el problema y poder observar que la solución propuesta cumplía con las expectativas planteadas.

La utilización de casos reales derivados de proyectos con empresas para realizar esta validación, no sólo ha servido para alcanzar dicho objetivo sino que se han abierto nuevas ideas y líneas de trabajo entorno a esta temática y con una importante carga de investigación aplicada a la industria, con la continuidad que esto supone en la temática de los procesos.

Todas y cada una de estas aportaciones nos han permitido alcanzar los objetivos planteados al comienzo del trabajo de tesis, reforzando las hipótesis y evidenciado las principales fortalezas del marco de referencia para facilitar la interoperabilidad y mantenibilidad de los procesos de software aquí expuesto.

9.3. Trabajos futuros

Una vez recapituladas las principales aportaciones realizadas con el desarrollo de este trabajo de tesis, es importante poner en valor que dicho trabajo no es algo estanco, sino que sus resultados, en confluencia con los otros trabajos del grupo IWT2 que hemos comentado anteriormente, han propiciado la apertura de nuevas ideas y líneas para avanzar en esta temática. A continuación se plantean algunos de los trabajos futuros más representativos en el ámbito de los procesos.

9.3.1. La incorporación de nuevos lenguajes en el marco de referencia

Para poder utilizar el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los lenguajes de modelado de procesos de software es necesario incorporar esos lenguajes mediante el método definido en el marco. Para poder utilizar el marco y validarlo en entornos reales, durante el desarrollo de este trabajo de tesis se incorporaron, a modo de ejemplo, los lenguajes de modelado de procesos BPMN, los diagramas de actividad de UML e IDEF3. Hemos

podido comprobar que los dos primeros son habitualmente utilizados en las organizaciones para modelar sus procesos de negocio, independientemente de si este negocio es software u otro, e IDEF3 tiene un uso bastante extendido en la industria aeronáutica. Sin embargo, para poder obtener mayor rendimiento de este marco de referencia, un mayor número de lenguajes deberían ser incorporados.

De hecho, se podría ampliar el alcance en cuanto al tipo de procesos se refiere, y abrir las puertas a su utilización en áreas como la energía, la automoción o la salud, puesto que hemos podido comprobar que el marco de referencia cumple las expectativas para las que fue desarrollado en un ámbito más general de procesos de negocio, sin particularizar únicamente en el caso de los procesos de software.

9.3.2. La definición de un mecanismo para asegurar la calidad de los procesos y los productos relacionados con ellos en entornos empresariales

Los procesos, una vez definidos y modelados, se convierten en uno de los principales activos de una organización. Ellos aspiran a ser un reflejo operativo de las particularidades e idiosincrasia de la propia organización. La mayor parte de estos procesos, a excepción de aquellos que son transversales a la organización, se ponen en marcha durante la ejecución de los proyectos. Sin embargo, es una situación bastante habitual encontrarse con que existe una notable distancia entre el proceso definido (o modelado) por los ingenieros de procesos y el proceso real (o actual) que verdaderamente se está utilizando en el desarrollo de los proyectos. Esta cuestión suele ser solventada con la instauración de una figura o rol dentro de una empresa que se encarga del aseguramiento de la calidad de los procesos, observando, habitualmente de forma manual, las diferencias entre los procesos definidos y los reales en las evidencias (o productos procedentes de los procesos) obtenidas de los proyectos en curso y finalizados. Estas actividades, que suponen un coste para las empresas, son asumidas dentro de lo que se llama coste de la calidad.

Más allá de disponer de un mecanismo de orquestación y ejecución de los procesos modelados, el enfoque que se otorga a esta línea de trabajo consiste en establecer una forma de asegurar el seguimiento de los procesos mediante la aplicación de técnicas de testeo de software. Habitualmente estas técnicas son utilizadas para comprobar la calidad de los productos resultado de los proyectos, pero rara vez se utilizan para comprobar la calidad de los procesos definidos. De esta forma, estas actividades manuales pasarían a tener un soporte importante, eliminando en gran medida la subjetividad de una tarea meramente manual y personal.

9.3.3. La definición de un mecanismo de medición, análisis y toma de decisiones en base a datos de ejecución de los procesos

Otro aspecto importante al que dirigir una línea de trabajo consiste en la disponibilidad en los procesos de una organización de los elementos necesarios para establecer indicadores, a partir de métricas y mediciones fruto de la ejecución de los procesos en el desarrollo de los proyectos, y utilizar dichas métricas para la toma de decisiones dentro de una organización.

Ampliar y extender los conceptos de métricas e indicadores introducidos en el lenguaje INROMA que permitan tener en cuenta estos aspectos a la hora de modelar los procesos, va a permitir la obtención de un cuadro de mando organizacional que, unidos con entornos de gestión estadística de los procesos como Six Sigma o Lean, podrían ser de gran utilidad en las empresas, de ahí su importancia como línea de investigación aplicada. Unido a este enfoque se podría contemplar la incorporación de aspectos de simulación de los procesos a partir de su modelado como otro elemento a incorporar en el análisis y toma de decisiones.

9.3.4. La definición de un mecanismo que facilite las auditorias y certificaciones de calidad de los procesos

Habitualmente las organizaciones, una vez definidos e institucionalizados los procesos, suelen querer acceder a las certificaciones y auditorías que acreditan la obtención de alguna norma o estándar por la importante imagen comercial que éstas suponen. Sin embargo, las tareas relacionadas específicamente a la preparación y ejecución de estas auditorías es un coste bastante elevado para cualquier organización, puesto que los objetivos de estas tareas están dirigidos a recopilar las evidencias necesarias para poder pasar con éxito las evaluaciones de tales certificaciones.

Una línea de trabajo en este sentido consistiría en marcar en los procesos a la hora de modelarlos los aspectos susceptibles de ser tomados como evidencias a la hora de realizar las certificaciones, estableciendo un marco de evaluación en el que se pudieran incorporar diferentes normas y estándares de todo tipo. Los resultados de este trabajo podrían tener gran aceptación por parte de las empresas, puesto que suelen aceptar a regañadientes los costes internos asociados a estas evaluaciones sumados a los propios costes de las mismas.

9.3.5. La definición de un mecanismo para la interacción de procesos en la cadena de suministro

En esta línea se plantea la utilización de los procesos de negocio en el ámbito de la cadena de suministro. Hoy en día la mayoría de productos y servicios existentes son fruto de la colaboración de un gran número de compañías diferentes que forman una cadena de valor. Cada organización que forma parte de la cadena ofrece, de alguna u otra forma, valor al cliente final, que se materializa como resultado de la colaboración efectiva de los miembros de la cadena. Por ello es necesario que, manteniendo la privacidad y autonomía en los procesos individuales, cada organización conozca qué papel desempeña en una cadena de suministro y cómo puede o le pueden afectar las decisiones tomadas por otras empresas con las que colabora en la cadena de valor.

En este contexto, la línea de trabajo que se plantea y que se encuentra ya en desarrollo se centra en, tomando como base el lenguaje INROMA, ser capaces de reutilizar modelos de procesos de negocio creados en diversos lenguajes para crear, en un primer paso, un modelo de interacción de procesos que especifique cómo un proceso se puede relacionar con otros, y en un

segundo paso, definir un modelo de proceso de cadena de suministro en el que se aborden las actividades con un enfoque inter-organizacional.

Las líneas de trabajo futuras aquí referidas son las más representativas actualmente, pero cabe destacar que la sincronización de los resultados obtenidos en dos trabajos que se están desarrollando en el grupo IWT2 de forma simultánea, ampliaría en gran medida horizontes y surgirían nuevas ideas que explorar.

9.4. Conclusiones

A lo largo de todo esta memoria hemos presentado el trabajo realizado en los diferentes capítulos que la componen, desde la definición del problema basada en los resultados obtenidos en la revisión sistemática de literatura, hasta la propuesta de solución, inicialmente desde un punto de vista más teórico, para llegar a un enfoque totalmente práctico con la herramienta de soporte y la manifestación de su utilidad mediante la exposición de los casos de estudio.

En este capítulo han quedado reflejado el contexto dentro del grupo IWT2 en el que se ha desarrollado esta tesis doctoral, lo que ha influido notablemente en los enfoques establecidos durante la ejecución del trabajo de tesis. Asimismo, se han destacado las principales relaciones con otros grupos de investigación y empresas con los que se trabaja en el ámbito de los procesos de negocio, conformando todo ello un marco estratégico para esta línea de investigación. Además de las principales aportaciones realizadas por esta tesis doctoral, dentro de este marco estratégico se han abierto un conjunto de ideas de trabajo futuro que complementarían los resultados aquí obtenidos.

Bibliografía

[Ahern *et al.*, 2008] Ahern, D. M., Clouse, A., y Turner, R. (2008). *CMMI distilled: a practical introduction to integrated process improvement.* Addison-Wesley Professional.

[Aoussat *et al.*, 2011] Aoussat, F., Oussalah, M., y Nacer, M. A. (2011). SPEM extension with software process architectural concepts. En *Proceedings of the International Computers, Software and Applications Conference (COMPSAC 2011)*, pp. 215 – 223, Munich, Germany.

[Atkinson *et al.*, 2004] Atkinson, D., Weeks, D., y Noll, J. (2004). The design of evolutionary process modeling languages. En *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, pp. 73 – 82.

[Atkinson *et al.*, 2007] Atkinson, D., Weeks, D., y Noll, J. (2007). Tool support for iterative software process modeling. *Information and Software Technology*, 49(5):493–514.

[Ayed *et al.*, 2012] Ayed, H., Vanderose, B., y Habra, N. (2012). A metamodel-based approach for customizing and assessing agile methods. En *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012)*, pp. 66–74.

[Banhesse *et al.*, 2012] Banhesse, E. L., Salviano, C. F., y Jino, M. (2012). Towards a metamodel for integrating multiple models for process improvement. *Proceedings of the 38th Euromicro Conference On Software Engineering and Advanced Applications (SEAA 2012).*

[Bendraou, 2007] Bendraou, R. (2007). *UML4SPM: Un Langage De Modélisation De Procédés De Développement Logiciel Exécutable Et Orienté Modèle.* Tesis doctoral, Paris 6.

[Bendraou *et al.*, 2007a] Bendraou, R., Combemale, B., Crégut, X., y Gervais, M. (2007a). Definition of an Executable SPEM 2.0. En *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007)*, pp. 390–397. IEEE.

[Bendraou y Gervais, 2007] Bendraou, R. y Gervais, M.-P. (2007). A Framework for Classifying and Comparing Process Technology Domains. En *Proceedings of the International Conference on Software Engineering Advances (ICSEA 2007)*, p. 5.

[Bendraou *et al.*, 2005] Bendraou, R., Gervais, M.-P., y Blanc, X. (2005). UML4SPM: A UML2.0-based metamodel for software process modelling. En *Model Driven Engineering Languages and Systems*, pp. 17–38. Springer.

[Bendraou *et al.*, 2006] Bendraou, R., Gervais, M.-P., y Blanc, X. (2006). UML4SPM: An executable software process modeling language providing high-level abstractions. En *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006)*, pp. 297–306. IEEE.

[Bendraou *et al.*, 2009] Bendraou, R., Jezéquél, J.-M., y Fleurey, F. (2009). Combining aspect and model-driven engineering approaches for software process modeling and execution. En *Trustworthy Software Development Processes*, pp. 148–160. Springer.

[Bendraou *et al.*, 2012] Bendraou, R., Jezéquél, J.-M., y Fleurey, F. (2012). Achieving process modeling and execution through the combination of aspect and model-driven engineering approaches. *Journal of Software: Evolution and Process*, 24(7):765–781.

[Bendraou *et al.*, 2010] Bendraou, R., Jezéquél, J.-M., Gervais, M.-P., y Blanc, X. (2010). A Comparison of Six UML-Based Languages for Software Process Modeling. *IEEE Transactions on Software Engineering*, 36(5):662–675.

[Bendraou *et al.*, 2007b] Bendraou, R., Sadovykh, A., Gervais, M.-P., y Blanc, X. (2007b). Software process modeling and execution: The UML4SPM to WS-BPEL approach. En *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)*, pp. 314 – 321.

[Berger *et al.*, 2010] Berger, S., Grossmann, G., Stumptner, M., y Schrefl, M. (2010). *Metamodel-based information integration at industrial scale*. Springer.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2):171–188.

[Birkmeier y Overhage, 2010] Birkmeier, D. y Overhage, S. (2010). Is BPMN really first choice in joint architecture development? an empirical study on the usability of BPMN and UML activity diagrams for business users. En *Research into Practice–Reality and Gaps*, pp. 119–134. Springer.

[BPMI.org y OMG, 2006] BPMI.org y OMG (2006). Business Process Modeling Notation Specification. Final Adopted Specification.

[Brambilla *et al.*, 2012] Brambilla, M., Cabot, J., y Wimmer, M. (2012). *Model-driven software engineering in practice*. Morgan & Claypool Publishers.

[Briand *et al.*, 2005] Briand, L. C., Labiche, Y., Di Penta, M., y Yan-Bondoc, H. (2005). An experimental investigation of formality in UML-based development. *IEEE Transactions on Software Engineering*, 31(10):833–849.

[Bruneliere *et al.*, 2010] Bruneliere, H., Cabot, J., Clasen, C., Jouault, F., y Bézivin, J. (2010). Towards model driven tool interoperability: bridging eclipse and microsoft modeling tools. En *Modelling foundations and applications*, pp. 32–47. Springer.

[Cares *et al.*, 2006] Cares, C., Mayol, E., Franch, X., y Alvarez, E. (2006). Goal-driven agent-oriented software processes. En *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2006)*, pp. 336 – 343.

[Castro *et al.*, 2014] Castro, S., García, V., y Crespo, R. (2014). Software processes and methodologies modeling language spmml, a holistic solution for software engineering. *IEEE Latin America Transactions*, 12(4):818–824.

[Chaghrouchni *et al.*, 2014] Chaghrouchni, T., Kabbaj, I., y Bakkoury, Z. (2014). Towards dynamic adaptation of the software process. En *Proceedings of the 9th International Conference on Intelligent Systems: Theories and Applications (SITA 2014)*, pp. 1–7.

[Chen *et al.*, 2002] Chen, C., Shen, B.-J., y Gu, Y.-Q. (2002). Flexible and formalized process modeling language. *Journal of Software*, 13(8):1374 – 1381.

[Chen *et al.*, 2000] Chen, J., Chou, S., y Liu, W. (2000). APER-2: A developer-centered, object-oriented process language. En *Proceedings of the International Symposium on Multimedia Software Engineering*, pp. 297–303.

[Chou, 2004] Chou, S.-C. (2004). DPEM: A decentralized software process enactment model. *Information and Software Technology*, 46(6):383 – 395.

[Chrissis *et al.*, 2011] Chrissis, M. B., Konrad, M., y Shrum, S. (2011). *CMMI for development: guidelines for process integration and product improvement*. Pearson Education.

[Cibran, 2009] Cibran, M. A. (2009). Translating BPMN Models into UML Activities. En *Business Process Management Workshops*, pp. 236–247. Springer.

[Combemale *et al.*, 2006] Combemale, B., Cregut, X., Caplain, A., y Coulette, B. (2006). Towards a rigorous process modeling with SPEM. En *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, volumen ISAS, pp. 530 – 533.

[Curtis *et al.*, 1992] Curtis, B., Kellner, M. I., y Over, J. (1992). Process modeling. *Communications of the ACM*, 35(9):75–90.

[Cutilla *et al.*, 2011] Cutilla, C., García-García, J., Alba, M., Escalona, M., Ponce, J., y Rodriguez, L. (2011). Aplicacion del paradigma MDE para la generacion de pruebas funcionales; Experiencia dentro del proyecto AQUA-WS. En *Actas de la Sexta Conferencia Iberica de Sistemas e Tecnologias de InformaÃ§ao*.

[Czarnecki y Helsen, 2006] Czarnecki, K. y Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

[Da Silva *et al.*, 2011] Da Silva, F., Santos, A., Soares, S., França, C., Monteiro, C., y Maciel, F. F. (2011). Six years of systematic literature reviews in software engineering: An updated tertiary study. *Information and Software Technology*, 53(9):899–913.

[Debnath *et al.*, 2006] Debnath, N., Riesco, D., Montejano, G., Perez-Cota, M., Garcia, B. J., Romero, D., y Uva, M. (2006). Supporting the SPEM with a UML extended workflow metamodel. En *Proceedings of the IEEE International Conference on Computer Systems and Applications (AICCSA 2006)*, volumen 2006, pp. 1151 – 1154.

[Del Fabro *et al.*, 2005] Del Fabro, Marcos Didonet and Bézivin, Jean and Jouault, Frédéric and Breton, Erwann and Gueltas, Guillaume and others (2005). AMW: a generic model weaver. *Journées sur l'Ingénierie Dirigée par les Modèles*, pp. 105–114.

[Demuth, 2012] Demuth, A. (2012). Enabling dynamic metamodels through constraint-driven modeling. En *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, pp. 1622–1624.

[Di Nitto *et al.*, 2002] Di Nitto, E., Lavazza, L., Schiavoni, M., Tracanella, E., y Trombetta, M. (2002). Deriving executable process descriptions from UML. En *Proceedings of the 24rd International Conference on Software Engineering (ICSE 2002)*, pp. 155 – 165.

[Di Ruscio *et al.*, 2012] Di Ruscio, D., Eramo, R., y Pierantonio, A. (2012). Model transformations. En *Formal Methods for Model-Driven Engineering*, pp. 91–136. Springer.

[Ellner *et al.*, 2010] Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., y Philippsen, M. (2010). eSPEM – A SPEM Extension for Enactable Behavior Modeling. En *Modelling Foundations and Applications*, pp. 116–131. Springer.

[Ellner *et al.*, 2011] Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., y Philippsen, M. (2011). A FUML-based distributed execution machine for enacting software process models. En *Modelling Foundations and Applications*, pp. 19–34. Springer.

[Escalona, 2004] Escalona, M. J. (2004). *Modelos y tecnicas para la especificación y el análisis de la Navegación en Sistemas Software - European Thesis.* Tesis doctoral, Department of Computer Language and Systems. University of Seville.

[Escalona y Aragon, 2008] Escalona, M. J. y Aragon, G. (2008). NDT. A Model-Driven Approach for Web Requirements. *IEEE Transactions on Software Engineering*, 34(3):377–390.

[Evans y Kent, 1999] Evans, A. y Kent, S. (1999). Core meta-modelling semantics of UML: the pUML approach. En *UML99: The Unified Modeling Language*, pp. 140–155. Springer.

[Ferreira *et al.*, 2011] Ferreira, A. L., Machado, R. J., y Paulk, M. C. (2011). An Approach to Software Process Design and Implementation Using Transition Rules. En *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2011)*, pp. 330 –333.

[Fondement y Silaghi, 2004] Fondement, F. y Silaghi, R. (2004). Defining Model Driven Engineering Processes. En *Proceedings of the 3rd International Workshop in Software Model Engineering (WiSME 2004)*. Citeseer.

[Fontoura *et al.*, 2000] Fontoura, M., Pree, W., y Rumpe, B. (2000). *The UML profile for framework architectures.* Addison-Wesley Longman Publishing Co., Inc.

[Franch y Ribó, 2003] Franch, X. y Ribó, J. M. (2003). A UML-based approach to enhance reuse within process technology. En *Software Process Technology*, pp. 74–93. Springer.

[García *et al.*, 2014] García, M. T., Barcelona, M. A., García-Borgoñón, L., Escalona, M. J., Ruiz, M., y Ramos, I. (2014). A Discrete-Event Simulation Metamodel for obtaining Simulation Models from Business Process Models. En *Information System Development - Improving Enterprise Communication*, pp. 167–178. Springer.

[García-Borgoñón *et al.*, 2014a] García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Alba, M., y Escalona, M. J. (2014a). Software process modeling languages: A systematic literature review. *Information and Software Technology*, 56(2):103–116.

[García-Borgoñón *et al.*, 2014b] García-Borgoñón, L., Barcelona, M. A., Peña, P., y Escalona, M. J. (2014b). SoftAragón: a methodological framework for Software Process Improvement in SMEs. En *Proceedings of the 22nd International Conference on Software Quality Management (SQM 2014)*.

[García-Borgoñón *et al.*, 2014c] García-Borgoñón, L., Blanco, R., García-García, J. A., y Barcelona, M. A. (2014c). Applying testing techniques to software process assessment: A model-based perspective. En *Information System Development - Improving Enterprise Communication*, pp. 167–178. Springer.

[García-Borgoñón *et al.*, 2013a] García-Borgoñón, L., García-García, J. A., Alba, M., y Domínguez-Mayo, F. J. (2013a). Gestión de Procesos en Organizaciones de Software: Un Enfoque Basado en Modelos. En *Actas de las Jornadas de Ingenieria del Software y Bases de Datos (JISBD2013)*.

[García-Borgoñón *et al.*, 2013b] García-Borgoñón, L., García-García, J. A., Alba, M., y Escalona, M. J. (2013b). Software Process Management: A Model-Based Approach. En *Information System Development - Building Sustainable Information Systems*, pp. 167–178. Springer.

[García-García *et al.*, 2014] García-García, J. A., Escalona, M. J., Domínguez-Mayo, F. J., y Salido, A. (2014). NDT-Suite: A Methodological Tool Solution in the Model-Driven Engineering Paradigm. *Journal of Software Engineering and Applications*, 2014.

[García-García *et al.*, 2015] García-García, J. A., Escalona, M. J., Martínez-García, A., Aragón, G., Moreno-Conde, A., y Parra, C. (2015). Improving patient care through a clinical process management based on the MDE paradigm. *The Scientific World Journal. En revision*.

[García-García *et al.*, 2012] García-García, J. A., Ortega, M. A., García-Borgoñon, L., y Escalona, M. J. (2012). NDT-Suite: a model-based suite for the application of NDT. En *Web Engineering*, pp. 469–472. Springer.

[García-García *et al.*, 2013] García-García, J. A., Victorio, J., García-Borgoñon, L., Barcelona, M. A., Domínguez-Mayo, F. J., y Escalona, M. J. (2013). A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology. En *Proceedings of the 21st International Conference on Software Quality Management (SQM 2013)*.

[García Guzmán *et al.*, 2013] García Guzmán, J., Martín, D., Urbano, J., y de Amescua, A. (2013). Practical experiences in modelling software engineering practices: The project patterns approach. *Software Quality Journal*, 21(2):325–354.

[García-Molina *et al.*, 2013] García-Molina, J., García-Rubio, F., Pelechano, V., Vallecillo, A., Vara, J., y Vicente-Chicote, C. (2013). *Desarrollo de Software Dirigido por Modelos: Conceptos, Metodos y Herramientas*.

[Gardner *et al.*, 2003] Gardner, T., Griffin, C., Koehler, J., y Hauser, R. (2003). A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard. En *MetaModelling for MDA Workshop*, pp. 178–197. Citeseer.

[Ge *et al.*, 2006] Ge, J., Hu, H., Gu, Q., y Lu, J. (2006). Modeling multi-view software process with object Petri nets. En *Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006)*, pp. 41–41. IEEE.

[Ge *et al.*, 2008] Ge, J., Hu, H., y Lu, J. (2008). Order Constraints for Multi-view Software Process Model. En *Proceedings of the International Conference on Computer Science and Software Engineering (CSSE 2008)*, volumen 2, pp. 639 –642.

[Golra y Dagnat, 2011] Golra, F. y Dagnat, F. (2011). Component-oriented Multi-metamodel Process Modeling Framework (CoMProM). En *First Workshop on Process-based approaches for Model-Driven Engineering (PMDE 2011)*, p. 44.

[Hagen M., 2004] Hagen M., G. V. (2004). Towards flexible software processes by using process patterns. En *Proceedings of the 8th IASTED International Conference on Software Engineering and Applications (SEA2004)*, pp. 436–441.

[Henderson-Sellers y Gonzalez-Perez, 2005] Henderson-Sellers, B. y Gonzalez-Perez, C. (2005). A comparison of four process metamodels and the creation of a new generic standard. *Information and Software Technology*, 47(1):49–65.

[Hsueh *et al.*, 2008] Hsueh, N.-L., Shen, W.-H., Yang, Z.-W., y Yang, D.-L. (2008). Applying UML and software simulation for process definition, verification, and validation. *Information and Software Technology*, 50(9-10):897 – 911.

[Humphrey, 1988] Humphrey, W. S. (1988). The software engineering process: definition and scope. En *Notes of the ACM SIGSOFT Software Engineering*, volumen 14, pp. 82–83.

[IDEF, 1995] IDEF (1995). Integration DEFinition, http://www.idef.com.

[IEEE, 1990] IEEE (1990). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. Institute of Electrical and Electronics Engineers.

[IKV++, 2011] IKV++ (2011). QVTMedini.

[ISO/IEC, 2007a] ISO/IEC (2007a). ISO/IEC 24744:2007 Software Engineering – Metamodel for Development Methodologies. International Organization for Standardization.

[ISO/IEC, 2007b] ISO/IEC (2007b). ISO/IEC TR 24744:2007 Software and systems engineering – Life cycle management – Guidelines for process description. International Organization for Standardization.

[ISO/IEC, 2008] ISO/IEC (2008). ISO/IEC 9001:2008 Quality management systems, Requirements. International Organization for Standardization.

[ISO/IEC, 2009] ISO/IEC (2009). ISO/IEC 27000:2009 Information technology, Security techniques, Information security management systems and Overview and vocabulary. International Organization for Standardization.

[ISO/IEC, 2014] ISO/IEC (2014). ISO/IEC 29119 Software Testing. The new international software testing standard. International Organization for Standardization.

[Istoan, 2014] Istoan, P. A. (2014). *Methodology for the derivation of product behaviour in a Software Product Line.* Tesis doctoral, Rennes 1.

[ITIL, 2014] ITIL (2014). Information Technology Infrastructure Library v3, http://www.itil-officialsite.com.

[IWT2 y Consejeria de Cultura de la Junta de Andalucia, 2013] IWT2 y Consejeria de Cultura de la Junta de Andalucia (2013). Comparativa de herramientas de modelado. Proyecto de Calidad.

[Jablonski *et al.*, 2008] Jablonski, S., Volz, B., y Dornstauder, S. (2008). A Meta Modeling Framework for Domain Specific Process Management. En *Proceedings of the 32nd Annual IEEE International Computer Software and Applications (COMPSAC 2008)*, pp. 1011 –1016.

[Jouault *et al.*, 2008] Jouault, F., Allilaire, F., Bézivin, J., y Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1):31–39.

[Jouault y Kurtev, 2006] Jouault, F. y Kurtev, I. (2006). Transforming models with ATL. En *Satellite Events at the MoDELS 2005 Conference*, pp. 128–138. Springer.

[Kang *et al.*, 2008] Kang, H., Dai, F., y Huang, B. (2008). Evolution process component description language. En *Proceedings of the International Conference on MultiMedia and Information Technology (MMIT 2008)*, pp. 306 – 309.

[Kedji *et al.*, 2014] Kedji, K. A., Lbath, R., Coulette, B., Nassar, M., Baresse, L., y Racaru, F. (2014). Supporting collaborative development using process models: a tooled integration-focused approach. *Journal of Software: Evolution and Process*.

[Kellner *et al.*, 1991] Kellner, M., Feiler, P., Finkelstein, A., Katayama, T., Osterweil, L., Penedo, M., y Rombach, D. (1991). *ISPW-6 software process example*. IEEE Computer Society Press.

[Kerzazi *et al.*, 2013] Kerzazi, N., Lavallée, M., y Robillard, P.-N. (2013). A knowledge-based perspective for software process modeling. *E-Informatica Software Engineering Journal*, 7(1):25–33.

[Kim *et al.*, 2003] Kim, C.-H., Weston, R. H., Hodgson, A., y Lee, K.-H. (2003). The complementary use of idef and uml modelling approaches. *Computers in Industry*, 50(1):35–56.

[Kitchenham y Brereton, 2013] Kitchenham, B. y Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049 – 2075.

[Kitchenham y Charters, 2007] Kitchenham, B. y Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

[Kitchenham *et al.*, 2010] Kitchenham, B., Pretorius, R., Budgen, D., Pearl Brereton, O., Turner, M., Niazi, M., y Linkman, S. (2010). Systematic literature reviews in software engineering–a tertiary study. *Information and Software Technology*, 52(8):792–805.

[Kleppe *et al.*, 2003] Kleppe, A. G., Warmer, J. B., y Bast, W. (2003). *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

[Kolovos *et al.*, 2008] Kolovos, D. S., Paige, R. F., y Polack, F. A. (2008). The epsilon transformation language. En *Theory and practice of model transformations*, pp. 46–60. Springer.

[Koudri y Champeau, 2010] Koudri, A. y Champeau, J. (2010). MODAL: A SPEM extension to improve co-design process models. En *New Modeling Concepts for Todayś Software Processes*, pp. 248–259. Springer.

[Kuhrmann *et al.*, 2013] Kuhrmann, M., Fernandez, D. M., y Steenweg, R. (2013). Systematic software process development: Where do we stand today? En *Proceedings of the 2013 International Conference ons Software and System Process (ICSSP 2013)*, pp. 166 – 170.

[Lee *et al.*, 2002] Lee, S., Shim, J., y Wu, C. (2002). A meta model approach using UML for task assignment policy in software process. En *Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC 2002)*, pp. 376 – 382.

[Lima Reis *et al.*, 2002] Lima Reis, C., Quites Reis, R., Abreu, M., Schlebbe, H., y Nunes, D. (2002). Flexible software process enactment support in the APSEE model. En *Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments (HCCLE 2002)*, pp. 112 – 121.

[Lonchamp, 1993] Lonchamp, J. (1993). A structured conceptual and terminological framework for software process engineering. En *Proceedings of the 2nd International Conference on the Continuous Software Process Improvement*, pp. 41–53.

[Maciel *et al.*, 2009] Maciel, R., da Silva, B., Magalhaes, P., y Rosa, N. (2009). An Integrated Approach for Model Driven Process Modeling and Enactment. En *Proceedings of the 23rd Brazilian Symposium on Software Engineering (SBES 2009)*, pp. 104 –114.

[Martınez-Ruiz *et al.*, 2011] Martınez-Ruiz, T., Garcıa, F., Piattini, M., y Munch, J. (2011). Modelling software process variability: an empirical study. *IET Software*, 5(2):172–187.

[Martinez-Ruiz *et al.*, 2011] Martinez-Ruiz, T., Garcia, F., Piattini, M., y Munch, J. (2011). Applying AOSE Concepts to Model Crosscutting Variability in Variant-Rich Processes. En *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2011)*, pp. 334 –338.

[Martinho *et al.*, 2008] Martinho, R., Varajao, J., y Domingos, D. (2008). A two-step approach for modelling flexibility in software processes. En *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pp. 427 – 430.

[Martinho *et al.*, 2009] Martinho, R., Varajao, J., y Domingos, D. (2009). FlexSPMF: A framework for modelling and learning flexibility in software processes. En *Visioning and Engineering the Knowledge Society. A Web Science Perspective*, pp. 78–87. Springer.

[Mas *et al.*, 2013] Mas, F., Rios, J., Menendez, J., y Gomez, A. (2013). A process-oriented approach to modeling the conceptual design of aircraft assembly lines. *The International Journal of Advanced Manufacturing Technology*, 67(1-4):771–784.

[McCartney, 1999] McCartney, S. (1999). *ENIAC: The Triumphs and Tragedies of the World's First Computer*. Walker & Company.

[McUmber y Cheng, 2001] McUmber, W. E. y Cheng, B. H. (2001). A general framework for formalizing uml with formal languages. En *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2011)*, pp. 433–442.

[Mellor, 2004] Mellor, S. J. (2004). *MDA distilled: principles of model-driven architecture.* Addison-Wesley Professional.

[Menzel y Mayer, 1998] Menzel, C. y Mayer, R. J. (1998). The IDEF family of languages. En *Handbook on architectures of information systems*, pp. 209–241. Springer.

[Metzger, 2005] Metzger, A. (2005). A systematic look at model transformations. En *Model-driven Software Development*, pp. 19–33. Springer.

[Min *et al.*, 2000] Min, S.-Y., Lee, H.-D., y Bae, D.-H. (2000). SoftPM: A software process management system reconciling formalism with easiness. *Information and Software Technology*, 42(1):1–16.

[Moreno y Vallecillo, 2008] Moreno, N. y Vallecillo, A. (2008). Towards interoperable web engineering methods. *Journal of the American Society for Information Science and Technology*, 59(7):1073–1092.

[Muller *et al.*, 2005] Muller, P.-A., Fleurey, F., y Jézéquel, J.-M. (2005). Weaving executability into object-oriented meta-languages. En *Model Driven Engineering Languages and Systems*, pp. 264–278. Springer.

[Noack, 2000] Noack, J. (2000). Extending the software development process with a toolkit of UML-centred techniques. En *Proceedings of the International Conference on Software Methods and Tools (SMT 2000)*, pp. 87–96.

[Noran, 2000] Noran, O. S. (2000). Business modelling: UML vs. IDEF. *School of CIT, Griffith University.*

[OMG, 2008] OMG (2008). Software & Systems Process Engineering Metamodel specification (SPEM)Version 2.0. Object Management Group. [Online; accessed 3-July-2013].

[OMG, 2010] OMG (2010). Business Process Model and Notation (BPMN) Version 2.0. Object Management Group. [Online; accessed 3-July-2014].

[OMG, 2011a] OMG (2011a). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (QVT) Version 1.1. Object Management Group. [Online; accessed 18-July-2014].

[OMG, 2011b] OMG (2011b). Unified Modeling Language (UML) - Superstructure. Version 2.4.1. Object Management Group. [Online; accessed 8-July-2013].

[OMG, 2012] OMG (2012). Object Constraint Language (OCL) Core Version 2.3.1. Object Management Group. [Online; accessed 8-June-2014].

[OMG, 2014a] OMG (2014a). Meta Object Facility (MOF) Core Version 2.4.2. Object Management Group. [Online; accessed 8-June-2014].

[OMG, 2014b] OMG (2014b). XML Metadata Interchange (XMI) Core Version 2.4.2. Object Management Group. [Online; accessed 8-June-2014].

[Osterweil, 1987] Osterweil, L. (1987). Software processes are software too. En *Proceedings of the 9th International Conference on Software Engineering (ICSE 1987)*, volumen 3, pp. 2–13.

[Pastor *et al.*, 2013] Pastor, O., Giachetti, G., Marín, B., y Valverde, F. (2013). Automating the Interoperability of Conceptual Models in Specific Development Domains. En *Domain Engineering*, pp. 349–373. Springer.

[Pillain *et al.*, 2011] Pillain, P.-Y., Champeau, J., y Tran, H. N. (2011). Towards an Enactment Mechanism for MODAL Process Models. En *Proceedings of the 1st Workshop on Process-based approaches for Model-Driven Engineering (PMDE 2011)*, p. 33.

[Pillat *et al.*, 2012] Pillat, R., Oliveira, T., y Fonseca, F. (2012). Introducing Software Process Tailoring to BPMN: BPMNt. En *Proceedings of the International Conference on Software and System Process (ICSSP 2012)*, pp. 58–62.

[PMI, 2008] PMI (2008). A Guide to the Project Management Body of Knowledge, Fourth Edition (PMBOK Guide). Project Management Institute.

[Podnar *et al.*, 2000] Podnar, I., Mikac, B., y Caric, A. (2000). SDL based approach to software process modeling. En *Software Process Technology*, volumen 1780, pp. 190–202.

[Ponce *et al.*, 2013] Ponce, J., García-Borgoñon, L., García-García, J. A., Escalona, M. J., Dominguez-Mayo, F. J., Alba, M., y Aragon, G. (2013). A Model-Driven Approach for Business Process Management. *Covenant University*.

[Pressman y Maxim, 2014] Pressman, R. S. y Maxim, B. (2014). *Ingeniería Del software un enfoque practico. Quinta edición*. MacGraw-Hill.

[Recker, 2010] Recker, J. (2010). Opportunities and constraints: the current struggle with BPMN. *Business Process Management Journal*, 16(1):181–201.

[Recker, 2012] Recker, J. (2012). *BPMN Research: What We Know and What We Donť Know*. Springer.

[Reis *et al.*, 2002] Reis, R. Q., Reis, C. A., Schlebbe, H., y Nunes, D. J. (2002). Early experiences on promoting explicit separation of details to improve software processes reusability. En *Proceedings of the IEEE International Conference on Computers, Software and Applications (COMPSAC 2002)*, pp. 373 – 378.

[Romeikat *et al.*, 2008] Romeikat, R., Roser, S., Müllender, P., y Bauer, B. (2008). Translation of QVT relations into QVT operational mappings. En *Theory and Practice of Model Transformations*, pp. 137–151. Springer.

[Rouille *et al.*, 2012] Rouille, E., Combemale, B., Barais, O., Touzet, D., y Jezequel, J.-M. (2012). Leveraging CVL to Manage Variability in Software Process Lines. En *Proceedings of the 19th Asia-Pacific Software Engineering Conference(APSEC 2012)*.

[Rouille *et al.*, 2013] Rouille, E., Combemale, B., Barais, O., Touzet, D., y Jezequel, J.-M. (2013). Integrating Software Process Reuse and Automation. En *Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC 2013)*, volumen 1, pp. 380–387.

[Ruiz *et al.*, 2003] Ruiz, F., Vizcaino, A., Garcia, F., y Piattini, M. (2003). Using XMI and MOF for representation and interchange of software processes. En *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pp. 739–744.

[Ruiz-González y Canfora, 2004] Ruiz-González, F. y Canfora, G. (2004). Software Process: Characteristics, Technology and Environments. *UPGrade, The European Journal for the Informatics Professional*, 5(5):6–10.

[Russell *et al.*, 2006] Russell, N., van der Aalst, W. M., Ter Hofstede, A. H., y Wohed, P. (2006). On the suitability of UML 2.0 activity diagrams for business process modelling. En *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006)*, pp. 95–104.

[Salido *et al.*, 2013] Salido, A., García-García, J. A., y Escalona, M. J. (2013). Managing CALIPSOneo Project: Learning from Trenches. En *Proceedings of the 23rd International Conference on Information Systems Development (ISD 2014)*.

[Schmidt, 2006] Schmidt, D. C. (2006). Model-driven engineering. *IEEE Computer*, 39(2):25.

[Schürr, 1995] Schürr, A. (1995). Specification of graph translators with triple graph grammars. pp. 151–163.

[Sommerville, 2007] Sommerville, I. (2007). *Software Engineering Eighth Edition*. Addison-Wesley.

[SparxSystems, 2014] SparxSystems (2014). Enterprise Architect.

[Strnadl, 2006] Strnadl, C. F. (2006). Aligning business and IT: The process-driven architecture model. *Information Systems Management*, 23(4):67–77.

[Sutton Jr. y Osterweil, 1997] Sutton Jr., S. M. y Osterweil, L. J. (1997). The Design of a Next-generation Process Language. *SIGSOFT Software Engineering0 Notes*, 22(6):142–158.

[Taentzer, 2004] Taentzer, G. (2004). AGG: A graph transformation environment for modeling and validation of software. En *Applications of Graph Transformations with Industrial Relevance*, pp. 446–453. Springer.

[The Eclipse Foundation, 2014a] The Eclipse Foundation (2014a). BPMN2 Modeler.

[The Eclipse Foundation, 2014b] The Eclipse Foundation (2014b). Eclipse Luna.

[The Eclipse Foundation, 2014c] The Eclipse Foundation (2014c). EuGENia.

[The Eclipse Foundation, 2015] The Eclipse Foundation (2015). Papyrus.

[Thiry y Thirion, 2009] Thiry, L. y Thirion, B. (2009). Functional metamodels for systems and software. *Journal of Systems and Software*, 82(7):1125–1136.

[Thomas y Nejmeh, 1992] Thomas, I. y Nejmeh, B. A. (1992). Definitions of tool integration for environments. *IEEE Software*, 9(2):29–35.

[Thu *et al.*, 2005] Thu, T. D., Nhi, T. H., Thuy, D. T. B., Coulette, B., y Cregut, X. (2005). Topological properties for characterizing well-formedness of process components. *Software Process Improvement and Practice*, 10(2):217 – 247.

[Tran *et al.*, 2007] Tran, H. N., Coulette, B., y Dong, B. T. (2007). Modeling process patterns and their application. En *Proceedings of the 2nd International Conference on Software Engineering Advances (ICSEA 2007)*.

[Vallecillo, 2010] Vallecillo, A. (2010). On the combination of domain specific modeling languages. En *Modelling Foundations and Applications*, pp. 305–320. Springer.

[Van Der Aalst *et al.*, 2003a] Van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., y Barros, A. P. (2003a). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.

[Van Der Aalst *et al.*, 2003b] Van Der Aalst, W. M., Ter Hofstede, A. H., y Weske, M. (2003b). Business process management: A survey. En *Business Process Management*, pp. 1–12. Springer.

[Varró *et al.*, 2002] Varró, D., Varró, G., y Pataricza, A. (2002). Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227.

[Washizaki, 2007] Washizaki, H. (2007). Deriving project-specific processes from process line architecture with commonality and variability. pp. 1301 – 1306.

[Weske, 2007] Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures. 2nd edition*. Springer.

[Wohed *et al.*, 2005] Wohed, P., Van der Aalst, W. M., Dumas, M., Ter Hofstede, A. H., y Russell, N. (2005). Pattern-based analysis of the control-flow perspective of UML activity diagrams. En *Conceptual Modeling*, pp. 63–78. Springer.

[Wohlin y Prikladniki, 2013] Wohlin, C. y Prikladniki, R. (2013). Systematic Literature Reviews in Software Engineering. *Information and Software Technology*, (0).

[Wu *et al.*, 2007] Wu, M., Li, G., Ying, J., y Yan, H. (2007). A metamodel approach to software process modeling based on UML extension. volumen 6, pp. 4508 – 4512.

[Zaki *et al.*, 2011] Zaki, M. Z. M., Isa, M., y Jawawi, D. N. A. (2011). Meta-model validation of integrated MARTE and component-based methodology component model for embedded real-time software. *Communications in Computer and Information Science*, 181 CCIS(PART 3):246 – 256.

[Zamenhof, 1905] Zamenhof, L. L. (1905). *Fundamento de Esperanto*.

[Zamli y Lee, 2001] Zamli, K. y Lee, P. (2001). Taxonomy of process modeling languages. En *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2001)*, pp. 435 –437.

[Zamli y Lee, 2003] Zamli, K. y Lee, P. (2003). Modeling and enacting software processes using VRPML. En *Proceedings of the 10th Asia-Pacific Software Engineering Conference(APSEC 2003)*, pp. 243 – 252.

[Zamli, 2001] Zamli, K. Z. (2001). Process Modeling Languages: A Literature Review. *Malaysian Journal of Computer Science*, 14(2):26–37.

[Zamli e Isa, 2004] Zamli, K. Z. e Isa, N. M. (2004). A survey and analysis of process modeling languages. *Malaysian Journal of Computer Science*, 17(2):68–89.

[Zhang y Ali Babar, 2013] Zhang, H. y Ali Babar, M. (2013). Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology*, 55(7):1341–1354.

[Zhang *et al.*, 2011] Zhang, H., Babar, M. A., y Tell, P. (2011). Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637.

Anexo A

Transformaciones bidireccionales de UML-DA a INROMA

En este anexo se presentan las relaciones para acometer la transformación bidireccional entre los diagramas de actividad de UML e INROMA. Para ello, haremos uso de la sintaxis concreta gráfica y textual de QVT Relations, tal y como fueron descritas en el capítulo 6. También se recoge la implementación que se ha hecho de todas estas transformaciones en MONETA, de forma similar a lo expuesto en el capítulo 7.

La tabla A.1 expone las correspondencias entre las relaciones y los diagramas de transformación QVT que se muestran a continuación. Las relaciones, atendiendo a la sintaxis textual de QVT Relations, se detallan en el algoritmo A.1. La implementación de dichas transformaciones se recogen en los algoritmos A.2 y A.3.

Relación	**Sintaxis Gráfica QVT Relacional**
Activity2Process	Figura A.1
MapActivities	Figura A.2
MapInputProductsOfActivity	Figura A.3
MapOutputProductsOfActivity	Figura A.4
MapIncomingSequence	Figura A.5
MapOutgoingSequence	Figura A.6
MapInitial	Figura A.7
MapFinal	Figura A.8
MapConditional	Figura A.9

Tabla A.1: Relaciones para acometer la transformación UML2INROMA

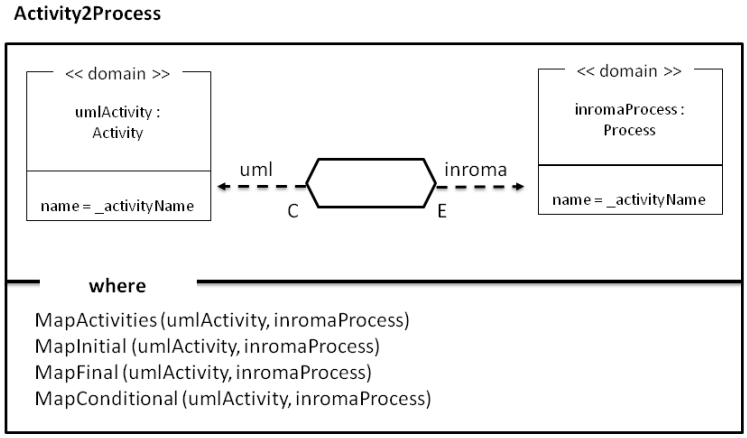


Figura A.1: Vista QVT Relations de la relación Activity2Process en la transformación UML2INROMA

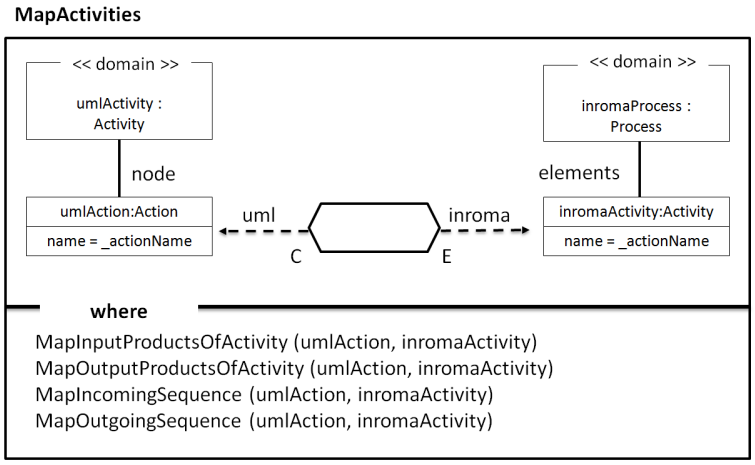


Figura A.2: Vista QVT Relations de la relación MapActivities en la transformación UML2INROMA

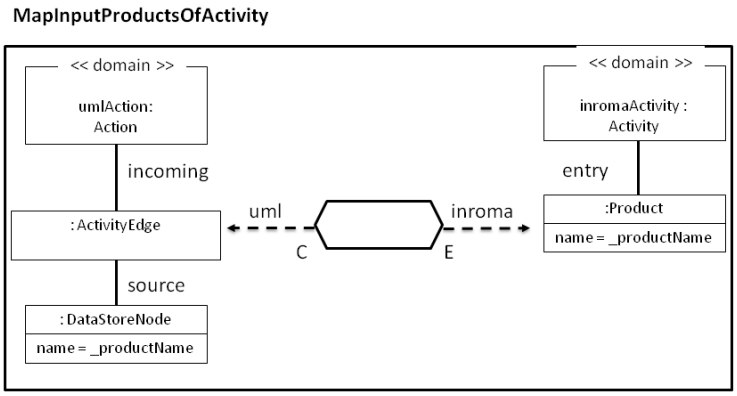


Figura A.3: Vista QVT Relations de la relación MapInputProductsOfActivity en la transformación UML2INROMA

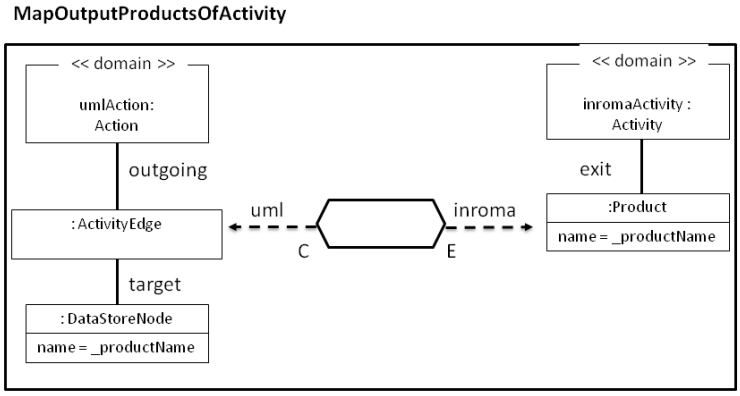


Figura A.4: Vista QVT Relations de la relación MapOutputProductsOfActivity en la transformación UML2INROMA

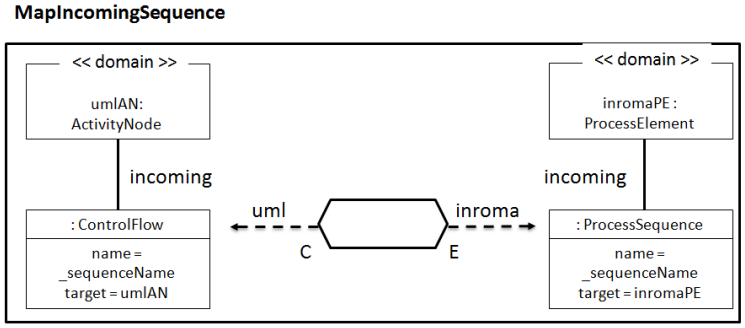


Figura A.5: Vista QVT Relations de la relación MapIncomingSequence en la transformación UML2INROMA

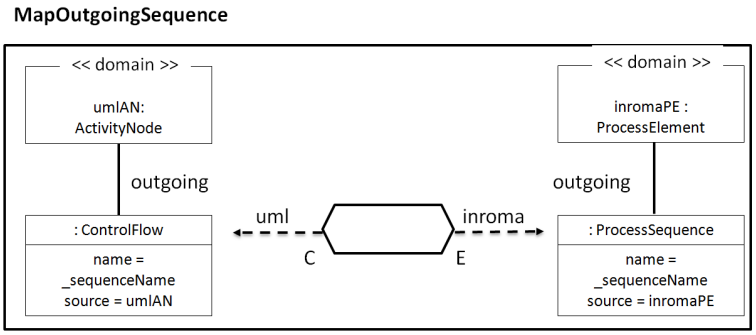


Figura A.6: Vista QVT Relations de la relación MapOutgoingSequence en la transformación UML2INROMA

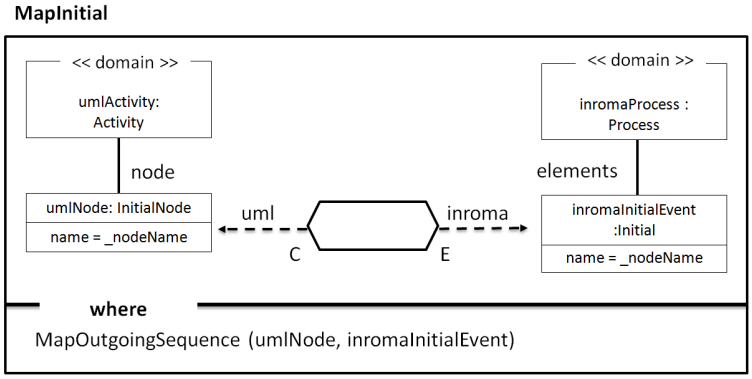


Figura A.7: Vista QVT Relations de la relación MapInitial en la transformación UML2INROMA

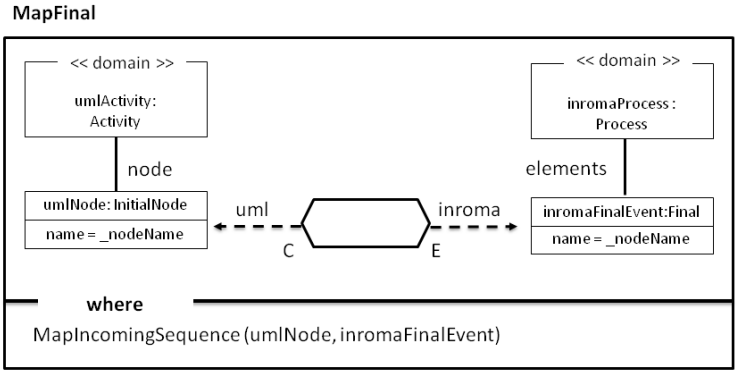


Figura A.8: Vista QVT Relations de la relación MapFinal en la transformación UML2INROMA

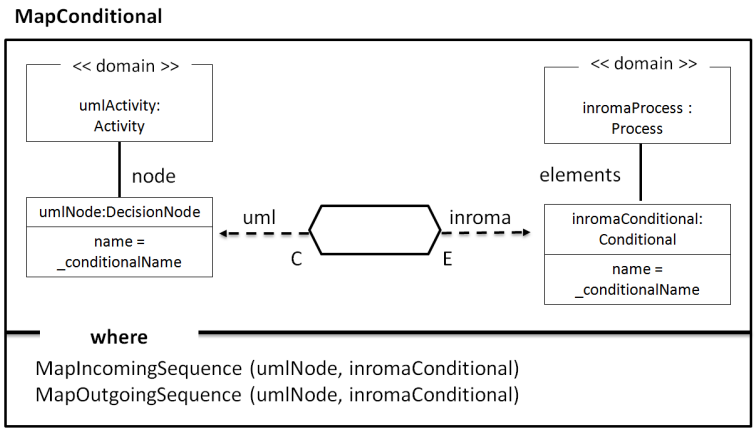


Figura A.9: Vista QVT Relations de la relación MapConditional en la transformación UML2INROMA

```
1  transformation uml2inroma (umlmodel:uml, inromamodel:inroma) {
2
3     top relation Activity2Process
4     {
5        _activityName : String;
6        checkonly domain umlmodel umlActivity : uml::Activity
7        {
8           name = _activityName
9        };
10       enforce domain inromamodel inromaProcess : inroma::Process
11       {
12          name = _activityName
13       };
14       where {
15          MapActivities (umlActivity, inromaProcess);
16          MapInitial (umlActivity, inromaProcess);
17          MapFinal (umlActivity, inromaProcess);
18          MapConditional (umlActivity, inromaProcess);
19          -- No existen Indicators en UML, no es posible realizar el mapeo
20          -- MapIndicators (bpmnProcess, inromaProcess);
21       }
22    }
23
24    relation MapActivities
25    {
26       _actionName : String;
27       checkonly domain umlmodel umlActivity : uml::Activity
28       {
29          node = umlAction : uml::Action {name = _actionName}
30       };
31       enforce domain inromamodel inromaProcess : inroma::Process
32       {
33          elements = inromaActivity : inroma::Activity { name = _actionName }
34       };
35       where {
36          MapInputProductsOfActivity (umlAction, inromaActivity);
37          MapOutputProductsOfActivity (umlAction, inromaActivity);
38          -- No existe relación entre un Actor y los diagramas de actividad en UML,
                no es posible realizar el mapeo
39          --MapStakeholdersOfActivity (umlAction, inromaActivity);
40          MapIncomingSequence (umlAction, inromaActivity);
41          MapOutgoingSequence (umlAction, inromaActivity);
42       }
43    }
44
45    relation MapInputProductsOfActivity
46    {
47       _productName : String;
48      checkonly domain umlmodel umlAction : uml::Action
49       {
50          incoming = umlActionEdges : uml::ActivityEdge {
51            source = umlDataStoreNode : uml::DataStoreNode {name = _productName}}
52       };
53      enforce domain inromamodel inromaActivity : inroma::Activity
54       {
55          entry = inromaProduct : inroma::Product {name = _productName}
56       };
```

```
57      }
58
59      relation MapOutputProductsOfActivity
60      {
61          _productName : String;
62        checkonly domain umlmodel umlAction : uml::Action
63          {
64              outgoing = umlActionEdges : uml::ActivityEdge {
65                target = umlDataStoreNode : uml::DataStoreNode {name = _productName}}
66          };
67        enforce domain inromamodel inromaActivity : inroma::Activity
68          {
69              exit = inromaProduct : inroma::Product {name = _productName}
70          };
71      }
72
73      relation MapIncomingSequence
74      {
75          _sequenceName : String;
76        checkonly domain umlmodel umlAN : uml::ActivityNode
77          {
78              incoming = umlCF: uml::ControlFlow {name = _sequenceName, target = umlAN}
79          };
80          enforce domain inromamodel inromaPE : inroma::ProcessElement
81          {
82              incoming = inromaIncoming : inroma::ProcessSequence {name = _sequenceName
                    , target = inromaPE}
83          };
84      }
85
86      relation MapOutgoingSequence
87      {
88          _sequenceName : String;
89        checkonly domain umlmodel umlAN : uml::ActivityNode
90          {
91              outgoing = umlCF: uml::ControlFlow {name = _sequenceName, source = umlAN}
92          };
93          enforce domain inromamodel inromaPE : inroma::ProcessElement
94          {
95              outgoing = inromaOutgoing : inroma::ProcessSequence {name = _sequenceName
                    , source = inromaPE}
96          };
97      }
98
99      relation MapInitial
100     {
101         _nodeName : String;
102         checkonly domain umlmodel umlActivity : uml::Activity
103         {
104             node = umlNode : uml::InitialNode {name = _nodeName}
105         };
106         enforce domain inromamodel inromaProcess : inroma::Process
107         {
108             elements = inromaInitialEvent : inroma::Initial { name = _nodeName }
109         };
110         where {
111             MapOutgoingSequence (umlNode, inromaInitialEvent);
112         }
```

```
113     }
114
115     relation MapFinal
116     {
117       _nodeName : String;
118        checkonly domain umlmodel umlActivity : uml::Activity
119        {
120           node = umlNode : uml::FinalNode {name = _nodeName}
121        };
122         enforce domain inromamodel inromaProcess : inroma::Process
123        {
124           elements = inromaFinalEvent : inroma::Final { name = _nodeName }
125        };
126        where {
127           MapIncomingSequence (umlNode, inromaFinalEvent);
128        }
129     }
130
131     relation MapConditional
132     {
133        _conditionalName : String;
134        checkonly domain umlmodel umlActivity : uml::Activity
135        {
136           node = umlNode : uml::DecisionNode {name = _conditionalName}
137        };
138        enforce domain inromamodel inromaProcess : inroma::Process
139        {
140           elements = inromaConditional : inroma::Conditional { name =
                  _conditionalName }
141        };
142        where {
143           MapIncomingSequence (umlNode, inromaConditional);
144           MapOutgoingSequence (umlNode, inromaConditional);
145        }
146     }
147
148 }
```

Algoritmo A.1: Transformación UML2INROMA

```
1     class UML2INROMA : INROMA, ITransform
2     {
3
4         Hashtable _idMaps = new Hashtable(); // inputModelElementID →
              outputModelElementID
5
6         public EA.Package transform(EA.Repository repository, EA.Package
              inputModel, EA.Package outputModelView)
7         {
8             _idMaps.Clear();
9             addNewRoot(repository, outputModelView);
10            foreach (EA.Element e in inputModel.Elements)
11            {
12                if (e.Type == "Activity")
13                    Activity2Process(e, findElement(e.Name));
14            }
15
16            _package.Update();
17            repository.RefreshModelView(_package.PackageID);
18
19            return _package;
20        }
21
22        public int getOutputDiagramID() { return getDiagramID(); }
23
24        void Activity2Process(EA.Element umlActivity, EA.Element inromaProcess)
25        {
26            if (inromaProcess == null)
27                inromaProcess = addNewProcess(umlActivity.Name);
28
29            if (inromaProcess == null)
30                return;
31
32            MapActivities(umlActivity, inromaProcess);
33            MapInitial(umlActivity, inromaProcess);
34            MapFinal(umlActivity, inromaProcess);
35            MapConditional(umlActivity, inromaProcess);
36        }
37
38        void MapActivities(EA.Element umlActivity, EA.Element inromaProcess)
39        {
40            foreach (EA.Element e in umlActivity.Elements)
41            {
42                if (e.Type == "Action")
43                {
44                    if (findElement(e.Name) == null)
45                    {
46                        EA.Element inromaActivity = addNewActivity(inromaProcess.
                              Elements, e.Name);
47                        if (inromaActivity != null)
48                        {
49                            _idMaps[e.ElementID] = inromaActivity.ElementID;
50                            MapInputProducts(e, inromaActivity, inromaProcess);
51                            MapOutputProducts(e, inromaActivity, inromaProcess);
52                            MapStakeholders(e, inromaActivity, inromaProcess);
53                            MapIncomingSequence(e, inromaActivity);
54                            MapOutcomingSequence(e, inromaActivity);
```

```
55                      }
56                  }
57              }
58          }
59      }
60
61      void MapInitial(EA.Element umlActivity, EA.Element inromaProcess)
62      {
63          foreach (EA.Element e in umlActivity.Elements)
64          {
65              if (e.Type == "StateNode" && e.Stereotype == "Initial")
66              {
67                  if (findElement(e.Name) == null)
68                  {
69                      EA.Element inromaInitial = addNewInitial(inromaProcess.
                            Elements, e.Name);
70                      if (inromaInitial != null)
71                      {
72                          _idMaps[e.ElementID] = inromaInitial.ElementID;
73                          MapOutcomingSequence(e, inromaInitial);
74                      }
75                  }
76              }
77          }
78      }
79
80      void MapFinal(EA.Element umlActivity, EA.Element inromaProcess)
81      {
82          foreach (EA.Element e in umlActivity.Elements)
83          {
84              if (e.Type == "StateNode" && e.Stereotype == "Final")
85              {
86                  if (findElement(e.Name) == null)
87                  {
88                      EA.Element inromaFinal = addNewFinal(inromaProcess.
                            Elements, e.Name);
89                      if (inromaFinal != null)
90                      {
91                          _idMaps[e.ElementID] = inromaFinal.ElementID;
92                          MapIncomingSequence(e, inromaFinal);
93                      }
94                  }
95              }
96          }
97      }
98
99      void MapConditional(EA.Element umlActivity, EA.Element inromaProcess)
100     {
101         foreach (EA.Element e in umlActivity.Elements)
102         {
103             if (e.Type == "Decision")
104             {
105                 if (findElement(e.Name) == null)
106                 {
107                     EA.Element inromaConditional = addNewConditional(
                            inromaProcess.Elements, e.Name);
108                     if (inromaConditional != null)
109                     {
```

```
110                         _idMaps[e.ElementID] = inromaConditional.ElementID;
111                         MapIncomingSequence(e, inromaConditional);
112                         MapOutcomingSequence(e, inromaConditional);
113                     }
114                 }
115             }
116         }
117     }
118
119     void MapInputProducts(EA.Element umlAction, EA.Element inromaActivity, EA.
        Element inromaProcess)
120     {
121         foreach (EA.Connector c in umlAction.Connectors)
122         {
123             int elementID = c.ClientID;
124             EA.Element e = _repository.GetElementByID(elementID);
125             if (e != null && e.Stereotype == "datastore")
126             {
127                 EA.Element product = null;
128                 if (_idMaps.ContainsKey(e.ElementID))
129                 {
130                     product = _repository.GetElementByID((int)_idMaps[e.
                        ElementID]);
131                 }
132                 else
133                 {
134                     product = addNewProduct(inromaProcess.Elements, e.Name);
135                     _idMaps[e.ElementID] = product.ElementID;
136                 }
137                 addNewEntry(product, inromaActivity, c.Name);
138             }
139         }
140     }
141
142     void MapOutputProducts(EA.Element umlAction, EA.Element inromaActivity, EA
        .Element inromaProcess)
143     {
144         foreach (EA.Connector c in umlAction.Connectors)
145         {
146             int elementID = c.SupplierID;
147             EA.Element e = _repository.GetElementByID(elementID);
148             if (e != null && e.Stereotype == "datastore")
149             {
150                 EA.Element product = null;
151                 if (_idMaps.ContainsKey(e.ElementID))
152                 {
153                     product = _repository.GetElementByID((int)_idMaps[e.
                        ElementID]);
154                 }
155                 else
156                 {
157                     product = addNewProduct(inromaProcess.Elements, e.Name);
158                     _idMaps[e.ElementID] = product.ElementID;
159                 }
160                 addNewExit(inromaActivity, product, c.Name);
161             }
162         }
163     }
```

```
164
165         void MapStakeholders(EA.Element umlAction, EA.Element inromaActivity, EA.
                Element inromaProcess)
166         {
167             foreach (EA.Connector c in umlAction.Connectors)
168             {
169                 EA.Element e = _repository.GetElementByID(c.SupplierID);
170                 if (e == null || e.Type != "Actor")
171                 {
172                     e = _repository.GetElementByID(c.ClientID);
173                     if (e == null || e.Type != "Actor")
174                         continue;
175                 }
176
177                 EA.Element stakeholder = addNewStakeholder(inromaProcess.Elements,
                        e.Name);
178                 addNewParticipes(inromaActivity, stakeholder, "");
179             }
180         }
181
182         void MapIncomingSequence(EA.Element umlElement, EA.Element inromaElement)
183         {
184             foreach (EA.Connector c in umlElement.Connectors)
185             {
186                 if (c.Type != "ControlFlow" || c.SupplierID != umlElement.
                        ElementID)     // not incoming
187                     continue;
188
189                 int sourceID = c.ClientID;
190                 int targetID = c.SupplierID;
191                 if (_idMaps.ContainsKey(sourceID) == false
192                     || _idMaps.ContainsKey(targetID) == false)
193                     continue;
194
195                 int inromaSourceID = (int)_idMaps[sourceID];
196                 int inromaTargetID = (int)_idMaps[targetID];
197
198                 EA.Element source = _repository.GetElementByID(inromaSourceID);
199                 EA.Element target = inromaElement; // _repository.GetElementByID(
                        inromaTargetID);
200
201                 addNewProcessSequence(source, target, c.Name);
202             }
203         }
204
205         void MapOutcomingSequence(EA.Element umlElement, EA.Element inromaElement)
206         {
207             foreach (EA.Connector c in umlElement.Connectors)
208             {
209                 if (c.Type != "ControlFlow" || c.ClientID != umlElement.ElementID)
                            // not incoming
210                     continue;
211
212                 int sourceID = c.ClientID;
213                 int targetID = c.SupplierID;
214                 if (_idMaps.ContainsKey(sourceID) == false
215                     || _idMaps.ContainsKey(targetID) == false)
216                     continue;
```

```
217
218                 int inromaSourceID = (int)_idMaps[sourceID];
219                 int inromaTargetID = (int)_idMaps[targetID];
220
221                 EA.Element source = inromaElement; // _repository.GetElementByID(
                        inromaSourceID);
222                 EA.Element target = _repository.GetElementByID(inromaTargetID);
223
224                 addNewProcessSequence(source, target, c.Name);
225             }
226         }
227     }
```

Algoritmo A.2: Implementación de las transformaciones de UML-DA a INROMA en C#

```
1   class INROMA2UML : UML, ITransform
2   {
3       Hashtable _idMaps = new Hashtable(); // inputModelElementID →
            outputModelElementID
4
5       public EA.Package transform(EA.Repository repository, EA.Package
            inputModel, EA.Package outputModelView)
6       {
7           _idMaps.Clear();
8           addNewRoot(repository, outputModelView);
9           foreach (EA.Element e in inputModel.Elements)
10          {
11              if (e.Stereotype.Contains("Process"))
12                  Process2Activity(e, findElement(e.Name));
13          }
14
15          _package.Update();
16          repository.RefreshModelView(_package.PackageID);
17
18          return _package;
19      }
20
21      public int getOutputDiagramID() { return getDiagramID(); }
22
23      void Process2Activity(EA.Element inromaProcess, EA.Element umlActivity)
24      {
25          if (umlActivity == null)
26              umlActivity = addNewActivity(inromaProcess.Name);
27
28          if (umlActivity == null)
29              return;
30
31          MapActivities(inromaProcess, umlActivity);
32          MapInitial(inromaProcess, umlActivity);
33          MapFinal(inromaProcess, umlActivity);
34          MapConditional(inromaProcess, umlActivity);
35      }
36
37      void MapActivities(EA.Element inromaProcess, EA.Element umlActivity)
38      {
39          foreach (EA.Element activity in inromaProcess.Elements)
40          {
41              if (activity.Stereotype.Contains("Activity"))
42              {
43                  if (findElement(activity.Name) == null)
44                  {
45                      EA.Element umlAction = addNewAction(umlActivity.Elements,
                            activity.Name);
46                      if (umlAction != null)
47                      {
48                          _idMaps[activity.ElementID] = umlAction.ElementID;
49                          MapInputProducts(activity, umlAction, umlActivity);
50                          MapOutputProducts(activity, umlAction, umlActivity);
51                          MapStakeholders(activity, umlAction, umlActivity);
52                          MapIncomingSequence(activity, umlAction);
53                          MapOutcomingSequence(activity, umlAction);
54                      }
```

```
55                      }
56                  }
57              }
58          }
59
60          void MapInitial(EA.Element inromaProcess, EA.Element umlActivity)
61          {
62              foreach (EA.Element e in inromaProcess.Elements)
63              {
64                  if (e.Stereotype.Contains("Initial"))
65                  {
66                      if (findElement(e.Name) == null)
67                      {
68                          EA.Element umlInitial = addNewActivityInitial(umlActivity.
                                Elements, e.Name);
69                          if (umlInitial != null)
70                          {
71                              _idMaps[e.ElementID] = umlInitial.ElementID;
72                              MapOutcomingSequence(e, umlInitial);
73                          }
74                      }
75                  }
76              }
77          }
78
79          void MapFinal(EA.Element inromaProcess, EA.Element umlActivity)
80          {
81              foreach (EA.Element e in inromaProcess.Elements)
82              {
83                  if (e.Stereotype.Contains("Final"))
84                  {
85                      if (findElement(e.Name) == null)
86                      {
87                          EA.Element umlFinal = addNewActivityFinal(umlActivity.
                                Elements, e.Name);
88                          if (umlFinal != null)
89                          {
90                              _idMaps[e.ElementID] = umlFinal.ElementID;
91                              MapIncomingSequence(e, umlFinal);
92                          }
93                      }
94                  }
95              }
96          }
97
98          void MapConditional(EA.Element inromaProcess, EA.Element umlActivity)
99          {
100             foreach (EA.Element e in inromaProcess.Elements)
101             {
102                 if (e.Stereotype.Contains("Conditional"))
103                 {
104                     if (findElement(e.Name) == null)
105                     {
106                         EA.Element umlDecision = addNewDecision(umlActivity.
                                Elements, e.Name);
107                         if (umlDecision != null)
108                         {
109                             _idMaps[e.ElementID] = umlDecision.ElementID;
```

```
110                         MapIncomingSequence(e, umlDecision);
111                         MapOutcomingSequence(e, umlDecision);
112                     }
113                 }
114             }
115         }
116     }
117
118     void MapInputProducts(EA.Element inromaActivity, EA.Element umlAction, EA.
            Element umlActivity)
119     {
120         foreach (EA.Connector c in inromaActivity.Connectors)
121         {
122             if (c.Stereotype.Contains("Entry"))
123             {
124                 EA.Element e = _repository.GetElementByID(c.ClientID);
125                 if (e == null || !e.Stereotype.Contains("Product"))
126                 {
127                     e = _repository.GetElementByID(c.SupplierID);
128                     if (e == null || !e.Stereotype.Contains("Product"))
129                         continue;
130                 }
131
132                 EA.Element product = null;
133                 if (_idMaps.ContainsKey(e.ElementID))
134                 {
135                     product = _repository.GetElementByID((int)_idMaps[e.
                            ElementID]);
136                 }
137                 else
138                 {
139                     product = addNewDataStore(umlActivity.Elements, e.Name);
140                     _idMaps[e.ElementID] = product.ElementID;
141                 }
142                 addNewDependency(product, umlAction, "");
143             }
144         }
145     }
146
147     void MapOutputProducts(EA.Element inromaActivity, EA.Element umlAction, EA
            .Element umlActivity)
148     {
149         foreach (EA.Connector c in inromaActivity.Connectors)
150         {
151             if (c.Stereotype.Contains("Exit"))
152             {
153                 EA.Element e = _repository.GetElementByID(c.ClientID);
154                 if (e == null || !e.Stereotype.Contains("Product"))
155                 {
156                     e = _repository.GetElementByID(c.SupplierID);
157                     if (e == null || !e.Stereotype.Contains("Product"))
158                         continue;
159                 }
160
161                 EA.Element product = null;
162                 if (_idMaps.ContainsKey(e.ElementID))
163                 {
164                     product = _repository.GetElementByID((int)_idMaps[e.
```

```
                        ElementID]);
165                 }
166                 else
167                 {
168                     product = addNewDataStore(umlActivity.Elements, e.Name);
169                     _idMaps[e.ElementID] = product.ElementID;
170                 }
171                 addNewDependency(umlAction, product, "");
172             }
173         }
174     }
175
176     void MapStakeholders(EA.Element inromaActivity, EA.Element umlAction, EA.
        Element umlActivity)
177     {
178         foreach (EA.Connector c in inromaActivity.Connectors)
179         {
180             int elementID = c.SupplierID;
181             EA.Element e = _repository.GetElementByID(elementID);
182             if (e == null || !e.Stereotype.Contains("Stakeholder"))
183                 e = _repository.GetElementByID(c.ClientID);
184             if (e != null && e.Stereotype.Contains("Stakeholder"))
185             {
186                 EA.Element actor = addNewActor(umlActivity.Elements, e.Name);
187                 addNewDependency(umlAction, actor, "");
188             }
189         }
190     }
191
192     void MapIncomingSequence(EA.Element inromaElement, EA.Element umlElement)
193     {
194         foreach (EA.Connector c in inromaElement.Connectors)
195         {
196             if (!c.Stereotype.Contains("ProcessSequence") || c.SupplierID !=
                inromaElement.ElementID)     // not incoming
197                 continue;
198
199             int sourceID = c.ClientID;
200             int targetID = c.SupplierID;
201             if (_idMaps.ContainsKey(sourceID) == false
202                 || _idMaps.ContainsKey(targetID) == false)
203                 continue;
204
205             int umlSourceID = (int)_idMaps[sourceID];
206             int umlTargetID = (int)_idMaps[targetID];
207
208             EA.Element source = _repository.GetElementByID(umlSourceID);
209             EA.Element target = umlElement; // _repository.GetElementByID(
                bpmnTargetID);
210
211             addNewControlFlow(source, target, c.Name);
212         }
213     }
214
215     void MapOutcomingSequence(EA.Element inromaElement, EA.Element umlElement)
216     {
217         foreach (EA.Connector c in inromaElement.Connectors)
218         {
```

```
219                 if (!c.Stereotype.Contains("ProcessSequence") || c.ClientID !=
                        inromaElement.ElementID)     // not incoming
220                     continue;
221
222                 int sourceID = c.ClientID;
223                 int targetID = c.SupplierID;
224                 if (_idMaps.ContainsKey(sourceID) == false
225                     || _idMaps.ContainsKey(targetID) == false)
226                     continue;
227
228                 int umlSourceID = (int)_idMaps[sourceID];
229                 int umlTargetID = (int)_idMaps[targetID];
230
231                 EA.Element source = umlElement; // _repository.GetElementByID(
                        bpmnSourceID);
232                 EA.Element target = _repository.GetElementByID(umlTargetID);
233
234                 addNewControlFlow(source, target, c.Name);
235             }
236         }
237     }
```

Algoritmo A.3: Implementación de las transformaciones de INROMA a UML-DA en C#

Anexo B

Transformaciones bidireccionales de BPMN a INROMA

Seguidamente se expone con mayor detalle la transformación bidireccional entre BPMN e INROMA, siguiendo el mismo esquema que en el anexo A.

La tabla B.1 expone las correspondencias entre las relaciones y los diagramas de transformación QVT que se muestran a continuación. Las relaciones, atendiendo a la sintaxis textual de QVT Relations, se detallan en el algoritmo B.1. La implementación de dichas transformaciones se recogen en los algoritmos B.2 y B.3.

Relación	**Sintaxis Gráfica QVT Relacional**
Process2Process	Figura B.1
MapActivities	Figura B.2
MapInputProductsOfActivity	Figura B.3
MapOutputProductsOfActivity	Figura B.4
MapStakeholdersOfActivity	Figura B.5
MapIncomingSequence	Figura B.6
MapOutgoingSequence	Figura B.7
MapInitial	Figura B.8
MapFinal	Figura B.9
MapConditional	Figura B.10

Tabla B.1: Relaciones para acometer la transformación BPMN2INROMA

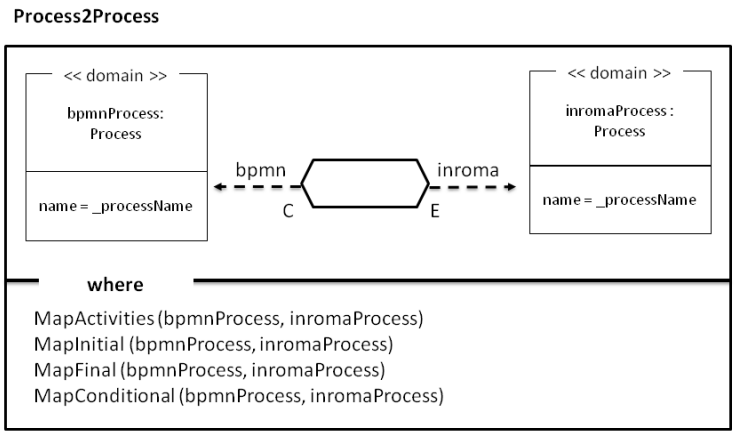


Figura B.1: Vista QVT Relations de la relación Process2Process en la transformación BPMN2INROMA

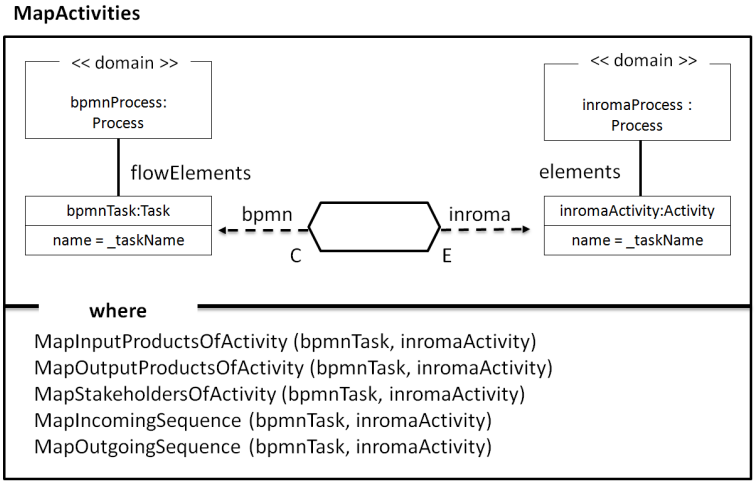


Figura B.2: Vista QVT Relations de la relación MapActivities en la transformación BPMN2INROMA

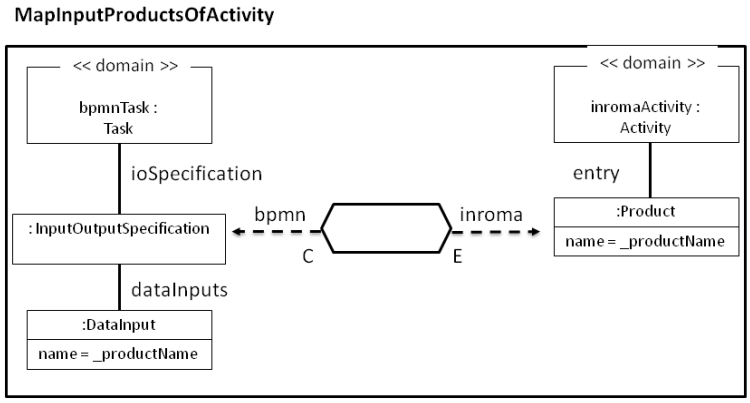


Figura B.3: Vista QVT Relations de la relación MapInputProductsOfActivity en la transformación BPMN2INROMA

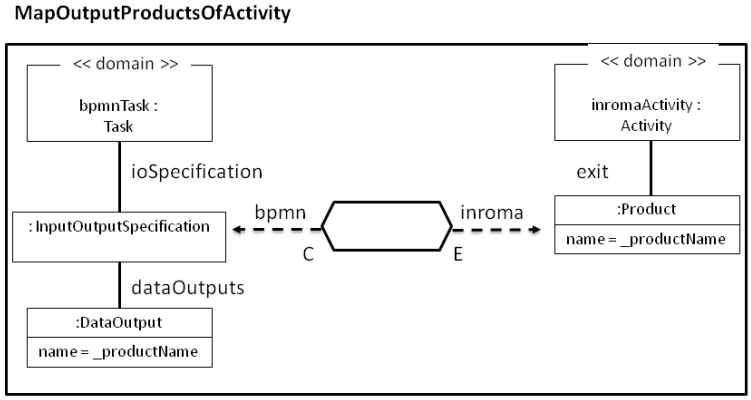


Figura B.4: Vista QVT Relations de la relación MapOutputProductsOfActivity en la transformación BPMN2INROMA

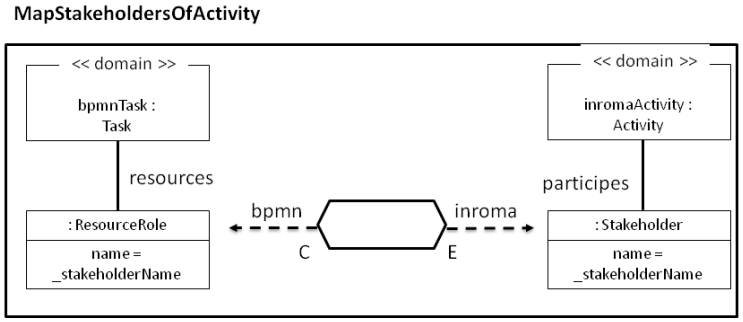


Figura B.5: Vista QVT Relations de la relación MapStakeholdersOfActivity en la transformación BPMN2INROMA

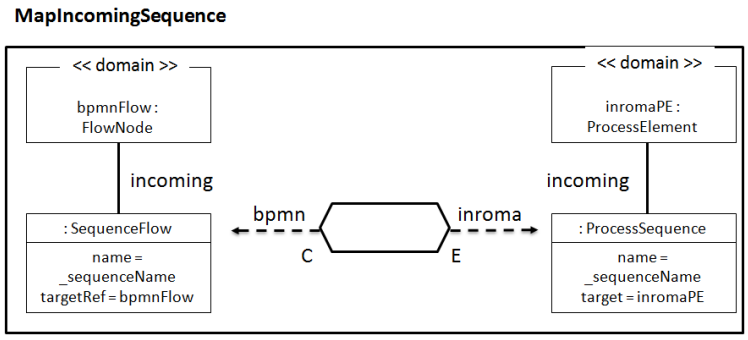


Figura B.6: Vista QVT Relations de la relación MapIncomingSequence en la transformación BPMN2INROMA

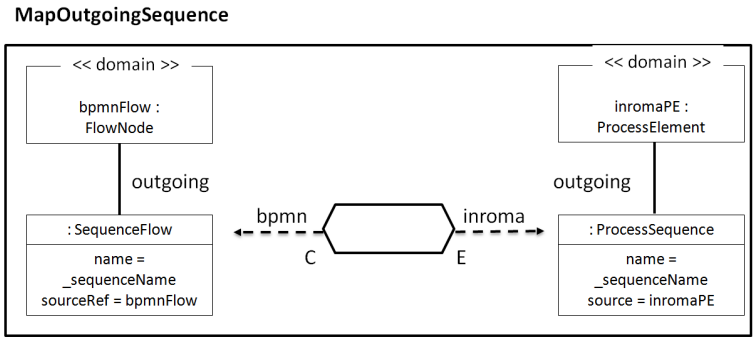


Figura B.7: Vista QVT Relations de la relación MapOutgoingSequence en la transformación BPMN2INROMA

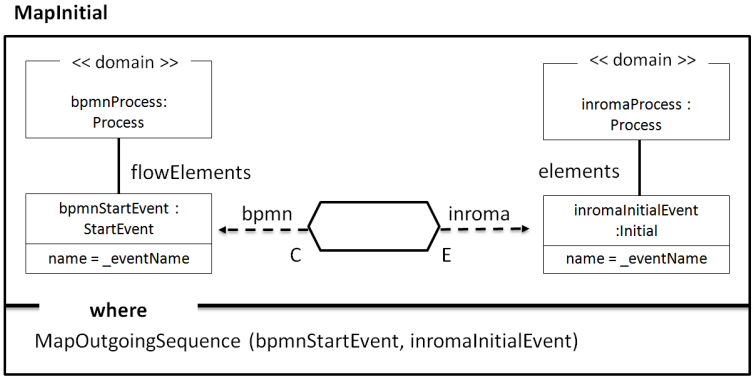


Figura B.8: Vista QVT Relations de la relación MapInitial en la transformación BPMN2INROMA

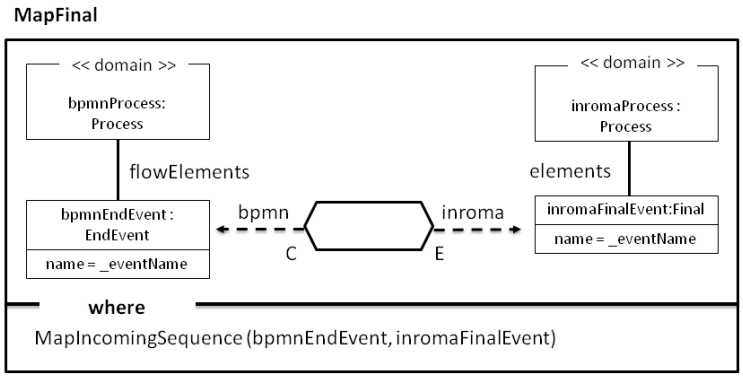


Figura B.9: Vista QVT Relations de la relación MapFinal en la transformación BPMN2INROMA

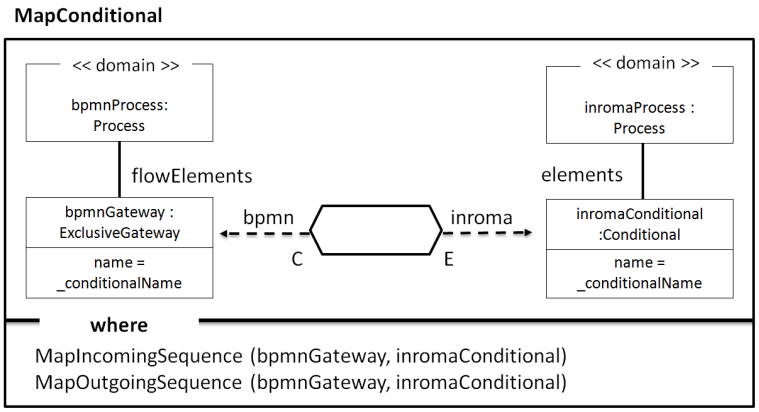


Figura B.10: Vista QVT Relations de la relación MapConditional en la transformación BPMN2INROMA

```
1  transformation bpmn2inroma (bpmnmodel:bpmn2, inromamodel:inroma) {
2
3     top relation Process2Process
4     {
5        _processName : String;
6        checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
7        {
8           name = _processName
9        };
10       enforce domain inromamodel inromaProcess : inroma::Process
11       {
12          name = _processName
13       };
14       where {
15          MapActivities (bpmnProcess, inromaProcess);
16          MapInitial (bpmnProcess, inromaProcess);
17          MapFinal (bpmnProcess, inromaProcess);
18          MapConditional (bpmnProcess, inromaProcess);
19          -- No existen Indicators en BPMN, no es posible realizar el mapeo
20          -- MapIndicators (bpmnProcess, inromaProcess);
21       }
22    }
23
24    relation MapActivities
25    {
26       _taskName : String;
27       checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
28       {
29          flowElements = bpmnTask : bpmn2::Task {name = _taskName}
30       };
31       enforce domain inromamodel inromaProcess : inroma::Process
32       {
33          elements = inromaActivity : inroma::Activity { name = _taskName }
34       };
35       where {
36          MapInputProductsOfActivity (bpmnTask, inromaActivity);
37          MapOutputProductsOfActivity (bpmnTask, inromaActivity);
38          MapStakeholdersOfActivity (bpmnTask, inromaActivity);
39          MapIncomingSequence (bpmnTask, inromaActivity);
40          MapOutgoingSequence (bpmnTask, inromaActivity);
41       }
42    }
43
44    relation MapInputProductsOfActivity
45    {
46       _productName : String;
47       checkonly domain bpmnmodel bpmnTask : bpmn2::Task
48       {
49          ioSpecification = bpmnIOSpecification : bpmn2::InputOutputSpecification {
50              dataInputs = bpmnDataInput : bpmn2::DataInput {name = _productName}}
51       };
52      enforce domain inromamodel inromaActivity : inroma::Activity
53       {
54          entry = inromaProduct : inroma::Product {name = _productName}
55       };
56    }
57
```

```
58    relation MapOutputProductsOfActivity
59    {
60       _productName : String;
61       checkonly domain bpmnmodel bpmnTask : bpmn2::Task
62       {
63          ioSpecification = bpmnIOSpecification : bpmn2::InputOutputSpecification {
64              dataOutputs = bpmnDataOutput : bpmn2::DataOutput {name = _productName
                    }}
65       };
66      enforce domain inromamodel inromaActivity : inroma::Activity
67       {
68          exit = inromaProduct : inroma::Product {name = _productName}
69       };
70    }
71
72    relation MapStakeholdersOfActivity
73    {
74       _stakeholderName : String;
75       checkonly domain bpmnmodel bpmnTask : bpmn2::Task
76       {
77          resources = bpmnResourceRole : bpmn2::ResourceRole {name =
                _stakeholderName}
78       };
79       enforce domain inromamodel inromaActivity : inroma::Activity
80       {
81          participes = inromaStakeholder : inroma::Stakeholder {name =
                _stakeholderName}
82       };
83    }
84
85    relation MapIncomingSequence
86    {
87       _sequenceName : String;
88       checkonly domain bpmnmodel bpmnFlow : bpmn2::FlowNode
89       {
90          incoming = bpmnIncoming : bpmn2::SequenceFlow {name = _sequenceName,
                targetRef = bpmnFlow}
91       };
92       enforce domain inromamodel inromaPE : inroma::ProcessElement
93       {
94          incoming = inromaIncoming : inroma::ProcessSequence {name = _sequenceName
                , target = inromaPE}
95       };
96    }
97
98    relation MapOutgoingSequence
99    {
100      _sequenceName : String;
101      checkonly domain bpmnmodel bpmnFlow : bpmn2::FlowNode
102      {
103         outgoing = bpmnOutgoing : bpmn2::SequenceFlow {name = _sequenceName,
               sourceRef = bpmnFlow}
104      };
105      enforce domain inromamodel inromaPE : inroma::ProcessElement
106      {
107         outgoing = inromaOutgoing : inroma::ProcessSequence {name = _sequenceName
               , source = inromaPE}
108      };
```

```
109     }
110
111     relation MapInitial
112     {
113         _eventName : String;
114         checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
115         {
116             flowElements = bpmnStartEvent : bpmn2::StartEvent {name = _eventName}
117         };
118         enforce domain inromamodel inromaProcess : inroma::Process
119         {
120             elements = inromaInitialEvent : inroma::Initial { name = _eventName }
121         };
122         where {
123             MapOutgoingSequence (bpmnStartEvent, inromaInitialEvent);
124         }
125     }
126
127     relation MapFinal
128     {
129         _eventName : String;
130         checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
131         {
132             flowElements = bpmnEndEvent : bpmn2::EndEvent {name = _eventName}
133         };
134         enforce domain inromamodel inromaProcess : inroma::Process
135         {
136             elements = inromaFinalEvent : inroma::Final { name = _eventName }
137         };
138         where {
139             MapIncomingSequence (bpmnEndEvent, inromaFinalEvent);
140         }
141     }
142
143     relation MapConditional
144     {
145         _conditionalName : String;
146         checkonly domain bpmnmodel bpmnProcess : bpmn2::Process
147         {
148             flowElements = bpmnGateway : bpmn2::ExclusiveGateway {name =
                    _conditionalName}
149         };
150         enforce domain inromamodel inromaProcess : inroma::Process
151         {
152             elements = inromaConditional : inroma::Conditional { name =
                    _conditionalName }
153         };
154         where {
155             MapIncomingSequence (bpmnGateway, inromaConditional);
156             MapOutgoingSequence (bpmnGateway, inromaConditional);
157         }
158     }
159 }
```

Algoritmo B.1: Transformación BPMN2INROMA

```
1     class BPMN2INROMA : INROMA, ITransform
2     {
3
4         Hashtable _idMaps = new Hashtable(); // inputModelElementID →
              outputModelElementID
5
6         public EA.Package transform(EA.Repository repository, EA.Package
              inputModel, EA.Package outputModelView)
7         {
8             _idMaps.Clear();
9             addNewRoot(repository, outputModelView);
10            foreach (EA.Element e in inputModel.Elements)
11            {
12                if (e.Stereotype.Contains("BusinessProcess"))
13                    Process2Process(e, findElement(e.Name));
14            }
15
16            _package.Update();
17            repository.RefreshModelView(_package.PackageID);
18
19            return _package;
20        }
21
22        public int getOutputDiagramID() { return getDiagramID(); }
23
24        void Process2Process(EA.Element bpmnProcess, EA.Element inromaProcess)
25        {
26            if (inromaProcess == null)
27                inromaProcess = addNewProcess(bpmnProcess.Name);
28
29            if (inromaProcess == null)
30                return;
31
32            MapActivities(bpmnProcess, inromaProcess);
33            MapInitial(bpmnProcess, inromaProcess);
34            MapFinal(bpmnProcess, inromaProcess);
35            MapConditional(bpmnProcess, inromaProcess);
36        }
37
38        void MapActivities(EA.Element bpmnProcess, EA.Element inromaProcess)
39        {
40            foreach (EA.Element bpmnTask in bpmnProcess.Elements)
41            {
42                if (bpmnTask.Stereotype.Contains("Activity")) // BPMN::Task!
43                {
44                    if (findElement(bpmnTask.Name) == null)
45                    {
46                        EA.Element inromaActivity = addNewActivity(inromaProcess.
                              Elements, bpmnTask.Name);
47                        if (inromaActivity != null)
48                        {
49                            _idMaps[bpmnTask.ElementID] = inromaActivity.ElementID
                                  ;
50                            MapInputProducts(bpmnTask, inromaActivity,
                                  inromaProcess);
51                            MapOutputProducts(bpmnTask, inromaActivity,
                                  inromaProcess);
```

```
52                          MapStakeholders(bpmnTask, inromaActivity,
                                inromaProcess);
53                          MapIncomingSequence(bpmnTask, inromaActivity);
54                          MapOutcomingSequence(bpmnTask, inromaActivity);
55                      }
56                  }
57              }
58          }
59      }
60
61      void MapInitial(EA.Element bpmnProcess, EA.Element inromaProcess)
62      {
63          foreach (EA.Element e in bpmnProcess.Elements)
64          {
65              if (e.Stereotype.Contains("StartEvent"))
66              {
67                  if (findElement(e.Name) == null)
68                  {
69                      EA.Element inromaInitial = addNewInitial(inromaProcess.
                            Elements, e.Name);
70                      if (inromaInitial != null)
71                      {
72                          _idMaps[e.ElementID] = inromaInitial.ElementID;
73                          MapOutcomingSequence(e, inromaInitial);
74                      }
75                  }
76              }
77          }
78      }
79
80      void MapFinal(EA.Element bpmnProcess, EA.Element inromaProcess)
81      {
82          foreach (EA.Element e in bpmnProcess.Elements)
83          {
84              if (e.Stereotype.Contains("EndEvent"))
85              {
86                  if (findElement(e.Name) == null)
87                  {
88                      EA.Element inromaFinal = addNewFinal(inromaProcess.
                            Elements, e.Name);
89                      if (inromaFinal != null)
90                      {
91                          _idMaps[e.ElementID] = inromaFinal.ElementID;
92                          MapIncomingSequence(e, inromaFinal);
93                      }
94                  }
95              }
96          }
97      }
98
99      void MapConditional(EA.Element bpmnProcess, EA.Element inromaProcess)
100     {
101         foreach (EA.Element e in bpmnProcess.Elements)
102         {
103             if (e.Stereotype.Contains("Gateway") && getTagValue(e, "
                    gatewayType") == "Exclusive")
104             {
105                 if (findElement(e.Name) == null)
```

```
106                     {
107                         EA.Element inromaConditional = addNewConditional(
                                inromaProcess.Elements, e.Name);
108                         if (inromaConditional != null)
109                         {
110                             _idMaps[e.ElementID] = inromaConditional.ElementID;
111                             MapIncomingSequence(e, inromaConditional);
112                             MapOutcomingSequence(e, inromaConditional);
113                         }
114                     }
115                 }
116             }
117         }
118 
119         void MapInputProducts(EA.Element bpmnTask, EA.Element inromaActivity, EA.
                Element inromaProcess)
120         {
121             foreach (EA.Connector c in bpmnTask.Connectors)
122             {
123                 int elementID = c.ClientID;
124                 EA.Element e = _repository.GetElementByID(elementID);
125                 if (e != null && e.Stereotype.Contains("DataObject"))
126                 {
127                     foreach (EA.TaggedValue tag in e.TaggedValues)
128                     {
129                         if (tag.Name == "dataInOut")
130                         {
131                             if (tag.Value == "Input" || tag.Value == "None")
132                             {
133                                 EA.Element product = addNewProduct(inromaProcess.
                                        Elements, e.Name);
134                                 addNewEntry(inromaActivity, product, c.Name);
135                             }
136                         }
137                     }
138                 }
139             }
140         }
141 
142         void MapOutputProducts(EA.Element bpmnTask, EA.Element inromaActivity, EA.
                Element inromaProcess)
143         {
144             foreach (EA.Connector c in bpmnTask.Connectors)
145             {
146                 int elementID = c.SupplierID;
147                 EA.Element e = _repository.GetElementByID(elementID);
148                 if (e != null && e.Stereotype.Contains("DataObject"))
149                 {
150                     foreach (EA.TaggedValue tag in e.TaggedValues)
151                     {
152                         if (tag.Name == "dataInOut")
153                         {
154                             if (tag.Value == "Output" || tag.Value == "None")
155                             {
156                                 EA.Element product = addNewProduct(inromaProcess.
                                        Elements, e.Name);
157                                 addNewExit(inromaActivity, product, c.Name);
158                             }
```

```
159                         }
160                     }
161                 }
162             }
163         }
164
165         void MapStakeholders(EA.Element bpmnTask, EA.Element inromaActivity, EA.
                Element inromaProcess)
166         {
167             foreach (EA.Connector c in bpmnTask.Connectors)
168             {
169                 int elementID = c.SupplierID;
170                 EA.Element e = _repository.GetElementByID(elementID);
171                 if (e != null && e.Stereotype.Contains("ResourceRole"))
172                 {
173                     EA.Element stakeholder = addNewStakeholder(inromaProcess.
                            Elements, e.Name);
174                     addNewParticipes(inromaActivity, stakeholder, "");
175                 }
176             }
177         }
178
179         void MapIncomingSequence(EA.Element bpmnElement, EA.Element inromaElement)
180         {
181             foreach (EA.Connector c in bpmnElement.Connectors)
182             {
183                 if (!c.Stereotype.Contains("SequenceFlow") || c.SupplierID !=
                        bpmnElement.ElementID)     // not incoming
184                     continue;
185
186                 int sourceID = c.ClientID;
187                 int targetID = c.SupplierID;
188                 if (_idMaps.ContainsKey(sourceID) == false
189                     || _idMaps.ContainsKey(targetID) == false)
190                     continue;
191
192                 int inromaSourceID = (int)_idMaps[sourceID];
193                 int inromaTargetID = (int)_idMaps[targetID];
194
195                 EA.Element source = _repository.GetElementByID(inromaSourceID);
196                 EA.Element target = inromaElement; // _repository.GetElementByID(
                        inromaTargetID);
197
198                 addNewProcessSequence(source, target, c.Name);
199             }
200         }
201
202         void MapOutcomingSequence(EA.Element bpmnElement, EA.Element inromaElement
                )
203         {
204             foreach (EA.Connector c in bpmnElement.Connectors)
205             {
206                 if (!c.Stereotype.Contains("SequenceFlow") || c.ClientID !=
                        bpmnElement.ElementID)     // not outcoming
207                     continue;
208
209                 int sourceID = c.ClientID;
210                 int targetID = c.SupplierID;
```

```
211                 if (_idMaps.ContainsKey(sourceID) == false
212                     || _idMaps.ContainsKey(targetID) == false)
213                     continue;
214
215                 int inromaSourceID = (int)_idMaps[sourceID];
216                 int inromaTargetID = (int)_idMaps[targetID];
217
218                 EA.Element source = inromaElement; // _repository.GetElementByID(
                        inromaSourceID);
219                 EA.Element target = _repository.GetElementByID(inromaTargetID);
220
221                 addNewProcessSequence(source, target, c.Name);
222             }
223         }
224     }
```

Algoritmo B.2: Implementación de las transformaciones de BPMN a INROMA en C#

```
1   class INROMA2IBPMN : BPMN, ITransform
2   {
3       Hashtable _idMaps = new Hashtable(); // inputModelElementID →
            outputModelElementID
4
5       public EA.Package transform(EA.Repository repository, EA.Package
            inputModel, EA.Package outputModelView)
6       {
7           _idMaps.Clear();
8           addNewRoot(repository, outputModelView);
9           foreach (EA.Element e in inputModel.Elements)
10          {
11              if (e.Stereotype.Contains("Process"))
12                  Process2BusinessProcess(e, findElement(e.Name));
13          }
14
15          _package.Update();
16          repository.RefreshModelView(_package.PackageID);
17
18          return _package;
19      }
20
21      public int getOutputDiagramID() { return getDiagramID(); }
22
23      void Process2BusinessProcess(EA.Element inromaProcess, EA.Element
            bpmnProcess)
24      {
25          if (bpmnProcess == null)
26              bpmnProcess = addNewBusinessProcess(inromaProcess.Name);
27
28          if (bpmnProcess == null)
29              return;
30
31          MapActivities(inromaProcess, bpmnProcess);
32          MapInitial(inromaProcess, bpmnProcess);
33          MapFinal(inromaProcess, bpmnProcess);
34          MapConditional(inromaProcess, bpmnProcess);
35      }
36
37      void MapActivities(EA.Element inromaProcess, EA.Element bpmnProcess)
38      {
39          foreach (EA.Element activity in inromaProcess.Elements)
40          {
41              if (activity.Stereotype.Contains("Activity")) // BPMN::Task!
42              {
43                  if (findElement(activity.Name) == null)
44                  {
45                      EA.Element bpmnActivity = addNewActivity(bpmnProcess.
                            Elements, activity.Name);
46                      // por defecto: activityType=Task, taskType=Script
47                      if (bpmnActivity != null)
48                      {
49                          _idMaps[activity.ElementID] = bpmnActivity.ElementID;
50                          MapInputProducts(activity, bpmnActivity, bpmnProcess);
51                          MapOutputProducts(activity, bpmnActivity, bpmnProcess)
                                ;
52                          MapStakeholders(activity, bpmnActivity, bpmnProcess);
```

```
53                          MapIncomingSequence(activity, bpmnActivity);
54                          MapOutcomingSequence(activity, bpmnActivity);
55                      }
56                  }
57              }
58          }
59      }
60
61      void MapInitial(EA.Element inromaProcess, EA.Element bpmnProcess)
62      {
63          foreach (EA.Element e in inromaProcess.Elements)
64          {
65              if (e.Stereotype.Contains("Initial"))
66              {
67                  if (findElement(e.Name) == null)
68                  {
69                      EA.Element bpmnInitial = addNewStartEvent(bpmnProcess.
                            Elements, e.Name);
70                      //eventDefinition=None
71                      if (bpmnInitial != null)
72                      {
73                          _idMaps[e.ElementID] = bpmnInitial.ElementID;
74                          MapOutcomingSequence(e, bpmnInitial);
75                      }
76                  }
77              }
78          }
79      }
80
81      void MapFinal(EA.Element inromaProcess, EA.Element bpmnProcess)
82      {
83          foreach (EA.Element e in inromaProcess.Elements)
84          {
85              if (e.Stereotype.Contains("Final"))
86              {
87                  if (findElement(e.Name) == null)
88                  {
89                      EA.Element bpmnFinal = addNewEndEvent(bpmnProcess.Elements
                            , e.Name);
90                      if (bpmnFinal != null)
91                      {
92                          setTagValue(bpmnFinal, "eventDefinition", "Terminate")
                                ;
93                          _idMaps[e.ElementID] = bpmnFinal.ElementID;
94                          MapIncomingSequence(e, bpmnFinal);
95                      }
96                  }
97              }
98          }
99      }
100
101     void MapConditional(EA.Element inromaProcess, EA.Element bpmnProcess)
102     {
103         foreach (EA.Element e in inromaProcess.Elements)
104         {
105             if (e.Stereotype.Contains("Conditional"))
106             {
107                 if (findElement(e.Name) == null)
```

```
108                     {
109                         EA.Element bpmnGateway = addNewGateway(bpmnProcess.
                                Elements, e.Name);
110                         if (bpmnGateway != null)
111                         {
112                             setTagValue(bpmnGateway, "gatewayType", "Exclusive");
113                             _idMaps[e.ElementID] = bpmnGateway.ElementID;
114                             MapIncomingSequence(e, bpmnGateway);
115                             MapOutcomingSequence(e, bpmnGateway);
116                         }
117                     }
118                 }
119             }
120         }
121
122         void MapInputProducts(EA.Element inromaActivity, EA.Element bpmnActivity,
                EA.Element bpmnProcess)
123         {
124             foreach (EA.Connector c in inromaActivity.Connectors)
125             {
126                 if (c.Stereotype.Contains("Entry"))
127                 {
128                     EA.Element e = _repository.GetElementByID(c.ClientID);
129                     if (e == null || !e.Stereotype.Contains("Product"))
130                     {
131                         e = _repository.GetElementByID(c.SupplierID);
132                         if (e == null || !e.Stereotype.Contains("Product"))
133                             continue;
134                     }
135
136                     EA.Element product = null;
137                     if (_idMaps.ContainsKey(e.ElementID))
138                     {
139                         product = _repository.GetElementByID((int)_idMaps[e.
                                ElementID]);
140                     }
141                     else
142                     {
143                         product = addNewProduct(bpmnProcess.Elements, e.Name);
144                         _idMaps[e.ElementID] = product.ElementID;
145                     }
146                     setTagValue(product, "DataInOut", "Input");
147                     addNewDataAssociation(product, bpmnActivity, "");
148                 }
149             }
150         }
151
152         void MapOutputProducts(EA.Element inromaActivity, EA.Element bpmnActivity,
                EA.Element bpmnProcess)
153         {
154             foreach (EA.Connector c in inromaActivity.Connectors)
155             {
156                 if (c.Stereotype.Contains("Exit"))
157                 {
158                     EA.Element e = _repository.GetElementByID(c.ClientID);
159                     if (e == null || !e.Stereotype.Contains("Product"))
160                     {
161                         e = _repository.GetElementByID(c.SupplierID);
```

```
162                     if (e == null || !e.Stereotype.Contains("Product"))
163                         continue;
164                 }
165
166                 EA.Element product = null;
167                 if (_idMaps.ContainsKey(e.ElementID))
168                 {
169                     product = _repository.GetElementByID((int)_idMaps[e.
                        ElementID]);
170                 }
171                 else
172                 {
173                     product = addNewProduct(bpmnProcess.Elements, e.Name);
174                     _idMaps[e.ElementID] = product.ElementID;
175                 }
176                 setTagValue(product, "DataInOut", "Output");
177                 addNewDataAssociation(bpmnActivity, product, "");
178             }
179         }
180     }
181
182     void MapStakeholders(EA.Element inromaActivity, EA.Element bpmnActivity,
        EA.Element bpmnProcess)
183     {
184         foreach (EA.Connector c in inromaActivity.Connectors)
185         {
186             int elementID = c.SupplierID;
187             EA.Element e = _repository.GetElementByID(elementID);
188             if (e == null || !e.Stereotype.Contains("Stakeholder"))
189                 e = _repository.GetElementByID(c.ClientID);
190             if (e != null && e.Stereotype.Contains("Stakeholder"))
191             {
192                 EA.Element resourceRole = addNewResourceRole(bpmnProcess.
                    Elements, e.Name);
193                 addNewAssociation(bpmnActivity, resourceRole, "");
194             }
195         }
196     }
197
198     void MapIncomingSequence(EA.Element inromaElement, EA.Element bpmnElement)
199     {
200         foreach (EA.Connector c in inromaElement.Connectors)
201         {
202             if (!c.Stereotype.Contains("ProcessSequence") || c.SupplierID !=
                inromaElement.ElementID)     // not incoming
203                 continue;
204
205             int sourceID = c.ClientID;
206             int targetID = c.SupplierID;
207             if (_idMaps.ContainsKey(sourceID) == false
208                 || _idMaps.ContainsKey(targetID) == false)
209                 continue;
210
211             int bpmnSourceID = (int)_idMaps[sourceID];
212             int bpmnTargetID = (int)_idMaps[targetID];
213
214             EA.Element source = _repository.GetElementByID(bpmnSourceID);
215             EA.Element target = bpmnElement; // _repository.GetElementByID(
```

```
                    bpmnTargetID);
216
217                 addNewSequenceFlow(source, target, c.Name);
218             }
219         }
220
221         void MapOutcomingSequence(EA.Element inromaElement, EA.Element bpmnElement
                )
222         {
223             foreach (EA.Connector c in inromaElement.Connectors)
224             {
225                 if (!c.Stereotype.Contains("ProcessSequence") || c.ClientID !=
                        inromaElement.ElementID)     // not incoming
226                     continue;
227
228                 int sourceID = c.ClientID;
229                 int targetID = c.SupplierID;
230                 if (_idMaps.ContainsKey(sourceID) == false
231                     || _idMaps.ContainsKey(targetID) == false)
232                     continue;
233
234                 int bpmnSourceID = (int)_idMaps[sourceID];
235                 int bpmnTargetID = (int)_idMaps[targetID];
236
237                 EA.Element source = bpmnElement; // _repository.GetElementByID(
                        bpmnSourceID);
238                 EA.Element target = _repository.GetElementByID(bpmnTargetID);
239
240                 addNewSequenceFlow(source, target, c.Name);
241             }
242         }
243     }
```

Algoritmo B.3: Implementación de las transformaciones de INROMA a BPMN en C#

Anexo C

Transformaciones bidireccionales de IDEF a INROMA

Para finalizar con las transformaciones en el presente anexo se exponen las relaciones para desarrollar una transformación bidireccional entre IDEF3 e INROMA. Al igual que en los anexos anteriores, mostraremos todas las relaciones atendiendo a la sintaxis gráficas de QVT y finalizaremos con el algoritmo siguiendo la sintaxis textual de QVT.

En la tabla C.1 podemos observar la equivalencia entre las relaciones y los diagramas de transformación de QVT que son expuestos seguidamente, finalizando en el algoritmo C.1. La implementación de dichas transformaciones se recogen en los algoritmos C.2 y C.3.

Relación	**Sintaxis Gráfica QVT Relacional**
Scenario2Process	Figura C.1
MapUnitOfBehaviors	Figura C.2
MapInputProductsOfActivity	Figura C.3
MapOutputProductsOfActivity	Figura C.4
MapIncomingSequence	Figura C.5
MapOutgoingSequence	Figura C.6
MapConditional	Figura C.7

Tabla C.1: Relaciones para acometer la transformación IDEF2INROMA

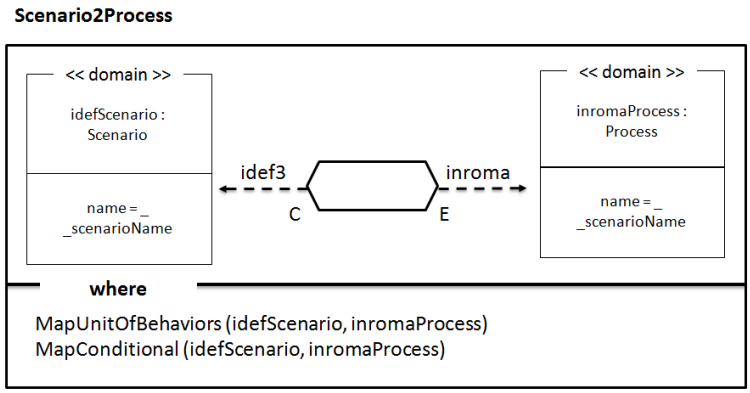


Figura C.1: Vista QVT Relations de la relación Scenario2Process en la transformación IDEF2INROMA

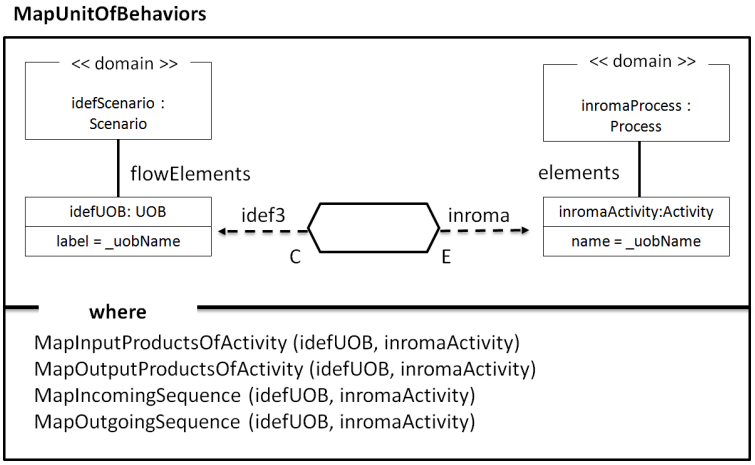


Figura C.2: Vista QVT Relations de la relación MapUnitOfBehaviors en la transformación IDEF2INROMA

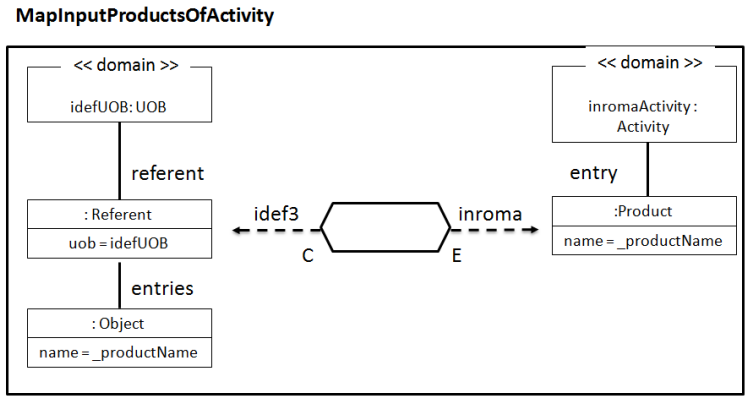


Figura C.3: Vista QVT Relations de la relación MapInputProductsOfActivity en la transformación IDEF2INROMA

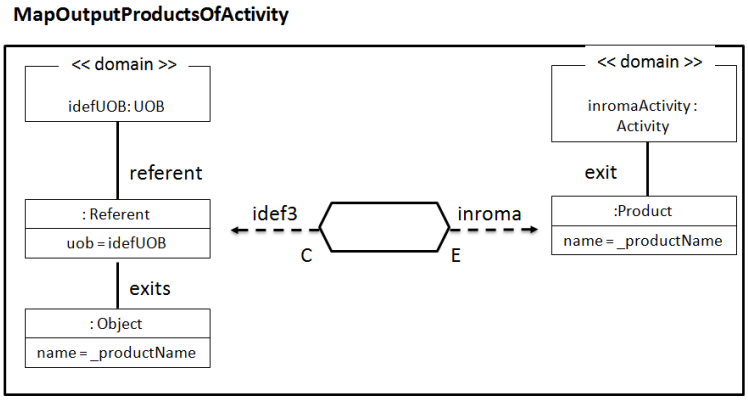


Figura C.4: Vista QVT Relations de la relación MapOutputProductsOfActivity en la transformación IDEF2INROMA

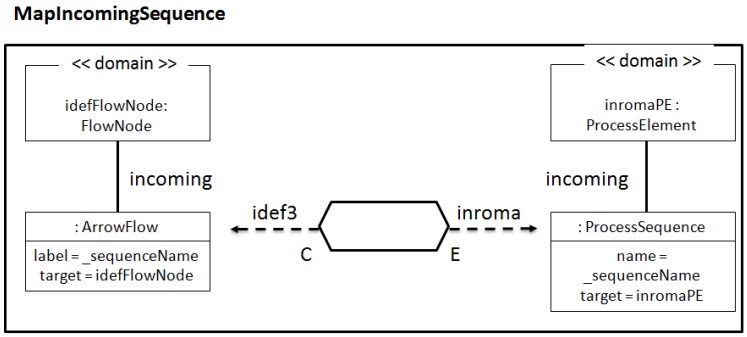


Figura C.5: Vista QVT Relations de la relación MapIncomingSequence en la transformación IDEF2INROMA

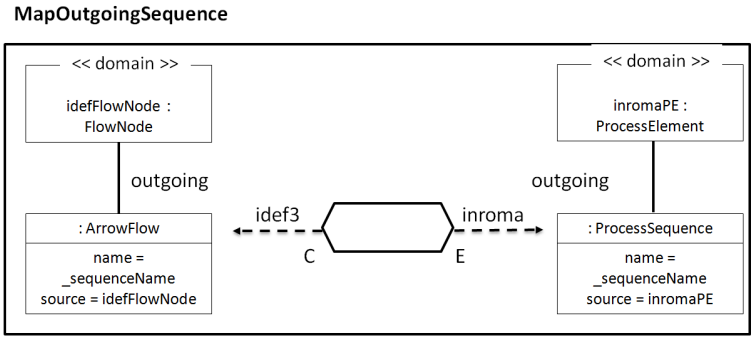


Figura C.6: Vista QVT Relations de la relación MapOutgoingSequence en la transformación IDEF2INROMA

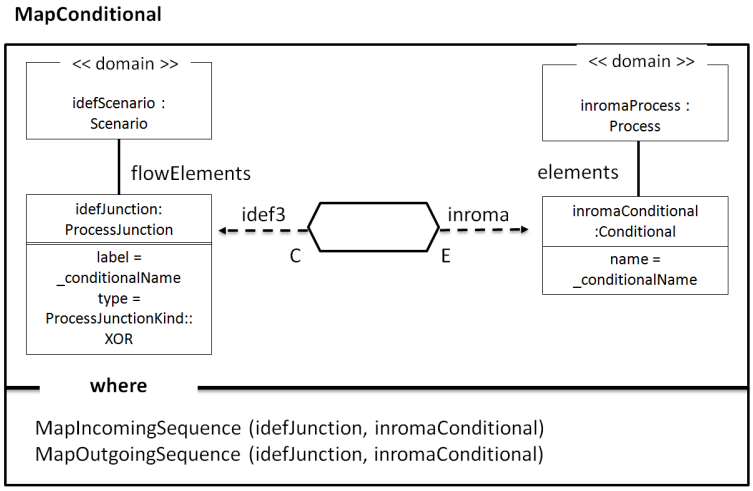


Figura C.7: Vista QVT Relations de la relación MapConditional en la transformación IDEF2INROMA

```
1  transformation idef2inroma (idefmodel:idef3, inromamodel:inroma) {
2
3     top relation Scenario2Process
4     {
5        _scenarioName : String;
6        checkonly domain idefmodel idefScenario : idef3::Scenario
7        {
8           name = _scenarioName
9        };
10       enforce domain inromamodel inromaProcess : inroma::Process
11       {
12          name = _scenarioName
13       };
14       where {
15          MapUnitOfBehaviors (idefScenario, inromaProcess);
16          -- No existe Inicial ni Final en IDEF3, no es posible realizar el mapeo
17          -- MapInitial (idefScenario, inromaProcess);
18          -- MapFinal (idefScenario, inromaProcess);
19          MapConditional (idefScenario, inromaProcess);
20          -- No existen Indicators en IDEF3, no es posible realizar el mapeo
21          -- MapIndicators (idefScenario, inromaProcess);
22       }
23    }
24
25    relation MapUnitOfBehaviors
26    {
27       _uobName : String;
28       checkonly domain idefmodel idefScenario : idef3::Scenario
29       {
30          flowElements = idefUOB : idef3::UOB {label = _uobName}
31       };
32       enforce domain inromamodel inromaProcess : inroma::Process
33       {
34          elements = inromaActivity : inroma::Activity { name = _uobName }
35       };
36       where {
37          MapInputProductsOfActivity (idefUOB, inromaActivity);
38          MapOutputProductsOfActivity (idefUOB, inromaActivity);
39          -- No existen Stakeholders en IDEF3, no es posible realizar el mapeo
40          -- MapStakeholdersOfActivity (idefUOB, inromaActivity);
41          MapIncomingSequence (idefUOB, inromaActivity);
42          MapOutgoingSequence (idefUOB, inromaActivity);
43       }
44    }
45
46    relation MapInputProductsOfActivity
47    {
48       _productName : String;
49       checkonly domain idefmodel idefUOB : idef3::UOB
50       {
51         referent = idefReferent : idef3::Referent {
52             uob = idefUOB,
53             entries = idefInputObject : idef3::Object {name = _productName}}
54       };
55      enforce domain inromamodel inromaActivity : inroma::Activity
56       {
57          entry = inromaProduct : inroma::Product {name = _productName}
```

```
58          };
59      }
60
61      relation MapOutputProductsOfActivity
62      {
63          _productName : String;
64          checkonly domain idefmodel idefUOB : idef3::UOB
65          {
66            referent = idefReferent : idef3::Referent {
67                uob = idefUOB,
68                exits = idefInputObject : idef3::Object {name = _productName}}
69          };
70        enforce domain inromamodel inromaActivity : inroma::Activity
71          {
72             exit = inromaProduct : inroma::Product {name = _productName}
73          };
74      }
75
76
77      relation MapIncomingSequence
78      {
79          _sequenceName : String;
80          checkonly domain idefmodel idefFlowNode : idef3::FlowNode
81          {
82             incoming = idefIncoming : idef3::ArrowFlow {label = _sequenceName, target
                    = idefFlowNode}
83          };
84          enforce domain inromamodel inromaPE : inroma::ProcessElement
85          {
86             incoming = inromaIncoming : inroma::ProcessSequence {name = _sequenceName
                   , target = inromaPE}
87          };
88      }
89
90      relation MapOutgoingSequence
91      {
92          _sequenceName : String;
93          checkonly domain idefmodel idefFlowNode : idef3::FlowNode
94          {
95             outgoing = idefOutgoing : idef3::ArrowFlow {label = _sequenceName, source
                    = idefFlowNode}
96          };
97          enforce domain inromamodel inromaPE : inroma::ProcessElement
98          {
99             outgoing = inromaOutgoing : inroma::ProcessSequence {name = _sequenceName
                   , source = inromaPE}
100         };
101     }
102
103     relation MapConditional
104     {
105         _conditionalName : String;
106         checkonly domain idefmodel idefScenario : idef3::Scenario
107          {
108            flowElements = idefJunction : idef3::ProcessJunction {label =
                    _conditionalName,
109               type = idef3::ProcessJunctionKind::XOR}
110         };
```

```
111         enforce domain inromamodel inromaProcess : inroma::Process
112         {
113             elements = inromaConditional : inroma::Conditional { name =
                    _conditionalName }
114         };
115         where {
116             MapIncomingSequence (idefJunction, inromaConditional);
117             MapOutgoingSequence (idefJunction, inromaConditional);
118         }
119     }
120 }
```

Algoritmo C.1: Transformación IDEF2INROMA

```
1   class IDEF2INROMA : INROMA, ITransform
2   {
3       Hashtable _idMaps = new Hashtable(); // inputModelElementID →
            outputModelElementID
4
5       public EA.Package transform(EA.Repository repository, EA.Package
            inputModel, EA.Package outputModelView)
6       {
7           _idMaps.Clear();
8           addNewRoot(repository, outputModelView);
9           foreach (EA.Element e in inputModel.Elements)
10          {
11              if (e.Stereotype.Contains("Scenario"))
12                  Scenario2Process(e, findElement(e.Name));
13          }
14          _package.Update();
15          repository.RefreshModelView(_package.PackageID);
16
17          return _package;
18      }
19
20      public int getOutputDiagramID() { return getDiagramID(); }
21
22      void Scenario2Process(EA.Element idefScenario, EA.Element inromaProcess)
23      {
24          if (inromaProcess == null)
25              inromaProcess = addNewProcess(idefScenario.Name);
26
27          if (inromaProcess == null)
28              return;
29
30          MapUnitOfBehaviors(idefScenario, inromaProcess);
31          MapConditional(idefScenario, inromaProcess);
32      }
33
34      void MapUnitOfBehaviors(EA.Element idefScenario, EA.Element inromaProcess)
35      {
36          foreach (EA.Element idefUOB in idefScenario.Elements)
37          {
38              if (idefUOB.Stereotype.Contains("UnitOfBehavior"))
39              {
40                  if (findElement(idefUOB.Name) == null)
41                  {
42                      EA.Element inromaActivity = addNewActivity(inromaProcess.
                            Elements, idefUOB.Name);
43                      if (inromaActivity != null)
44                      {
45                          _idMaps[idefUOB.ElementID] = inromaActivity.ElementID;
46                          MapInputProducts(idefScenario, idefUOB, inromaActivity
                                , inromaProcess);
47                          MapOutputProducts(idefScenario, idefUOB,
                                inromaActivity, inromaProcess);
48                          MapIncomingSequence(idefUOB, inromaActivity);
49                          MapOutcomingSequence(idefUOB, inromaActivity);
50                      }
51                  }
52              }
```

```
53              }
54          }
55
56          void MapConditional(EA.Element idefScenario, EA.Element inromaProcess)
57          {
58              foreach (EA.Element e in idefScenario.Elements)
59              {
60                  if (e.Type == "Decision" && e.Stereotype.Contains("XOR"))
61                  {
62                      if (findElement(e.Name) == null)
63                      {
64                          EA.Element inromaConditional = addNewConditional(
                                inromaProcess.Elements, e.Name);
65                          if (inromaConditional != null)
66                          {
67                              _idMaps[e.ElementID] = inromaConditional.ElementID;
68                              MapIncomingSequence(e, inromaConditional);
69                              MapOutcomingSequence(e, inromaConditional);
70                          }
71                      }
72                  }
73              }
74          }
75
76          void MapInputProducts(EA.Element idefScenario, EA.Element idefUOB, EA.
                Element inromaActivity, EA.Element inromaProcess)
77          {
78              // buscamos los referent de idefUOB
79              foreach (EA.Element e in idefScenario.Elements)
80              {
81                  if (e.IsAssociationClass() && e.Stereotype.Contains("Referent"))
82                  {
83                      string type = getTagValue(e, "type");
84                      if (type == null || type != "uob" || e.Name != idefUOB.Name ||
                            e.AssociationClassConnectorID < 1)
85                          continue;
86
87                      try
88                      {
89                          EA.Connector c = _repository.GetConnectorByID(e.
                                AssociationClassConnectorID);
90                          if (c != null)
91                          {
92                              EA.Element idefObject = _repository.GetElementByID(c.
                                    ClientID);
93
94                              if (idefObject == null)
95                              {
96                                  continue;
97                              }
98                              else if (idefObject.Type == "Class" && idefObject.
                                    Stereotype.Contains("ObjectAND"))
99                              {
100                                 foreach (EA.Connector cc in idefObject.Connectors)
101                                 {
102                                     int elementID = cc.ClientID;
103                                     EA.Element ee = _repository.GetElementByID(
                                            elementID);
```

```
104                                     if (ee != null && ee.Type == "Object" && ee.
                                            Stereotype.Contains("Object"))
105                                     {
106                                         addNewProductAndConnect(inromaProcess, ee,
                                                inromaActivity, true);
107                                     }
108                                 }
109                             }
110                             else if (idefObject.Type == "Object" && idefObject.
                                    Stereotype.Contains("Object"))
111                             {
112                                 addNewProductAndConnect(inromaProcess, idefObject,
                                        inromaActivity, true);
113                             }
114                         }
115                     }
116                     catch
117                     {
118                     }
119                 }
120             }
121         }
122
123         void MapOutputProducts(EA.Element idefScenario, EA.Element idefUOB, EA.
                Element inromaActivity, EA.Element inromaProcess)
124         {
125             // buscamos los referent de idefUOB
126             foreach (EA.Element e in idefScenario.Elements)
127             {
128                 if (e.IsAssociationClass() && e.Stereotype.Contains("Referent"))
129                 {
130                     string type = getTagValue(e, "type");
131                     if (type == null || type != "uob" || e.Name != idefUOB.Name ||
                            e.AssociationClassConnectorID < 1)
132                         continue;
133
134                     try
135                     {
136                         EA.Connector c = _repository.GetConnectorByID(e.
                                AssociationClassConnectorID);
137                         if (c != null)
138                         {
139                             EA.Element idefObject = _repository.GetElementByID(c.
                                    SupplierID);
140                             if (idefObject == null || !idefObject.Stereotype.
                                    Contains("Object"))
141                                 continue;
142
143                             addNewProductAndConnect(inromaProcess, idefObject,
                                    inromaActivity, false);
144                         }
145                     }
146                     catch
147                     {
148                     }
149                 }
150             }
151         }
```

```
152
153         void addNewProductAndConnect(EA.Element inromaProcess, EA.Element
                idefObject, EA.Element inromaActivity, bool isEntry )
154         {
155             if (inromaProcess == null || idefObject == null)
156                 return;
157
158             int idefObjectID = idefObject.ElementID;
159             EA.Element inromaProduct = null;
160             if (_idMaps.ContainsKey(idefObjectID))
161             {
162                 inromaProduct = _repository.GetElementByID((int)_idMaps[
                        idefObjectID]);
163             }
164             else
165             {
166                 inromaProduct = addNewProduct(inromaProcess.Elements, idefObject.
                        Name);
167                 _idMaps[idefObject.ElementID] = inromaProduct.ElementID;
168             }
169
170             if (isEntry)
171                 addNewEntry(inromaProduct, inromaActivity, "");
172             else
173                 addNewExit(inromaProduct, inromaActivity, "");
174         }
175
176         void MapIncomingSequence(EA.Element idefElement, EA.Element inromaElement)
177         {
178             foreach (EA.Connector c in idefElement.Connectors)
179             {
180                 if (!(c.Stereotype.Contains("SimplePrecedence") || c.Stereotype.
                        Contains("ConstrainedPrecedence"))
181                         || c.SupplierID != idefElement.ElementID)     // not
                            incoming
182                     continue;
183
184                 int sourceID = c.ClientID;
185                 int targetID = c.SupplierID;
186                 if (_idMaps.ContainsKey(sourceID) == false
187                     || _idMaps.ContainsKey(targetID) == false)
188                     continue;
189
190                 int inromaSourceID = (int)_idMaps[sourceID];
191                 int inromaTargetID = (int)_idMaps[targetID];
192
193                 EA.Element source = _repository.GetElementByID(inromaSourceID);
194                 EA.Element target = inromaElement; // _repository.GetElementByID(
                        inromaTargetID);
195
196                 addNewProcessSequence(source, target, c.Name);
197             }
198         }
199
200         void MapOutcomingSequence(EA.Element idefElement, EA.Element inromaElement
                )
201         {
202             foreach (EA.Connector c in idefElement.Connectors)
```

```
203             {
204                 if (!(c.Stereotype.Contains("SimplePrecedence") || c.Stereotype.
                        Contains("ConstrainedPrecedence"))
205                         || c.ClientID != idefElement.ElementID)     // not incoming
206                     continue;
207
208                 int sourceID = c.ClientID;
209                 int targetID = c.SupplierID;
210                 if (_idMaps.ContainsKey(sourceID) == false
211                     || _idMaps.ContainsKey(targetID) == false)
212                     continue;
213
214                 int inromaSourceID = (int)_idMaps[sourceID];
215                 int inromaTargetID = (int)_idMaps[targetID];
216
217                 EA.Element source = inromaElement; // _repository.GetElementByID(
                        inromaSourceID);
218                 EA.Element target = _repository.GetElementByID(inromaTargetID);
219
220                 addNewProcessSequence(source, target, c.Name);
221             }
222         }
223
224
225     }
```

Algoritmo C.2: Implementación de las transformaciones de IDEF a INROMA en C#

```
1     class INROMA2IDEF3 : IDEF3, ITransform
2     {
3         int _uidUnitOfBehavior;
4         Hashtable _idMaps = new Hashtable(); // inputModelElementID →
              outputModelElementID
5
6         public EA.Package transform(EA.Repository repository, EA.Package
              inputModel, EA.Package outputModelView)
7         {
8             _uidUnitOfBehavior = 1;
9             _idMaps.Clear();
10            addNewRoot(repository, outputModelView);
11            foreach (EA.Element e in inputModel.Elements)
12            {
13                if (e.Stereotype.Contains("Process"))
14                    Process2Scenario(e, findElement(e.Name));
15            }
16
17            _package.Update();
18            repository.RefreshModelView(_package.PackageID);
19
20            return _package;
21        }
22
23        public int getOutputDiagramID() { return getDiagramID(); }
24
25        void Process2Scenario(EA.Element inromaProcess, EA.Element idefProcess)
26        {
27            if (idefProcess == null)
28                idefProcess = addNewScenario(inromaProcess.Name);
29
30            if (idefProcess == null)
31                return;
32
33            MapActivities(inromaProcess, idefProcess);
34            MapConditional(inromaProcess, idefProcess);
35        }
36
37        void MapActivities(EA.Element inromaProcess, EA.Element idefProcess)
38        {
39            foreach (EA.Element inromaActivity in inromaProcess.Elements)
40            {
41                if (!inromaActivity.Stereotype.Contains("Activity"))
42                    continue;
43
44                EA.Element idefUOB = addNewUnitOfBehavior(idefProcess.Elements,
                      _uidUnitOfBehavior.ToString(), _uidUnitOfBehavior.ToString(),
                      inromaActivity.Name);
45                _uidUnitOfBehavior++;
46                if (idefUOB != null)
47                {
48                    _idMaps[inromaActivity.ElementID] = idefUOB.ElementID;
49                    MapProducts(inromaProcess, inromaActivity, idefProcess,
                          idefUOB);
50                    MapIncomingSequence(inromaActivity, idefUOB);
51                    MapOutcomingSequence(inromaActivity, idefUOB);
52                }
```

```
53              }
54
55          }
56
57          void MapConditional(EA.Element inromaProcess, EA.Element idefScenario)
58          {
59              foreach (EA.Element e in inromaProcess.Elements)
60              {
61                  if (e.Stereotype.Contains("Conditional"))
62                  {
63                      if (findElement(e.Name) == null)
64                      {
65                          EA.Element idefJunction = addNewProcessJunctionXOR(
                                idefScenario.Elements, e.Name);
66                          if (idefJunction != null)
67                          {
68                              _idMaps[e.ElementID] = idefJunction.ElementID;
69                              MapIncomingSequence(e, idefJunction);
70                              MapOutcomingSequence(e, idefJunction);
71                          }
72                      }
73                  }
74              }
75          }
76
77          void MapProducts(EA.Element inromaProcess, EA.Element inromaActivity, EA.
                Element idefProcess, EA.Element idefUOB)
78          {
79              int count = inromaActivity.Connectors.Count;
80              EA.Element[] entries = new EA.Element[count];
81              EA.Element[] exits = new EA.Element[count];
82              int entryCount = 0;
83              int exitCount = 0;
84              foreach (EA.Connector c in inromaActivity.Connectors)
85              {
86                  if (c.Stereotype.Contains("Entry"))
87                  {
88                      int inromaProductID = c.ClientID == inromaActivity.ElementID ?
                            c.SupplierID : c.ClientID;
89                      if (_idMaps.ContainsKey(inromaProductID))
90                      {
91                          entries[entryCount] = _repository.GetElementByID((int)
                                _idMaps[inromaProductID]);
92                          entryCount++;
93                      }
94                      else
95                      {
96                          EA.Element inromaProduct = _repository.GetElementByID(
                                inromaProductID);
97                          if (inromaProduct == null || !inromaProduct.Stereotype.
                                Contains("Product"))
98                              continue;
99
100                         EA.Element o = addNewObject(idefProcess.Elements,
                                inromaProduct.Name);
101                         _idMaps[inromaProductID] = o.ElementID;
102                         entries[entryCount] = o;
103                         entryCount++;
```

```
104                     }
105                 }
106                 else if (c.Stereotype.Contains("Exit"))
107                 {
108                     int inromaProductID = c.ClientID == inromaActivity.ElementID ?
                            c.SupplierID : c.ClientID;
109                     if (_idMaps.ContainsKey(inromaProductID))
110                     {
111                         exits[exitCount] = _repository.GetElementByID((int)_idMaps
                                [inromaProductID]);
112                         exitCount++;
113                     }
114                     else
115                     {
116                         EA.Element inromaProduct = _repository.GetElementByID(
                                inromaProductID);
117                         if (inromaProduct == null || !inromaProduct.Stereotype.
                                Contains("Product"))
118                             continue;
119 
120                         EA.Element o = addNewObject(idefProcess.Elements,
                                inromaProduct.Name);
121                         _idMaps[inromaProductID] = o.ElementID;
122                         exits[exitCount] = o;
123                         exitCount++;
124                     }
125                 }
126             }
127 
128             if (exitCount < 1 || entryCount < 1)
129                 return;
130 
131             EA.Element junctionEntry = null;
132             EA.Element junctionExit = null;
133 
134             if (entryCount > 1)
135             {
136                 junctionEntry = addNewObjectJunctionAND(idefProcess.Elements, "");
137                 for (int i = 0; i < entryCount; i++)
138                     addNewWeakTransition(entries[i], junctionEntry, "");
139             }
140             else
141             {
142                 junctionEntry = entries[0];
143             }
144 
145             if (exitCount > 1)
146             {
147                 junctionExit = addNewObjectJunctionAND(idefProcess.Elements, "");
148                 for (int i = 0; i < exitCount; i++)
149                     addNewWeakTransition(exits[i], junctionExit, "");
150             }
151             else
152             {
153                 junctionExit = exits[0];
154             }
155 
156             addNewReference(idefProcess.Elements, idefUOB.Name, junctionEntry,
```

```
                junctionExit);
157         }
158
159         void MapIncomingSequence(EA.Element inromaElement, EA.Element idefElement)
160         {
161             foreach (EA.Connector c in inromaElement.Connectors)
162             {
163                 if (!c.Stereotype.Contains("ProcessSequence") || c.SupplierID !=
                        inromaElement.ElementID)     // not incoming
164                     continue;
165
166                 int sourceID = c.ClientID;
167                 int targetID = c.SupplierID;
168                 if (_idMaps.ContainsKey(sourceID) == false
169                     || _idMaps.ContainsKey(targetID) == false)
170                     continue;
171
172                 int bpmnSourceID = (int)_idMaps[sourceID];
173                 int bpmnTargetID = (int)_idMaps[targetID];
174
175                 EA.Element source = _repository.GetElementByID(bpmnSourceID);
176                 EA.Element target = idefElement; // _repository.GetElementByID(
                        bpmnTargetID);
177
178                 addNewSimplePrecedence(source, target, c.Name);
179             }
180         }
181
182         void MapOutcomingSequence(EA.Element inromaElement, EA.Element idefElement
                )
183         {
184             foreach (EA.Connector c in inromaElement.Connectors)
185             {
186                 if (!c.Stereotype.Contains("ProcessSequence") || c.ClientID !=
                        inromaElement.ElementID)     // not incoming
187                     continue;
188
189                 int sourceID = c.ClientID;
190                 int targetID = c.SupplierID;
191                 if (_idMaps.ContainsKey(sourceID) == false
192                     || _idMaps.ContainsKey(targetID) == false)
193                     continue;
194
195                 int bpmnSourceID = (int)_idMaps[sourceID];
196                 int bpmnTargetID = (int)_idMaps[targetID];
197
198                 EA.Element source = idefElement; // _repository.GetElementByID(
                        bpmnSourceID);
199                 EA.Element target = _repository.GetElementByID(bpmnTargetID);
200
201                 addNewSimplePrecedence(source, target, c.Name);
202             }
203         }
204     }
```

Algoritmo C.3: Implementación de las transformaciones de INROMA a IDEF en C#

Anexo D

Manual de MONETA

Este anexo tiene como objetivo describir el uso de la herramienta MONETA, a modo de un manual de usuario. Esta herramienta fue introducida en el capítulo 7, donde se explicó en detalle cómo había sido desarrollada y cómo se habían incorporado cada uno de los elementos claves que conforman el marco de referencia para facilitar la interoperabilidad y mantenibilidad de los modelos de procesos de software, con el objetivo de constituir la herramienta soporte del mismo. Sin embargo, en dicho capítulo no se hacía hincapié en el manejo de MONETA, siendo este manejo el objeto de este manual de usuario.

En dicho capítulo se detalló el entorno de trabajo sobre el que se desarrolla MONETA y la justificación de los motivos que nos han llevado a esa elección, por lo que en este manual de usuario se va a observar que la interfaz gráfica de MONETA está integrada en el propio entorno de Enterprise Architect. A continuación se establecen los pasos que determinan el uso de MONETA.

D.1. Crear un modelo de proceso en MONETA

Como primer paso para la creación de un modelo en MONETA es necesario crear un proyecto en Enterprise Architect, utilizando el asistente que dicha herramienta nos proporciona. No es necesario establecer ningún tipo específico de proyecto concreto para MONETA, por lo que basta con elegir la opción de creación de un nuevo proyecto sin detallar ningún aspecto añadido. Como resultado se obtiene un modelo (*Model*) vacío.

Una vez creado el modelo, se realizan las acciones encaminadas a completarlo con los elementos del proceso que lo conforman. En el capítulo 7 se ha explicado que para poder utilizar MONETA es necesario tener disponibles los lenguajes de modelado de procesos de software que queremos utilizar en el marco de referencia. Por ello, en dicho capítulo se detallaba la necesidad de incorporar un perfil UML para hacer accesibles los lenguajes dentro de MONETA. Como en este manual está enfocado a una perspectiva de uso, se supone que dichos perfiles ya han sido

incorporados.

Para incluir los elementos del proceso en el modelo se genera una vista (*Add* - *AddView*, pulsando el botón derecho del ratón sobre el icono del modelo) a la que se establece un nombre (*Name*). Una vez generada dicha vista, se añade un diagrama a la misma (*Add* - *AddDiagram*, pulsando el botón derecho del ratón sobre el icono de la vista), siendo éste el momento en el que se especifica el lenguaje de modelado de procesos que se va a utilizar para modelar el proceso concreto. Para ello, además del nombre del diagrama (*Name*), el usuario elige el tipo (*Type*) de diagrama que se va a utilizar.

En este momento ya tenemos disponible el diagrama para modelar todos los elementos del proceso, estando disponible para ello el editor con la caja de herramientas (*Toolbox*) correspondiente al lenguaje de modelado de procesos que se ha elegido.

En la figura D.1 se muestra un proceso modelado utilizando como lenguaje el diagrama de actividad de UML, en la que se aprecia la estructura que se genera para contener el modelo, la vista y el diagrama que se han creado dentro de MONETA. En ella se puede observar un detalle muy importante para hacer uso de MONETA y es el hecho de que todos los elementos del proceso pertenecen al proceso, es decir, en el árbol de estructura lo tienen como padre. Esto permite diferenciar en MONETA los elementos propios del proceso de aquellos que se incluyen a modo explicativo.

Actualmente MONETA permite modelar procesos en UML-DA, BPMN, IDEF3 e INROMA, aunque cualquier otro lenguaje puede ser incorporado siguiendo las explicaciones detalladas en el capítulo 7.

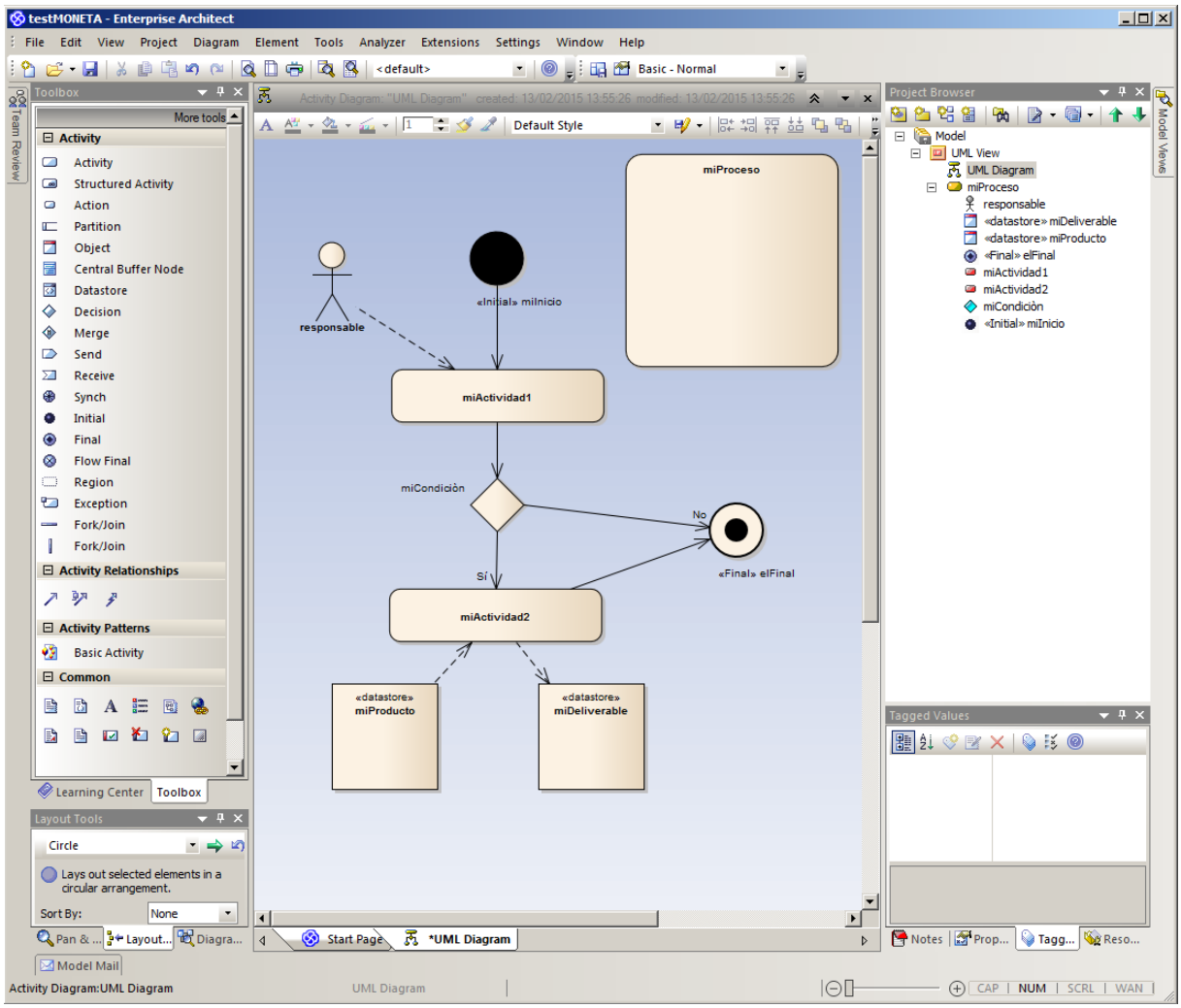


Figura D.1: Proceso modelado con el lenguaje UML-DA con MONETA

D.2. Obtener el modelo de proceso equivalente en otro lenguaje

Una vez creado el modelo de proceso en un lenguaje de modelado de procesos concreto con MONETA, podemos obtener el modelo equivalente del proceso en otro de los lenguajes incluidos en el marco de referencia, puesto que las transformaciones propias de dicho marco se encuentran implementadas en MONETA.

La ejecución de dichas transformaciones es totalmente transparente al usuario, el cual únicamente tiene que seguir los pasos para obtener el nuevo modelo de proceso. En la figura D.2 se muestran los pasos a seguir para transformar un modelo en BPMN a otro lenguaje. Hay que tener en cuenta que para poder hacer uso de esta funcionalidad, es necesario encontrarse sobre el diagrama donde se encuentra modelado el proceso.

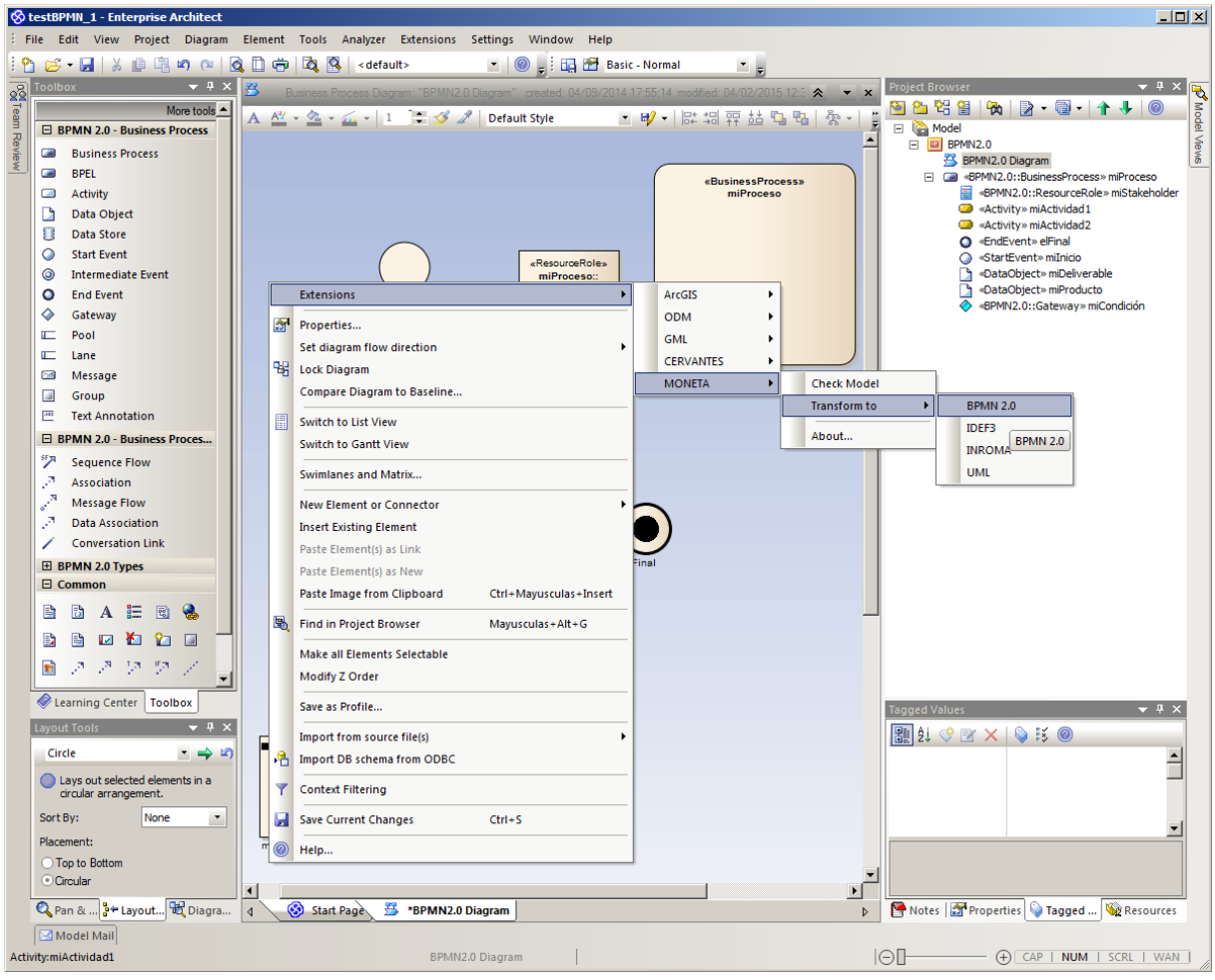


Figura D.2: Uso de MONETA para transformar un modelo en BPMN a otro lenguaje

Una vez ejecutada esta funcionalidad, se crea un modelo intermedio, denominado MONETA, que alberga una vista que contiene el modelo de proceso obtenido a partir de la transformación en el lenguaje elegido, y otra vista con el modelo del proceso en INROMA, puesto que es el lenguaje base utilizado y puede ser útil en actividades de interoperabilidad de procesos. En la figura D.3 se observa este hecho.

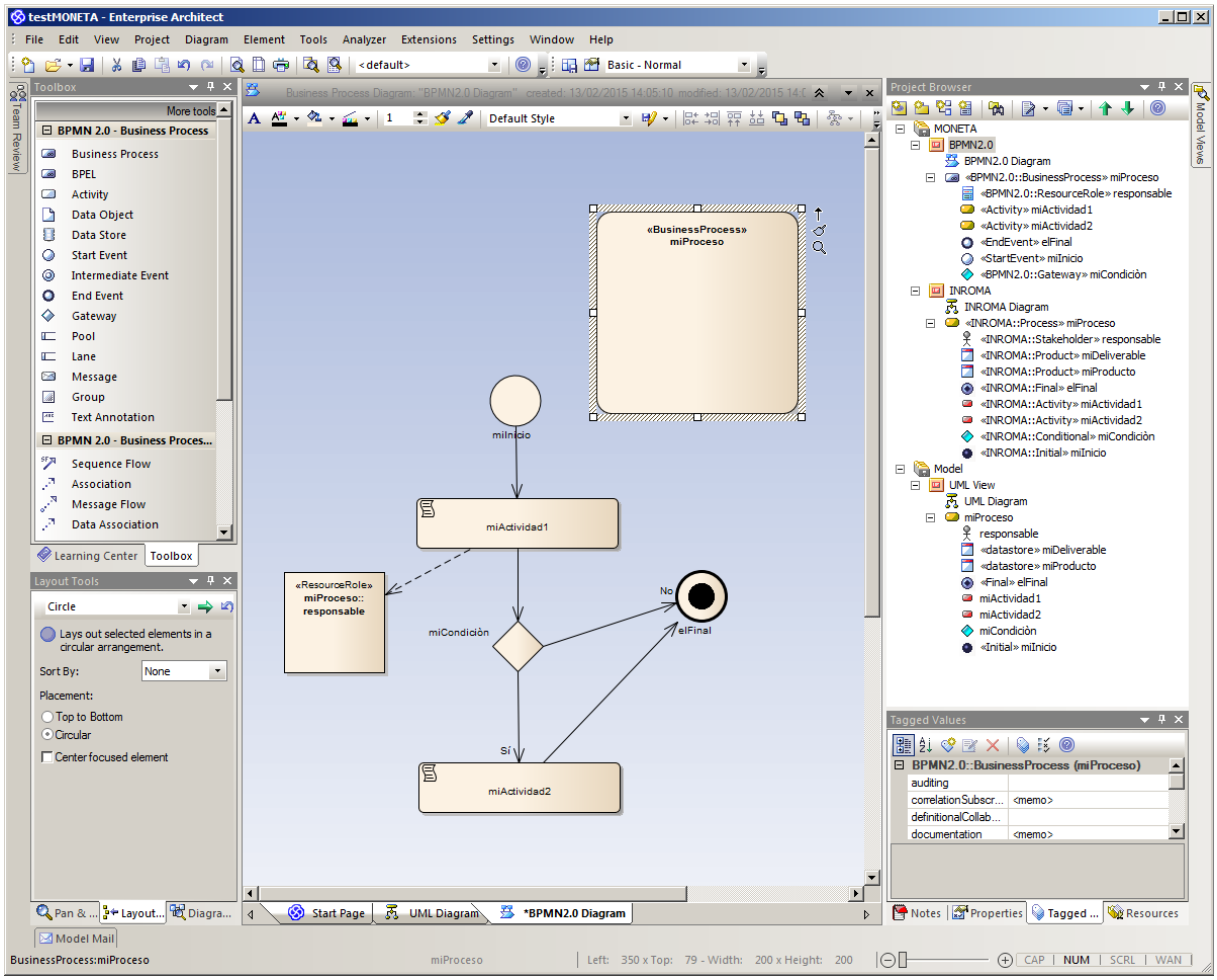


Figura D.3: Modelo MONETA después de su ejecución

Una vez revisadas, estas vistas deberían ser incorporadas al modelo en el que se está trabajando puesto que MONETA se seguirá utilizando como modelos intermedio para futuras transformaciones.

Anexo E

Actividad investigadora

Este anexo recoge la producción y la actividad investigadora de la doctoranda, clasificándola en diferentes apartados para facilitar su lectura. Así, en un primer apartado se recopila la información relacionada con la producción investigadora, esto es, capítulos de libros, artículos en revistas y artículos publicados en conferencias internacionales y nacionales.

En un segundo apartado se reúne la información referente a la actividad investigadora, es decir, estancias de investigación, comités, proyectos de I+D y redes de investigación en los que la doctoranda ha colaborado.

E.1. Producción investigadora

E.1.1. Capítulos de libro

García-García, J. A., Alba, M., García-Borgoñón, L., Escalona, M. J. *NDT-Suite: A model-based suite for the application of NDT*. Proceedings of the 12nd International Conference on Web Engineering (ICWE 2012) Lecture Notes in Computer Science. 2012 Vol. 7387, No. 1, pp. 469-472. 2012.

García-Borgoñón, L., García-García, J. A., Alba, M., Escalona, M. J. *Software Process Management: A Model-Based Approach*. Information Systems Development: Building Sustainable Information Systems (Proceedings of the 21st International Conference on Information Systems Development (ISD 2012)), pp 167-178. Editors: Henry Linger, Julie Fisher, Andrew Barnden, Chris Barry, Michael Lang, Christoph Schneider. ISBN: 978-1-4614-7539-2. 2013.

García-Borgoñón, L., Blanco, R., García-García, J. A., Barcelona, M. A. *Applying Testing Techniques to Software Process Assessment: A model-based perspective*. Information Systems Development: Improving Enterprise Communication (Proceedings of the 22nd International Conference on Information Systems Development (ISD 2013)), pp 333-344. Editors: María José Escalona, Gustavo Aragón, Henry Linger, Michael Lang, Chris Barry, Christoph Schneider. ISBN: ISBN 978-3-319-07214-2. 2014.

García, M. T., Barcelona, M. A., Ruiz, M., García-Borgoñón, L., Ramos, I. *A Discrete-Event Simulation Metamodel for obtaining Simulation Models from Business Process Models*. Information Systems Development: Improving Enterprise Communication (Proceedings of the 22nd International Conference on Information Systems Development (ISD 2013)), pp 333-344. Editors: María José Escalona, Gustavo Aragón, Henry Linger, Michael Lang, Chris Barry, Christoph Schneider. ISBN: ISBN 978-3-319-07214-2. 2014.

E.1.2. Revistas

Ponce, J., García-Borgoñón, L., García-García, J. A., Escalona, M. J., Domínguez-Mayo, F. J., Alba, M., Aragon, G. *A Model-Driven Approach for Business Process Management*. Covenant Journal of Engineering & Technology (CJICT). Vol. 1, No. 2, pp. 32-52. 2013.

García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Alba, M., Escalona, M. J. *Software Process Modeling Languages: A Systematic Literature Review*. Information and Software Technology (IST). Vol. 56, No. 2, pp. 103-116. 2014.

García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Escalona, M. J. *Software Process Accessibility in Practice: A Case Study*. Procedia Computer Science (Proceedings of the 5th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion, DSAI 2013) . Vol. 27, pp. 292-301. 2014.

García-García, J. A., Escalona, M. J., Mejías, M., García-Borgoñón, L. *A Model-Based Approach for Software Processes Modeling with PLM4BS*. Journal Information and Software Technology . Bajo Revisión. 2014.

Barcelona M.A., García-Borgoñón L., Escalona, M.J., Ramos I., Mejías M. *Supply Chain Business Process Modeling Languages: a Systematic Literature Review*. Journal of Systems and Software. Bajo Revisión. 2015.

E.1.3. Conferencias internacionales

García-Borgoñón, L., Escalona, M. J. *Globally Distributed Software Process Engineering*. Doctoral Symposium at the 6th International Conference on Global Software Engineering (ICGSE 2011) ISBN: 978-1-4577-1140-4. 2011.

García-García, J. A., Alba, M., García-Borgoñón, L., Escalona, M. J. *NDT-Suite: A model-based suite for the application of NDT*. Proceedings of the 12nd International Conference on Web Engineering (ICWE 2012) Lecture Notes in Computer Science. 2012 Vol. 7387, No. 1, pp. 469-472. 2012.

García-García, J. A., Victorio, J., García-Borgoñón, L., Barcelona, M. A., Domínguez-Mayo, F. J., Escalona, M. J. *A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology*. Proceedings of the 21st Annual Software Quality Management Conference (SQM 2013) ISBN: 978-0-9563140-8-6. 2013.

Domínguez-Mayo, F. J., García-García, J. A., García-Borgoñón, L., Escalona, M. J., Mejías, M. *QuEF framework and quality management of software products in organizations*. Proceedings of the 22nd Annual Software Quality Management Conference (SQM 2014) ISBN: 978-0-9926958-1-1. 2014.

García-Borgoñón, L., Barcelona, M. A., Peña, P., Escalona, M. J. *SoftAragón: a methodological framework for Software Process Improvement in SMEs*. Proceedings of the 22nd Annual Software Quality Management Conference (SQM 2014) ISBN: 978-0-9926958-1-1. 2014.

García-Borgoñón, L., Barcelona, M. A., Calvo, J. I., Ramos, I., Escalona, M. J. *CERVANTES: A Model-Based Approach for Service-Oriented Systems Development*. Information Systems Development: Transforming Organisations and Society through Information Systems (Proceedings of the 23rd International Conference on Information Systems Development (ISD 2014)), pp 298-305. Editors: Vjeran Strahonja, Neven Vrcek, Dijana Plantak Vukovac, Chris Barry, Michael Lang, Henry Linger, Christoph Schneider. ISBN: ISBN 978-953-6071-43-2. 2014.

E.1.4. Conferencias nacionales

García-Borgoñón, L., García-García, J. A., Alba, M. , Domínguez-Mayo, F. J. *Gestión de Procesos en Organizaciones de Desarrollo de Software: Un Enfoque Basado en Modelos*. Actas de las XVIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 113-126, ISBN: 978-84-695-8310-4. 2013.

Barcelona, M. A., García-Borgoñón, L., Calvo, J. I., Escalona, M. J. *CERVANTES: Un framework para el diseño y desarrollo de sistemas distribuidos*. Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 113-126, ISBN: 978-84-697-1152-1. 2014.

Guessous, M., Barcelona, M. A., García-Borgoñón, L., Alba, M. *Un Enfoque Basado en Modelos para incorporar Requisitos No Funcionales y de Integración de Software en el Diseño de Arquitecturas Orientadas a Servicios*. Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 113-126, ISBN: 978-84-697-1152-1. 2014.

E.2. Actividad investigadora

E.2.1. Estancias de investigación

- Agosto 2013. Una semana de estancia de investigación en el Laboratorio de Investigación y Formación en Informática Avanzada de la Universidad Nacional de La Plata, Argentina, bajo la supervisión de Dr. Gustavo Rossi.
- Julio-Agosto 2013. Seis semanas de estancia de investigación en la Southampton Solent University, Maritime and Technology Faculty, bajo la supervisón de Professor Margaret Ross.
- Abril 2013. Tres dias de estancia de investigación en el Laboratoire des Usages en Techniques d'Information Numériques, University of Paris 8, bajo la supervisión del Professor Charles Tijus.

E.2.2. Comités

- Miembro del Comité de Programa de la conferencia ISD2014 (23rd International Conference on Information Systems Development), que se ha llevado a cabo en Croacia en Septiembre del 2014.
- Miembro del Comité de Programa de la conferencia COMTEL (Congreso Internacional en Computación y Comunicaciones) que se celebró en Lima (Perú) en Octubre de 2014.
- Miembro del Comité de Programa y del Comité de Organización de la conferencia ISD2013 (22nd International Conference on Information Systems Development), que se llevó a cabo en Sevilla en Septiembre de 2013.
- Miembro del Comité de Programa de la conferencia COMTEL (Congreso Internacional en Computación y Comunicaciones) que se celebró en Lima (Perú) en Octubre de 2013.
- Miembro del Commité de Programa de la conferencia SEPGLA (Software Engineering Process Group, Latin America) desde 2005-2009.
- Miembro del Commité de Programa de la conferencia EUROMED SPI (Europe-Mediterranean Process Improvement Software) en 2010.

E.2.3. Redes de investigación

- **Redes nacionales**
  - Red CaSA - Calidad del software aplicada (TIN2010-12312-E9).
  - Grupo de Calidad de Software - INES
  - Red Temática para el desarrollo de soluciones de software de calidad en entornos PLM. Redes de Excelencia. Convocatoria 2015. (TIN2015-71938-REDT).

- **Redes internacionales**
  - AST - Red Internacional sobre arquitecturas de Testing.
  - Red Temática Mexicana en Ingeniería del Software (244467) del Consejo Nacional de Ciencia y Tecnología de Méxio en la convocatoria Conacit Redes Tematicas 2014.

E.2.4. Proyectos

Proyecto Empower	
Información del proyecto	Responsable: Julián Alberto García García, José Ponce Tipo de proyecto: Contrato 68/83 Referencia: P047-14/E09 Fecha de comienzo: 01/03/2015 Fecha de finalización: 31/12/2016

Red Temática Mexicana en Ingeniería del Software	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Consejo Nacional de Ciencia y Tecnología, CONACIT redes temáticas Referencia: 0244467 Fecha de comienzo: 01-12-2014
Financiado por	Consejo Nacional de Ciencia y Tecnología, México

Mecanismos Guiados en Etapas Tempranas para la Mejora del Software. Megus	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Ministerio de Economía y Competitividad Referencia: TIN2013-46928-C3-3-R Fecha de comienzo: 01/01/2014 Fecha de finalización: 31/12/2016
Financiado por	Ministerio de Economía y Competitividad

Mérimée Project: Paris-Seville	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Programme Mérimée de collaboration entre Ecoles Doctorales françaises et espagnoles Referencia: ED224 Fecha de comienzo: 05/11/2012 Fecha de finalización: 31/12/2014
Financiado por	European Union

MIDAS: Model and Inference Driven Automated Testing of Services Architectures	
Información del proyecto	Tipo de proyecto: FP7-ICT Strep Referencia: FP7-ICT-2012-8-318786 Fecha de comienzo: 01/09/2012 Fecha de finalización: 31/10/2015
Financiado por	European Union

Red sobre Calidad del Software Aplicada. Red CaSA	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Ministerio de Ciencia e Innovación Referencia: TIN2010-12312-E Fecha de comienzo: 21/07/2011 Fecha de finalización: 20/07/2012
Financiado por	Ministerio de Ciencia e Innovación

Testing temprano y modelos de simulación híbrida en la producción de software	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Plan Nacional del 2010 Referencia: TIN2010-20057-C03-02 Fecha de comienzo: 01-01-2011 Fecha de finalización: 31-12-2013
Financiado por	Ministerio de Ciencia e Innovación

ITChain: Tecnologías de la Información para la Cadena de Suministro Colaborativa	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Plan AVANZA Referencia: TSI-020302-2010-80 Fecha de comienzo: 02/11/2009 Fecha de finalización: 29/03/2013
Financiado por	Ministerio de Industria

NOVARED: Sistema para distribuir e incrementar el valor creado en internet	
Información del proyecto	Tipo de proyecto: Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica Referencia: IPT-2011-1942-430000 Fecha de comienzo: 01/01/2008 Fecha de finalización: 31/12/2010
Financiado por	Ministerio de Economía y Competitividad

LOGysTIC-A: Centro Nacional de Conocimiento en la Aplicación de las Tecnologías de la Información y las Comunicaciones a la resolución de problemas en el ámbito de la Logística	
Información del proyecto	Tipo de proyecto: Plan AVANZA Centros de Conocimiento Referencia: TSI-070200-2008-76 Fecha de comienzo: 01/01/2008 Fecha de finalización: 31/12/2010
Financiado por	Ministerio de Industria

DBE: Digital Business Ecosystem	
Información del proyecto	Tipo de proyecto: FP7-ICT Strep Referencia: IST-2002-507953 Fecha de comienzo: 03/11/2003 Fecha de finalización: 31/01/2007
Financiado por	European Union

IST@HOME: Delivering video-based IST services into European HOMEs	
Información del proyecto	Tipo de proyecto: FP7-ICT Strep Referencia: IST-2000-28406 Fecha de comienzo: 01/02/2002 Fecha de finalización: 29/06/2004
Financiado por	European Union

VULCANO: Promoción del desarrollo de software libre en un entorno de calidad y confianza adaptando las metodologías, procesos, modelos de negocio y últimas tecnologías	
Información del proyecto	Tipo de proyecto: Plan AVANZA Referencia: TSI-020301-2009-1 Fecha de comienzo: 01/01/2007 Fecha de finalización: 01/01/2010
Financiado por	Ministerio de Industria

Acrónimos

BPM Business Process Management

COBIT Control Objectives for Information and Related Technology

CIM Computation Independent Model

CMMI Capability Maturity Model Integration

EAI Enterprise Application Integration

EFQM European Foundation for Quality Management

INROMA INteroperabilidad, RObustez y MAntenibilidad (INteroperability, RObustness and MAntenibility)

ITIL Information Technology Infrastructure Library

ISE Information Systems Engineering

ISO International Organization for Standardization

IWT2 Ingeniería Web y Testing Temprano

MDA Model Driven Architecture

MDE Model Driven Engineering

MOF Meta Object Facility

M2M Model-to-Model Transformation

M2T Model-to-Text Transformation

NDT Navigational Development Techniques

OCL Object Constraint Language

OMG Object Management Group

PCI DSS Payment Card Industry Data Security Standard

PIM Platform Independent Model

PMBOOK A Guide to the Project Management Body of Knowledge

PML Process Modeling Language

PRINCE2 PRojects IN Controlled Environments

PSM Platform Specific Model

QGS Quasi-Gold Standard

QVT Query View Transformation

RQ Research Question

SQA Software Quality Assurance

SPE Software Process Engineering

SPEM Software Process Engineering Metamodel

SPICE Software Process Improvement and Capability Determination

SPML Software Process Modeling Language

UML Unified Modeling Language

TMMi Test Maturity Model integration

WfM Workflow Management

Glosario de Términos

Capacidad de abstraccion (Abstraction) Capacidad de un lenguaje de aislar ciertos aspectos de los modelos de procesos concretos de forma que faciliten su comprensión

Capacidad de analisis (Analyzability) Capacidad de un lenguaje para disponer de los elementos necesarios para comparar diferentes mo- delos de procesos, tomar decisiones y actividades similares

Comprensible (Understandability)) Capacidad de un lenguaje de ser entendido por los usuarios del mismo

Comprobable (Testability) Capacidad de un lenguaje para realizar de forma automática pruebas que supervisen cómo se promulga el modelo

Ejecutabilidad (Executability) Capacidad de un lenguaje para soportar construcciones con semántica operacional que ofrezcan el soporte para la ejecución de esas construcciones

Escalabilidad (Scalability) Capacidad de un lenguaje para describir procesos de diferentes tamaños, tanto grandes como pequeños

Estandarizacion (Standardization) Grado de cercanía de un lenguaje a estándares o lenguajes de uso generalizado como UML

Expresividad (Expressiveness) Capacidad de un lenguaje para expresar lo que realmente se realiza durante los procesos de desarrollo de software

Flexibilidad (Flexibility) Capacidad de un lenguaje para poder ser aplicado en una variedad de entornos de procesos, no únicamente en el ámbito del software

Formalidad (Formality) Capacidad de un lenguaje para expresar formalmente la sintaxis y semántica

Gestion de Procesos de Negocio (Business Process Management) Conjunto de métodos, técnicas y herramientas que soportan el diseño, aprobación, gestión, análisis y mejora de los procesos de negocio

Ingenieria del Software Disciplina o área de la informática que ofrece métodos y técnicas para el desarrollo y el mantenimiento de calidad que resuelva todo tipo de problemas

Ingenieria orientada a modelos (Model Driven Development) Paradigma de desarrollo de software que se centra en la creación y explotación de modelos de dominio

Modelo de Proceso Descripción abstracta de un proceso actual o propuesto que representa elementos seleccionados del mismo, que son considerados importantes para el propósito del modelo y pueden ser ejecutados por una persona o una máquina

Modularidad (Modularity) Capacidad de un lenguaje para combinar diferentes pedazos o componentes de procesos con el objetivo de construir uno nuevo

Navigational Development Techniques Propuesta metodológica web guiada por modelos que ofrece un soporte completo, incluyendo herramientas, para todo el ciclo de vida de un sistema software: desarrollo, pruebas, aseguramiento de la calidad y gestión del proyecto

NDTQ-Framework Framework de trabajo que define todos los procesos actualmente soportados por NDT

Proceso Consiste en el uso de los recursos para transformar entradas en salidas debido a que algún tipo de trabajo, actividad o función se ha llevado a cabo

Proceso de Negocio Conjunto de tareas relacionadas de manera lógica que se deben realizar para proporcionar valor a los clientes o cumplir otros objetivos estratégicos

Proceso de Software Conjunto de tecnologías, mejores prácticas y estrategias que permitan gestionar de manera efectiva el desarrollo realizado por grandes equipos de ingenieros software a lo largo de todo el ciclo de vida de los productos

Promulgable (Enactabiliy) Capacidad de un lenguaje para que un modelo pueda imitar la ejecución real del proceso

Reflexion (Reflection) Capacidad de un lenguaje para soportar directamente la evolución de los modelos de procesos

Representacion grafica (Graphical representation) Capacidad de un lenguaje para disponer de una semántica gráfica que facilite su descripción e intercambio entre los usuarios del mismo

Simulable (Simulability) Capacidad de un lenguaje para que un modelo pueda ser simulado previo a su ejecución

Sintaxis Abstracta Describe la estructura del lenguaje y la forma en la que se pueden combinar los distintos elementos, independientemente de su representación

Sintaxis Concreta Especifica la representación legible (gráfica o textual) de los elementos abstractos

Soportado por herramientas (Tooling Support) Disponibilidad de herramientas que nos permitan definir los procesos con el lenguaje

Soporte a multiples vistas (Support Multiple Views) Capacidad de un lenguaje para disponer de los elementos semánticos necesarios para la definición de vistas desde varias perspectivas del modelo de procesos

Transformacion Mecanismo que nos permite derivar unos modelos a partir de otros modelos ya existentes

Transformacion Horizontal Aquella en la que el modelo derivado y el origen tienen el mismo nivel de abstracción

Transformacion M2M Transformación Modelo-a-Modelo: representa una relación entre dos sintaxis abstractas y se define mediante un conjunto de relaciones entre los elementos de los correspondientes metamodelos

Transformacion Vertical Aquella en la que el modelo derivado tiene un menor nivel de abstracción que el modelo inicial