UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

Doctorado en Ingeniería

Mención en Ingeniería en Sistemas de Información

Tesis Doctoral

"Diseño y Gestión de Modelos de Procesos de Negocio en Colaboraciones Inter-organizacionales"

Lic. Ivanna Maricruz Lazarte

Director Dr. Pablo D. Villarreal

Codirector Dr. Omar Chiotti

Santa Fe, Argentina. Abril de 2013

Lazarte, Ivanna Maricruz
Diseño y gestión de modelos de procesos de negocio en colaboraciones inter-organizacionales. - 1a ed. - Santa Fe : el autor, 2013.
332 p. ; 29x21 cm.

ISBN 978-987-33-3845-8

1. Negocios. 2. Aplicaciones Informáticas. I. Título
CDD 005.3

Fecha de catalogación: 19/09/2013

UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

Comisión de Posgrado

Se presenta esta Tesis en cumplimiento de los requisitos exigidos por la Universidad Tecnológica Nacional para la obtención del grado académico de Doctor en Ingeniería, mención Sistemas de Información

"Diseño y Gestión de Modelos de Procesos de Negocio en Colaboraciones Inter-organizacionales"

por

Lic. Ivanna Maricruz Lazarte

Director Dr. Pablo D. Villarreal

Codirector Dr. Omar Chiotti

Jurados de Tesis

Dr. Daniel Riesco

Dr. Germán Montejano

Dra. Ma. Laura Caliusco

Santa Fe, Argentina. Abril de 2013

A mi familia.

Índice General

Índice de Tablas

Índice de Figuras

Prólogo

La globalización, los mercados modernos, las nuevas filosofías de gestión de organizaciones y los avances en las Tecnologías de Información y Comunicación, alientan a las organizaciones a formar redes colaborativas y establecer colaboraciones inter-organizacionales entre los miembros de dichas redes. Las colaboraciones inter-organizacionales están impulsadas por la necesidad de agilidad, adaptabilidad y flexibilidad de las organizaciones para mantener o mejorar su desempeño y competitividad en el mercado global.

Una colaboración inter-organizacional implica una integración orientada a procesos entre organizaciones heterogéneas y autónomas, que debe ser alcanzada tanto a *nivel organizacional* como a *nivel tecnológico*. A nivel organizacional, las organizaciones se centran en el diseño de procesos (de negocio) colaborativos para acordar el comportamiento de la colaboración. Un *proceso colaborativo* define una vista global de las interacciones entre las organizaciones para alcanzar metas comunes. Estos procesos colaborativos sirven como una base contractual para la colaboraciones inter-organizacionales, pero no son ejecutables. Para implementar y ejecutar un proceso colaborativo en forma descentralizada, cada organización debe definir sus procesos de negocio internos, denominados *procesos de interfaz* (públicos) y *procesos de integración* (privados). La solución definida en este nivel se denomina *solución inter-organizacional*.

A nivel tecnológico, las organizaciones generan las especificaciones (código) ejecutables de procesos de integración y las interfaces de los sistemas de las organizaciones usando estándares Business-to-Business. La solución definida en este nivel se denomina *solución tecnológica*. Esta solución permite dar soporte a la ejecución de los procesos colaborativos. En consecuencia, las soluciones definidas en ambos niveles deben tener una mutua correspondencia.

La solución inter-organizacional debe definirse usando modelos conceptuales para lograr el entendimiento y la comunicación de los procesos definidos entre todos los "stakeholders" participantes. La definición de estos modelos conceptua-

les de procesos con independencia de la tecnología de implementación posibilita su implementación en diferentes plataformas, facilitando y fomentando su re-uso. Además, los modelos de procesos de interfaz e integración deben ser interoperables y mantenerse sincronizados y consistentes con los modelos de procesos colaborativos.

Por lo tanto, la tesis propone métodos y herramientas que posibilitan el diseño y la gestión de los modelos que conforman la solución inter-organizacional.

En primer lugar, propone una metodología que identifica las fases y artefactos requeridos para el desarrollo e implementación de colaboraciones inter-organizacionales. Esta metodología incluye métodos basados en el Desarrollo Dirigido por Modelos - Model-Driven Development (MDD) y la Arquitectura Dirigida por Modelos - Model-Driven Architecture (MDA). Los modelos de procesos de negocio definidos como la solución inter-organizacional son usados para generar y construir los artefactos de software de la solución tecnológica de una colaboración inter-organizacional.

En segundo lugar, propone un método basado en MDA para generar modelos de procesos de interfaz interoperables entre sí y consistentes con los modelos de procesos colaborativos. Los modelos de procesos de interfaz se definen con el lenguaje BPMN y se generan automáticamente a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP, aplicando una transformación de modelo-a-modelo. La definición de transformación del método se especificó analizando la similitud semántica de ambos lenguajes y aplicando la teoría de patrones de workflows para definir el flujo de control del modelo generado.

En tercer lugar, propone un método basado en MDA para asistir en el diseño de modelos de procesos de integración interoperables y consistentes. Este método es utilizado para generar en forma automática plantillas y modelos de procesos de integración definidos con el lenguaje BPMN, a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP, aplicando una transformación de modelo-a-modelo. La definición de transformación del método se especificó analizando la similitud semántica de ambos lenguajes, aplicando la teoría de patrones de workflows para definir el flujo de control del modelo generado. Además, se aplicó la teoría de patrones de actividades para derivar las actividades públicas y principalmente privadas que una organización requiere realizar para dar soporte al intercambio de mensajes definidos en un proceso

colaborativo.

En último lugar, propone un repositorio distribuido para gestionar los modelos de procesos colaborativos, de interfaz e integración. El repositorio distribuido fue definido de acuerdo a los principios del diseño orientado a servicios y está basado en la Arquitectura Orientada a Servicios - Service-Oriented Architecture (SOA) para definir servicios interoperables, reusables, distribuidos y débilmente acoplados. El repositorio posibilita implementar un repositorio global público, en el que las organizaciones gestionan en forma conjunta modelos de procesos colaborativos y de interfaz. El repositorio también permite a las organizaciones gestionar sus modelos de procesos de integración en forma local y privada. El repositorio integra los métodos que generan modelos de procesos de interfaz e integración, garantizando que los mismos son interoperables y consistentes, como así también provee servicios para verificarlos, validarlos y mantener sus versiones en forma sincronizadas.

Los resultados parciales del trabajo realizado en la tesis han sido divulgados a través de las siguientes publicaciones:

- Lazarte, Ivanna M.; Thom, Lucinéia H.; Iochpe, Cirano; Chiotti, Omar y Villarreal, Pablo D. (2013). «A distributed repository for managing business process models in cross-organizational collaborations». *Computers in Industry*, **64**, pp. 252-267. ISSN: 0166-3615 (ISI Impact Factor 2010 = 1,529).

- Thom, Lucinéia H.; Lazarte, Ivanna M.; Iochpe, Cirano; Priego, Luz M.; Verdier, Christine; Chiotti, Omar y Villarreal, Pablo D. (2011). «On the Capabilities of BPMN for Workflow Activity Patterns Representation». En: Dijkman, Remco; Hofstetter, Jörg y Koehler, Jana (Eds.), *3rd International Workshop and Practitioner Day on Business Process Model and Notation (BPMN 2011)*. Lecture Notes in Business Information Processing (LNBIP), volumen 95(2). Springer, pp. 172-177.

- Lazarte, Ivanna M.; Villarreal, Pablo D.; Chiotti, Omar; Thom, Lucinéia H. y Iochpe, Cirano (2011). «An MDA-based Method for Designing Integration Process Models in B2B Collaborations». En: *International Conference on Enterprise Information Systems (ICEIS 2011)*. INSTICC, pp. 55-65.

- Lazarte, Ivanna M.; Tello-Leal, Edgar; Roa, Jorge; Chiotti, Omar y Villarreal, Pablo D. (2010). «Model-Driven Development Methodology for B2B Collaborations». En: *International Workshop on Models and Model-driven Methods for Service Engineering (3M4SE 2010)*. IEEE Computer Society, pp. 69-78.

- Villarreal, Pablo D.; Lazarte, Ivanna M.; Roa, Jorge y Chiotti, Omar (2010). «A Modeling Approach for Collaborative Business Processes based on the UP-ColBPIP Language». En: Rinderle-Ma, Stefanie; Sadiq, Shazia y Leymann, Frank (Eds.) *Business Process Management Workshops*. Lecture Notes in Business Information Processing (LNBIP), volumen 43. Springer, pp. 318-329.

- Lazarte, Ivanna M.; Chiotti, Omar y Villarreal, Pablo D. (2009). «Transforming Collaborative Process Models into Interface Process Models by Applying an MDA Approach». En: *AIS Transactions on Enterprise Systems*, **2**. GITO Publishing GMBH, pp. 13-23. ISSN: 1867-7134.

- Lazarte, Ivanna M.; Chiotti, Omar y Villarreal, Pablo D. (2009). «Transforming Collaborative Process Models into Interface Process Models by Applying an MDA Approach». En: Godart, Claude; Gronau, Norbert; Sharma, Sushil y Canals, Gérôme (Eds.) *Software Services for e-Business and e-Society (I3E 2009)*. IFIP Advances in Information and Communication Technology, volumen 305. Springer, pp. 301-315.

- Lazarte, Ivanna M. (2009). «Method for Modeling and Specification of Interface/Integration Business Processes and Services Oriented Architectures for B2B Collaborations». En: *10th Argentine Symposium on Software Engineering (ASSE 2009)*, 38 JAIIO. Publicado en CD-ROM.

Asimismo, cabe señalar que la autora de esta tesis realizó una estadía de investigación de tres meses en el Instituto de Informática, de la Universidad Federal de Río Grande del Sur (UFRGS) de Porto Alegre, Brasil. Para ello, le fue otorgada una beca en el marco del Proyecto CAPG-BA ME 032/07, para el intercambio académico entre el Instituto de Informática de la Universidad Federal de Río Grande del Sur y el Doctorado en Ingeniería de la UTN Facultad Regio-

nal Santa. Supervisores durante la estadía: Dra. Lucinéia H. Thom y Dr. Cirano Iochpe.

Convenciones Tipográficas

Los estilos de letra que se muestran a continuación se utilizan en esta tesis para distinguir elementos de lenguajes de modelado o elementos de interfaces de usuario, del castellano común.

- Los elementos de lenguajes de modelado (meta-modelos, clases y atributos) se escriben en `typewriter`.
- Los elementos de interfaces de usuario se escriben en sans serif.

Así mismo, las palabras que pertenecen a un idioma diferente al castellano, se escriben en "este estilo".

Resumen

En una colaboración inter-organizacional, las organizaciones se centran en el diseño de procesos (de negocio) colaborativos para acordar el comportamiento de la colaboración. No obstante, el diseño de procesos internos (de interfaz y de integración) constituyen un desafío importante para que las organizaciones puedan implementar y gestionar colaboraciones inter-organizacionales.

Esta tesis propone métodos y herramientas que posibilitan el diseño y la gestión de los modelos de procesos de negocio internos con el propósito de integrar los mismos con los procesos de negocio colaborativos acordados en el marco de colaboraciones inter-organizacionales.

Con el propósito de guiar el proceso de desarrollo e implementación de colaboraciones inter-organizacionales se propone una metodología que sigue un enfoque "top-down" basada en los principios del desarrollo dirigido por modelos. La metodología identifica las fases, actividades y artefactos requeridos para generar soluciones tecnológicas a partir de soluciones inter-organizacionales.

Para dar soporte al diseño de los procesos de negocio internos se proponen dos métodos de desarrollo dirigidos por modelos que permiten generar automáticamente los modelos de procesos de interfaz e integración que cada organización requiere para implementar colaboraciones inter-organizacionales. Estos métodos permiten definir modelos de procesos de interfaz e integración interoperables y consistentes con el comportamiento definido en los modelos de procesos colaborativos.

Para permitir a las organizaciones gestionar los modelos conceptuales de procesos de negocio que se definen en la solución inter-organizacional en forma distribuida, manteniendo a los mismos consistentes, interoperables, sincronizados y libres de errores lógicos, se propone un repositorio distribuido de modelos de procesos de negocio involucrados en colaboraciones inter-organizacionales.

Agradecimientos

Quisiera dedicar la finalización de esta tesis a todas aquellas personas que me han acompañado y facilitado su apoyo, consejo y ánimo a lo largo de este proceso, sin las cuales no hubiera sido posible lograr este objetivo.

En primer lugar quiero agradecer a mi director, Dr. Pablo David Villarreal, no sólo por ofrecerme sus valiosos conocimientos y experiencia profesional, sino también por su paciencia, dedicación y orientación durante el desarrollo de la tesis.

También quiero expresar mi gratitud a mi codirector, Dr. Omar Chiotti, por su apoyo permanente, sus aportes y colaboración en el desarrollo de la tesis.

Agradezco a la Dra. Lucinéia Thom, Profesora Asistente del Instituto de Informática de la Universidad Federal de Rio Grande del Sur (UFRGS), por compartir sus conocimientos que fueron fuente de inspiración de trabajos en conjunto.

Mi agradecimiento al Lic. Juan Antonio Verón, Profesor Titular de la Facultad de Tecnología y Ciencias Aplicadas de la Universidad Nacional de Catamarca (UNCa), quien creyó en mi capacidad para recorrer con éxito este camino antes que yo misma.

Agradezco a la Universidad Nacional de Catamarca (UNCa), a la que debo mi formación profesional y mis inicios en la investigación.

Mi agradecimiento al CONICET por haber hecho posible la realización de esta tesis a través del soporte económico facilitado con una Beca de Posgrado, y a la UTN Facultad Regional Santa Fe por el espacio y material brindado.

A las instituciones que brindaron aporte económico para la realización de este trabajo mediante subsidios a los siguientes proyectos:

- *Tecnología de Información para el Desarrollo de Procesos de Negocio Colaborativos*. Ente financiador: Agencia Nacional de Promoción Científica y Técnica, PAE 37122 - PICT-118. Período: 2009-2012.

- *TI para Desarrollar Procesos de integración que implementan Procesos Colaborativos*. Ente financiador: CONICET, PIP 112-200801-02421. Período: 2009-2011.

- *Herramientas de Software para el Diseño de Procesos de Negocio Colaborativos y el Desarrollo de Sistemas de Información Business-to-Business*. Ente financiador: UTN, 25/O116. Período: 2010-2012.

Agradezco a los integrantes del CIDISI, especialmente a las "brujas", por crear un ambiente de trabajo cálido y brindar el apoyo emocional tan necesario en esta tarea.

Quiero agradecer especialmente a mis amigos Jorge Roa, Mariano Rubiolo y Edgar Tello-Leal por su cordialidad, apoyo y gratos momentos compartidos.

Por último, y muy especialmente, agradezco a Verónica Torrente, Edith Carrizo, Aurelia Luna y María Inés Lazarte por su cariño, comprensión, apoyo y aliento constante.

Ivanna M. Lazarte
Santa Fe, Argentina
Abril 2013

CAPÍTULO 1

Introducción

El capítulo describe el contexto en el que se enmarca la tesis (Sección 1.1), los problemas a resolver y los objetivos planteados en el trabajo de investigación (Sección 1.2). Se mencionan las principales contribuciones alcanzadas (Sección 1.3), y se presenta la organización general de la tesis (Sección 1.4).

1.1. El contexto

En las últimas décadas, debido a la globalización de los mercados, la apertura económica y el entorno competitivo generado, las organizaciones se han enfocado en la gestión de sus procesos de negocio como base para desarrollar la tarea de gestión de las mismas. La Gestión de Procesos de Negocio - Business Process Management (BPM) es una disciplina de gestión organizacional que se enfoca en los procesos de una organización (también llamados procesos de negocio) y define conceptos, métodos, y técnicas para el diseño, administración, configuración, ejecución y análisis de dichos procesos (Weske, 2007). Su objetivo es aplicar cambios incrementales o radicales a procesos de negocio, enfatizando la mejora continua, la satisfacción del cliente y la participación de los empleados (Reijers y otros, 2010).

Un *proceso de negocio* consiste de un conjunto de actividades que son ejecutadas en forma coordinada en una organización para lograr una meta organizacional o de negocio (Ouyang y otros, 2009; Weske, 2007; zur Muehlen y Indulska, 2010). Dichas actividades pueden ser realizadas por empleados de la organización manualmente o con la ayuda de sistemas de información. También hay actividades que pueden ser ejecutadas automáticamente por sistemas de información, sin ninguna intervención humana (Weske, 2007).

Las nuevas Tecnologías de Información y Comunicación (TICs), principal-

mente aquellas basadas en Internet, han posibilitado a las organizaciones extender sus procesos de negocio para colaborar con otras organizaciones, estableciendo *colaboraciones inter-organizacionales*. Las colaboraciones inter-organizacionales están impulsadas por la necesidad de agilidad, adaptabilidad y flexibilidad de las organizaciones para mantener o mejorar su desempeño y competitividad en el mercado global; presionando a las mismas a crear sistemas de información capaces de adaptarse y responder rápidamente a las condiciones de negocio cambiantes (Xu y otros, 2011).

Mediante las colaboraciones inter-organizacionales, las organizaciones forman redes colaborativas. Una *red colaborativa* consiste de organizaciones autónomas y heterogéneas que colaboran para alcanzar una meta común más rápidamente y/o a un menor costo (Camarinha-Matos y otros, 2009; Chituc y otros, 2009; Jiménez y otros, 2005; Li y otros, 2010; Roser y otros, 2011).

El comportamiento de una colaboración inter-organizacional entre dos o más organizaciones se define a través de procesos de negocios colaborativos (también denominados coreografías de procesos (Hofreiter, 2008; OMG, 2011a; Weske, 2007) o procesos de negocio inter-organizacionales (Bauer y otros, 2005)). Un *proceso de negocio colaborativo*, o simplemente proceso colaborativo (Figura 1.1) define, desde un punto de vista global, las interacciones entre las organizaciones participantes que ejecutan diferentes roles, para lograr una meta de negocio común (Bauer y otros, 2005; Villarreal y otros, 2007b; Weske, 2007). Dichas interacciones describen la coreografía de mensajes intercambiados entre los roles desempeñados por las organizaciones, y sirven como una base contractual para la colaboración inter-organizacional (Decker y Weske, 2007; Weske, 2007).

Un proceso colaborativo es un proceso abstracto, no ejecutable directamente (Lazarte y otros, 2009). Para implementar y ejecutar un proceso colaborativo en forma descentralizada, cada organización debe definir sus procesos de negocio internos. Un proceso de negocio interno puede ser *público*: proceso de interfaz (Lazarte y otros, 2009) (también denominado proceso abstracto (OASIS, 2007), proceso público (OMG, 2011a), interfaz de comportamiento (Weske, 2007)); o *privado*: proceso de integración (Lazarte y otros, 2011) (también denominado proceso ejecutable (Bauer y otros, 2005; OASIS, 2007), proceso privado (OMG, 2011a) o proceso de orquestación (Weske, 2007)). Un *proceso de interfaz* (Figura 1.1) representa la vista particular que una organización tiene del proceso colabo-

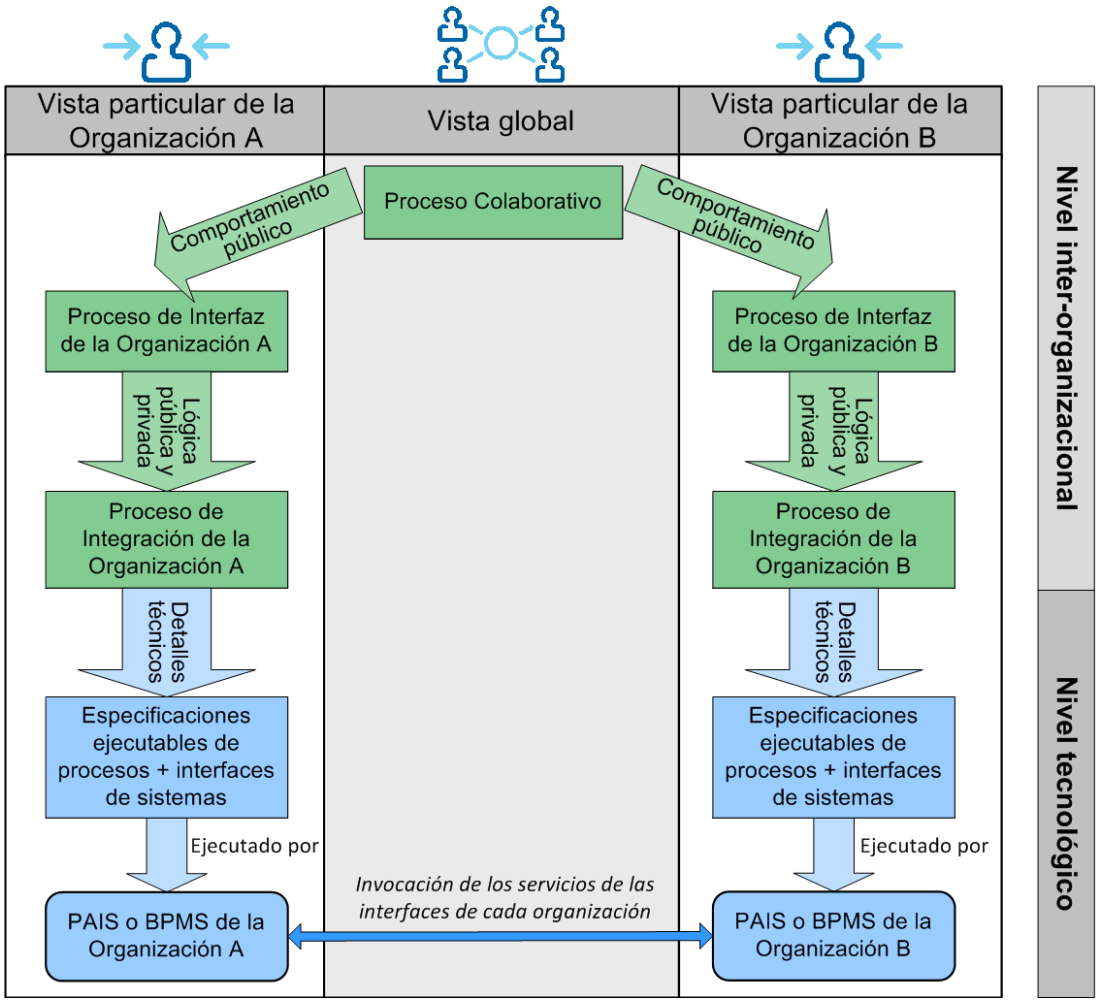


Figura 1.1: Procesos de negocio requeridos para implementar una colaboración inter-organizacional.

rativo. Un proceso de interfaz define el comportamiento público y externamente visible de una organización, el cual es expresado en términos de las actividades que soportan el envío y la recepción de mensajes con otras organizaciones, de acuerdo con el comportamiento definido en el proceso colaborativo del que se deriva. Un *proceso de integración* (Figura 1.1) define y combina el comportamiento y las actividades públicas (derivadas de un proceso de interfaz) con las actividades privadas que una organización debe ejecutar para dar soporte al rol que ésta desempeña en un proceso colaborativo. Un proceso de integración incorpora las actividades privadas, las cuales permiten generar y procesar la información intercambiada entre las organizaciones, realizar la transformación de datos e invocar a sistemas de información internos. De esta manera, la ejecución descentralizada de los procesos colaborativos se realiza a través de la ejecución distribuida y

concurrente de los procesos de integración de las organizaciones.

En resumen, para implementar una colaboración inter-organizacional son necesarios tres tipos de procesos: procesos colaborativos, procesos de interfaz y procesos de integración. Dichos procesos deben ser definidos usando conceptos del dominio del problema, a nivel organizacional (también denominado nivel de negocio), modelados con alto nivel de abstracción e independiente de la plataforma tecnológica (Lazarte y otros, 2010). Estos modelos conceptuales forman la *solución inter-organizacional*.

Los modelos conceptuales de procesos de integración son refinados en diferentes iteraciones hasta obtener modelos con suficientes detalles de implementación. Estos modelos detallados son usados luego en un nivel tecnológico para generar las especificaciones (código) ejecutables de procesos de integración y las interfaces de los sistemas de información de las organizaciones (Figura 1.1) usando estándares Business-to-Business (B2B) tales como Web Services Business Process Execution Language (WS-BPEL) (OASIS, 2007), Web Services Choreography Description Language (WS-CDL) (W3C, 2007b), Electronic Business using eXtensible Markup Language (ebXML) (Bernauer y otros, 2003). Los artefactos definidos en este nivel forman la *solución tecnológica*, la cual puede ser implementada a través de la configuración de Sistemas de Información Orientados a Procesos - Process-Aware Information Systems (PAISs) (Dumas y otros, 2005), o de Sistemas de Gestión de Procesos de Negocio - Business Process Management Systems (BPMS) (Karagiannis, 1995), que interpretan las especificaciones B2B y ejecutan los procesos de integración (Figura 1.1) de acuerdo al comportamiento acordado en un proceso colaborativo. Esto es realizado mediante la invocación a los servicios provistos por las interfaces de los sistemas de información que permiten la generación y procesamiento de la información intercambiada y el intercambio de mensajes de negocio.

De esta manera, el desarrollo e implementación de colaboraciones inter-organizacionales y de los procesos de negocio involucrados implica la consideración y cumplimiento de los siguientes requisitos (Bauer y otros, 2005; Roser y Bauer, 2005; Villarreal y otros, 2007b; Xu y otros, 2011):

- **Vista global de las interacciones entre las organizaciones**. Una colaboración inter-organizacional no debe definirse desde el punto de vista de una única organización, sino que debe describir la vista de cada una de las

mismas. La vista global posibilita que las organizaciones entiendan y visualicen de la misma forma la colaboración, y conduce al establecimiento de confianza entre ellas. Esta vista debe ser expresada a través de los procesos colaborativos.

- **Gestión descentralizada**. Cada organización debe gestionar el rol que va a desempeñar en una colaboración inter-organizacional, a través de una vista particular de la organización con respecto a los procesos colaborativos en los que participa. Esto implica que cada organización debe definir, implementar y gestionar las actividades públicas y privadas que permiten desempeñar el rol que cumple en los procesos colaborativos. Esto es soportado a través de los procesos de interfaz y los procesos de integración, los cuales son internos a cada organización.

- **Interacciones "peer-to-peer"**. Los sistemas de información de las organizaciones deben interactuar de manera directa, sin la mediación de terceros, mediante la especificación de mecanismos de interacción que habiliten el intercambio y uso de información entre nodos heterogéneos, de tal manera de posibilitar la gestión descentralizada de los procesos colaborativos, a través de la ejecución distribuida y concurrente de los procesos de integración de las organizaciones.

- **Interoperabilidad**. Una colaboración inter-organizacional requiere la integración de los sistemas de información de las organizaciones participantes. Para lograrlo, se requiere garantizar la interoperabilidad en diferentes capas (protocolos de comunicación, información, procesos) a través del uso de lenguajes comunes y/o estándares de intercambio de datos y especificaciones de procesos.

- **Autonomía de las organizaciones**. En una colaboración inter-organizacional, se asume que cada organización participante es autónoma. Esto significa que además de las metas comunes a alcanzar, cada organización tiene sus propias metas y por lo tanto tiene el control de sus acciones internas y puede decidir diferentes cursos de acción en las interacciones con las demás organizaciones. Esto significa que aquellas actividades internas de cada organización, requeridas para el procesamiento de la información

recibida o para la generación de la información a enviarse, no deben ser definidas en un modelo de proceso colaborativo, debido a que forman parte de los aspectos privados de la organización. Las mismas deben ser mantenidas en los procesos de integración de manera privada.

- **Soporte para negociaciones complejas**. En una colaboración inter-organizacional, se requiere de mecanismos de negociación que permita a las organizaciones definir compromisos y acuerdos mediante el intercambio de mensajes para poder tomar decisiones en forma conjunta.

- **Capacidad de adaptación a cambios**. Las colaboraciones inter-organizacionales son impulsadas por la necesidad de agilidad, flexibilidad y adaptabilidad de las organizaciones al mercado global. Esto se traduce en la necesidad de cambios frecuentes en sus sistemas de información, para responder rápidamente a los cambios de los requerimientos de negocio. Para ello se requiere de mecanismos ágiles, eficientes y eficaces que permitan modificar y gestionar los modelos de procesos de negocio para que los sistemas de información puedan interpretar los nuevos requerimientos de negocio.

- **Alineación entre la solución inter-organizacional y la solución tecnológica**. En una colaboración organizacional se requiere alcanzar la alineación entre la solución inter-organizacional y la solución tecnológica con el propósito de garantizar que la colaboración se lleva a cabo según se acordó en los procesos colaborativos.

1.2. Los problemas a resolver y los objetivos

En el desarrollo de colaboraciones inter-organizacionales, la definición de los modelos de procesos difiere en: los puntos de vista en que son descritos (global o interno), el lenguaje de modelado usado para expresarlos (UP-ColBPIP, Let's Dance, BPMN, BPEL4Chor, EPC y UML), la fase de desarrollo en que son usados los modelos de procesos (análisis, diseño, implementación), las personas destinatarias de los modelos (analistas de negocio, diseñadores y desarrolladores de sistemas, etc.), o simplemente el nivel de abstracción (conceptuales, ejecutables) y la granularidad de los mismos. El uso de modelos conceptuales para definir

los tres tipos de procesos involucrados en colaboraciones inter-organizacionales es un requisito importante para lograr el entendimiento y la comunicación de los procesos definidos entre todos los "stakeholders" participantes. La definición de estos modelos conceptuales de procesos con independencia de la tecnología de implementación posibilita su implementación en diferentes plataformas, facilitando y fomentando su re-uso.

Par alcanzar estos requerimientos, las organizaciones deben disponer de métodos y herramientas de software para diseñar y gestionar los tres tipos de modelos de procesos de negocio. Esta tarea se torna compleja y desafiante para las organizaciones, particularmente cuando las mismas integran varias redes colaborativas y establecen varias colaboraciones inter-organizacionales. La necesidad de adaptación a cambios frecuentes en los procesos de negocio, lleva a que se requieran herramientas ágiles, eficientes y eficaces que permitan definir, modificar y gestionar dichos modelos de procesos para que los sistemas de información de las organizaciones puedan interpretar los nuevos requerimientos de las colaboraciones. Se deben considerar también los aspectos mencionados anteriormente acerca de las colaboraciones inter-organizacionales.

Lo expresado se traduce en los siguientes requerimientos principales a resolver:

- Soporte al *diseño de modelos conceptuales de procesos internos.*
- Garantía de *consistencia e interoperabilidad* entre los procesos colaborativos y los procesos internos de las organizaciones.
- Soporte a la *gestión de los modelos de procesos.*
- Mantenimiento de la *privacidad* de los modelos de procesos de integración de las organizaciones.
- Soporte a la *sincronización de los modelos de procesos.*

El diseño y modelado de procesos colaborativos ha sido un área de investigación intensiva en los últimos años en la cual se pueden encontrar diversas propuestas (Bauer y otros, 2005; Huemer y otros, 2008; OMG, 2011a; Villarreal y otros, 2006b). No obstante, no ha ocurrido lo mismo con el *diseño y modelado* de los procesos internos que las organizaciones requieren para llevar adelante la

ejecución de los procesos colaborativos. No se dispone de métodos y herramientas que posibiliten diseñar correctamente los procesos internos de las organizaciones e integrarlos con los procesos colaborativos. Esto se refiere a poder derivar el comportamiento de los mismos a partir del comportamiento de los procesos colaborativos.

El diseño de los procesos de integración es un aspecto clave para lograr la ejecución exitosa de un proceso colaborativo a través de la ejecución descentralizada de los mismos. Los procesos de interfaz e integración deben ser consistentes con su correspondiente proceso colaborativo e interoperables con los procesos de integración de las demás organizaciones participantes. *Consistencia* se refiere a la coherencia que debe existir entre el comportamiento definido en un proceso de interfaz o de integración y su correspondiente proceso colaborativo, de tal manera que la lógica de comportamiento de la colaboración definida en un proceso colaborativo se refleje en la lógica del proceso de interfaz y/o de integración (Lazarte y otros, 2010). *Interoperabilidad* (también denominado compatibilidad (Decker y Weske, 2007; Weske, 2007)) se refiere a la capacidad de los procesos de interfaz e integración de interactuar, a través de un intercambio sincronizado de mensajes, de acuerdo al comportamiento definido en un proceso colaborativo (Lazarte y otros, 2010).

Por otra parte, los modelos de procesos colaborativos y de interfaz requieren ser compartidos y editados por "stakeholders" distribuidos en las organizaciones involucradas en la colaboración inter-organizacional, mientras que los modelos de procesos de integración requieren ser compartidos y editados por "stakeholders" de una organización, manteniéndolos en el ámbito de la misma. Esto requiere de herramientas de software que permitan la *gestión* (almacenamiento, acceso, compartición, versionamiento, etc.) de estos modelos de procesos, posibilitando el acceso compartido de los modelos de procesos colaborativos y de procesos de interfaz, y ofreciendo mecanismos de acceso privado a los modelos de procesos de integración de las organizaciones. *Privacidad* se refiere a la necesidad de evitar que organizaciones externas puedan acceder a la lógica de negocio interna definida en un proceso de integración de una organización.

Finalmente, el requerimiento de *sincronización* refiere a que un cambio en un modelo de proceso colaborativo debe verse reflejado en los correspondientes modelos de procesos de interfaz e integración, de tal manera de satisfacer una

correcta implementación de los procesos colaborativos y mantener la consistencia e interoperabilidad entre los modelos de procesos.

A partir de estos problemas a resolver, se plantea el objetivo general de la tesis.

Objetivo general:
Desarrollar métodos y herramientas que posibiliten el diseño y la gestión de los modelos de procesos de negocio que las organizaciones requieren para integrar sus procesos de negocio internos con los procesos de negocio colaborativos acordados en el marco de colaboraciones inter-organizacionales.

Por un lado, se pretende construir métodos, basados en los principios del desarrollo dirigido por modelos, que asistan a las organizaciones en el diseño y la generación automática o semi-automática de los modelos de procesos de interfaz y de integración a partir de modelos de procesos colaborativos. También se pretende proveer guías acerca de las fases y actividades requeridas para la definición de estos procesos. Por otro lado, se pretende proveer herramientas que posibiliten la gestión de estos modelos de procesos. El propósito es que estos métodos y herramientas puedan garantizar la consistencia e interoperabilidad entre los procesos de integración de las organizaciones, como así también la sincronización de éstos con los procesos colaborativos. Además, se espera que los métodos y herramientas a generar posibiliten a las organizaciones disminuir los tiempos, costos y complejidad en el desarrollo de la solución inter-organizacional para ambientes de colaboración inter-organizacionales y faciliten la generación de la solución tecnológica.

Si bien se propusieron varias metodologías para el desarrollo de colaboraciones inter-organizacionales, tales como (Bauer y otros, 2005; Huemer y otros, 2008; Roser y otros, 2006; Villarreal y otros, 2006b), las mismas se enfocan en la definición de procesos colaborativos y la generación de soluciones tecnológicas a partir de éstos, sin dar soporte a la definición de los procesos de negocio internos requeridos para la implementación de colaboraciones inter-organizacionales.

Esto da lugar a la definición del primer objetivo específico de la tesis.

Objetivo 1:
Proveer una metodología, basada en el desarrollo dirigido por modelos, que describa las fases, actividades y artefactos (modelos de procesos de negocio) requeridos para el desarrollo de colaboraciones inter-organizacionales.

El propósito de esta metodología es proveer una representación más clara de los artefactos de desarrollo requeridos, la separación de incumbencias y los diferentes niveles de abstracción. Se pretende que la metodología utilice lenguajes, métodos y herramientas que exploten los beneficios del desarrollo dirigido por modelos, para disminuir el tiempo y la complejidad en el desarrollo de colaboraciones inter-organizacionales.

El Desarrollo Dirigido por Modelos - Model-Driven Development (MDD) es un enfoque para la construcción de software que permite un desarrollo eficiente mediante el modelado en diferentes niveles de abstracción (Anneke G. Kleppe, 2003; Selic, 2003). Aplicando este enfoque, los modelos son los principales artefactos en la construcción de software para simular, estimar, entender, comunicar y producir código (Gherbi y otros, 2009). Tanto MDD como la Arquitectura Dirigida por Modelos - Model-Driven Architecture (MDA) (OMG, 2003b) permiten explotar los beneficios de las transformaciones de modelos automatizadas para lograr una rápida propagación de cambios en el diseño (nivel organizacional) a cambios en la implementación (nivel tecnológico), con el fin de permitir a las organizaciones adaptarse más rápidamente a los entornos dinámicos en el que se desenvuelven, reduciendo la complejidad y costos de desarrollo, y mejorando la calidad del software generado.

En el desarrollo de colaboraciones inter-organizacionales, las organizaciones deben definir sus procesos de interfaz para que puedan entender y enfocarse en los requerimientos de negocio que deben cumplir según el rol que desempeñan en los procesos colaborativos.

Esto da lugar a la definición del segundo objetivo específico de la tesis:

Objetivo 2:
Proveer un método para la generación de los modelos de procesos de interfaz de las organizaciones a partir de los modelos de procesos colaborativos.

El propósito es dar soporte a la generación de la solución inter-organizacional para producir modelos de procesos que representen el comportamiento público de la colaboración inter-organizacional desde el punto de vista del rol que cada organización desempeña.

Para completar la solución inter-organizacional, cada organización debe diseñar sus procesos de integración para poder implementar y ejecutar los procesos colaborativos. Los requerimientos de interoperabilidad y consistencia hacen que

el diseño de los mismos sea una tarea compleja, costosa y propensa a errores. Un proceso de integración definido incorrectamente imposibilitará la ejecución del proceso colaborativo, o en el mejor de los casos, se ejecutará con fallas, violando el acuerdo definido en el proceso colaborativo.

El diseño de procesos de integración requiere del conocimiento del analista de negocio para identificar y agregar en forma correcta las actividades públicas y privadas necesarias para dar soporte al intercambio de mensajes. Facilitar y agilizar la tarea de diseño, permitirá a las organizaciones adaptarse más rápidamente y con menor esfuerzo a nuevos requerimientos organizacionales o de negocio.

Esto da lugar a la definición del tercer objetivo específico de la tesis.

Objetivo 3:
Proveer un método para la generación automática de los modelos de procesos de integración de las organizaciones a partir de los modelos de procesos colaborativos.

El propósito de este método es generar modelos de procesos de integración interoperables y consistentes con la lógica global acordada en un proceso colaborativo. El modelo de proceso de integración obtenido servirá como un modelo inicial para los analistas de negocio y diseñadores de sistemas involucrados, los cuales deberán refinarlo en sucesivas iteraciones hasta obtener un modelo con detalles de implementación.

Además de métodos de diseño de procesos de interfaz e integración, para dar soporte a una metodología de desarrollo e implementación de colaboraciones inter-organizacionales, se requieren herramientas que posibiliten la gestión de los modelos de procesos. La tecnología de repositorio proporciona una infraestructura adecuada para la gestión de colecciones de modelos de procesos de negocio (Dijkman y otros, 2012). Los sistemas de repositorios que proveen funcionalidades específicas para la gestión de modelos de procesos de negocio se denominan *repositorios de modelos de procesos de negocio* (Yan y otros, 2012). Dichos repositorios proveen y explotan las funcionalidades comúnmente provistas por repositorios y bases de datos en general, tales como almacenamiento, recuperación, "check in/out", control de versiones y control de acceso. Para la gestión de modelos de procesos en colaboraciones inter-organizacionales, el repositorio debe satisfacer los requerimientos de acceso y almacenamiento compartido para los modelos de procesos colaborativos y de interfaz, como así también acceso y almacenamiento

privado para los modelos de procesos de integración. Debe garantizar la sincronización, consistencia e interoperabilidad de los modelos de procesos colaborativos con los correspondientes modelos de procesos de integración de las organizaciones. Si bien existen varias propuestas de repositorios para la gestión de modelos de procesos de negocio, tales como Chituc y otros (2009); Hofreiter (2008); La Rosa y otros (2011b); Ma y otros (2007); Theling y otros (2005); Vanhatalo y otros (2006), ninguna ofrece todas las funcionalidades requeridas en el dominio de las colaboraciones inter-organizacionales, en particular la gestión descentralizada de los mismos, privacidad, sincronización, interoperabilidad y consistencia.

Esto da lugar a la definición del cuarto objetivo específico de la tesis.

Objetivo 4:
Desarrollar un repositorio distribuido para la gestión de modelos de procesos de negocio involucrados en colaboraciones inter-organizacionales.

El propósito del repositorio distribuido es ofrecer una herramienta que les permita a las organizaciones gestionar (almacenar, acceder, mantener, compartir, versionar) los modelos conceptuales de procesos de negocio que se definen en las diferentes fases del desarrollo de colaboraciones inter-organizacionales, manteniendo a los mismos consistentes, interoperables, sincronizados y libres de errores lógicos. Para satisfacer algunos de estos aspectos, se propone implementar, automatizar e integrar en el repositorio los métodos de diseño de procesos de interfaz e integración para proveer nuevos servicios y funcionalidades al repositorio.

Finalmente, se requiere validar el repositorio distribuido que da soporte a la metodología, como así también de los métodos para la generación de procesos de interfaz e integración. Esto da lugar a la definición del quinto objetivo específico de la tesis.

Objetivo 5:
Validar el repositorio distribuido, junto con los métodos para la generación de procesos de interfaz e integración, a través del desarrollo de casos de estudio.

El propósito es demostrar la funcionalidad, aplicabilidad, factibilidad y utilidad del repositorio y los métodos propuestos para llevar adelante el diseño y la gestión de modelos de procesos de negocio en el desarrollo e implementación de colaboraciones inter-organizacionales.

1.3. Principales contribuciones

Para alcanzar el primer objetivo se propone una **metodología para el desarrollo de colaboraciones inter-organizacionales**, que aplica un enfoque "top-down". La metodología permite identificar, definir y representar claramente los artefactos requeridos para el desarrollo de colaboraciones inter-organizacionales, como así también indica las técnicas, lenguajes y métodos a usar, los cuales están basados en los principios y conceptos de MDD y MDA.

La metodología propone el uso del lenguaje UP-ColBPIP (Villarreal y otros, 2010, 2007b). Dicho lenguaje permite modelar procesos colaborativos mediante protocolos de interacción, posibilitando la definición del comportamiento global a través de una coreografía de mensajes.

Aplicando la metodología propuesta, los modelos de procesos de negocio definidos como la solución inter-organizacional son usados para construir los artefactos de software de la solución tecnológica de una colaboración inter-organizacional.

Para alcanzar el segundo objetivo, se propone un **método para la generación de modelos de procesos de interfaz**, que aplica los principios de MDA. Este método es utilizado para generar en forma automática modelos de procesos de interfaz definidos con el lenguaje BPMN, a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP, aplicando una transformación de modelo-a-modelo.

El modelo de proceso de interfaz generado se define con el lenguaje BPMN, el cual es un lenguaje estándar que permite modelar procesos de negocio orientados a actividades independientes de la plataforma, usando una notación que es fácilmente entendible por todos los usuarios de negocio, reduciendo la brecha entre el diseño de procesos de negocio y la implementación de los mismos (OMG, 2011a).

Para alcanzar el tercer objetivo, se propone un **método para la generación de modelos de procesos de integración**, que aplica los principios de MDA. Este método es utilizado para generar en forma automática procesos de integración (definidos con el lenguaje BPMN) interoperables y consistentes con la lógica global acordada en un proceso colaborativo (definido con el lenguaje UP-ColBPIP). La consistencia es garantizada siguiendo un proceso de transfor-

maciones de modelo-a-modelo. La interoperabilidad en el intercambio de mensajes definido en los modelos de proceso de integración es garantizada mediante la aplicación de *Patrones de Actividades de Workflow* (Thom y otros, 2009) en las reglas de transformación del método propuesto. Estas reglas permiten generar las actividades públicas y privadas de un modelo de proceso de integración, las cuales son requeridas por cada organización para dar soporte al intercambio de mensajes inter-organizacionales definidos en los procesos colaborativos.

Para alcanzar el cuarto objetivo, se propone un **sistema de repositorio distribuido para la gestión de modelos de procesos de negocios involucrados en colaboraciones inter-organizacionales**, basado en una Arquitectura Orientada a Servicios - Service-Oriented Architecture (SOA) (Erl, 2007).

La arquitectura definida para el repositorio distribuido permite a las organizaciones acceder a un repositorio global compartido para gestionar las colaboraciones inter-organizacionales y los modelos de los procesos colaborativos que se definen en las mismas. También permite que cada organización implemente repositorios locales para almacenar sus modelos de procesos de integración. SOA brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente Servicios Web (W3C, 2004)), que facilita la interacción entre el repositorio público y los repositorios locales, y permite la gestión distribuida de los modelos de procesos de negocio involucrados en una colaboración inter-organizacional.

El repositorio provee servicios que extienden las funcionalidades provistas por repositorios de modelos de procesos tradicionales, con el propósito de ofrecer herramientas de diseño de modelos de procesos de negocio y asegurar los requerimientos de consistencia, interoperabilidad y sincronización entre los procesos colaborativos y los procesos de integración, preservando los aspectos privados de las organizaciones. Para ello, el repositorio integra en los servicios los métodos basado en MDA propuestos para la generación de procesos de interfaz e integración, como así también métodos de verificación de modelos de procesos colaborativos (Roa y otros, 2012) y de chequeo de consistencia de modelos de procesos de integración (Martens, 2005), además de mecanismos de sincronización de los modelos de procesos colaborativos con los procesos de interfaz y de integración.

1.4. Organización de la tesis

El Capítulo 2 tiene por objetivo establecer el marco teórico de los conceptos usados a lo largo de la presente tesis. Presenta los principales trabajos relacionados asociados a cada una de las contribuciones de la tesis.

El Capítulo 3 describe la metodología para el desarrollo de las colaboraciones inter-organizacionales basada en el desarrollo dirigido por modelos.

El Capítulo 4 describe un método basado en MDA para la generación automática de procesos de interfaz definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP. Presenta las reglas de transformación requeridas para la ejecución automática del método.

El Capítulo 5 describe un método basado en MDA para la generación automática de procesos de integración definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP. Presenta las reglas de transformación requeridas para la ejecución automática del método.

El Capítulo 6 describe el sistema de repositorio distribuido para la gestión de modelos de procesos de negocio en el contexto de colaboraciones inter-organizacionales. Presenta la arquitectura basada en SOA de dicho repositorio, sus funcionalidades y las entidades que almacena.

El Capítulo 7 muestra la funcionalidad, aplicabilidad, viabilidad y utilidad del repositorio mediante casos de estudio.

El Capítulo 8 destaca las principales contribuciones de esta tesis y describe los aspectos de las mismas que constituyen el punto de partida para el desarrollo de trabajos futuros.

CAPÍTULO 2

Marco Teórico y Trabajos Relacionados

El presente capítulo describe el estado del arte en el que se sustenta la investigación, exponiendo conceptos y detalles de lenguajes, paradigmas, metodologías y plataformas de desarrollo de software referidos en el Capítulo 1. Se presentan conceptos relacionados con la gestión de procesos de negocio, sus fases y los lenguajes de modelado de procesos de negocio que se utilizan en el presente trabajo de tesis (Sección 2.1). Se describen los principios del desarrollo dirigido por modelos, el funcionamiento de las transformaciones de modelos, y lenguajes y herramientas que lo soportan (Sección 2.2). Se describen los principios de la arquitectura orientada a servicios (Sección 2.3). Se analizan y discuten los trabajos de investigación relacionados con el enfoque del presente trabajo de tesis (Sección 2.4).

2.1. Gestión de procesos de negocio

Las organizaciones han adoptado la *Gestión de Procesos de Negocio - Business Process Management (BPM)* para estandarizar, integrar y optimizar sus procesos de negocio con el propósito de responder rápidamente a los cambios de requerimientos del mercado, mejorando su competitividad.

Para soportar estos procesos de negocio, los sistemas de información de las organizaciones deben conocer dichos procesos y el contexto organizacional en el que se ejecutan. Esto ha producido un cambio de sistemas de información orientado a datos a *Sistemas de Información Orientados a Procesos - Process-Aware Information Systems (PAISs)* (Aalst, 2004).

BPM incluye conceptos, métodos y técnicas para soportar el diseño, gestión, configuración, ejecución y análisis de procesos de negocio. El ciclo de vida de BPM consta de cuatro fases (Figura 2.1): Diseño y Análisis, Configuración, Ejecución

Figura 2.1: Ciclo de vida de BPM.

y Evaluación (Aalst, 2004; Weske, 2007). Dicho ciclo se aplica en forma separada a cada uno de los procesos de negocio de una organización o entre organizaciones

En la fase *Diseño y Análisis*, se identifican, revisan, validan los procesos de negocio y se representan mediante modelos de procesos de negocio, usando una notación gráfica para facilitar la comunicación y entendimiento de los mismos. En esta fase se aplican técnicas de modelado de procesos, de validación, simulación y verificación.

En la fase *Configuración*, los diseños de procesos son implementados mediante la configuración de PAISs. Para lo cual, los modelos de procesos de negocio son refinados con detalles técnicos que permiten su ejecución. La configuración incluye las interacciones de los usuarios con el sistema así como la integración de sistemas existentes con los PAISs.

La fase *Ejecución* comprende la ejecución real de los procesos de negocio usando los sistemas configurados. En esta fase, los procesos de negocio son instanciados y automatizados para cumplir las metas de negocio de una organización.

En la fase *Evaluación* se utiliza la información disponible para evaluar y mejorar los modelos de procesos de negocio y sus implementaciones.

2.1.1. Clasificación de procesos de negocio

En esta sección se clasifican los procesos de negocio de acuerdo a sus principales dimensiones (Weske, 2007).

2.1.1.1. Organizacionales vs Operacionales

En BPM, se pueden identificar diferentes niveles, que van desde las estrategias de negocio a los procesos de negocio implementados (Weske, 2007). En el primer nivel, se encuentran las estrategias de la organización que le permitirán alcanzar una ventaja competitiva sustentable en el mercado. En el segundo nivel, se encuentran las metas de negocio, las cuales pueden dividirse en sub-metas.

En el tercer nivel, se encuentran los *procesos de negocio organizacionales*, que son procesos de alto nivel especificados típicamente en forma textual indicando sus entradas, sus salidas, los resultados esperados y sus dependencias de otros procesos de negocio de la organización.

En el cuarto nivel, se encuentran los *procesos de negocio operacionales*, en el cual se especifican las actividades y sus relaciones, ignorando los aspectos de implementación. Los procesos operacionales se especifican en modelos de procesos de negocio y son la base para el desarrollo de procesos de negocio automatizados e implementados.

En el quinto nivel, se encuentran los *procesos de negocio implementados*, los cuales contienen información sobre la ejecución de las actividades del proceso y el entorno técnico y organizacional en el que se ejecutará.

2.1.1.2. Intra-organizacionales vs Inter-organizacionales

Los *procesos intra-organizacionales* son aquellos en donde las actividades son realizadas por recursos de una organización y sin la interacción con otras organizaciones. El objetivo principal de estos procesos es la simplificación de los procesos internos de las organizaciones, eliminando las actividades que no aportan valor.

Los *procesos inter-organizacionales* son aquellos que interactúan con los procesos de negocio de otras organizaciones. En esta categoría se encuentran los procesos colaborativos, procesos de interfaz y procesos de integración, los cuales son el foco de la tesis.

2.1.1.3. Según el grado de automatización

Los procesos de negocio pueden diferir en el grado de automatización. Hay procesos de negocio que son completamente automatizados, lo que significa que no

se requiere de la intervención humana para su ejecución. También hay procesos de negocio que tienen actividades manuales pero que también incluyen actividades automatizadas.

2.1.1.4. Según el grado de repetición

Los procesos de negocio pueden ser altamente repetitivos (generalmente aquellos completamente automatizados) como por ejemplo, la venta de pasajes de una línea aérea. En el otro extremo, están los procesos de negocio que se ejecutan sólo unas pocas veces como por ejemplo, el diseño de un buque.

2.1.1.5. Según el grado de estructuración

Los procesos de negocio pueden ser estructurados, en los cuales se especifican todas las actividades y las restricciones para su ejecución, o no estructurados, en los cuales sus actividades se pueden derivar en tiempo de ejecución o ejecutar en cualquier orden o incluso varias veces.

2.1.2. Modelado de procesos de negocio

El *modelado de procesos de negocio* es un pre-requisito fundamental para las organizaciones que deseen adoptar BPM (Indulska y otros, 2009a) y se realiza en la primera fase de su ciclo de vida para separar la lógica del proceso de la lógica de las aplicaciones, de tal manera que el proceso de negocio pueda ser automatizado mediante PAISs (Lu y Sadiq, 2007). Con el propósito de ofrecer productos de alta calidad o servicios de manera eficiente, las organizaciones dividen el trabajo en un número de funciones específicas. El modelado de procesos de negocio tiene como objetivo optimizar estas funciones para lograr la eficiencia de la organización y mantener un alto nivel de calidad.

El resultado del modelado de procesos de negocio son *modelos de procesos de negocio* que describen gráficamente las actividades, eventos/estados y la lógica de flujo de control que constituye un proceso de negocio. También pueden incluir información sobre los datos involucrados, recursos organizacionales y de Tecnologías de la Información (TI), y otros artefactos como “stakeholders” externos, metas, métricas de riesgo y desempeño (Indulska y otros, 2009b). Estos modelos de procesos de negocio se usan como un medio de entendimiento común

entre los "stakeholders" involucrados y sirven para la identificación de problemas en el proceso (por ejemplo, cuellos de botella) y el descubrimiento de nuevas oportunidades de mejora de los mismos (Aalst y otros, 2003b).

En las siguientes sub-secciones se describen los lenguajes usados para modelar procesos de negocio y luego los patrones de procesos que permiten re-usar el conocimiento capturado en los mismos para obtener modelos de procesos de negocio de mayor calidad.

2.1.2.1. Lenguajes de modelado de procesos de negocio

Existen diferentes lenguajes de modelado de procesos de negocio, por ejemplo, BPMN, EPC, UML, UP-ColBPIP, Let's Dance, Workflow Nets. Un lenguaje de modelado proporciona la sintaxis y la semántica adecuada para especificar con precisión los requerimientos de un proceso de negocio. La sintaxis proporciona la gramática de los elementos del lenguaje para especificar los objetos y sus dependencias, mientras que la semántica define una interpretación coherente del modelo para reflejar la lógica del proceso subyacente.

Una cuestión importante a tener en cuenta en el desarrollo e implementación de colaboraciones inter-organizacionales es el modelado de procesos colaborativos. Esto requiere de un lenguaje de modelado que satisfaga los requerimientos del dominio de las colaboraciones inter-organizacionales y que además permita modelar la vista global de las interacciones entre las organizaciones (como se discutió en el capitulo anterior). Actualmente, el lenguaje BPMN ofrece un tipo de diagrama denominado *Coreografía* (OMG, 2011a) que soporta estas características. Sin embargo, este lenguaje no provee semántica a los mensajes intercambiados entre las organizaciones, lo cual es un requerimiento importante para identificar las actividades privadas requeridas para soportar el intercambio de mensajes, como se discute en el Capítulo 5. Villarreal y otros (2010, 2007b) proponen el lenguaje UP-ColBPIP, el cual es un lenguaje orientado a mensajes que además de proveer semántica a los mensajes intercambiados definidos en una vista global, permite definir los requerimientos de negocio.

Por otro lado, para definir los procesos de interfaz e integración requeridos para implementar una colaboración inter-organizacional, se precisa de un lenguaje de modelado orientado a actividades que permita representar procesos de negocio independientes de la plataforma, desde el punto de vista particular de

una organización. El lenguaje BPMN (OMG, 2011a) incorpora los conceptos de procesos de interfaz e integración a través de lo que denomina procesos públicos y privados, respectivamente.

A continuación se describen los lenguajes UP-ColBPIP y BPMN utilizados en el marco de esta tesis.

Lenguaje UP-ColBPIP

El lenguaje UP-ColBPIP extiende la semántica de UML para modelar procesos colaborativos con independencia de la tecnología (Villarreal y otros, 2010, 2007b). Fue definido como un perfil UML proporcionando notaciones gráficas fáciles de comprender por los analistas de negocio y diseñadores de sistemas. Este lenguaje aplica un enfoque de desarrollo "top-down" y ofrece elementos conceptuales para el modelado de cinco vistas:

- *Vista de colaboración inter-organizacional*: captura la información que identifica a las organizaciones y sus relaciones de comunicación. También captura los roles que las organizaciones desempeñan en el acuerdo de colaboración y describe en forma jerárquica las metas de negocio comunes que han acordado.

- *Vista de procesos colaborativos*: en esta vista se representan los procesos colaborativos requeridos para alcanzar las metas de negocio acordadas y se definen los documentos de negocio que contienen la información a ser intercambiada por las organizaciones.

- *Vista de protocolos de interacción*: en esta vista se define formalmente el comportamiento de los procesos colaborativos mediante protocolos de interacción. Estos protocolos describen patrones de comunicación de alto nivel a través de una coreografía de mensajes de negocio. Los aspectos de coordinación y comunicación de la colaboración inter-organizacional son representados en la vista del protocolo de interacción mediante el uso de actos de comunicación ("speech acts") (Tabla 2.1). Cada mensaje de negocio tiene un acto de comunicación asociado que representa la intención del emisor con respecto al documento de negocio asociado al mismo.

- *Vista de documentos de negocio*: en esta vista se representan los documentos de negocio que se intercambian en los procesos de colaboración. Son representados mediante diagramas de clases y se hace referencia a estos documentos en varias de las vistas del lenguaje.

- *Vista de interfaz de negocio*: es una vista estática de la colaboración que describe las interfaces de negocio que representan los roles desempeñados por las organizaciones. Las interfaces de negocio contienen los servicios necesarios para el intercambio de mensajes definidos en la vista de protocolos de interacción.

Tabla 2.1: Principales actos de comunicación usados por el lenguaje UP-ColBPIP.

Accept-Proposal: representa la acción de aceptar una propuesta para realizar una acción.
Agree: representa la acción de acordar ejecutar alguna acción, posiblemente en el futuro.
Call-for-Proposal: representa la acción de un rol i solicitando a otro rol j una propuesta para realizar alguna acción.
Inform: representa la acción de un rol de informar a otro que una proposición determinada es verdadera.
Propose: representa la acción de enviar una propuesta para realizar una cierta acción, dadas ciertas precondiciones.
Refuse: representa la acción de negarse a realizar una acción dada y explicar las razones del rechazo.
Reject-Proposal: representa la acción de rechazar una propuesta (previamente presentada) para realizar una acción durante una negociación.
Request: representa la acción de un rol de solicitarle a otro que realice una acción. La proposición asociada al acto describe la acción a ser ejecutada.

A continuación se describe la *Vista de protocolos de interacción*, la cual es usada para definir el comportamiento de los procesos colaborativos. Más detalles de este lenguaje pueden encontrarse en (Villarreal y otros, 2010, 2007b).

Los principales elementos conceptuales, cuya sintaxis concreta se muestra en la Figura 2.2, usados para definir protocolos de interacción son:

- *Trading Partner (Socio de negocio)*: representa una organización que forma parte de una colaboración inter-organizacional, en la cual dicha organización entabla una relación estrecha con otras organizaciones.

- *Partner Role (Rol del socio)*: define el rol que desempeña un socio de negocio u organización en una colaboración inter-organizacional.

- *Business Message (Mensaje de negocio)*: define una interacción o comunicación entre dos roles, el que envía el mensaje (el remitente) y el que lo recibe (el receptor). La semántica concreta de cada mensaje está dada por la semántica del acto de comunicación asociado al mensaje. Además, un mensaje de negocio indica la transferencia de información de un rol hacia otro. El contenido que transporta el mensaje es indicado por el documento de negocio asociado al mismo.

- *Protocol Reference (Referencia de protocolo)*: hace referencia a otro protocolo de interacción, denominado el sub-protocolo o protocolo anidado. Una referencia de protocolo indica que un protocolo de interacción contiene una llamada a un sub-protocolo, la cual representa la inserción del comportamiento del sub-protocolo dentro del protocolo. Cuando el sub-protocolo es ejecutado, el protocolo que lo invoca queda esperando hasta que el sub-protocolo finalice.

- *Termination (Terminación)*: representa una finalización explícita del protocolo en un punto dado. La semántica concreta de la terminación está dada por el atributo `endState`, el cual puede tomar dos valores: `Success` o `Failure`. Un protocolo siempre tiene una terminación exitosa implícita, como consecuencia de la ejecución del último mensaje.

- *Control Flow Segments (Segmentos de flujo de control)*: representan una secuencia de mensajes compleja. Un segmento de flujo control está compuesto de uno o varios caminos de interacción, los cuales a su vez pueden contener otros segmentos de flujo de control, sub-protocolos, secuencia de mensajes, etc., es decir, cualquier otro elemento de un protocolo de interacción. La semántica de un segmento de flujo de control depende del operador de flujo de control utilizado: *And*, *Xor*, *Or*, *Loop*, *Exception*, *Cancel*, *Multiple Instances*, and *If*.

- *Interaction Path (Camino de interacción)*: representa un camino posible de ejecución en un segmento de flujo de control.

- *Time Constraint (Restricción de tiempo)*: representa el tiempo límite disponible para la ejecución del elemento del protocolo al cual está asociada.
- *Condition (Condición)*: representa una expresión booleana que restringe la ejecución de un camino de interacción o el envío de un mensaje de negocio.

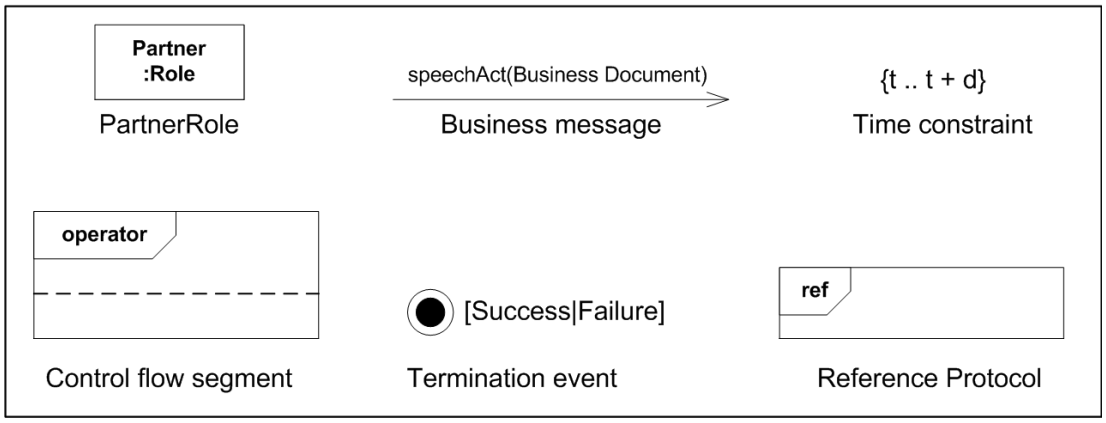


Figura 2.2: Resumen de la sintaxis concreta del lenguaje UP-ColBPIP.

La sintaxis abstracta del lenguaje UP-ColBPIP puede encontrarse en el Apéndice A.1.

Lenguaje BPMN

BPMN es un lenguaje estándar que proporciona a las organizaciones de una notación gráfica que les permite comprender sus procesos de negocio internos y comunicar estos procesos de manera estándar. El lenguaje BPMN permite representar modelos de procesos independientes de la plataforma en un alto nivel de abstracción creando un puente estandarizado para disminuir la brecha entre los procesos de negocio y la implementación de estos (OMG, 2011a).

Si bien el lenguaje BPMN permite representar múltiples diagramas (Procesos, Colaboración, Conversación y Coreografías), en esta tesis se describen los elementos correspondientes a un diagrama *Procesos* y a un diagrama *Colaboración*, los cuales permiten describir el comportamiento interno de las actividades de una organización, como así también indicar los puntos de interacción entre dos o más organizaciones.

Los principales elementos usados para definir procesos de interfaz e integración en un diagrama de Colaboración, cuya sintaxis concreta se muestra en la Figura 2.3, son:

- *Pool*: representa cada organización participando en una colaboración inter-organizacional.
- *Message (Mensaje)*: representa el contenido de una comunicación entre dos participantes.
- *Sequence Flow (Flujo de secuencia)*: indica el orden de secuencia en que las actividades del proceso deben ejecutarse.
- *Message Flow (Flujo de mensaje*: indica el flujo de mensajes entre dos participantes que están preparados para enviarlos y recibirlos.
- *Send Task (Tarea de envío)*: representa el envío de un mensaje a un participante externo (en relación al proceso). Una vez que el mensaje se ha enviado, la tarea se completó.
- *Receive Task (Tarea de recepción)*: representa la espera a que llegue un mensaje desde un participante externo (en relación al proceso). Una vez que el mensaje se ha recibido, la tarea se completó.
- *Service Task (Tarea de servicio)*: es una tarea que se ejecuta sin intervención humana, ya sea por una aplicación o por un servicio Web.
- *Script Task (Tarea de script)*: es una tarea que se ejecuta mediante un motor de procesos de negocio. Esta tarea permite el ingreso de un script en un lenguaje que el motor de procesos de negocio pueda ejecutar.
- *User Task (Tarea de usuario)*: es una tarea típica de "workflow", donde un intérprete humano realiza dicha tarea con la ayuda de una aplicación de software, generalmente programada mediante un gestor de listas de tareas ("task list manager").
- *Manual Task (Tarea manual)*: es una tarea que se lleva a cabo sin la ayuda de ningún motor de ejecución de procesos de negocio o cualquier otra aplicación.
- *Abstract Task (Tarea abstracta)*: es una tarea genérica o indefinida.
- *Call Activity (Actividad de llamada)*: es una referencia a un subproceso o tarea, definida de forma global, que se re-utiliza en el proceso actual. La

activación de esta actividad resulta en la transferencia de control al proceso o tarea invocado.

- *None Start Event (Evento de inicio)*: es un evento de inicio que no tiene definido su disparador. El evento de inicio comienza el flujo del proceso y por consiguiente no tiene flujo de secuencia de entrada.

- *None End Event (Evento de fin)*: es un evento de fin que no tiene definido el resultado. Un evento de fin indica dónde el proceso finaliza y por consiguiente no tiene flujo de secuencia de salida.

- *Timer Intermediate Event (Evento intermedio de tipo temporizador)*: actúa como un mecanismo de retardo sobre la base de una determinada fecha-hora o un ciclo específico que puede establecerse para disparar el evento.

- *Escalation Intermediate Event (Evento intermedio de tipo escalamiento)*: representan la activación de flujos excepcionales que ocurren en el proceso.

- *Conditional Intermediate Event (Evento intermedio de tipo condicional)*: es un evento que se dispara cuando una condición se hace verdadera.

- *Exclusive Gateway (Gateway exclusivo)*: puede ser de tipo divergente o convergente. Un "gateway" exclusivo divergente crea dos o más caminos alternativos dentro de un flujo de proceso. Para una instancia determinada del proceso, sólo se puede tomar un camino. La selección del camino está basada en datos que son evaluados en expresiones condicionales asociadas a cada camino. Un "gateway" exclusivo convergente fusiona caminos alternativos.

- *Event-Based Gateway (Gateway basado en eventos)*: representa un punto de decisión mutuamente excluyente en el proceso, pero dicha decisión no depende de la evaluación de expresiones condicionales basadas en datos del proceso sino de eventos que pueden ocurrir.

- *Parallel Gateway (Gateway paralelo)*: puede ser de tipo divergente o convergente. Un "gateway" paralelo divergente crea dos o más flujos paralelos. Un "gateway" paralelo convergente sincroniza flujos paralelos.

- *Inclusive Gateway (Gateway inclusivo)*: puede ser de tipo divergente o convergente. Un "gateway" inclusivo divergente crea dos o más caminos alternativos dentro de un flujo de proceso. Para una instancia determinada del proceso, se puede tomar uno o más caminos. Un "gateway" inclusivo convergente sincroniza uno o todos los flujos de secuencia entrantes al "gateway".

- *Complex Gateway (Gateway complejo)*: puede ser de tipo divergente o convergente. El "gateway" complejo se utiliza para manejar situaciones donde los otros tipos de "gateways" no proveen soporte para el comportamiento deseado.

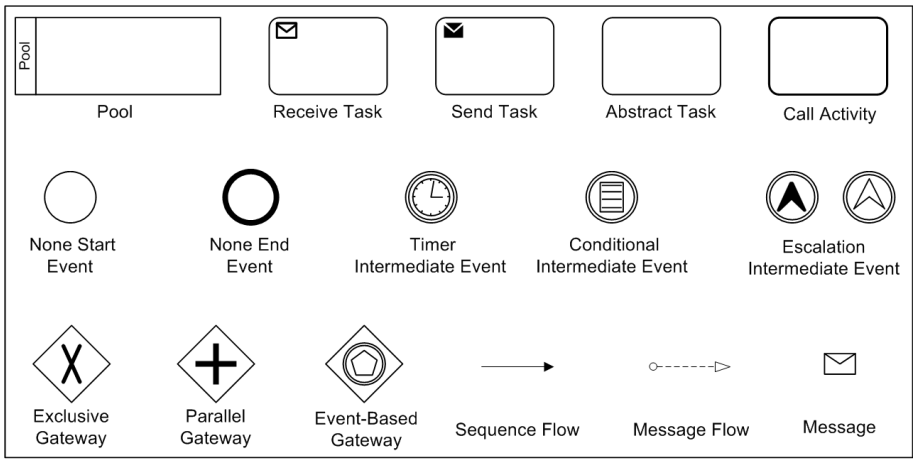


Figura 2.3: Resumen de la sintaxis concreta del lenguaje BPMN.

La sintaxis abstracta del lenguaje BPMN puede encontrarse en el Apéndice A.2.

2.1.2.2. Patrones de procesos de negocio

Existe un amplio consenso que los patrones pueden acelerar el proceso de diseño de una solución y reducir el tiempo de modelado, mientras que al mismo tiempo, permiten a una organización adoptar más fácilmente las mejores prácticas (Gschwind y otros, 2008). Un *patrón* es una solución generalizada que puede ser implementada y aplicada en un contexto dado para resolver un problema recurrente. El uso de patrones permiten a los participantes de una comunidad comunicarse más efectivamente, con mayor precisión y menor ambigüedad.

Los patrones de procesos son patrones funcionales y de comportamiento que intentan mejorar la calidad de los modelos de procesos (Eriksson y Penker, 2000). El uso de patrones de procesos facilita el diseño de modelos de procesos, reduce el tiempo y costo de modelado y fomenta el re-uso de (fragmentos de) modelos de procesos (Gschwind y otros, 2008). Además, se considera un medio efectivo para reducir la brecha negocio-TI.

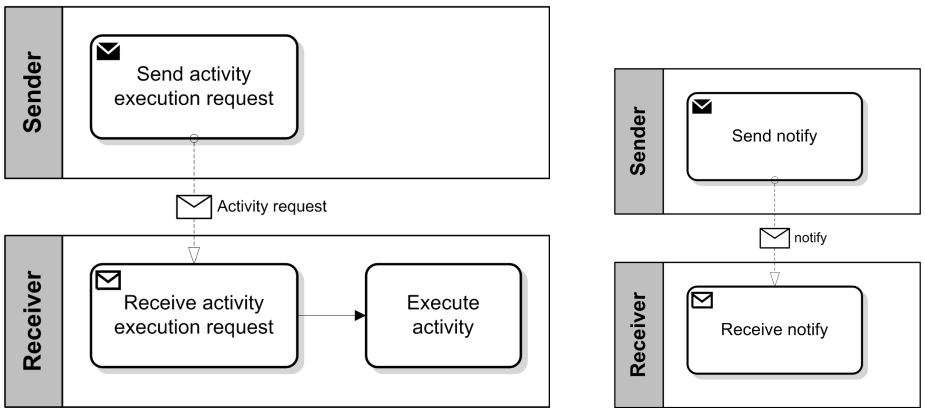
Existen numerosos patrones de procesos que pueden usarse en el modelado de procesos de negocio, tales como Workflow Patterns (Aalst y otros, 2003a), Workflow Data Patterns (Russell y otros, 2005), Service Interaction Patterns (Barros y otros, 2005), Workflow Activity Patterns (Thom y otros, 2009), entre otros. En la tesis, se hace uso de *Workflow Activity Patterns (Patrones de Actividades de Workflow)* (Thom y otros, 2009) en el método descrito en el Capítulo 5, para definir las actividades que soportan el intercambio de mensajes de negocio inter-organizacional y garantizar la interoperabilidad entre los procesos de integración.

Un *patrón de actividades de workflow* se refiere a la descripción de una función de negocio recurrente (por ejemplo, una notificación) que se encuentran frecuentemente en procesos de negocio (Thom y otros, 2009). Basado en un estudio intensivo sobre los tipos de procesos de negocio, se identificaron siete patrones de actividades de workflow, los cuales son (Tabla 2.2): *Approval*, *Question-answer*, *Uni- / Bi-directional Performative*, *Information Request*, *Notification* y *Decision Making*. Estos patrones son cercanos al vocabulario y nivel de abstracción en que los procesos de negocio son usualmente descritos por expertos del dominio. Esto favorece el re-uso de patrones cuando se modelan los procesos de negocio y por consiguiente contribuye a obtener modelos de procesos de negocio más estándares. Generalmente, un modelo de proceso de negocio se compone de múltiples patrones de actividades de workflow combinados mediante patrones de workflow de flujo de control (Aalst y otros, 2003a), tales como *Sequence*, *Parallel Split*, *Synchronization* o *Exclusive Choice*.

A continuación se describen dos ejemplos de patrones de actividades de workflow (Figura 2.4) usados por el método propuesto en el Capítulo 5, los cuales están ilustrados con el lenguaje BPMN. El conjunto completo de patrones de actividades de workflow puede encontrarse en (Thom y otros, 2009, 2011). Para cada patrón se provee el nombre, una descripción, un ejemplo ilustrativo, una descripción del problema que resuelve, cuestiones específicas, opciones de

Tabla 2.2: Patrones de Actividades de Workflow

WAP1 Approval: un objeto tiene que ser aprobado por uno o más roles organizacionales.
WAP2 Question-answer: permite formular una pregunta en el contexto de un proceso, identificar el rol organizacional que es capaz de responderla, enviar la pregunta a dicho rol y esperar por la respuesta respectiva.
WAP3 Unidirectional Performative: un emisor solicita la ejecución de una tarea particular de otro rol involucrado en el proceso. El emisor continúa la ejecución del proceso inmediatamente después de enviar la solicitud.
WAP4 Bi-directional Performative: un emisor solicita la ejecución de una tarea particular de otro rol involucrado en el proceso. El emisor espera hasta que dicho rol notifique que la tarea requerida ha sido ejecutada.
WAP5 Notification: el estado o resultado de una actividad ejecutada se comunica a uno o más participantes del proceso.
WAP6 Information Request: el emisor solicita cierta información a un participante del proceso. El emisor continúa la ejecución del proceso después de haber recibido la información.
WAP7 Decision Making: permite incluir una actividad de decisión en el flujo del proceso con conexiones a diferentes branches. Se ejecutarán sólo aquellos branches cuya condición sea verdadera.



(a) Unidirectional Performative (Single-Request variant).

(b) Notification (Single-Notification variant).

Figura 2.4: Ejemplos de patrones de actividades de workflow.

diseño (determinando diferentes variantes del patrón), una referencia a patrones relacionados y comentarios sobre la implementación del patrón.

Patrón de actividad Unidirectional Performative

Este patrón indica que un emisor requiere la ejecución de una actividad particular de un receptor involucrado en un proceso de negocio. El emisor continua la ejecución de su proceso inmediatamente después de enviar la solicitud.

- **Ejemplo ilustrativo**. En un proceso de contratación ("procurement"), un manager puede requerir la ejecución de una actividad para cancelar parcialmente una orden si ocurren algunas irregularidades. El flujo continúa inmediatamente después que se requiere la actividad de cancelación.

- **Descripción del problema**. En el curso de un proceso una solicitud de ejecución de una actividad puede incluirse como un paso en el proceso; el emisor de la solicitud debe continuar la ejecución sin esperar por una respuesta.

- **Cuestiones específicas**. Estas cuestiones deben considerarse al aplicar el patrón:

  - No se requiere una respuesta del receptor.
  - El proceso del emisor continúa su ejecución sin esperar la finalización de la actividad requerida.
  - La actividad requerida puede ser realizada por un humano o un software.

- **Opciones de diseño**. La elección dependerá de si la solicitud de ejecución de la actividad se enviará a uno o varios participantes.

  - *Single-Request* (Figura 2.4(a)): un emisor envía una solicitud de ejecución de una actividad a un receptor y continúa la ejecución de su proceso sin esperar una respuesta.
  - *Multi-Request*: un emisor envía simultáneamente una solicitud de ejecución a múltiples receptores y después continúa la ejecución sin esperar por ninguna respuesta.

- **Patrones relacionados**. *Bi-directional Performative* (WAP4), *Send* y *One-to-Many Send* (Barros y otros, 2005).

- **Implementación**. Este patrón puede implementarse basado en el patrón *Send* o basado en el patrón *One-to-Many* (Barros y otros, 2005).

Patrón de actividad Notification

El estado o resultado de una actividad se comunica a uno o más participantes del proceso.

- **Ejemplo ilustrativo**. Cuando se planifica una reunión en el contexto de un proceso de ingeniería, una notificación debe enviarse a los ingenieros informándoles sobre los detalles de la reunión (por ejemplo, lugar, fecha, hora de reunión, asunto).
- **Descripción del problema**. Durante la ejecución del proceso los participantes tienen que ser informados sobre el estado (por ejemplo, completado, en ejecución, esperando) o el resultado (por ejemplo, documento aceptado, rechazado) de la ejecución de una actividad.
- **Cuestiones específicas**. Estas cuestiones deben considerarse al aplicar el patrón:
  - La notificación debe enviarse electrónicamente a uno o más participantes del proceso.
  - El proceso no tiene que esperar alguna respuesta de los participantes que reciben la notificación.
  - La notificación informa sobre el estado o resultado de una actividad del proceso a ser monitoreado.
- **Opciones de diseño**. La elección de diseño depende de si la notificación es para enviarse a uno o varios participantes.
  - *Single-Notification* (Figura 2.4(b)): un emisor envía una notificación a un único receptor.
  - *Multi-Notification*: un emisor envía simultáneamente una notificación a múltiples receptores.
- **Patrones relacionados**. *One-Way Send* y *One-to-Many Send* (Barros y otros, 2005).

- **Implementación**. Este patrón es soportado por varios sistemas de gestión de workflow. Puede implementarse basado en el patrón *One-Way-Send* o *One-to-Many Send*.

2.2. Desarrollo dirigido por modelos

El Desarrollo Dirigido por Modelos - Model-Driven Development (MDD) es un nuevo paradigma para el desarrollo de software, en el cual los modelos son los principales artefactos en el proceso de desarrollo. La principal ventaja de este paradigma es que los modelos se expresan usando conceptos mucho más cercanos al dominio del problema, elevando el nivel de abstracción de los mismos, en lugar de usar conceptos ligados a la tecnología de implementación. Esto hace que los modelos sean más fáciles de especificar, entender y mantener; y sean menos sensibles a la tecnología de implementación elegida y a los cambios en dicha tecnología (Selic, 2003).

Otra característica importante de MDD es que el código es generado automáticamente (mediante transformaciones de modelos) a partir de sus correspondientes modelos (Selic, 2003). De esta manera, los modelos no sólo son utilizados para documentar el sistema sino también para construir el producto final. Esto es diferente al enfoque tradicional de desarrollo de software en donde los modelos eran meramente utilizados para propósitos de documentación.

Los conceptos claves de MDD son los modelos, meta-modelos y transformaciones de modelos.

Un *modelo* es una descripción de un sistema escrita en un lenguaje bien definido (Anneke G. Kleppe, 2003). Para que un modelo sea útil y eficaz, debe poseer las siguientes características (Selic, 2003, 2006):

- *Abstracto*: debe ocultar o remover los detalles irrelevantes con el fin de destacar los esenciales.

- *Comprensible*: debe ser expresado de forma que transmita la información esencial directamente y con precisión.

- *Exacto*: debe reflejar correctamente las propiedades de interés del sistema modelado.

- *Predictivo*: debe ser capaz de predecir con exactitud el comportamiento y otras propiedades del sistema modelado.
- *Económico*: debe ser significativamente más barato de construir y analizar que el sistema mismo.

Un *meta-modelo* es un tipo especial de modelo que especifica la sintaxis abstracta de un lenguaje de modelado (Anneke G. Kleppe, 2003; OMG, 2011b). Debido a que un meta-modelo es también un modelo, debe estar escrito en un lenguaje bien definido, denominado *meta-lenguaje*. La relación entre un modelo expresado en un lenguaje y el meta-modelo de dicho lenguaje se denomina *conformeA* ("conformsTo") (Jouault y otros, 2008).

Una *transformación de modelo* implica ingresar un modelo de entrada a un proceso de transformación y generar como salida otro modelo, o diferentes niveles de código ejecutable (Brown y otros, 2005). Debido a que definir y aplicar transformaciones de modelos es clave en MDD, los tipos de transformaciones se describen en detalle en la Sección 2.2.2.

En el ámbito de MDD, una de las propuestas más conocida y utilizada es MDA, desarrollada por la OMG (OMG, 2003b), la cual se describe en la siguiente sección.

2.2.1. Arquitectura dirigida por modelos

La Arquitectura Dirigida por Modelos - Model-Driven Architecture (MDA) se basa en un conjunto de estándares acerca de cómo definir un conjunto de modelos, notaciones y reglas de transformación (Brown y otros, 2005). Provee una infraestructura abierta, neutral e independiente de enfoques propietarios, garantizando la interoperabilidad de los sistemas a través de los estándares de modelado establecidos por la OMG: *Unified Modeling Language (UML)* (OMG, 2011d), *Meta-Object Facility (MOF)* (OMG, 2011b) y *Common Warehouse Metamodel (CWM)* (OMG, 2003a). Usando estos estándares, se pueden desarrollar soluciones organizacionales basadas en descripciones independientes de la plataforma y transformarlas a una plataforma abierta o propietaria, tales como CORBA, J2EE, .NET, XMI/XML y plataformas basadas en servicios Web.

La idea principal de MDA es la separación de la especificación del sistema de los aspectos puntuales de la implementación del mismo con el objetivo de

alcanzar portabilidad, interoperabilidad, calidad, re-usabilidad, productividad y mantenibilidad. Para este fin, MDA provee un marco conceptual y un vocabulario a usarse durante el proceso de desarrollo, en el cual se definen tres tipos de modelos (Anneke G. Kleppe, 2003; OMG, 2003b; Pons y otros, 2010):

- *Modelo Independiente de la Computación - Computation-Independent Model (CIM)*: es una vista del sistema independiente de la computación que describe los requerimientos del mismo y el contexto de negocio en el cual será utilizado, sin mostrar detalles de la estructura del sistema. Utiliza un vocabulario que es familiar para los especialistas del dominio y también es conocido como modelo de dominio.

- *Modelo Independiente de la Plataforma - Platform-Independent Model (PIM)*: es una vista del sistema independiente de la plataforma tecnológica. Es un modelo que representa la lógica del negocio y su funcionalidad, independientemente de los detalles de la implementación. En este modelo se puede observar los aspectos que no cambiarán de una plataforma a otra, con el fin de permitir su mapeo a una o más plataformas tecnológicas.

- *Modelo Específico de la Plataforma - Platform-Specific Model (PSM)*: es una vista del sistema desde la perspectiva de la plataforma tecnológica específica. Combina las especificaciones del PIM con los detalles y características propias del uso de dicha plataforma. En este modelo se puede observar la manera en la cual un sistema usa la plataforma para el cumplimiento de los objetivos trazados en el CIM.

El proceso de desarrollo aplicando el marco conceptual propuesto por MDA indica que se debe: (a) especificar un sistema con independencia de la plataforma que lo soporta, es decir, definir el PIM; (b) seleccionar una o varias plataformas para el sistema; (c) transformar el PIM hacia uno o más PSMs en las plataformas seleccionadas; y (d) generar el código, es decir, la definición de la transformación de un PSM a código, para la generación automática del mismo.

2.2.2. Transformaciones de modelos

Las transformaciones de modelos desempeñan un papel importante en MDD. Mediante una serie de transformaciones, se reduce el nivel de abstracción

de los modelos con el propósito de producir un modelo con suficientes detalles que permita la generación automática de código ejecutable (Jouault y otros, 2008).

Las transformaciones de modelos más comunes son (Brown y otros, 2005):

- *Transformación de tipo refactorización*: reorganiza un modelo basado en un criterio bien definido. La salida de esta transformación es una revisión del modelo original, denominado modelo refactorizado.
- *Transformación modelo-a-modelo*: convierte información de un modelo o modelos a otro modelo o conjunto de modelos.
- *Transformación modelo-a-código*: convierte un modelo a código.

Una transformación de modelos puede realizarse en forma *vertical*, en el cual la transformación se realiza en diferentes niveles de abstracción (por ejemplo, PIM a PSM), u *horizontal*, en el cual la transformación se realiza en el mismo nivel de abstracción (por ejemplo, PIM a PIM).

En MDD las transformaciones de modelos siguen un patrón común conocido como *patrón de transformación de modelo* (Jouault y otros, 2008) (Figura 2.5). El patrón de transformación consta de una definición de transformación (DTab) ejecutado por una herramienta de transformación (HT) para generar el modelo de salida (Mb) a partir del modelo de entrada (Ma). Estas tres entidades (DTab, Ma y Mb) son conformes a los meta-modelos MMt, MMa y MMb, respectivamente. MMt corresponde a la sintaxis abstracta del lenguaje de transformación usado. Estos tres meta-modelos son conformes a un meta-meta-modelo MMM (por ejemplo, MOF o Ecore). Los modelos, meta-modelos y meta-meta-modelos corresponden a los niveles M1, M2 y M3 de la arquitectura de MOF.

Una *definición de transformación* está compuesta por un conjunto de reglas de transformación que describen cómo un modelo de entrada (definido con un lenguaje origen) se transforma en un modelo de salida (definido con un lenguaje destino). Una *regla de transformación* es una descripción de cómo uno o más constructores del lenguaje origen (patrón origen) se transforma en uno o más constructores del lenguaje destino (patrón destino). La *herramienta de transformación* ejecuta la definición de transformación para generar automáticamente el modelo de salida a partir del modelo de entrada (Anneke G. Kleppe, 2003).

Existen varios lenguajes de transformación que permiten definir la definición de transformación, tales como QVT (OMG, 2011c), ATL (Jouault y otros, 2008),

MOFM2T (OMG, 2008). En el presente trabajo de tesis se utilizó el lenguaje ATL para las transformaciones modelo-a-modelo requeridos por los métodos propuestos en los Capítulos 4 y 5. Se seleccionó ATL para definir las transformaciones debido a que es un lenguaje ampliamente utilizado en el desarrollo dirigido por modelos, tanto en la academia como en la industria, y provee herramientas de soporte maduras. Dicho lenguaje se describe en la siguiente sub-sección.

2.2.2.1. Lenguaje ATL

ATL es un lenguaje de transformación de modelos desarrollado sobre la plataforma Eclipse, que proporciona un conjunto de herramientas que facilitan el desarrollo de definiciones de transformaciones, denominado ATL Development Tools (ADT) (Jouault y otros, 2008).

ATL es un lenguaje híbrido, es decir, contiene constructores declarativos e imperativos, basado en OCL (OMG, 2012) para definir sus tipos de datos y expresiones declarativas (Jouault y otros, 2008; Jouault y Kurtev, 2006).

Las transformaciones en ATL son unidireccionales, es decir, operan en modelos de entrada de sólo lectura y producen modelos de salida de sólo escritura. Dichos modelos pueden serializarse en el formato XMI (OMG, 2011e).

ATL soporta tres tipos de unidades (Wagelaar y otros, 2010): módulos, librerías y consultas. Los *módulos* representan las definiciones de transformación y son los únicos que pueden generar un modelo de salida. Las *librerías* contienen "helpers" que pueden ser usados en otros módulos. Las *consultas* ("query")

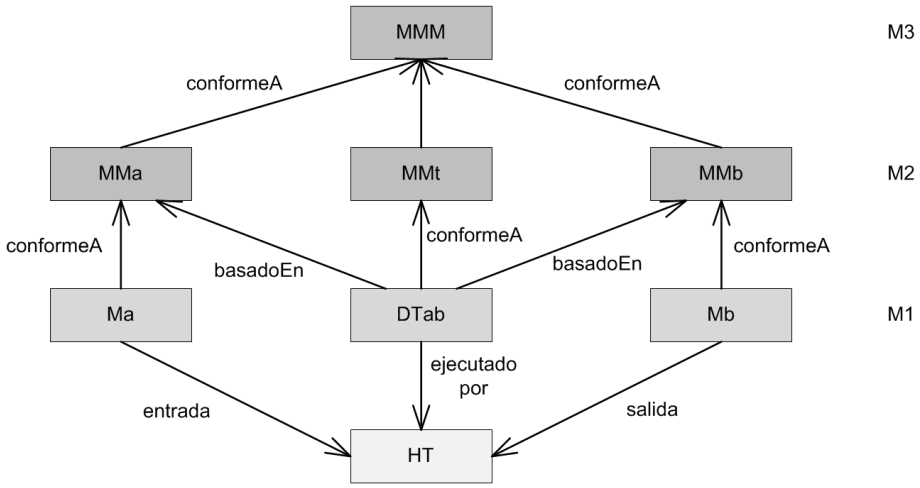


Figura 2.5: Patrón de transformación de modelo.

aceptan uno o más modelos de origen y retornan un único valor de algún tipo de dato primitivo soportado.

Un *módulo* ("module") contiene una sección *encabezado* ("header"), una sección *importación* ("import") y varios *métodos* ("helpers") y *reglas* ("rules").

La sección *encabezado* indica el nombre del módulo y declara los modelos de entrada y salida y sus meta-modelos respectivos. El Listado 2.1 muestra un ejemplo de un encabezado, la cual comienza con la palabra clave `module` seguido por el nombre del módulo, en este ejemplo, *Author2Person* (línea 1).

Listado 2.1: Ejemplo de un encabezado de un módulo ATL

```
1 module Author2Person;
2 create OUT : MMPerson from IN : MMAuthor;
```

Luego, como se muestra en la línea 2, los modelos de entrada y salida se declaran como variables tipadas por sus meta-modelos. La palabra clave `create` indica los modelos de salida y la palabra clave `from` indica los modelos de entrada. En el ejemplo, el modelo de salida ligado a la variable `OUT` es creado a partir del modelo de entrada ligado a la variable `IN`. Los modelos de entrada y salida son conformes a los meta-modelos *MMAuthor* y *MMPerson*, respectivamente.

La sección *importación* declara las librerías que serán importadas. En el Listado 2.2 se muestra un ejemplo que permite importar la librería *strings*. La palabra clave `uses` indica la librería a importar.

Listado 2.2: Ejemplo de una sección importación de un módulo ATL

```
1 uses strings;
```

Los *helpers* pueden verse como un equivalente a métodos. Permiten definir código factorizado que se puede llamar desde diferentes puntos de un módulo. Existen dos tipos de "helpers": *Operation* y *Attribute*. Un *helper Operation* define operaciones en el contexto de un elemento del modelo o en el contexto de un módulo, puede contener parámetros de entrada y usar recursión. Un *helper Attribute* se usa para asociar atributos a elementos del modelo de entrada. Estos tipos de "helpers" no aceptan parámetros.

En el Listado 2.3 se muestra un ejemplo de un "helper". Como se ve en la línea 1, el "helper" *firstToLower* se define en el contexto del tipo `String` (indicado por la palabra clave `context`) y los valores que devuelve son de tipo

`String`. Después del símbolo "=" se define la expresión OCL usada para calcular el valor del "helper" (línea 2). Cuando el "helper" se ejecuta sobre un determinado "string", devuelve el mismo "string" con la primera letra en minúscula.

Listado 2.3: Ejemplo de un helper de un módulo ATL

```
1 helper context String def: firstToLower() : String =
2 self.substring(1, 1).toLower() + self.substring(2, self.size());
```

Una *regla de transformación* es el constructor básico de ATL usado para expresar la lógica de transformación y puede especificarse en forma declarativa o imperativa.

Las reglas declarativas se denominan *matched*. Una regla "matched" se compone de un patrón origen y un patrón destino. El *patrón origen* indica qué elemento del modelo de entrada se va a mapear. También puede tener un *guard* (expresión booleana en OCL) que permite definir filtros sobre el patrón. El *patrón destino* se compone de los tipos de elementos que se van a generar en el modelo de salida, y un conjunto de *bindings*, que indican cómo se van a inicializar dichos elementos en el modelo de salida.

Las reglas "matched" se ejecutan por cada coincidencia de sus patrones origen encontrada en el modelo de entrada, creando en el modelo de salida los elementos especificados en el patrón destino.

Existen varios tipos de reglas "matched", que difieren en la forma en que se disparan (Jouault y otros, 2008; Jouault y Kurtev, 2006):

- *Reglas standard*: se disparan automáticamente por cada coincidencia del patrón origen encontrada en los modelos de entrada.
- *Reglas lazy*: a diferencia de las reglas "standard", estas reglas sólo se disparan cuando son invocadas por otras reglas, es decir, deben ser explícitamente invocadas. Las reglas "lazy" producen un nuevo patrón destino cada vez que se disparan.
- *Reglas unique lazy*: a diferencia de las reglas "lazy", estas reglas cada vez que se disparan producen el mismo patrón destino para un mismo patrón origen.

En el Listado 2.4 se muestra un ejemplo de una regla "matched" denominada *Author*. La regla transforma elementos de la clase `Author` del modelo

de entrada a elementos de la clase `Person` en el modelo de salida. El patrón origen (definida por la palabra clave `from`) tiene un elemento denominado *author* de tipo `Author`, pero no tiene definido ningún "guard", por lo cual todas las instancias de la clase `Author` del modelo de entrada serán mapeadas por la regla.

Listado 2.4: Ejemplo de una regla *matched*

```
1 rule Author {
2   from
3     author : MMAuthor!Author
4   to
5     person : MMPerson!Person (
6       name <- author.name,
7       surname <- author.surname
8     )
9 }
```

El patrón destino (definido por la palabra clave `to`) contiene un único elemento denominado *person* de tipo `Person` cuyo objetivo es crear una instancia de la clase `Person` en el modelo de salida por cada elemento del modelo de entrada coincidente con el patrón origen. A los atributos `name` y `surname` de la clase `Person` se le asignan los valores de los atributos `name` y `surname` de la clase `Author`.

La parte imperativa de ATL se basa en dos constructores (Jouault y otros, 2008; Jouault y Kurtev, 2006):

- *Reglas called*: es básicamente un procedimiento; se invoca explícitamente desde un bloque imperativo para generar un patrón destino. Este tipo de regla puede contener argumentos.

- *Bloque action*: es una secuencia de sentencias imperativas que puede utilizarse en lugar de o en combinación con un patrón destino en reglas "matched" o "called".

ATL permite aplicar *herencia de reglas* como un mecanismo de re-uso de código y también como un mecanismo para especificar reglas polimórficas (Jouault y otros, 2008; Jouault y Kurtev, 2006). La herencia de reglas permite definir reglas de transformación generales (super-reglas) que pueden ser extendidas por reglas específicas (sub-reglas). La sub-regla mapea un subconjunto de los elementos que

su super-regla mapea. En las sub-reglas se pueden reemplazar los elementos del patrón origen de la super-regla. El "guard" de la sub-regla actúa en conjunto con el "guard" de la super-regla. La principal ventaja de la herencia de reglas es que se evita la duplicación de código y, por consiguiente, los problemas de mantenimiento.

ATL provee también otro mecanismo de re-uso denominado *superimposición de módulos* (Wagelaar y otros, 2010), que permite la composición de dos o más módulos y ejecutarlos como si fuera un único módulo (Figura 2.6).

La superimposición consiste en dividir un módulo en varios módulos de tamaño y alcance manejable (técnica conocida como *factorización* (Sánchez Cuadrado y García Molina, 2008)), que luego se pueden usar para la composición. La factorización es el proceso de encontrar una funcionalidad común compartida entre dos o más módulos, y de la extracción de las partes comunes en un módulo base (Sánchez Cuadrado y García Molina, 2008). De esta manera, las reglas contenidas en cada módulo son altamente re-usables y tienen un alto grado de cohesión.

El módulo compuesto (*UML2Profiles* en la Figura 2.6) contiene la unión de todas las reglas de transformación. Además de agregar más reglas, también se pueden sobreescribir las reglas del módulo superpuesto (*UML2Copy* en la Figura 2.6), en cuyo caso las reglas originales no se ejecutan. De esta manera, se permite la adaptación de un módulo a nivel de regla y mejorar la capacidad de reutilización de los módulos.

La superimposición además mejora la escalabilidad y mantenibilidad de los módulos ya que cada vez que se realiza un cambio a un módulo, sólo dicho módulo debe ser recompilado.

Otro mecanismo para mejorar la calidad de los módulos es la *refactorización* (Wimmer y otros, 2012), la cual es una técnica, similar a la aplicada en la programación orientada a objetos, para mejorar la estructura de un módulo existente preservando su comportamiento externo. La preservación de comportamiento se logra si para un modelo de entrada, el modelo de salida producido es el mismo antes y después de refactorizar el módulo.

La refactorización se estructura en cuatro categorías (Wimmer y otros, 2012):

- *Renaming*: se realizan para renombrar identificadores así como sus refe-

rencias dentro de las transformaciones. El objetivo es asignarle nombres apropiados a las reglas y "helpers" para que provean una idea precisa de su funcionalidad.

- *Restructuring*: se realizan para mejorar la estructura de una transformación mediante su re-estructuración y la introducción de reglas y "helpers". El objetivo es transformar reglas grandes en varias de menor tamaño ya sea dividiendo una regla "matched" en varias o delegando funcionalidad a reglas "lazy" adicionales.

- *Inheritance-related*: se realizan para extraer o eliminar las partes comunes entre reglas introduciendo o removiendo herencia entre ellas.

- *OCL Expression Optimization*: se realizan para optimizar las expresiones OCL.

En resumen, aplicando herencia de reglas, superimposición de módulos y refactorización, es posible lograr la escalabilidad, mantenibilidad y reusabilidad de módulos. Dichas propiedades son claves para lograr una solución basada en modelos de alta calidad (Sánchez Cuadrado y García Molina, 2008; Wimmer y otros, 2012). En los métodos propuestos en los Capítulos 4 y 5 se hacen uso de estas propiedades de las transformaciones de modelos.

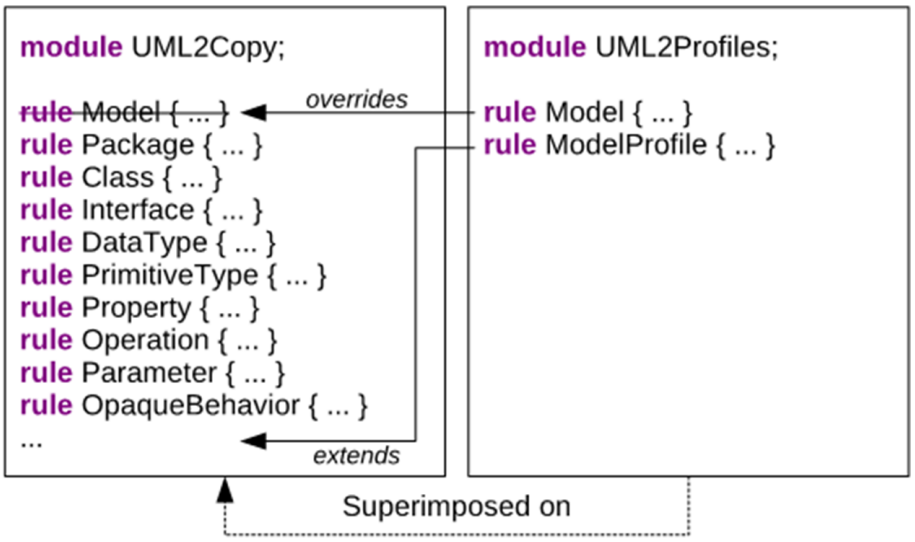


Figura 2.6: Ejemplo de superimposición de módulos en ATL.

2.3. Arquitectura orientada a servicios

Como se mencionó anteriormente, la capacidad para responder rápidamente a los cambios de mercado y optimizar los procesos de negocio es un factor clave para la competitividad y el crecimiento de las organizaciones. La agilidad y optimización está muy ligada a la capacidad con que la TI de las organizaciones puede responder de forma flexible a los cambios que afectan a los procesos de negocio. Un paradigma que permite abordar estas cuestiones críticas es la Arquitectura Orientada a Servicios - Service-Oriented Architecture (SOA) (Erl, 2007; Papazoglou y Heuvel, 2007).

Básicamente, SOA es un enfoque que permite la alineación negocio-TI en el que las aplicaciones dependen de los servicios disponibles para facilitar los procesos de negocio (Zhang y otros, 2007). Un *servicio* es un módulo bien definido, autónomo que provee una funcionalidad de negocio estándar y es independiente del estado o contexto de otros servicios (Papazoglou y Heuvel, 2007). Los servicios se describen con un lenguaje de definición estándar, tienen una interfaz publicada y se comunican entre sí invocando la ejecución de sus operaciones con el fin de soportar colectivamente una tarea o proceso de negocio común.

SOA es un modelo arquitectónico a nivel conceptual, lo que significa que debe ser implementado y realizado por una tecnología TI. La tecnología *servicios Web* es la más utilizada para implementar SOA (Zhang y otros, 2007). Los *servicios Web* (W3C, 2004) se basan en un conjunto de estándares de comunicación, como son *Extensible Markup Language (XML)* (W3C, 2008) para la representación de datos, *Simple Object Access Protocol (SOAP)* (W3C, 2007a) para el intercambio de datos, *Web Services Description Language (WSDL)* (W3C, 2007b) para describir las funcionalidades de un servicio Web, y *Universal Description, Discovery and Integration (UDDI)* (OASIS, 2004) para registrar los servicios disponibles.

Los servicios en una solución SOA deben cumplir con los siguientes principios (Erl, 2005):

- *Reutilización*, la cual asegura que cada funcionalidad del servicio se llevará a cabo sólo una vez y se puede usar repetidamente.

- *Contrato formal* (descripción de la interfaz de un servicio), el cual define

cómo comunicarse con un servicio particular.

- *Bajo acoplamiento* entre servicios, para minimizar las dependencias entre los mismos. Esto conduce a una mayor flexibilidad de la arquitectura de servicios.

- *Abstracción*, los servicios actúan como cajas negras, es decir, ocultan sus detalles al mundo exterior y sólo se puede acceder a través de sus interfaces.

- *Composición*, la cual se refiere al hecho de que los servicios pueden ser utilizados por otros servicios compuestos. Puede entenderse como otra forma de reutilización porque un servicio particular puede actuar en varias composiciones.

- *Autonomía*, la cual se refiere a que los servicios tienen el control sobre la lógica que encapsulan. Esto es compatible con otros principios, como el bajo acoplamiento.

- *Sin estado*. Un servicio simple no debe tener estado, es decir, no debe guardar ningún tipo de información luego de la invocación de sus operaciones por otros servicios o aplicaciones cliente. Esto es así porque si un servicio almacena algún tipo de información, se pueden producir problemas de inconsistencia de datos. La solución es que un servicio sólo contenga lógica.

Los servicios pueden desempeñar tres roles (Figura 2.7): *service provider*, *service requestor* y *service registry*, que interactúan realizando las operaciones: *publish/unpublish/update*, *discover/find* y *invoke/bind* (Erl, 2007; Papazoglou, 2003; Zhang y otros, 2007).

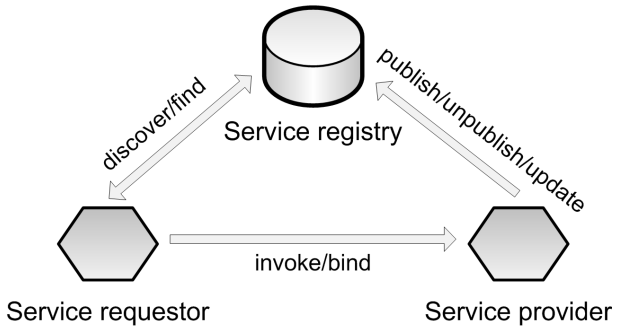


Figura 2.7: Roles de la arquitectura SOA.

- *Service provider*: provee funcionalidades de aplicaciones y negocio como servicios. Es responsable de publicar ("publish"), anular la publicación ("unpublish") y actualizar ("update") sus servicios para que estén disponibles en Internet. Desde una perspectiva de negocio, es el propietario del servicio. Desde un punto de vista arquitectónico, es el componente que implementa el servicio.

- *Service requestor*: tiene una necesidad que puede ser cumplida por un servicio disponible en Internet. Desde una perspectiva de negocio, es quien requiere una cierta función de negocio que cumplir. Desde el punto de vista arquitectónico, es la aplicación que está buscando e invocando un servicio.

- *Service registry*: provee un repositorio de descripciones de servicios para que los *service provider* publiquen sus servicios y los *service requestor* encuentren los servicios y obtengan información de enlace de estos servicios.

2.4. Trabajos relacionados

En esta sección se discuten las propuestas más relevantes que tienen relación con el enfoque propuesto en el presente trabajo de tesis. Primero se discuten las propuestas relacionadas al diseño de modelos de procesos de negocio y luego las relacionadas a la gestión de modelos de procesos de negocio en ambientes inter-organizacionales.

2.4.1. Trabajos relacionados al diseño de modelos de procesos de negocio

Existen varios métodos y metodologías, tales como (Bauer y otros, 2005; Huemer y otros, 2008; Roser y otros, 2006; Villarreal y otros, 2006b, 2007b), las cuales proveen herramientas y guías para definir modelos de procesos de negocio en el desarrollo e implementación de soluciones inter-organizacionales.

Bauer y otros (2005) proponen una arquitectura conceptual que sigue un enfoque "top-down" para el modelado de procesos colaborativos aplicando los principios de MDA. También proveen transformaciones de modelos de procesos colaborativos definidos en un nivel CIM a modelos de procesos de interfaz defi-

nidos en un nivel PIM. Dicha arquitectura usa un broker centralizado que actúa como un observador global que coordina a los participantes en la colaboración inter-organizacional.

Roser y otros (2006) proponen una metodología basada en MDA que sigue un enfoque "top-down" para el diseño e implementación de procesos colaborativos. La solución tecnológica generada por la metodología está basada en el paradigma SOA. A partir de modelos de procesos colaborativos se derivan los servicios que deben atender la ejecución de dichos procesos.

Villarreal y otros (2006b, 2007b) proponen métodos MDA para generar soluciones tecnológicas basadas en el estándar WS-BPEL de composición de servicios Web y el estándar ebXML de transacciones de negocio, respectivamente. La solución tecnológica se deriva directamente desde modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP.

Huemer y otros (2008) proponen la metodología UN/CEFACT Modeling Methodology (UMM) para el modelado de coreografías globales requeridas para la implementación de colaboraciones inter-organizacionales. Esta metodología aplica un enfoque "top-down" y se inicia en la definición de una coreografía global (procesos colaborativos) en una jerarquía de vistas.

Un aspecto a considerar de estas propuestas menciondas es que las mismas no identifican todos los artefactos o tipos de modelos de procesos de negocio requeridos para el desarrollo e implementación de colaboraciones inter-organizacionales. Sólo consideran el uso de modelos de procesos colaborativos para derivar la solución tecnológica, y dejan de lado el uso de modelos de procesos de negocio internos públicos (procesos de interfaz) y privados (procesos de integración), los cuales permiten expresar y analizar el comportamiento de cada organización en un alto nivel de abstracción, de manera independiente de la solución de sistemas y tecnológica.

Con respecto a la generación de procesos de interfaz, Hofreiter (2008) propuso un enfoque que permite derivar coreografías locales (procesos de interfaz), a partir de coreografías globales (procesos colaborativos) definidas con UMM y registradas en un repositorio global.

Villarreal y otros (2006a) propusieron un método basado en MDA para generar procesos abstractos (de interfaz) WS-BPEL a partir de procesos colaborativos UP-ColBPIP.

No obstante, la primera no usa un lenguaje estándar para el modelado de procesos de interfaz tal como BPMN, que facilita el entendimiento de los mismos y permite elevar el nivel de abstracción de los modelos. La segunda utiliza un lenguaje estándar, pero el mismo es de bajo nivel ya que es para una plataforma o tecnología específica como ser la de servicios Web.

Hofreiter (2009) propone un perfil UML que extiende UMM para representar coreografías locales (procesos de interfaz) en alineación con una coreografía global (proceso colaborativo) UMM.

Zaha y otros (2006) proponen un algoritmo para generar modelos de servicios locales a partir de un modelo de coreografía de servicios para proveer la vista local de cada organización que participa en la coreografía.

La propuesta de Zaha y otros (2006) se enfoca en modelos de procesos basados en interacciones de servicios en lugar de enfocarse en modelos de procesos independientes de la plataforma.

En resumen, el estado del arte actual es que las metodologías existentes para el desarrollo e implementación de colaboraciones inter-organizacionales no contemplan el diseño de todos los tipos de procesos de negocio necesarios. Si bien el modelado de procesos colaborativos ha sido sujeto de intensiva investigación, sin embargo, no ocurrió lo mismo con respecto al diseño de modelos de procesos de integración, los cuales son fundamentales para la ejecución de los procesos colaborativos.

Ninguna de las propuestas descritas anteriormente plantea o propone un método para el diseño de modelos de procesos integración, ya que las mismas no proveen mecanismos para poder identificar qué actividades privadas y qué comportamiento privado debe definirse para permitir la ejecución de los procesos colaborativos. Por lo tanto no se cuenta con métodos de diseño de procesos de integración que permitan derivar modelos de los mismos a partir de modelos de procesos colaborativos, siguiendo un enfoque “top-down”. Aunque existen métodos para derivar modelos de procesos de interfaz a partir de modelos de procesos colaborativos, muy pocos se enfocan en generar modelos de procesos de interfaz conceptuales, en un alto nivel de abstracción, expresados en un lenguaje que sea independiente de los paradigmas de sistemas y tecnologías de implementación.

2.4.2. Trabajos relacionados a la gestión de modelos de procesos de negocio

Debido a que las organizaciones que adoptan BPM deben expresar sus operaciones en términos de procesos de negocio, una organización puede lidiar con colecciones de cientos o incluso miles de modelos de procesos de negocio (Dijkman y otros, 2012; Yan y otros, 2012). Gestionar grandes colecciones de modelos de procesos de negocio es una tarea compleja, que presenta nuevos desafíos como gestionar varias versiones de modelos o encontrar un modelo particular. También presenta nuevas oportunidades como la extracción de conocimiento sobre las operaciones de la organización o el re-uso (considerado una buena práctica) de fragmentos de procesos para el diseño de nuevos procesos de negocio. Esto atrajo el interés de grupos de investigación los cuales se enfocan en la definición de diferentes técnicas para la gestión de grandes colecciones de modelos de procesos de negocio tales como consulta, búsqueda de similitud, gestión de variantes, fusión ("merging") de modelos de procesos de negocio, minería, factorización y re-uso (Dijkman y otros, 2012).

Como se mencionó en el capítulo anterior, la tecnología de repositorio proporciona una infraestructura adecuada para la gestión de colecciones de modelos de procesos de negocio (Dijkman y otros, 2012). Repositorios que proveen funcionalidades específicas para la gestión de modelos de procesos de negocio se denominan *repositorios de modelos de procesos de negocio* (Yan y otros, 2012). Dichos repositorios además proveen y explotan las funcionalidades comúnmente provistas por repositorios y bases de datos en general, tales como almacenamiento, recuperación, "check in/out", control de versiones y control de acceso.

2.4.2.1. Requerimientos para la gestión de modelos de procesos de negocio en colaboraciones inter-organizacionales

La gestión de colecciones de modelos de procesos de negocio es una tarea mucho más compleja en ambientes inter-organizacionales, particularmente cuando las organizaciones integran varias redes colaborativas y establecen diferentes colaboraciones inter-organizacionales en dichas redes. Esto se debe a que, como se mencionó en el capítulo anterior, el desarrollo e implementación de colaboraciones inter-organizacionales implica la definición de diferentes modelos de procesos de

negocio que difieren en el lenguaje de modelado, el nivel de abstracción, el nivel de detalle, la fase de modelado en que son usados, las personas destinatarias o la evolución (expresados en diferentes versiones) de los mismos. En un ambiente inter-organizacional, se requiere mantener la asociación y la sincronización entre los modelos de procesos colaborativos y los modelos de procesos de interfaz/integración, la consistencia entre los modelos de procesos de interfaz/integración y sus respectivos modelos de procesos colaborativos, y también garantizar la interoperabilidad de los modelos de procesos de integración para asegurar la ejecución de los procesos colaborativos, preservando los aspectos privados de cada organización.

De acuerdo a lo expresado anteriormente y basado en el análisis de varios casos de estudio, se definen los siguientes requerimientos esenciales para la gestión de modelos de procesos de negocio involucrados en colaboraciones inter-organizacionales (Lazarte y otros, 2013): interoperabilidad, consistencia, sincronización, privacidad, soporte en el diseño de los procesos de negocio e implementación distribuida, los cuales se describen a continuación:

- **Interoperabilidad.** Los modelos de procesos de integración de las organizaciones que participan de un proceso colaborativo deben ser interoperables para lograr la ejecución de dicho proceso. Para lograr la interoperabilidad, debe existir una correspondencia entre cada tarea de envío de mensaje de un modelo de proceso de interfaz/integración de una organización con la respectiva tarea de recepción de dicho mensaje en un modelo de proceso de interfaz/integración de otra organización. Además, los mensajes deben ser enviados en la secuencia que fue definida en el modelo de proceso colaborativo. En otras palabras, interoperabilidad implica que el intercambio de mensajes públicos entre los modelos de procesos de interfaz/integración debe estar sincronizado. Si no se logra la interoperabilidad, los procesos de interfaz/integración pueden llegar a puntos muertos ("deadlock") en tiempo de ejecución.

- **Consistencia.** El comportamiento definido en los modelos de procesos de interfaz/integración debe ser coherente con el comportamiento definido en el modelo de proceso colaborativo correspondiente. Debido a que un modelo de proceso colaborativo sirve como una base contractual para la colaboración

inter-organizacional, violaciones a dicho contrato podría tener consecuencias hasta legales (Decker y Weske, 2007), como así también la definición de modelos de procesos de interfaz/integración no operables.

- **Sincronización.** Un cambio realizado en un modelo de proceso colaborativo debe propagarse a los correspondientes modelos de procesos de interfaz/integración de las organizaciones. Si la sincronización no es realizada y mantenida, los modelos de procesos de interfaz/integración pueden quedar inconsistentes con respecto a los modelos de procesos colaborativos.

- **Privacidad.** Debido a que la lógica de negocio interna es un activo de gran valor de la organización, ésta debe mantenerse dentro de los límites de la organización. Es decir, las organizaciones no deben tener acceso a la lógica de negocio interna contenida en los modelos de procesos de integración de las demás organizaciones.

- **Asistencia en el diseño de los procesos de negocio.** Debido a que el modelado es una tarea costosa, propensa a errores y que lleva mucho tiempo, se debe asistir a las organizaciones en el diseño de procesos colaborativos y principalmente en el diseño de procesos de integración, garantizando que éstos sean consistentes e interoperables, como se discute en el Capítulo 5. También, se debe asistir en el modelado de procesos de interfaz con el propósito que las organizaciones entiendan el rol que desempeñan en la colaboración inter-organizacional, como se discute en el Capítulo 4.

- **Implementación distribuida.** Debido a la naturaleza distribuida de las colaboraciones inter-organizacionales, los modelos de procesos de negocio deben gestionarse en repositorios distribuidos entre las organizaciones participantes. Los modelos de procesos colaborativos deben ser compartidos en un lugar público y accesible por "stakeholders" de diferentes organizaciones, mientras que los modelos de procesos de integración de cada organización deben ser editados por "stakeholders" de la misma y mantenidos en forma privada en la organización.

Existen diferentes repositorios de modelos de procesos de negocio (Chituc y otros, 2009; Hofreiter, 2008; La Rosa y otros, 2011b; Ma y otros, 2007; Theling

y otros, 2005; Vanhatalo y otros, 2006), que varían con respecto a las técnicas de gestión y las formas de almacenamiento que soportan, ya sea mediante sistema de archivos o base de datos. A continuación se realiza un análisis comparativo de los mismos con respecto al soporte que proveen para satisfacer los requerimientos planteados.

La Rosa y otros (2011b) proponen el Advanced PROcess MOdel REpository (APROMORE), el cual ofrece funcionalidades para mantener, analizar y explotar el contenido de modelos de procesos. Sus funcionalidades se enfocan en el análisis basado en modelos, filtros ("filtering") y consolidación, las cuales pueden operar sobre uno o conjuntos relacionados de modelos de procesos de negocio.

Ma y otros (2007) proponen el Semantic Business Process Repository (SBPR), el cual almacena instancias de modelos de procesos de negocio basados en ontologías de procesos. SBPR soporta consultas, razonamiento, versionamiento y gestión de "check-in"/"check-out" de modelos.

Vanhatalo y otros (2006) proponen el WS-BPEL Repository, el cual es un plug-in de Eclipse que almacena procesos de negocio definidos con el lenguaje WS-BPEL y otros documentos XML relacionados. Este repositorio provee funcionalidades para almacenar, encontrar y usar estos documentos.

Chituc y otros (2009) proponen el Collaborative Interoperability Framework (CIbFw), el cual provee un enfoque orientado a servicios que soporta la interoperabilidad en un entorno interconectado. CIbFw permite a las organizaciones que han cargado en un repositorio centralizado su perfil de colaboración ser añadidas y eliminadas de una red de colaboración y realizar negocios electrónicos mediante el intercambio de documentos de negocio.

Hofreiter (2008) propone el ebXML Registry, el cual permite a las organizaciones registrar sus coreografías globales y locales, manteniendo las dependencias entre las mismas.

Theling y otros (2005) proponen el BPMN Repository Architecture, el cual permite planificar, ejecutar y controlar procesos colaborativos. El componente central de esta arquitectura es un repositorio distribuido que gestiona toda la información.

Sin embargo, estos repositorios no dan soporte a todos los requerimientos mencionados en la sección anterior. La Tabla 2.3 resume los resultados de comparar estos repositorios con respecto a su capacidad para satisfacer los reque-

Tabla 2.3: Comparación de repositorios de modelos de procesos de negocio. Nota: + (Soportado), +- (Parcialmente soportado), - (No soportado)

	Repositorios de modelos de procesos de negocio					
Criterio de comparación	La Rosa y otros (2011b)	Ma y otros (2007)	Vanhatalo y otros (2006)	Chituc y otros (2009)	Hofreiter (2008)	Theling y otros (2005)
Dominio inter-organizacional	-	-	-	+	+	+
Implementación distribuida	-	-	-	-	-	+
Sincronización	+-	-	-	-	-	-
Interoperabilidad	-	-	-	-	+-	-
Consistencia	+-	-	-	-	+-	-
Asistencia en el diseño	+-	+-	+-	+-	+-	+-
Privacidad	+	+	+	+	+	+

rimientos discutidos anteriormente para gestionar modelos de procesos de negocio involucrados en colaboraciones inter-organizacionales.

Por un lado, podemos mencionar los repositorios APROMORE (La Rosa y otros, 2011b), SBPR (Ma y otros, 2007) y WS-BPEL Repository (Vanhatalo y otros, 2006). Éstos se enfocan principalmente en gestionar modelos de procesos de negocio que no exceden los límites de una organización. Por consiguiente, éstos no proveen soporte para una implementación distribuida de repositorio.

SBPR y WS-BPEL Repository no proveen soporte para la sincronización entre modelos. APROMORE soporta la consolidación de procesos (La Rosa y otros, 2012) a través de un enfoque que permite unir semi-automáticamente una colección de modelos de procesos en un único modelo (denominado modelo fusionado). Los cambios realizados al modelo fusionado son propagados a cada variante del proceso de negocio, manteniéndolos sincronizados. Sin embargo, éste no considera la sincronización de diferentes tipos de modelos de procesos de negocio, tales como modelos de procesos colaborativos y de integración.

Ninguno de estos tres repositorios provee soporte para garantizar la interoperabilidad entre modelos de procesos de negocio de diferentes organizaciones.

SBPR y WS-BPEL Repository no soportan tampoco la verificación de consistencia. APROMORE identifica el análisis de conformidad ("conformance") como una funcionalidad para evaluar en qué medida un modelo de entrada se ajusta

a un modelo de proceso de referencia en un dominio determinado, pero este análisis no se considera entre diferentes tipos de modelos, como ser modelos de procesos colaborativos y modelos de procesos de integración.

Los tres repositorios proveen asistencia en el diseño de modelos de procesos de negocio basados en el re-uso de contenido existente, es decir, re-uso de fragmentos de procesos de negocio o procesos de negocio completos ya existentes en el repositorio. Esto es apropiado en enfoques de desarrollo "bottom-up" de colaboraciones inter-organizacionales, en los cuales cada organización primero define sus procesos privados o de integración en forma aislada y luego debe buscar por organizaciones o socios de negocio potenciales cuyos procesos sean compatibles, es decir, con los que pueda interoperar. En la práctica, descubrir organizaciones o socios potenciales requiere comparaciones complejas de procesos de interfaz/integración y es bastante improbable encontrar procesos de interfaz/integración compatibles (Hofreiter, 2008). Las colaboraciones inter-organizacionales comienzan generalmente con acuerdos de colaboración y metas de negocio comunes. En dicho caso, un enfoque "top-down" es más factible de usar, ya que a partir de lo anterior las organizaciones pueden definir los modelos de procesos colaborativos y a partir de los mismos derivar sus modelos de procesos de interfaz e integración, tal como se propone en los Capítulos 4 y 5, respectivamente.

APROMORE, SBPR y WS-BPEL Repository preservan los aspectos privados de la organización debido a que éstos se implementan dentro de una organización.

Por otro lado, los repositorios CIbFw (Chituc y otros, 2009), ebXML Registry (Hofreiter, 2008) y BPMN Repository Architecture (Theling y otros, 2005) se enfocan en gestionar modelos de procesos de negocio en ambientes inter-organizacionales.

Sólo BPMN Repository Architecture provee una implementación distribuida del repositorio.

La sincronización entre modelos de procesos de negocio no es soportada por ninguno de estos tres repositorios. ebXML Registry mantiene dependencias entre coreografías globales (procesos colaborativos) y coreografías locales (procesos de interfaz), pero no mantiene la sincronización de los mismos.

ebXML Registry garantiza la consistencia e interoperabilidad entre coreografías globales y locales debido a que cada organización deriva su coreografía

local de una coreografía global común basada en un perfil UML dedicado. No obstante, éste no considera la consistencia e interoperabilidad de modelos de procesos de integración (orquestación). CIbFw y BPMN Repository Architecture no proveen soporte para la consistencia e interoperabilidad de modelos de procesos de negocio.

CIbFw permite descargar una plantilla de acuerdo de colaboración, completarlo y cargarlo a un repositorio centralizado. ebXML Registry provee asistencia en el diseño permitiendo el re-uso de coreografías globales disponibles aplicando un enfoque "top-down". BPMN Repository Architecture gestiona modelos de referencia que pueden ser importados desde el repositorio a herramientas de modelado para crear modelos individuales. Sin embargo, éstos tres repositorios no proveen mecanismos para diseñar y gestionar modelos de procesos de interfaz y de integración.

CIbFw, ebXML Registry y BPMN Repository Architecture preservan los aspectos privados de las organizaciones a través del control de acceso a los datos del repositorio. ebXML Registry lo consigue almacenando sólo coreografías globales y locales, las cuales no incluyen actividades privadas.

En resumen, el estado del arte actual de repositorios de modelos de procesos de negocio es que los repositorios existentes no proveen funcionalidades para dar soporte a los principales requerimientos identificados en la tesis para gestionar modelos de procesos de negocio en colaboraciones inter-organizacionales.

CAPÍTULO 3

Metodología para el Desarrollo de Colaboraciones Inter-organizacionales

El capítulo presenta una metodología para el diseño e implementación de colaboraciones inter-organizacionales. La metodología propuesta sigue un enfoque "top-down" basado en los principios del desarrollo dirigido por modelos. Se proponen las fases, actividades y los artefactos requeridos para guiar un proceso de desarrollo que permita generar una solución tecnológica para colaboraciones inter-organizacionales a partir de modelos conceptuales de procesos colaborativos. De esta manera es posible determinar los modelos de procesos a ser gestionados en dicho proceso de desarrollo. Se presenta el marco conceptual en el que se basa la metodología propuesta (Sección 3.1). Se describen las fases y actividades de la metodología (Sección 3.2). Se muestra la aplicación de la metodología a un caso de estudio (Sección 3.3). Se exponen las conclusiones (Sección 3.4).

3.1. Marco conceptual para el desarrollo de colaboraciones inter-organizacionales

La metodología propuesta se basa en un marco conceptual que proporciona los diferentes dominios o niveles de abstracción en los que los analistas de negocio, diseñadores y desarrolladores de sistemas o software deben hacer foco para el desarrollo de TI para colaboraciones inter-organizacionales (Lazarte y otros, 2010). A partir de esta metodología se desarrollaron los métodos y herramientas de diseño y gestión de modelos de procesos para colaboraciones inter-organizacionales.

El marco conceptual se basa en la separación de dominios propuesta por Nikiforova (2009). Estos dominios son: el dominio del problema, el dominio de la

solución, y el dominio de la implementación (Figura 3.1).

El *dominio del problema* se refiere a la identificación de los requerimientos de una colaboración inter-organizacional desde un punto de vista organizacional o de negocio, los cuales se derivan del modelo de gestión o de negocio colaborativo que las organizaciones acuerdan llevar a cabo. Tales requerimientos son las metas de negocio comunes a ser alcanzadas, las organizaciones involucradas y los roles que cada organización va a desempeñar, así como la identificación de los procesos colaborativos a llevarse a cabo.

El *dominio de la solución* se refiere a cómo satisfacer los requerimientos identificados en el dominio del problema. En este dominio se definen y diseñan una *solución inter-organizacional* y una *solución de arquitectura de TI*. El propósito es separar las incumbencias relacionadas a la lógica de negocio de los procesos colaborativos de aquéllas relacionadas al diseño de los sistemas de información involucrados en la solución tecnológica. Las soluciones definidas en este dominio son independientes de la plataforma, y por lo tanto son representaciones de alto nivel de abstracción. De esta manera, las soluciones definidas pueden re-usarse para implementarlas en diferentes tecnologías.

La *solución inter-organizacional* consiste en definir cómo las organizaciones gestionan la colaboración inter-organizacional. Esto implica la definición del comportamiento explícito de los procesos colaborativos desde un punto de vista organizacional o de negocio, y la definición de los procesos de negocio internos que cada organización debe modelar para soportar los procesos colaborativos. Esta solución es definida por los analistas de negocio usando conceptos más cercanos al dominio del problema.

La *solución de arquitectura de TI* consiste en definir la arquitectura de sistemas que cada organización debe implementar para soportar los procesos de negocio definidos en la solución inter-organizacional. Esta solución es definida por los

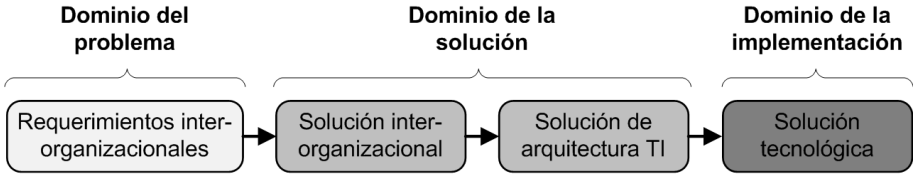


Figura 3.1: Marco conceptual para el desarrollo de colaboraciones inter-organizacionales.

diseñadores de sistemas o software usando conceptos más cercanos al dominio de la implementación, pero abstrayendo los detalles técnicos, en un nivel de abstracción independiente de la plataforma o tecnología de implementación. El propósito de esta solución es facilitar la transición de la solución inter-organizacional a la solución tecnológica.

El *dominio de la implementación* se refiere al desarrollo de la *solución tecnológica*, la cual consiste en definir las especificaciones ejecutables de los procesos de integración privados y de las interfaces de los sistemas usando tecnologías o estándares B2B. De esta manera los sistemas de las organizaciones pueden interoperar y ejecutar los procesos colaborativos. La solución tecnológica es implementada por desarrolladores de sistemas o software usando conceptos específicos de la plataforma de implementación seleccionada.

3.2. Metodología para el desarrollo de colaboraciones inter-organizacionales

En base al marco conceptual descrito en la sección anterior, la metodología propuesta define las fases a realizar y los artefactos de desarrollo a generar y/o usar. El propósito es guiar a las organizaciones en el desarrollo de colaboraciones inter-organizacionales.

Los principales artefactos de desarrollo de la metodología son los modelos, los cuales se definen y generan a través de transformaciones de modelos basadas en MDD y MDA. La metodología indica los métodos de transformación de modelos a ser usados en cada fase para generar los modelos correspondientes. El foco del esfuerzo va desde modelos definidos en altos niveles de abstracción, que describen la solución inter-organizacional y la solución de arquitectura de TI, a aquellos definidos en bajos niveles de abstracción, que describen la solución tecnológica. En consecuencia, la metodología usa modelos de procesos de negocio y sistemas para construir los artefactos de software de la solución tecnológica de una colaboración inter-organizacional. Los artefactos de software finales a generar son las especificaciones ejecutables correspondientes a los procesos de negocio y las interfaces de los sistemas de las organizaciones.

En la Figura 3.2 se presentan las cuatro fases que componen la metodología,

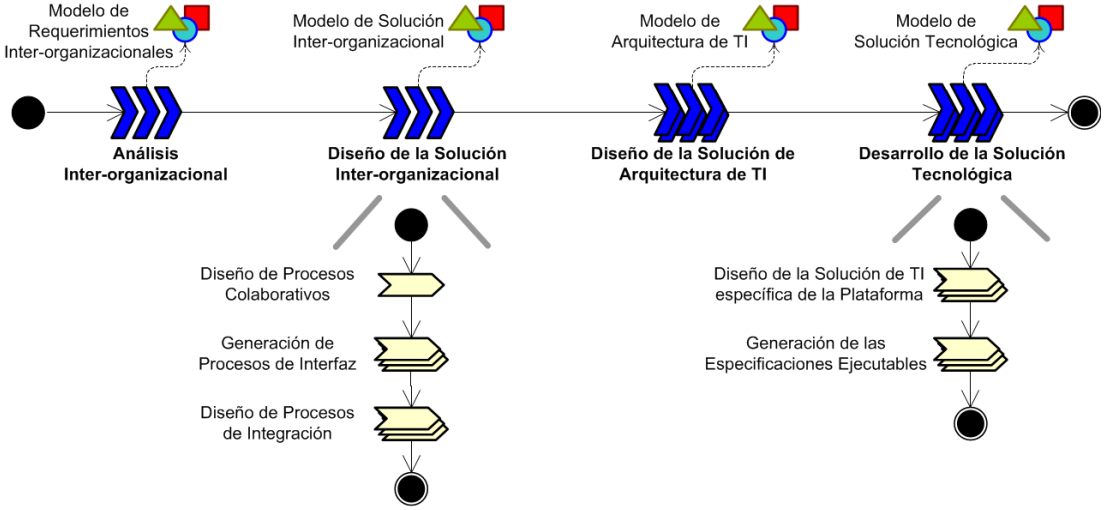


Figura 3.2: Fases de la metodología para el desarrollo de colaboraciones inter-organizacionales.

las cuales son: (1) Análisis Inter-organizacional; (2) Diseño de la Solución Inter-organizacional; (3) Diseño de la Arquitectura de TI; y (4) Desarrollo de la Solución Tecnológica. Estas fases son descritas en las siguientes sub-secciones.

La fase *Análisis Inter-organizacional* y la actividad *Diseño de Procesos Colaborativos* de la fase *Diseño de la Solución Inter-organizacional* se llevan a cabo en forma conjunta por las organizaciones participantes. En cambio, a partir de la actividad *Generación de Procesos de Interfaz* de la fase *Diseño de la Solución Inter-organizacional*, las actividades y fases que le siguen son realizadas en paralelo por cada organización.

En cada fase de la metodología los modelos obtenidos se clasifican como CIM, PIM y PSM de acuerdo a la propuesta de OMG (2003b) y se indican las transformaciones de modelos a ser aplicadas (Figura 3.3). La metodología pretende servir de guía para el desarrollo de colaboraciones inter-organizacionales, siendo independiente de los lenguajes de modelado y métodos de transformación concretos que puedan aplicarse en cada fase. En particular, en este capítulo se describen lenguajes y métodos que pueden ser utilizados en las fases y actividades para alcanzar los objetivos planteados en la tesis con respecto al diseño y gestión de modelos de procesos de negocio.

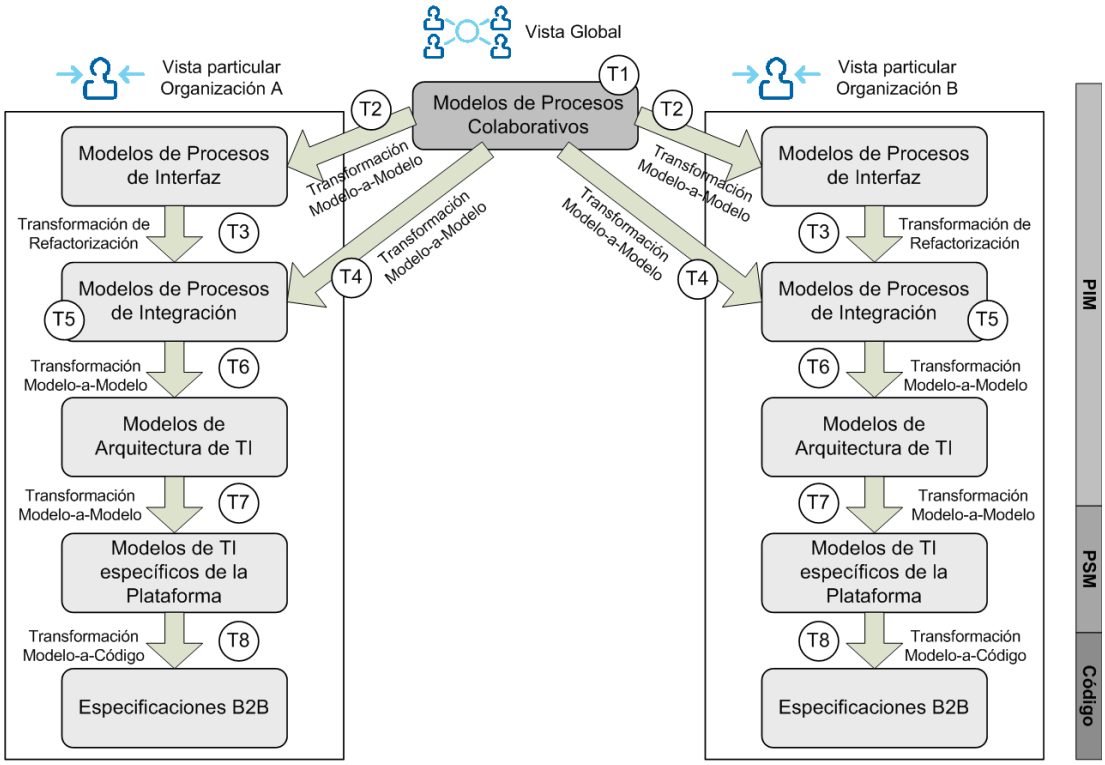


Figura 3.3: Transformaciones de modelos sugeridas por la metodología.

3.2.1. Fase: Análisis Inter-organizacional

Esta fase consiste en analizar el dominio del problema e identificar los requerimientos inter-organizacionales o de negocio. Se lleva a cabo en forma conjunta por las organizaciones participantes de una colaboración para definir el *modelo de requerimientos inter-organizacionales*, el cual se clasifica como CIM.

El modelo de requerimientos se puede definir usando el lenguaje UP-ColBPIP, ya que el mismo soporta el modelado del contexto de negocio o inter-organizacional de los procesos colaborativos a través de la definición de la *Vista de Colaboración Inter-organizacional* y la *Vista de Procesos Colaborativos* (Capítulo 2). La *Vista de Colaboración Inter-organizacional* captura los participantes, esto es las organizaciones y los roles que éstas desempeñan, como así también sus relaciones de colaboración. Esta vista también define los parámetros del acuerdo de colaboración y la jerarquía de las metas comunes. La *Vista de Procesos Colaborativos* identifica los procesos colaborativos requeridos para alcanzar cada una de las metas de la colaboración.

En consecuencia, usando el lenguaje UP-ColBPIP, la salida de esta fase es un

modelo de requerimientos inter-organizacionales, representado mediante ambas vistas. Este modelo ayuda a los analistas de negocio a entender el problema e identificar los procesos colaborativos a ser diseñados en la siguiente fase.

3.2.2. Fase: Diseño de la Solución Inter-organizacional

Esta fase se enfoca en el diseño de los procesos de negocio requeridos para llevar a cabo una colaboración inter-organizacional. Las personas involucradas son los analistas de negocio y los diseñadores de sistemas o software responsables de definir los aspectos organizacionales o de negocio de la colaboración.

El foco inicial es la definición detallada de los *procesos colaborativos*, los cuales describen el comportamiento común de las interacciones entre las organizaciones desde un punto de vista global (Bauer y otros, 2005; Villarreal y otros, 2007b; Weske, 2007). También se deben definir los procesos de negocio que cada organización debe implementar para desempeñar su rol en el proceso colaborativo. Por lo cual es necesario definir los *procesos de interfaz* y los *procesos de integración* que describen dicho rol.

Estos tres tipos de procesos que conforman la *solución inter-organizacional* difieren entre si en el punto de vista y el nivel de detalle en que son expresados y se definen con independencia de la plataforma de implementación.

En consecuencia, en esta fase se realizan tres actividades (Figura 3.2): (1) Diseño de Procesos Colaborativos; (2) Generación de Procesos de Interfaz; y (3) Diseño de Procesos de Integración. Estas actividades se describen en las siguientes sub-secciones.

Las contribuciones de la tesis están orientadas a dar soporte a las actividades de esta fase en relación con el diseño y gestión de los modelos de procesos que definen las colaboraciones inter-organizacionales.

3.2.2.1. Actividad: Diseño de Procesos Colaborativos

En esta actividad, los analistas de negocio de las organizaciones trabajan en conjunto para modelar los procesos colaborativos desde una perspectiva inter-organizacional o de negocio. Dicha actividad puede realizarse usando el lenguaje UP-ColBPIP (Villarreal y otros, 2010, 2007b). Una de las principales contribuciones de este lenguaje es el uso de protocolos de interacción para modelar el

comportamiento de los procesos colaborativos. Un *protocolo de interacción* describe un patrón de comunicación de alto nivel a través de una coreografía de mensajes entre las organizaciones (Villarreal y otros, 2010). De manera que el diseño de los procesos colaborativos se centra en el modelado del flujo de control de los mensajes que representan el intercambio de documentos de negocio entre las organizaciones. Un mensaje contiene un documento de negocio, la semántica del mensaje es descrita por un acto de comunicación ("speech act") que permite explicitar la intención del emisor en relación al documento que se envía. Decisiones y compromisos entre las partes pueden ser establecidos y conocidos a partir de actos de comunicación. Esto posibilita la definición de negociaciones complejas y evita la ambigüedad en la semántica y entendimiento de las interacciones entre organizaciones.

Con el lenguaje UP-ColBPIP, el comportamiento de los procesos colaborativos es expresado a través de la *Vista Protocolos de Interacción* de una colaboración inter-organizacional, la cual implica modelar un protocolo de interacción por cada proceso colaborativo definido. Este lenguaje soporta también la definición de la *Vista Documentos de Negocio*, que describe los documentos de negocio electrónicos a ser intercambiados, y la *Vista Interfaces de Negocio*, que describe las interfaces de cada rol desempeñado por las organizaciones. Cada interfaz de negocio (servicio) contiene las operaciones que soportan el intercambio de mensajes definidos en los protocolos de interacción.

Un requerimiento importante es que las organizaciones puedan asegurar que el comportamiento de los modelos de procesos colaborativos esté bien definido y libre de errores estructurales y lógicos. Esto requiere del uso de técnicas de verificación y validación (V&V). A efectos de no trasladar errores, la V&V de estos modelos debe realizarse en etapas tempranas de desarrollo de la colaboración inter-organizacional, que es cuando se toman la mayoría de las decisiones fundamentales, previo a la generación de la solución de arquitectura de TI (Villarreal y otros, 2007a).

Para la V&V se pueden aplicar técnicas formales como el método propuesto por Roa y otros (2012) el cual transforma los modelos de procesos colaborativos a Redes de Petri Coloreadas (T1 en la Figura 3.3). Cada Red de Petri Coloreada se verifica y valida usando herramientas apropiadas para determinar si el modelo está bien definido. Si no lo está, se hacen las modificaciones necesarias y se vuelve

a verificar.

La salida de la actividad *Diseño de Procesos Colaborativos* son *modelos de procesos colaborativos* verificados y validados. Éstos se clasifican como PIM.

3.2.2.2. Actividad: Generación de Procesos de Interfaz

En esta actividad los analistas de negocio de cada organización generan los procesos de interfaz correspondientes a cada proceso colaborativo en el que participa la organización. Se definen las actividades públicas que le permiten a la organización el intercambio de mensajes en el orden establecido en cada proceso colaborativo. Los modelos de procesos de interfaz pueden derivarse en forma completa a partir de los modelos de procesos colaborativos debido a que la información requerida se encuentra definida en estos últimos.

Para facilitar esta actividad, en la tesis se propone un método de transformación de modelos basado en MDA, el cual se describe en el Capítulo 4. Este método puede ser utilizado para llevar adelante la transformación de modelos T2 (Figura 3.3). Este método genera automáticamente modelos de procesos de interfaz definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP de acuerdo al rol que una organización cumple en cada proceso colaborativo. El método garantiza que los modelos de procesos de interfaz generados sean interoperables y consistentes con la lógica global definida en los procesos colaborativos.

Como se discutió en el capítulo anterior, se eligió el lenguaje BPMN (OMG, 2011a) para modelar los procesos de interfaz debido a que es un lenguaje de modelado orientado a actividades que permite representar procesos de negocio independientes de la plataforma, desde el punto de vista particular de una organización. El lenguaje BPMN incorpora los conceptos de procesos de interfaz e integración a través de lo que denomina procesos públicos y privados, respectivamente.

La salida de la actividad *Generación de Procesos de Interfaz* son los *modelos de procesos de interfaz* de cada organización. Éstos se clasifican como PIM.

3.2.2.3. Actividad: Diseño de Procesos de Integración

En esta actividad cada organización define la vista privada del rol que desempeña en un proceso colaborativo. Con este propósito, las organizaciones

definen las actividades privadas necesarias para procesar y generar la información intercambiada en los procesos colaborativos.

Esta actividad puede realizarse usando la transformación de modelos T3 o la transformación de modelos T4 (Figura 3.3). La transformación T3 implica realizar una refactorización del modelo de proceso de interfaz generado previamente, el cual se toma como un esqueleto o estructura para diseñar el correspondiente proceso de integración. Es decir, se agrega al modelo de proceso de interfaz las actividades internas y privadas requeridas para generar el correspondiente modelo de proceso de integración, sin modificar las actividades públicas y la lógica de flujo de control, ambas definidas en el modelo de entrada de la transformación. La desventaja de la refactorización de modelos es que debe ser realizada manualmente y, como consecuencia, los modelos de procesos de integración obtenidos podrían no ser interoperables entre sí y/o inconsistentes con la lógica definida en los modelos de procesos colaborativos.

Otra alternativa es usar la transformación T4, mediante la cual, a partir del modelo de proceso colaborativo, se generan en forma automática modelos de procesos de integración interoperables y consistentes para cada organización.

Con el propósito de facilitar esta transformación, en la tesis se propone un método basado en MDA, el cual se describe en el Capítulo 5. Este método permite diseñar y generar automáticamente modelos de procesos de integración definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP.

El uso de la transformación T3 o T4 para la definición de los modelos de procesos de integración puede requerir de la intervención de los diseñadores para definir los detalles de las actividades privadas. En dicho caso, se pueden introducir en los modelos errores por parte de los diseñadores (tales como “deadlocks”, “livelocks”, etc.), provocando inconsistencias en los mismos. Por lo tanto, en esta actividad también se requiere de métodos de V&V (T5 en la Figura 3.3) para determinar que los procesos están libres de errores, como así también garantizar que los modelos de procesos de integración satisfacen el comportamiento público definido en los procesos colaborativos. La V&V puede realizarse usando diferentes métodos de verificación de procesos, tales como los métodos propuestos por Gröner y otros (2012); Martens (2005).

La salida de la actividad *Diseño de Procesos de Integración* son *modelos de*

procesos de integración verificados y validados. Éstos se clasifican como PIM.

3.2.3. Fase: Diseño de la Arquitectura de TI

Esta fase la realiza cada organización para definir el *modelo de arquitectura de TI* (Figura 3.2). Este modelo se define de forma independiente de la plataforma para desacoplar la solución lógica de TI de su implementación tecnológica, permitiendo que tal implementación pueda ser producida en diferentes plataformas y tecnologías.

El modelo de arquitectura de TI de una organización se deriva de los modelos de procesos de integración definidos en la fase previa. Es definido por los diseñadores de sistemas o software para representar la parte de la solución de sistemas que permanecerá sin cambios en cualquier plataforma, considerando el paradigma de arquitectura de software que las organizaciones han acordado usar. Las arquitecturas SOA o Multi-Agent System (MAS) son buenas candidatas para usar en la definición de este modelo desde un punto de vista conceptual.

El modelo de arquitectura de TI de una organización principalmente define las interfaces de sus sistemas inter-organizacionales. Es útil para facilitar la comunicación entre analistas de negocio y diseñadores de sistemas o software, y fundamentalmente permite una transición más simple desde la solución inter-organizacional a la solución tecnológica.

En esta fase se puede aplicar un método de transformación de modelos, identificado como T6 (Figura 3.3), que genere un *modelo de arquitectura de TI* a partir de un *modelo de proceso de integración*. El modelo generado debe ser acorde a la arquitectura de software seleccionada por las organizaciones a fin de proporcionar la información necesaria para transformar tal modelo en el PSM que represente la solución tecnológica.

Para generar una solución basada en SOA se puede usar el método propuesto por Mayer y otros (2008). Si se requiere generar una solución basada en MAS, se puede usar el método propuesto por Zinnikus y otros (2008) o el propuesto por Tello-Leal (2012).

Los *modelos de arquitectura de TI* diseñados en esta fase se clasifican como PIM.

3.2.4. Fase: Desarrollo de la Solución Tecnológica

Esta fase define la solución tecnológica que soporta la ejecución descentralizada de procesos colaborativos a través de PAISs. Los desarrolladores de sistemas o software definen esta solución usando conceptos específicos de la plataforma para generar el código ejecutable requerido por los PAISs de la organización. Dicho código corresponde a las especificaciones ejecutables de los procesos de integración y de las interfaces de los sistemas de la organización, conocidas como especificaciones B2B. Las organizaciones deben acordar los estándares y las tecnologías a usar para implementar las especificaciones B2B.

Esta fase implica realizar dos actividades (Figura 3.2): (1) Diseño de la Solución de TI específica de la Plataforma; y (2) Generación de las Especificaciones Ejecutables. Dichas actividades se describen a continuación.

3.2.4.1. Actividad: Diseño de la Solución de TI específica de la Plataforma

Esta actividad consiste en la definición de los modelos de las especificaciones ejecutables basada en una tecnología o estándar B2B. Los *modelos de TI específicos de la plataforma* diseñados en esta actividad se clasifican como PSM. Dichos modelos deben contener la información necesaria requerida por la plataforma de implementación (por ejemplo, el formato concreto de los mensajes enviados y recibidos por las organizaciones, los protocolos de transporte a usar, etc.) para poner el modelo de arquitectura de TI en operación.

El modelo de TI específico de la plataforma es dependiente de la arquitectura de software que se utilizó para definir el modelo de arquitectura de TI realizado en la fase previa. De esta manera, si este último está basado en SOA, la plataforma de implementación podrá estar basada en los estándares WS-BPEL y/o WSDL. En cambio, si está basado en MAS, se podría usar la plataforma JADE (Bellifemine y otros, 2007) o la plataforma JADEX (Pokahr y otros, 2005).

Para llevar a cabo esta actividad se puede aplicar el método de transformación de modelos T7 (Figura 3.3) que toma como entrada un *modelo de arquitectura de TI* y genera un *modelo de TI específico de la plataforma*. El método de transformación a aplicarse dependerá de la plataforma acordada por las organizaciones y será ejecutado por los desarrolladores de sistemas o software de la organización.

Como ejemplo, esta actividad se puede realizar aplicando el enfoque MDD4SOA propuesto por Mayer y otros (2008). Dicho enfoque consiste en convertir el modelo de arquitectura de TI definido con el perfil UML4SOA como un modelo SOA, a un modelo PIM intermedio denominado Modelo de Orquestación Intermedio - Intermediate Orchestration Model (IOM). Luego, el modelo IOM se transforma en un modelo PSM que representa el modelo de TI específico de la plataforma, por ejemplo, WS-BPEL/WSDL o Java.

3.2.4.2. Actividad: Generación de las Especificaciones Ejecutables

Esta actividad consiste en generar los artefactos de software finales de la implementación, que comprenden las especificaciones ejecutables de los procesos de integración y las interfaces de los sistemas inter-organizacionales de la organización. Dichos artefactos se generan a partir del modelo de TI específico de la plataforma, el cual contiene la información necesaria para generar el código.

Cabe aclarar que la metodología propuesta no se enfoca en generar el código de los artefactos de software correspondientes a las aplicaciones internas de la organización que implementan las interfaces de los sistemas inter-organizacionales.

Para realizar esta actividad se puede aplicar el método de transformación de modelos T8 (Figura 3.3) que genera el código de las especificaciones ejecutables correspondientes a los procesos de integración y las interfaces de sistemas inter-organizacionales de la organización. El código a ser generado dependerá de la plataforma seleccionada. Si la plataforma está basada en SOA y los estándares WS-BPEL y WSDL son los seleccionados para la implementación, entonces se generará código XML. En este caso, la transformación del PSM al código XML correspondiente es más directa. Como ejemplo, se podría utilizar también aquí el enfoque MDD4SOA, el cual provee transformaciones de modelos PSM a código WS-BPEL/WSDL. Si la plataforma es Java para una arquitectura basada en MAS, por ejemplo JADEX, se generará código XML y Java. En este caso se podría aplicar el método propuesto en (Tello-Leal, 2012).

3.3. Aplicación de la metodología a un caso de estudio

En esta sección se muestra la aplicación de la metodología a un caso de ejemplo del dominio de la *Gestión Integrada de Cadenas de Suministro* ("Supply Chain Integrated Management"), el cual fue tomado del estándar CPFR (VICS, 2004) de modelos de colaboración inter-organizacionales. El ejemplo consiste de una organización, en este ejemplo denominada *Megatronic*, cuyo rol en la cadena es de minorista en la venta de productos electrónicos, y de otra organización, en este ejemplo denominada *Sanx*, la cual es el proveedor de productos electrónicos de Megatronic.

Para llevar adelante la primera fase de la metodología y definir el *modelo de requerimientos inter-organizacionales*, se ha usado el lenguaje UP-ColBPIP. El diagrama de la *Vista de Colaboración Inter-organizacional* (Figura 3.4) describe los participantes. La organización Sanx desempeña el rol de *Proveedor* ("Supplier") y la organización Megatronic desempeña el rol de *Minorista* ("Retailer"), ambos relacionados a través de una relación B2B. El diagrama muestra también las interfaces correspondientes.

Como parte del modelo de requerimientos inter-organizacionales, se definió un diagrama de clases de las metas de negocio comunes (Figura 3.5), el cual describe las métricas y los objetivos de las metas a alcanzar por las organizaciones en la colaboración. En dicho diagrama se muestra el Acuerdo de Colaboración, con detalles acerca de los parámetros definidos para la colaboración, y las metas asociadas a dicho acuerdo. Se definió la meta principal *Decrease Inventory Average Levels* y cuatro sub-metas: *Accurate Replenishment Plan*, *Reliable Order Schedule*, *Reliable Delivery Schedule* y *Order Fulfillment*. Cada una fue definida como meta cuantitativa, definiendo el valor a alcanzar, la métrica utilizada y la frecuencia de actualización.

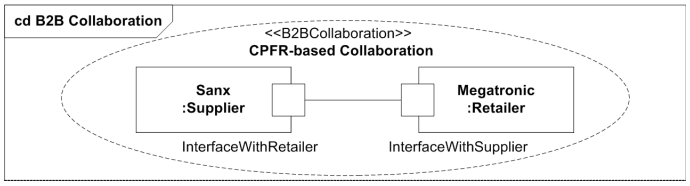


Figura 3.4: Vista de Colaboración Inter-organizacional.

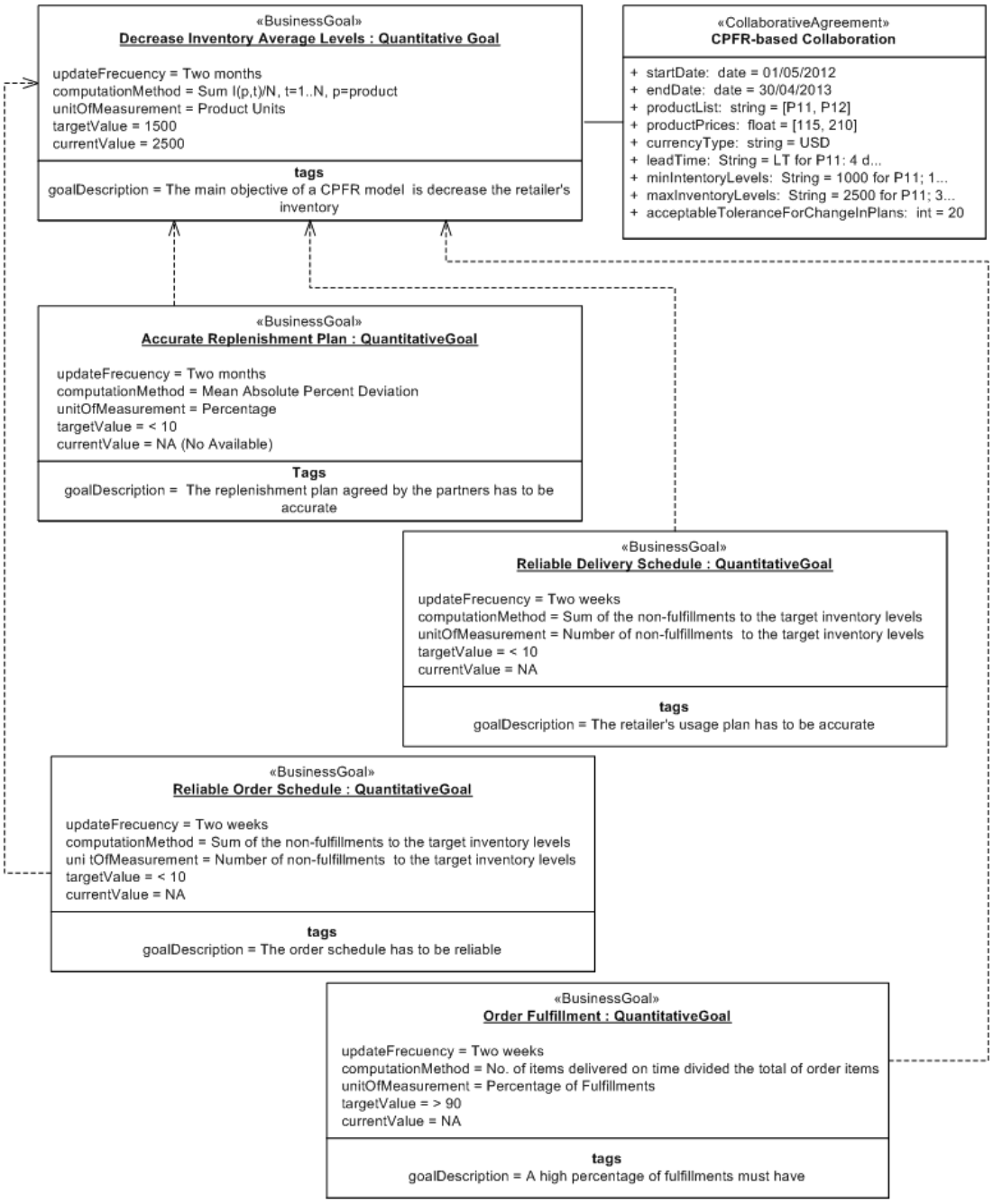


Figura 3.5: Diagrama de clases que describe el Acuerdo de Colaboración y las metas de negocio comunes.

De acuerdo al lenguaje y el método propuesto por UP-ColBPIP, por cada meta definida, se define un proceso colaborativo para alcanzarla. A través de las metas es posible evaluar los procesos. El diagrama de la *Vista de Procesos Colaborativos* (Figura 3.6) muestra los cinco procesos colaborativos a ser llevados a cabo durante la colaboración inter-organizacional, sus relaciones (sub-procesos),

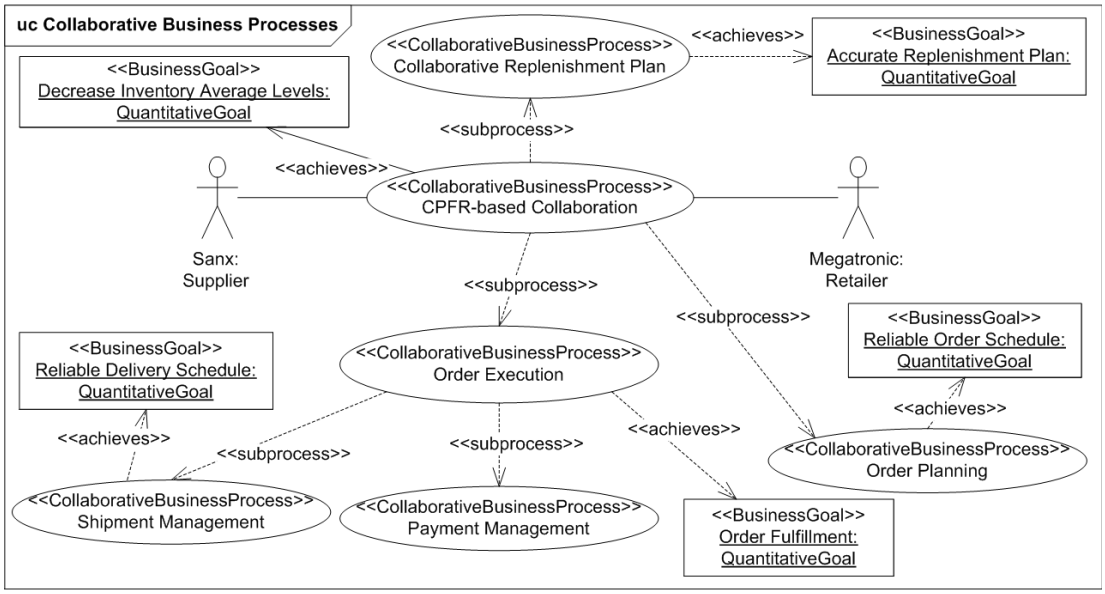


Figura 3.6: Vista de Procesos Colaborativos.

las organizaciones involucradas y la meta de negocio que cada proceso colaborativo debe alcanzar.

El lenguaje UP-ColBPIP también se utilizó para la actividad de *Diseño de Procesos Colaborativos* correspondiente a la fase de *Diseño de la Solución Inter-organizacional*. Por cada proceso colaborativo definido en la *Vista de Procesos Colaborativos*, se definió un protocolo de interacción. Como ejemplo de la *Vista de Protocolos de Interacción*, aquí se describe el *Protocolo de Interacción* del proceso colaborativo *Plan de Aprovisionamiento Colaborativo* (*Collaborative Replenishment Plan*) (Figura 3.7).

El propósito de este proceso colaborativo es que los participantes acuerden un plan de suministro a corto/mediano plazo. El proceso comienza con el proveedor quien propone un plan de suministro (SupplyPlan) de un producto, el cual debe ser negociado y acordado con el minorista. El minorista evalúa el plan con respecto a su plan de suministro generado internamente. El resultado puede dar lugar a tres respuestas alternativas, según que el responsable de logística decida aceptar, rechazar o hacer una contrapropuesta: (1) envía una respuesta de aceptación al proveedor y el proceso finaliza; (2) envía una respuesta de rechazo al proveedor y el proceso finaliza; o (3) propone cambios al proveedor, enviando su propuesta de plan de suministro.

Si el minorista propone un nuevo plan, el proveedor debe evaluarlo y pro-

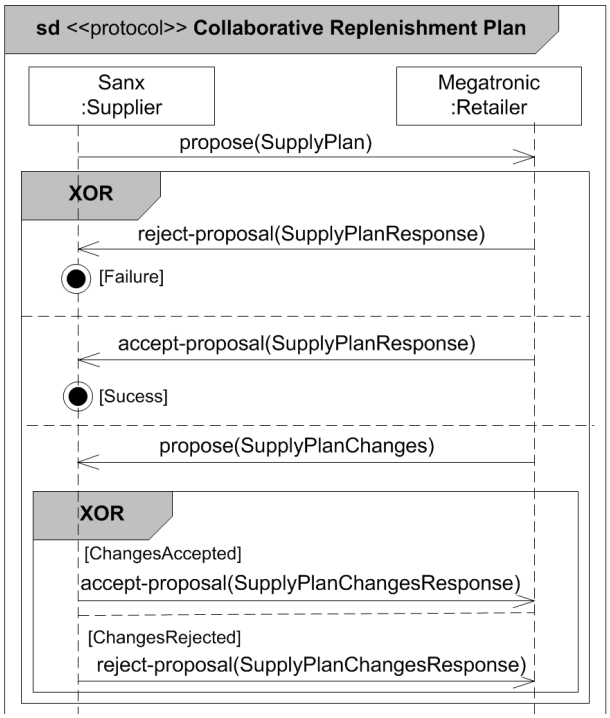


Figura 3.7: Modelo de proceso colaborativo *Collaborative Replenishment Plan*.

cesarlo teniendo en cuenta su plan de producción. Como resultado, el proveedor puede responder de dos maneras: (1) rechazar el plan de suministro propuesto por el minorista; o (2) aceptar el plan de suministro del minorista, en cuyo caso éste será el plan aceptado. Cualquiera sea la respuesta del proveedor, el proceso finaliza.

Siguiendo la metodología, la siguiente actividad es *Generación de Procesos de Interfaz* correspondiente a la fase de *Diseño de la Solución Inter-organizacional*, la cual es realizada por cada organización, tomando como entradas los modelos de procesos colaborativos definidos como protocolos de interacción en la etapa previa. Para ello, se ha aplicado el método basado en MDA descrito en el Capítulo 4.

Como ejemplo de esta actividad se muestra el *modelo de proceso de interfaz* del proveedor (Figura 3.8). A partir de los mensajes definidos en el protocolo y de acuerdo al rol proveedor del proceso colaborativo, se generan las actividades de envío y recepción de mensajes y la lógica de flujo de control del proceso de interfaz. Como ejemplo, puede verse que el mensaje *propose(SupplyPlan)* del protocolo, el cual es enviado por el proveedor al minorista, se ha transformado en una tarea de envío de mensaje (Tarea `SendTask` en BPMN) *Propose SupplyPlan*. Una

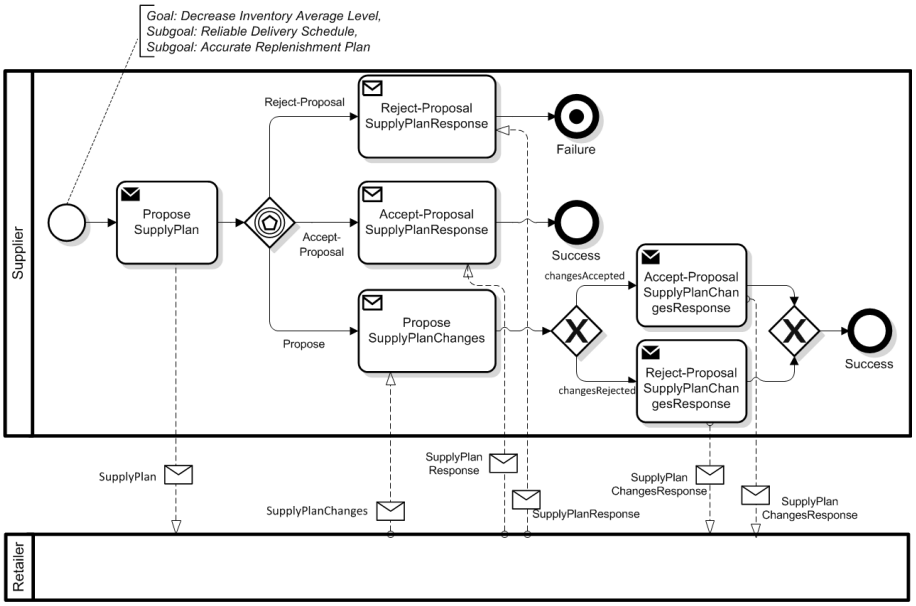


Figura 3.8: Modelo de proceso de interfaz del proveedor.

explicación detallada de cómo se generó este modelo de proceso de interfaz se presenta en el siguiente capítulo.

La siguiente actividad propuesta por la metodología es el *Diseño de Procesos de Integración* correspondiente a la fase de *Diseño de la Solución Inter-organizacional*. Para realizar esta actividad, se aplicó el método basado en MDA descrito en el Capítulo 5, el cual genera automáticamente los modelos de procesos de integración a partir del modelo de proceso colaborativo. Como ejemplo de esta actividad, se muestra el *modelo de proceso de integración* (Figura 3.9). A partir de los mensajes definidos en el protocolo y de acuerdo al rol proveedor del proceso, se generan junto con las actividades públicas de envío y recepción de mensajes, las actividades privadas requeridas para procesar o generar los documentos intercambiados en los mensajes y la lógica de flujo de control del proceso. Como ejemplo, se puede observar que para el mensaje *propose(SupplyPlan)* del protocolo, el cual es enviado por el proveedor al minorista, se ha generado una tarea privada (`Task` en BPMN) *Generate SupplyPlan*, la cual permite generar el documento a enviarse al minorista, y además una tarea de envío de mensaje (Tarea `SendTask` en BPMN) *Propose SupplyPlan*. Una explicación detallada de cómo se generó este modelo de proceso de integración se presenta en el Capítulo 5.

De esta manera se definió la solución inter-organizacional, la cual está con-

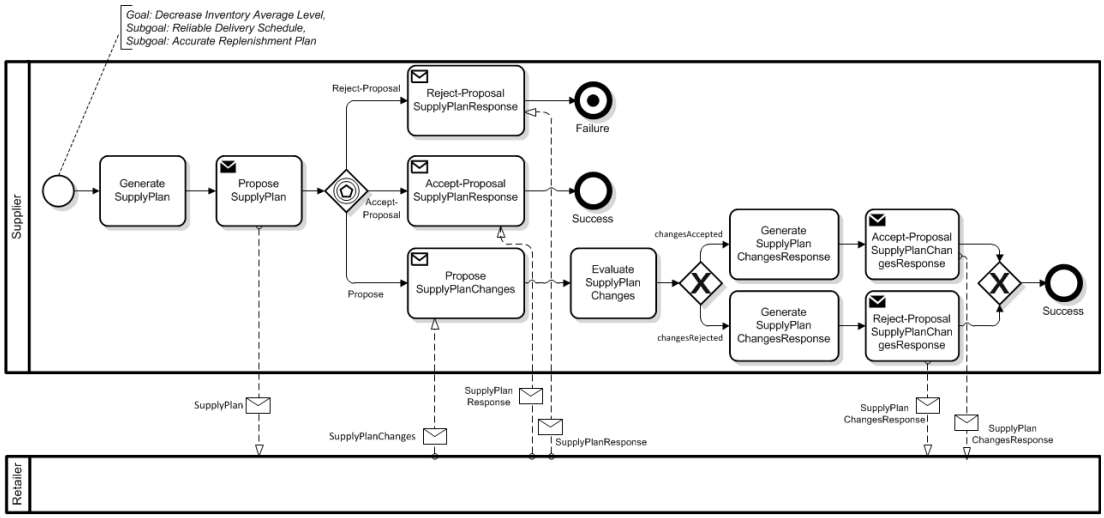


Figura 3.9: Modelo de proceso de integración del proveedor.

formada por tres tipos de modelos de procesos: los modelos de procesos colaborativos, expresados como protocolos de interacción con el lenguaje UP-ColBPIP, y los modelos de procesos de interfaz y los modelos de procesos de integración de ambas organizaciones, minorista y proveedor, expresados con el lenguaje BPMN.

La siguiente fase de la metodología corresponde al *Diseño de la Solución de Arquitectura de TI*. Se decidió aplicar el paradigma SOA para definir la arquitectura de sistemas de cada organización, por lo cual se usó el perfil UML4SOA (Mayer y otros, 2008) para expresar el *modelo de arquitectura de TI*. Si bien la metodología propone que este modelo se puede generar aplicando una transformación de modelos tomando como entrada los modelos de proceso de integración, para el caso de estudio esta actividad se realizó en forma manual, ya que la provisión de tal método de transformación está fuera del alcance de la tesis.

Como ejemplo, se muestra el *modelo de arquitectura de TI* del proveedor (Figura 3.10). El mismo se generó, tomando como entrada el modelo de proceso de integración del proveedor generado anteriormente, redefinido usando la notación provista por el perfil UML4SOA. El modelo generado describe la orquestación entre los servicios del proveedor y el minorista desde el punto de vista del proveedor, y la invocación de los servicios de los sistemas internos. Esto es, desde el punto de vista de sistemas, el modelo de proceso de integración se convirtió en un servicio compuesto que representa la orquestación entre los servicios (interfaces de sistemas) del proveedor y los servicios del minorista.

La siguiente fase de la metodología corresponde al *Desarrollo de la Solución*

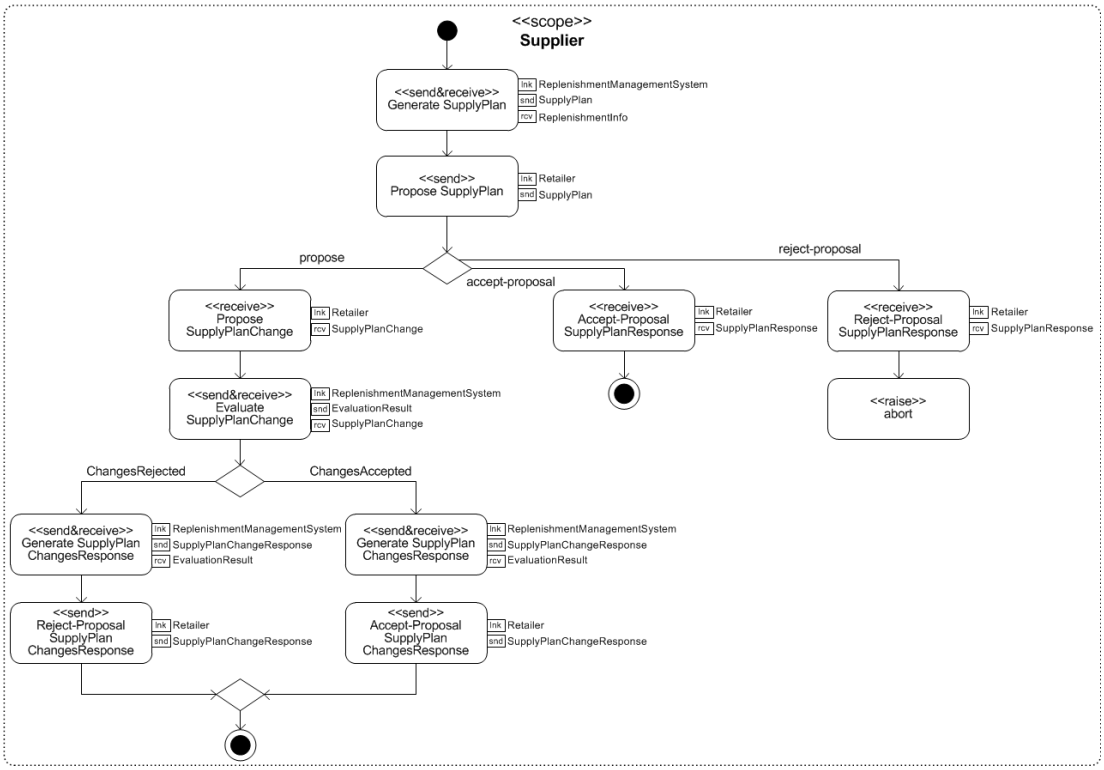


Figura 3.10: Modelo de Arquitectura de TI basada en SOA del proveedor.

Tecnológica. Para generar la solución tecnológica que implementa la colaboración inter-organizacional se decidió usar los estándares de Servicios Web WS-BPEL y WSDL. En este caso de estudio se aplicó el enfoque MDD4SOA propuesto por Mayer y otros (2008) para convertir el modelo de arquitectura de TI definido con el perfil UML4SOA a un modelo IOM. Luego, el modelo IOM se transformó en modelos PSM que representan el *modelo de TI específico de la plataforma*, expresados en términos de los meta-modelos de WS-BPEL y WSDL.

Finalmente, usando el mismo enfoque, a partir de los PSM se generó el *código WS-BPEL* y *WSDL*. Como ejemplo se muestra un fragmento del código WS-BPEL generado como parte de la solución tecnológica del proveedor (Listado 3.1), el cual representa la especificación ejecutable del proceso de integración del proveedor, que corresponde a la implementación del servicio compuesto definido anteriormente.

Listado 3.1: Fragmento del código WS-BPEL de la solución de TI específica de la plataforma del proveedor

```
1 <scope name= "Supplier">
2 <sequence name="sequence_inside_supplier">
3 <invoke name="Generate_SupplyPlan"
```

```
 4   inputVariable="ReplenishmentInfo"
 5   operation="generate_document"
 6   outputVariable="SupplyPlan"
 7   partnerLink="ReplenishmentManagementSystem"/>
 8
 9 <invoke name="Propose_SupplyPlan"
10     operation="send_document"
11     outputVariable="SupplyPlan"
12     partnerLink="Retailer"/>
13 ...
14 </sequence>
15 </scope>
```

3.4. Conclusiones

La metodología dirigida por modelos para el desarrollo e implementación de colaboraciones inter-organizacionales presentada en este capítulo aplica un enfoque "top-down" y provee una clara separación de los dominios y niveles de abstracción junto con los artefactos requeridos en el desarrollo de colaboraciones inter-organizacionales. Define los diferentes tipos de transformaciones de modelos que deben aplicarse en cada una de las fases o actividades. Además, la metodología establece qué actividades deben realizarse en forma compartida por las organizaciones y cuáles deben ser realizadas en paralelo y en forma privada por las organizaciones.

Permite a las organizaciones identificar los requerimientos inter-organizacionales o de negocio, definir la solución inter-organizacional necesaria para satisfacer dichos requerimientos, definir la arquitectura de TI de cada organización para implementar la solución inter-organizacional, y definir la solución tecnológica que implementa la colaboración inter-organizacional.

La metodología incrementa el nivel de abstracción en el diseño de colaboraciones inter-organizacionales debido a que los principales artefactos de desarrollo son los modelos conceptuales de procesos de negocio independientes de la plataforma y modelos de arquitectura de TI. Además, reduce el tiempo y costos de desarrollo debido a que la solución tecnológica requerida por cada organización puede ser automáticamente generada a partir de modelos conceptuales definidos en el dominio de la solución. Estos modelos evolucionan, mediante transformaciones de modelos, en forma consistente para generar las especificaciones ejecutables,

lo cual garantiza que la solución tecnológica es consistente y está alineada con la solución inter-organizacional y la solución de arquitectura de TI. Esto además permite a las organizaciones asegurar que las especificaciones ejecutables generadas realmente soportan los procesos colaborativos y, por consiguiente, las metas comunes definidas por las organizaciones.

La separación de la solución inter-organizacional de la solución de arquitectura de TI permite el re-uso de los modelos de procesos de negocio y la posibilidad de usar diferentes paradigmas de arquitectura de software para expresar un modelo de arquitectura de TI. Por otra parte, debido a que los modelos de arquitectura de TI de las organizaciones se expresan en forma independiente de la plataforma, los mismos pueden re-usarse para generar soluciones tecnológicas basadas en diferentes plataformas de implementación.

Finalmente, a través de esta metodología, se han identificado los modelos de procesos a diseñar y gestionar en el desarrollo de colaboraciones inter-organizacionales, como así también los métodos de transformación de modelos a emplear para generar y diseñar modelos de procesos. A partir de esto, los siguientes capítulos de la tesis muestran contribuciones concretas para realizar las fases y actividades de la metodología, desde el punto de vista de métodos de transformaciones de modelos para el diseño de los procesos y de herramientas de gestión de los modelos de procesos de negocio que conforman la solución inter-organizacional.

CAPÍTULO 4

Generación de Modelos de Procesos de Interfaz

Este capítulo presenta un método basado en MDA para la generación automática de procesos de interfaz definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP y las reglas de transformación del método (Sección 4.1). Se describe la implementación del método que permite la ejecución automática del mismo (Sección 4.2) y la aplicación a un caso de estudio (Sección 4.3). Finalmente, se presentan las conclusiones (Sección 7.3).

4.1. Método para la generación de modelos de procesos de interfaz

Para dar soporte a la actividad *Generación de Procesos de Interfaz* de la fase *Diseño de la Solución Inter-organizacional* de la metodología propuesta (Sección 3.2.2.2), se propone un método para la generación automática de modelos de procesos de interfaz, con el propósito de obtener y definir el comportamiento público del rol que una organización desempeña en un proceso colaborativo (Lazarte y otros, 2009).

El método aplica los principios de MDA para la generación automática de modelos de procesos de interfaz definidos con el lenguaje BPMN (OMG, 2011a) a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP.

El método se enfoca en transformaciones horizontales entre modelos de procesos de negocio definidos con estos lenguajes en un nivel PIM (Figura 4.1). Toma

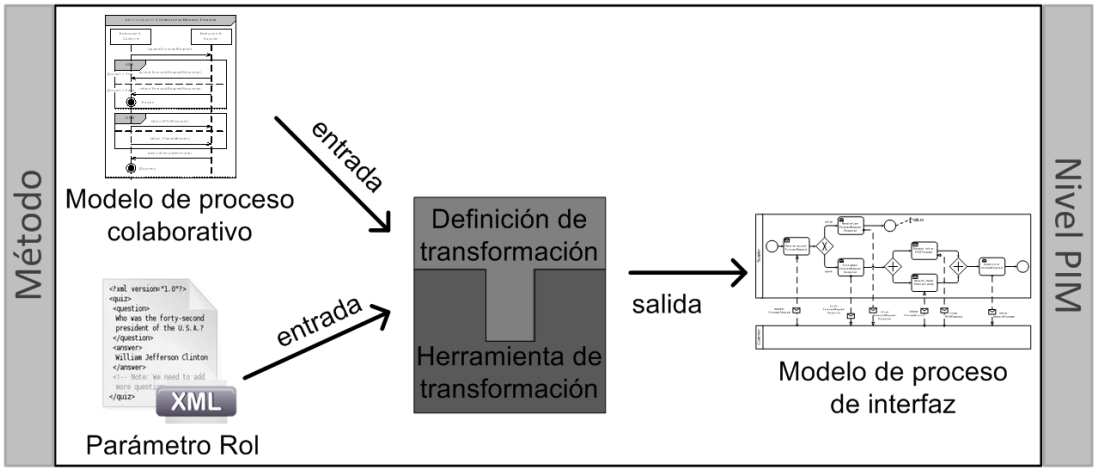


Figura 4.1: Método basado en MDA para la generación de procesos de interfaz a partir de procesos colaborativos.

como entrada un modelo de proceso colaborativo, representado como protocolo de interacción, y un modelo de parámetros el cual contiene el rol de la organización para la cual se quiere generar el modelo de proceso de interfaz.

Para llevar a cabo dicha transformación, se propone un conjunto de patrones destino definidos con el lenguaje BPMN para cada elemento del lenguaje UP-ColBPIP correspondiente a protocolos de interacción. De esta manera, la semántica de cada elemento del lenguaje UP-ColBPIP se representa en términos de los elementos y semántica provista por el lenguaje BPMN, desde el punto de vista particular de una organización.

El Algoritmo 1 muestra la estrategia general de los pasos definidos para generar modelos de procesos de interfaz a partir de modelos de procesos colaborativos. El proceso de transformación consiste en analizar cada elemento e de un modelo de proceso colaborativo M_{PC} desde el punto de vista del rol de una organización (rol R). Por cada elemento e se genera el patrón destino BPMN p, aplicando una regla de transformación. El modelo de salida M_{PI} se construye mediante la composición de los elementos generados a partir de los patrones p.

Los modelos de procesos de interfaz generados contienen las actividades públicas que permiten a la organización el envío y recepción de mensajes en el orden definido en el flujo de control del modelo de proceso colaborativo. El método garantiza que cada modelo de proceso de interfaz generado es consistente con el modelo de proceso colaborativo y, por consiguiente, es interoperable con los modelos de procesos de interfaz de las organizaciones involucradas. La consistencia

Algoritmo 1 Algoritmo para generar un modelo de proceso de interfaz M_{PI} a partir de un modelo de proceso colaborativo dado M_{PC} y un rol R.

Entrada: M_{PC} y R
  para cada elemento $e \in M_{PC}$ y punto de vista de R **hacer**
    transformar elemento e en patrón p
    insertar patrón p en modelo M_{PI}
  fin para
Salida: M_{PI}

es garantizada debido a que los modelos se generan automáticamente mediante un proceso de transformación modelo-a-modelo, aplicando un enfoque top-down.

La definición de transformación del método propuesto (Figura 4.1) usa el meta-modelo UP-ColBPIP (Apéndice A.1) para definir el patrón origen y el meta-modelo BPMN (Apéndice A.2) para definir el patrón destino de cada regla de transformación. El meta-modelo Parameters (Figura 4.2) es usado para definir como parámetro el rol a transformar. Si bien este método usa un único parámetro (el rol a transformar), el meta-modelo Parameters permite definir más parámetros. Este meta-modelo es descrito en detalle en el Capítulo 5 debido a que es utilizado en el método presentado en dicho capítulo para la definición de parámetros en la generación de modelos de procesos de integración.

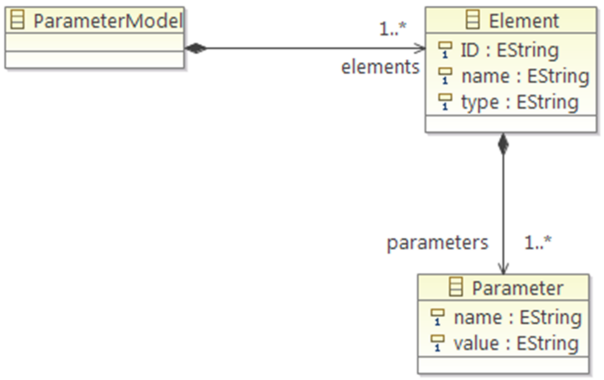


Figura 4.2: Meta-modelo Parameters.

A continuación se describen las reglas que componen la definición de transformación del método propuesto. Las reglas de transformación se describen conceptualmente y los elementos involucrados en cada una de ellas se identifican usando la notación punto.

4.1.1. Definición de transformación del método

Los patrones destino de las reglas de transformación del método propuesto permiten representar la semántica de los elementos del lenguaje UP-ColBPIP en términos de los elementos y la semántica provista por el lenguaje BPMN desde el punto de vista de un rol definido. En la ejecución de una definición de transformación, cada una de estas reglas es aplicada por cada patrón origen de las mismas que es encontrado en el modelo de proceso colaborativo de entrada.

4.1.1.1. Regla UPColBPIPModel2Definitions

Esta regla define que un modelo `UPCOLBPIP!UPColBPIPModel` se transforma en una definición `BPMN!Definitions`. A dicha definición se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor *definitionsId*; al atributo `name` se le asigna el valor `UPCOLBPIP!UPColBPIPModel.name`; al atributo `targetNamespace` se le asigna el valor *http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2*; y al atributo `rootElements` se le asocian todos los elementos contenidos en el modelo de proceso colaborativo.

4.1.1.2. Regla b2bCollaboration2Collaboration

Esta regla define que una colaboración inter-organizacional `UPCOLBPIP!B2BCollaboration` se transforma en una colaboración `BPMN!Collaboration` con el propósito de mostrar el flujo de mensajes entre las organizaciones participantes. A dicha colaboración se le asignan los siguientes valores a sus atributos: al atributo `name` se le asigna el valor `UPCOLBPIP!B2BCollaboration.name`; al atributo `isClosed` se le asigna el valor *true*; al atributo `participants` se le asigna el valor `UPCOLBPIP!B2BCollaboration.partners`; y al atributo `messageFlows` se le asocian los documentos de negocio a ser intercambiados entre las organizaciones.

4.1.1.3. Regla collaborativeProcess2Process

Esta regla define que un proceso colaborativo `UPCOLBPIP!CollaborativeBusinessProcess` se transforma en un

proceso BPMN!Process, correspondiente al proceso de interfaz de la organización desempeñando el rol definido como parámetro. A dicho proceso se le asignan los siguientes valores a sus atributos: al atributo name se le asigna el valor UPCOLBPIP!CollaborativeBusinessProcess.name; al atributo isExecutable se le asigna el valor *false* debido a que el proceso de interfaz es un proceso público; al atributo processType se le asigna el valor *Public* para indicar que se define un proceso de interfaz con contenido público; y al atributo flowElements se le asocian todos los elementos del proceso colaborativo siendo transformados.

Para indicar la meta pública a ser alcanzada por el proceso BPMN!Process, la regla genera un artefacto BPMN!TextAnnotation, a cuyo atributo id se le asigna el valor *businessGoal0* y a su atributo text se le asigna el valor de UPCOLBPIP!CollaborativeBusinessProcess.goalToFulfill. Este artefacto se asocia al atributo artifacts del proceso BPMN!Process creado por la regla.

Esta regla además genera un evento BPMN!StartEvent sin una definición de evento BPMN!EventDefiniton asociada para representar el inicio del proceso de interfaz.

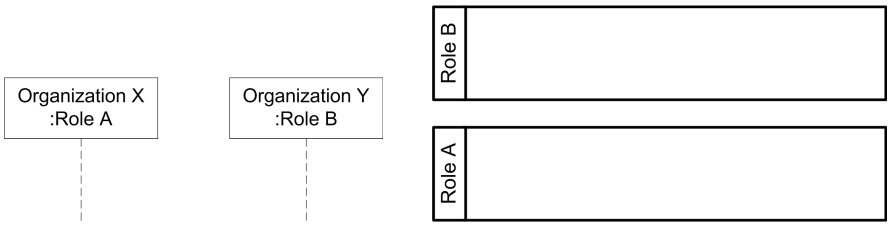
4.1.1.4. Regla tradingPartner2Participant

Esta regla define que cada organización representada en un UPCOLBPIP!TradingPartner (Figura 4.3(a)) se transforma en un participante BPMN!Participant y una entidad socio BPMN!PartnerEntity. Los participantes en BPMN representan organizaciones y se visualizan en "pools" (Figura 4.3(b)).

Los atributos de BPMN!Participant se establecen como sigue: al atributo name se le asigna el valor UPCOLBPIP!TradingPartner.name; y al atributo processRef se le asocia el proceso correspondiente al rol definido como parámetro; para el otro rol se asigna el valor *OclUndefined* debido a que el pool correspondiente a dicho rol se modela como *caja negra*, es decir, no se generan los elementos correspondientes a dicho rol. De esta manera cada organización se enfoca en sus requerimientos particulares de la colaboración inter-organizacional.

Los atributos de BPMN!PartnerEntity se establecen como sigue: al atributo name se le asigna el valor UPCOLBPIP!TradingPartner.name; y al atri-

buto `participantRef` se le asocia el correspondiente `BPMN!Participant`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.3: Representación gráfica de la regla *tradingPartner2Participant*.

4.1.1.5. Regla partnerRole2PartnerRole

Esta regla define que el rol que desempeña una organización en el proceso colaborativo, definido en un `UPCOLBPIP!PartnerRole` se transforma en un rol `BPMN!PartnerRole`. A dicho rol se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!PartnerRole.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!PartnerRole.name`; y al atributo `participantRef` se le asocia el correspondiente `BPMN!Participant`.

4.1.1.6. Reglas de transformación de mensajes de negocio

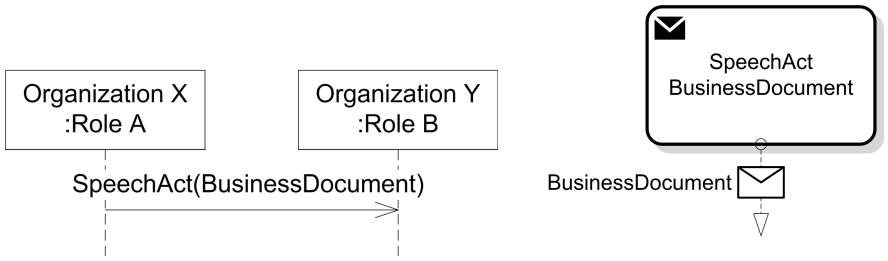
Con respecto a las reglas de transformación definidas para los mensajes de negocio de un modelo de proceso colaborativo, en el patrón destino las actividades públicas se definen como `Send Task` o `Receive Task` (OMG, 2011a), debido a que la semántica de las mismas refiere a que permiten enviar o recibir mensajes, respectivamente. A ambos tipos de tareas se asocia un `Message` (OMG, 2011a), el cual representa el documento de negocio intercambiado en un mensaje de un modelo de proceso colaborativo. Todas las actividades públicas definidas en cada patrón destino se etiquetan siguiendo el estilo *verbo-objeto* (Mendling y otros, 2010b), el cual es considerado menos ambiguo, facilitando la comunicación y el entendimiento de los modelos, y por consiguiente, mejorando la calidad del modelo de proceso de interfaz generado.

Aplicando el estilo *verbo-objeto*, la etiqueta de una actividad se forma por el acto de comunicación y el documento de negocio asociado al mensaje de negocio que representa la actividad. De esta manera, un mensaje enviado por una tarea `Send Task` o recibido por una tarea `Receive Task` no sólo será entendido por

la información que es transportada en el mismo, sino también por la intención que el rol remitente tiene con respecto a la información enviada en el mensaje.

Regla sendBusinessMessage2SendTask

Esta regla indica que un mensaje de negocio `UPCOLBPIP!BusinessMessage` enviado por el rol `UPCOLBPIP!ParnerRole = PARAMETERS!role`, cuyo documento intercambiado es `UPCOLBPIP!BusinessDocument` (Figura 4.4(a)), se transforma en una tarea de envío de mensaje `BPMN!SendTask` (Figura 4.4(b)), la cual representa el envío de un mensaje hacia una organización o participante externo.



(a) Patrón origen. (b) Patrón destino.

Figura 4.4: Representación gráfica de la regla *sendBusinessMessage2SendTask*.

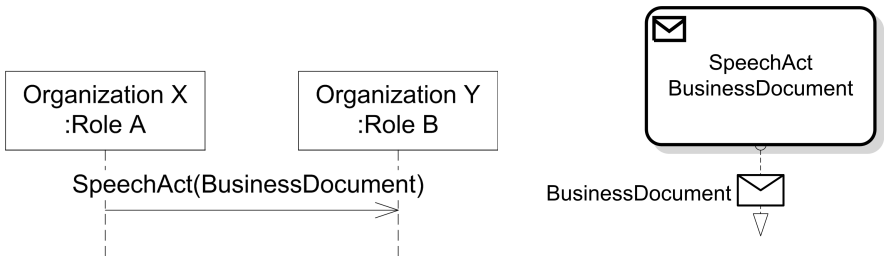
A la tarea de envío se le asignan los siguientes valores a sus atributos: al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessDocument.id`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false*; al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.

Regla receiveBusinessMessage2ReceiveTask

Esta regla indica que un mensaje de negocio `UPCOLBPIP!BusinessMessage` recibido por el rol `UPCOLBPIP!ParnerRole = PARAMETERS!role`, cuyo documento intercambiado es `UPCOLBPIP!BusinessDocument` (Figura 4.5(a)), se transforma

en una tarea de recepción `BPMN!ReceiveTask` (Figura 4.5(b)), la cual representa la recepción de un mensaje desde una organización o participante externo.

A la tarea de recepción se le asignan los siguientes valores a sus atributos: al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessDocument.id`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false*; al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.5: Representación gráfica de la regla *receiveBusinessMessage2ReceiveTask*.

4.1.1.7. Reglas de transformación de flujos de control

Debido a que el lenguaje UP-ColBPIP es estructurado, los patrones destino de las reglas correspondientes a los segmentos de flujo de control se definen en forma estructurada. Es decir, que cada conector "split" coincide con un conector "join" del mismo tipo (Mendling y otros, 2010a). Esto permite obtener modelos estructurados, los cuales son menos propensos a errores y son más fáciles de entender (Mendling y otros, 2007). En consecuencia, se incrementa la calidad de los modelos de procesos de interfaz generados. Otra de las ventajas de los modelos estructurados es que facilita la generación de las especificaciones ejecutables de los procesos de negocio, ya que la mayoría de los lenguajes ejecutables, tal como WS-BPEL, son estructurados (Mayer y otros, 2008).

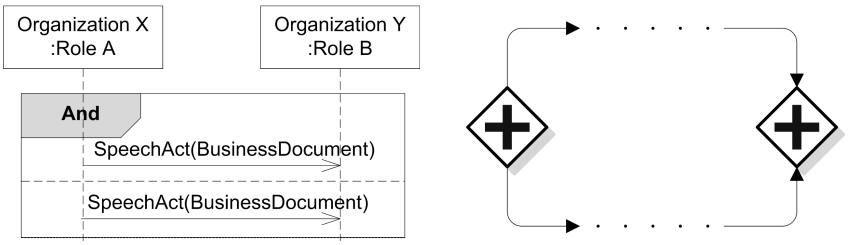
Para definir el patrón destino correspondiente a cada tipo de segmento de

flujo de control se utilizaron los patrones de flujo de control de workflows (Aalst y otros, 2003a), los cuales permiten entender y analizar los diferentes lenguajes con respecto a la definición del flujo de control de procesos/workflows. A partir del análisis de los lenguajes UP-ColBPIP (Villarreal y otros, 2010) y BPMN (OMG, 2011a) con respecto a dichos patrones de flujo de control, se estableció la relación semántica entre los constructores de ambos lenguajes, lo que permitió definir el mapeo de los constructores de los mismos.

Regla cfsAnd2ParallelGateway

Un segmento de flujo de control `UPCOLBPIP!And` (Figura 4.6(a)) hace referencia a la semántica de un patrón de workflow *Parallel Split* y un patrón de workflow *Synchronization*. Por lo tanto, esta regla transforma un segmento de flujo de control `UPCOLBPIP!And` en dos gateways `BPMN!ParallelGateway`.

El primer gateway `BPMN!ParallelGateway` (Figura 4.6(b)) permite representar la división de un flujo de control en dos o más flujos de control que pueden ejecutarse en paralelo, como lo indica el patrón de workflow *Parallel Split*. A dicho gateway se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!And.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!And.name`; al atributo `gatewayDirection` se le asigna el valor *Diverging*. Cada camino de interacción del segmento se transforma según la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14). Luego, los elementos de cada camino se transforman de acuerdo a su tipo, aplicando la regla respectiva.



(a) Patrón origen. (b) Patrón destino.

Figura 4.6: Representación gráfica de la regla *cfsAnd2ParallelGateway*.

Al segundo gateway `BPMN!ParallelGateway` se le asigna el valor *Converging* a su atributo `gatewayDirection`. Este gateway permite representar la sincronización de múltiples flujos de control, como indica el patrón de workflow *Synchronization*. Si algún camino del paralelismo de un protocolo contiene una

terminación explícita, no se genera un flujo de secuencia hacia el gateway representando la sincronización. Si todos los caminos de un paralelismo de un protocolo contienen una terminación explícita, no se genera el gateway que representa la sincronización.

Regla cfsXor2ExclusiveGateway

Un segmento de flujo de control `UPCOLBPIP!Xor` (Figura 4.7(a)) hace referencia a la semántica de un patrón de workflow *Exclusive Choice* y un patrón *Simple Merge*, para el rol que envía el primer mensaje de cada camino alternativo del segmento. En cambio, para el rol que recibe el primer mensaje de cada camino alternativo del segmento, el segmento de flujo de control `UPCOLBPIP!Xor` hace referencia a la semántica de un patrón de workflow *Deferred Choice* y un patrón de workflow *Simple Merge*.

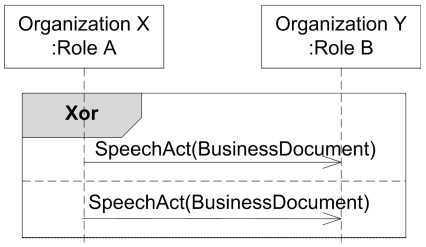
Por lo tanto, en el caso del rol que envía el primer mensaje de cada camino del segmento, la regla transforma un segmento de flujo de control `UPCOLBPIP!Xor` en dos gateways `BPMN!ExclusiveGateway` (Figura 4.7(b)). El primer gateway permite representar la evaluación de datos del proceso con el propósito de seleccionar uno de varios flujos alternativos, como lo indica el patrón de workflow *Exclusive Choice*. El segundo gateway `BPMN!ExclusiveGateway` permite representar la unión de varios caminos alternativos, como lo indica el patrón de workflow *Simple Merge*.

En cambio, para el caso que el rol recibe el primer mensaje de cada camino del segmento, la regla transforma un segmento de flujo de control `UPCOLBPIP!Xor` en dos gateways, un gateway basado en eventos `BPMN!EventBasedGateway` y un gateway `BPMN!ExclusiveGateway` (Figura 4.7(c)). El gateway `BPMN!EventBasedGateway` permite representar la selección de uno de varios flujos alternativos de acuerdo a la ocurrencia de un evento, como lo indica el patrón de workflow *Deferred Choice*. En este caso, los eventos que pueden ocurrir corresponden a la recepción de los mensajes de negocio alternativos.

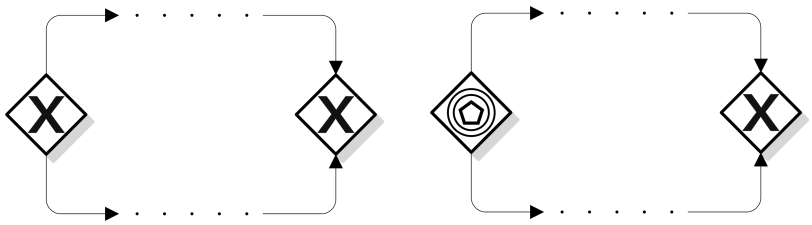
Además, en ambos casos, los atributos del primer gateway se establecen como sigue: al atributo `id` se le asigna el valor `UPCOLBPIP!Xor.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Xor.name`; al atributo `gatewayDirection` se le asigna el valor *Diverging*. Cada camino de interacción

del segmento se transforma según la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14). Luego, los elementos de cada camino se transforman de acuerdo a su tipo, aplicando la regla respectiva.

En ambos casos, al segundo gateway se le asigna el valor *Converging* a su atributo `gatewayDirection`, para representar la unión ("merge") de los caminos que no poseen una terminación explícita.



(a) Patrón origen.



(b) Patrón destino del rol emisor. (c) Patrón destino del rol receptor.

Figura 4.7: Representación gráfica de la regla *cfsXor2ExclusiveGateway*.

Regla cfsOrSynchronizingMerge2InclusiveGateway

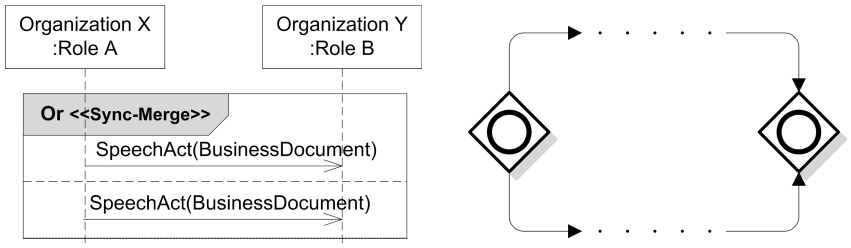
Un segmento de flujo de control `UPCOLBPIP!Or`, cuyo atributo `synchronizationType` es <<*Sync-Merge*>> (Figura 4.8(a)), hace referencia a la semántica de un patrón de workflow *Multi-Choice* y un patrón de workflow *Synchronizing Merge*. Por lo tanto, la regla transforma un segmento de flujo de control *Or* en dos gateways `BPMN!InclusiveGateway` (Figura 4.8(b)).

El primer gateway `BPMN!InclusiveGateway` permite representar que uno o varios flujos alternativos pueden seleccionarse de acuerdo a la evaluación de datos del proceso, como lo indica el patrón de workflow *Multi-Choice*. A este gateway se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!Or.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Or.name`; y al atributo `gatewayDirection` se le asigna el valor

Diverging. Cada camino de interacción del segmento se transforma según la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14).

Luego, los elementos de cada camino se transforman de acuerdo a su tipo, aplicando la regla respectiva.

El segundo gateway `BPMN!InclusiveGateway` permite sincronizar los caminos alternativos, como lo indica el patrón de workflow *Synchronizing Merge*. A este gateway se le asigna el valor *Converging* a su atributo `gatewayDirection`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.8: Representación gráfica de la regla *cfsOrSynchronizingMerge2InclusiveGateway*.

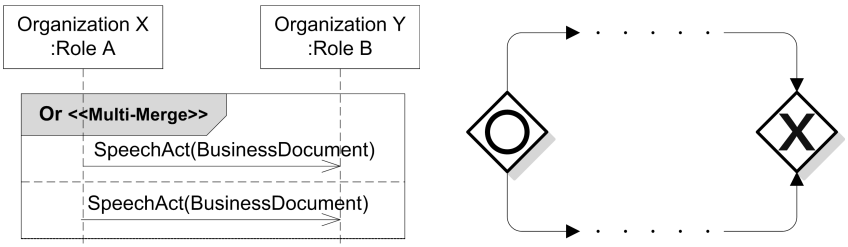
Regla cfsOrMultiMerge2InclusiveGateway

Un segmento de flujo de control `UPCOLBPIP!Or`, cuyo atributo `synchronizationType` es <<*Multi-Merge*>> (Figura 4.9(a)), hace referencia a la semántica de un patrón de workflow *Multi-Choice* y un patrón de workflow *Multi-Merge*. Por lo tanto, la regla transforma este segmento de flujo de control en dos gateways, un gateway inclusivo `BPMN!InclusiveGateway` y un gateway exclusivo `BPMN!ExclusiveGateway` (Figura 4.9(b)).

El gateway inclusivo `BPMN!InclusiveGateway` permite representar el patrón de workflow *Multi-Choice*. A este gateway se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!Or.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Or.name`; y al atributo `gatewayDirection` se le asigna el valor *Diverging*. Cada camino de interacción del segmento se transforma según la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14).

Luego, los elementos de cada camino se transforman de acuerdo a su tipo, aplicando la regla respectiva.

El gateway `BPMN!ExclusiveGateway` permite converger dos o más flujos alternativos, como lo indica el patrón de workflow *Multi-Merge*. A este gateway se le asigna el valor *Converging* a su atributo `gatewayDirection`, para converger los caminos que no poseen una terminación explícita.



(a) Patrón origen. (b) Patrón destino.

Figura 4.9: Representación gráfica de la regla *cfsOrMultiMerge2InclusiveGateway*.

Regla cfsOrNoutM2ComplexGateway

Un segmento de flujo de control `UPCOLBPIP!Or`, cuyo atributo `synchronizationType` es $<< NoutofM >>$ (Figura 4.10(a)), hace referencia a la semántica de un patrón de workflow *Multi-Choice* y un patrón de workflow *Structured Partial Join*. Aunque el lenguaje BPMN provee el constructor `BPMN!InclusiveGateway` para representar el patrón *Multi-Choice*, no provee un constructor explícito para el patrón *Structured Partial Join*, por lo cual se usa un gateway complejo `BPMN!ComplexGateway`, al cual se le asigna la semántica requerida por dicho patrón de workflow, debido a que en BPMN este tipo de gateway no tiene un comportamiento predefinido, sino que es el usuario quien le asigna un comportamiento. El patrón de workflow *Structured Partial Join* indica la sincronización de varios caminos alternativos y/o en paralelos, en donde de M caminos posibles se requiere la finalización de N caminos para continuar.

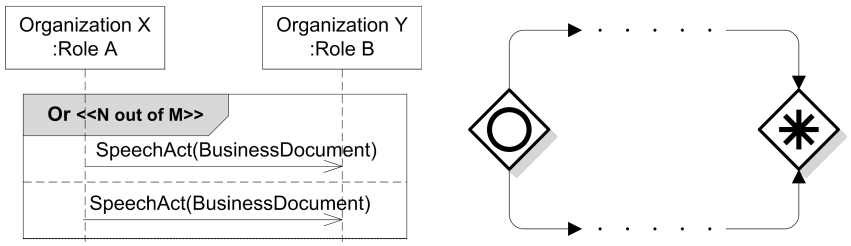
Por lo tanto, la regla transforma este segmento de flujo de control en dos gateways, un gateway inclusivo `BPMN!InclusiveGateway` y un gateway complejo `BPMN!ComplexGateway` (Figura 4.10(b)).

Al gateway inclusivo `BPMN!InclusiveGateway` se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!Or.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Or.name`; y al atributo `gatewayDirection` se le asigna

el valor *Diverging*. Cada camino de interacción del segmento se transforma según la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14).

Luego, los elementos de cada camino se transforman de acuerdo a su tipo, aplicando la regla respectiva.

Al gateway complejo `BPMN!ComplexGateway` se le asigna la semántica que representa el patrón de workflow *Structured Partial Join*. A este gateway se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!Or.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Or.name`; al atributo `gatewayDirection` se le asigna el valor *Converging*; y al atributo `activationExpression` se le asigna el valor *N out of M*, donde los valores de *N* y *M* se obtienen del modelo de entrada.



(a) Patrón origen. (b) Patrón destino.

Figura 4.10: Representación gráfica de la regla *cfsOrNoutM2ComplexGateway*.

Regla cfsLoopUntil2SubProcess

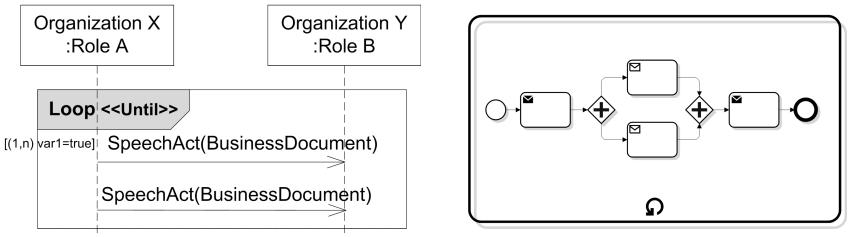
Esta regla define que un segmento de flujo de control `UPCOLBPIP!Loop`, cuyo atributo `loopType` es *Until* (Figura 4.11(a)), se transforma en un subproceso expandido `BPMN!SubProcess` (Figura 4.11(b)) con un marcador *loop* para indicar que todos los elementos contenidos en el subproceso se repiten secuencialmente. A dicho subproceso se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!Loop.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!Loop.name`; al atributo `triggeredByEvent` se le asigna el valor *false*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false*; y al atributo `startQuantity` se le asigna el valor *1*.

Los atributos `StandardLoopCharacteristics` se establecen como sigue: al atributo `LoopCondition` se le asigna el valor `not`

UPCOLBPIP!Loop.condition; al atributo loopMaximum se le asigna el valor UPCOLBPIP!Loop.iterationMaximum; y al atributo testBefore se le asigna el valor *false*.

El atributo LoopCondition es una expresión booleana que controla el bucle. El atributo loopMaximum sirve como un límite en el número de iteraciones. El atributo testBefore controla si la condición del bucle se evalúa al inicio (testBefore = true) o al final (testBefore = false) de la iteración del bucle.

Al subproceso se le agrega un evento de inicio BPMN!StartEvent sin una definición de evento BPMN!EventDefiniton asociada para representar el inicio del subproceso. Todos los elementos contenidos en el segmento de flujo de control UPCOLBPIP!Loop se transforman de acuerdo a su tipo, aplicando la regla respectiva. Luego, se agrega un evento de fin BPMN!EndEvent sin una definición de evento BPMN!EventDefiniton asociada para representar el fin del subproceso.



(a) Patrón origen. (b) Patrón destino.

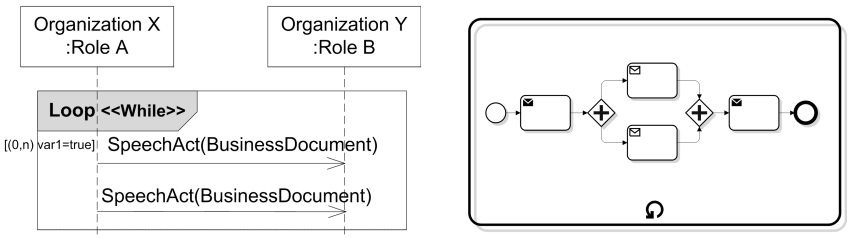
Figura 4.11: Representación gráfica de la regla *cfsLoopUntil2SubProcess*.

Regla cfsLoopWhile2SubProcess

Cada segmento de flujo de control UPCOLBPIP!Loop, cuyo atributo loopType es *While* (Figura 4.12(a)), se transforma en un subproceso expandido BPMN!SubProcess (Figura 4.12(b)). A dicho subproceso se le asignan los siguientes valores a sus atributos: al atributo id se le asigna el valor UPCOLBPIP!Loop.id; al atributo name se le asigna el valor UPCOLBPIP!Loop.name; al atributo triggeredByEvent se le asigna el valor *false*; al atributo completionQuantity se le asigna el valor *1*; al atributo isForCompensation se le asigna el valor *false*; y al atributo startQuantity se le asigna el valor *1*.

Los atributos `StandardLoopCharacteristics` se configuran como sigue: al atributo `LoopCondition` se le asigna el valor `UPCOLBPIP!Loop.condition`; al atributo `loopMaximum` se le asigna el valor `UPCOLBPIP!Loop.iterationMaximum`; y al atributo `testBefore` se le asigna el valor *true*.

Al subproceso se le agrega un evento de inicio `BPMN!StartEvent` sin una definición de evento `BPMN!EventDefiniton` asociada para representar el inicio del subproceso. Todos los elementos contenidos en el segmento de flujo de control `UPCOLBPIP!Loop` se transforman de acuerdo a su tipo, aplicando la regla respectiva. Luego, se agrega un evento de fin `BPMN!EndEvent` sin una definición de evento `BPMN!EventDefiniton` asociada para representar el fin del subproceso.



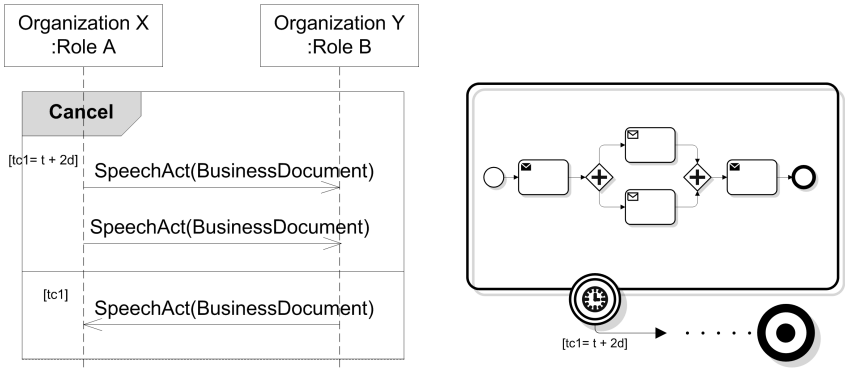
(a) Patrón origen. (b) Patrón destino.

Figura 4.12: Representación gráfica de la regla *cfsLoopWhile2SubProcess*.

Regla cfsCancel2SubProcess

Esta regla define que un segmento de flujo de control `UPCOLBPIP!Cancel` (Figura 4.13(a)) se transforma como sigue: el camino de interacción `UPCOLBPIP!InteractionPath` que encierra el alcance de la excepción (lógica o de tiempo) se transforma en un subproceso expandido `BPMN!SubProcess` (Figura 4.13(b)). A dicho subproceso se le agrega un evento de inicio `BPMN!StartEvent` sin una definición de evento `BPMN!EventDefiniton` asociada para representar el inicio del subproceso. Los elementos que se encuentran dentro de dicho camino de interacción `UPCOLBPIP!InteractionPath` se transforman de acuerdo a su tipo, aplicando la regla respectiva. Luego, se agrega un evento de fin `BPMN!EndEvent` sin una definición de evento `BPMN!EventDefiniton` asociada para representar el fin del subproceso.

Cada excepción se transforma en un `BPMN!BoundaryEvent` cuya definición `BPMN!EventDefinition` es de tipo `BPMN!TimerEventDefinition` para una excepción de tiempo o `BPMN!ConditionalEventDefinition` para una excepción lógica. A su atributo `cancelActivity` se le asigna el valor `true`. Los elementos del proceso colaborativo que se encuentran dentro de los caminos de interacción `UPCOLBPIP!InteractionPath` que manejan las excepciones se transforman de acuerdo a su tipo, aplicando la regla respectiva y se conectan al flujo de secuencia de salida del `BPMN!BoundaryEvent`. Finalmente, a cada camino se agrega un evento `BPMN!EndEvent` cuya definición `BPMN!EventDefinition` es de tipo `BPMN!TerminateEventDefiniton` para finalizar la ejecución del proceso colaborativo.



(a) Patrón origen. (b) Patrón destino.

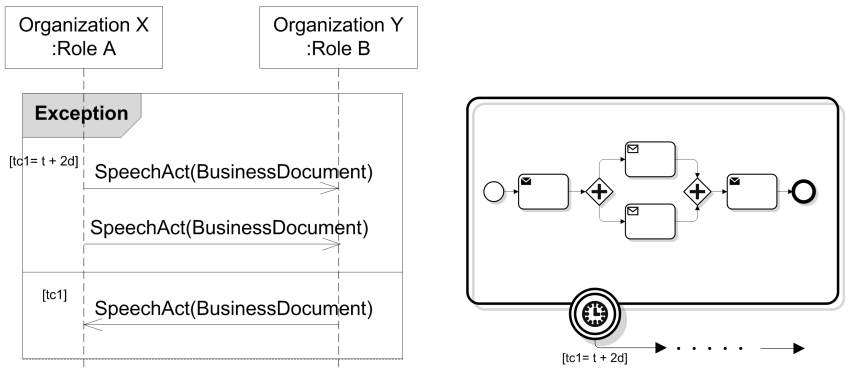
Figura 4.13: Representación gráfica de la regla *cfsCancel2SubProcess*.

Regla cfsException2SubProcess

Esta regla define que un segmento de flujo de control `UPCOLBPIP!Exception` (Figura 4.14(a)) se transforma como sigue: el camino de interacción `UPCOLBPIP!InteractionPath` que encierra el alcance de la excepción (lógica o de tiempo) se transforma en un subproceso expandido `BPMN!SubProcess` (Figura 4.14(b)). A dicho subproceso se le agrega un evento de inicio `BPMN!StartEvent` sin una definición de evento `BPMN!EventDefiniton` asociada para representar el inicio del subproceso. Los elementos que se encuentran dentro de dicho camino de interacción `UPCOLBPIP!InteractionPath` se transforman de acuerdo a su tipo, aplicando la regla respectiva. Luego, se agrega un evento de fin `BPMN!EndEvent` sin

una definición de evento `BPMN!EventDefiniton` asociada para representar el fin del subproceso.

Cada excepción se transforma en un `BPMN!BoundaryEvent` cuya definición `BPMN!EventDefinition` es de tipo `BPMN!TimerEventDefinition` para una excepción de tiempo o `BPMN!ConditionalEventDefinition` para una excepción lógica. A su atributo `cancelActivity` se le asigna el valor *true*. Los elementos del proceso colaborativo que se encuentran dentro de los caminos de interacción `UPCOLBPIP!InteractionPath` que manejan las excepciones se transforman de acuerdo a su tipo, aplicando la regla respectiva, y se conectan al flujo de secuencia de salida del `BPMN!BoundaryEvent` para finalizar la ejecución del subproceso y continuar el flujo del proceso principal.



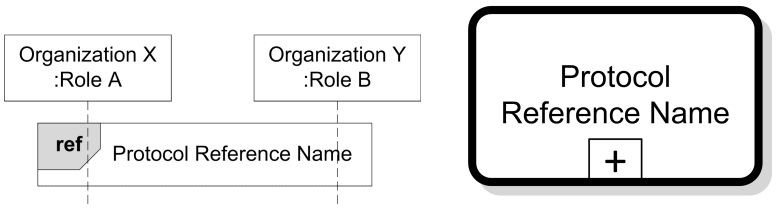
(a) Patrón origen. (b) Patrón destino.

Figura 4.14: Representación gráfica de la regla *cfsException2SubProcess*.

4.1.1.8. Regla protocolReference2CallActivity

Esta regla define que una referencia de protocolo `UPCOLBPIP!ProtocolReference` (Figura 4.15(a)) se transforma en una actividad de llamada `BPMN!CallActivity` (Figura 4.15(b)), debido a que estos constructores representan la invocación a un subprotocolo o subproceso respectivamente. A la actividad `BPMN!CallActivity` se le asignan los siguientes valores a sus atributos: al atributo `name` se le asigna el valor `UPCOLBPIP!ProtocolReference.name`; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false*; al atributo `startQuantity` se le asigna el valor *1*; y al atributo `calledElementRef` se le asigna el valor

UPCOLBPIP!ProtocolReference.protocol.

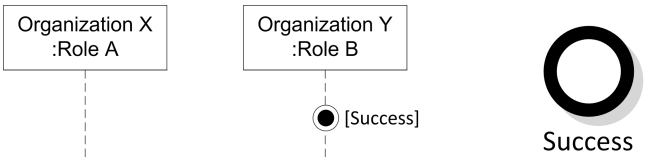


(a) Patrón origen. (b) Patrón destino.

Figura 4.15: Representación gráfica de la regla *protocolReference2CallActivity*.

4.1.1.9. Regla successExplicitTermination2EndEvent

Esta regla define que una terminación explícita UPCOLBPIP!Termination (Figura 4.16(a)), cuyo UPCOLBPIP!Termination.terminationElement es *Success*, se transforma en un evento de fin BPMN!EndEvent sin una definición de evento BPMN!EventDefiniton asociada (Figura 4.16(b)), para indicar la finalización exitosa del proceso de interfaz. A su atributo name se le asigna el valor UPCOLBPIP!Termination.name.

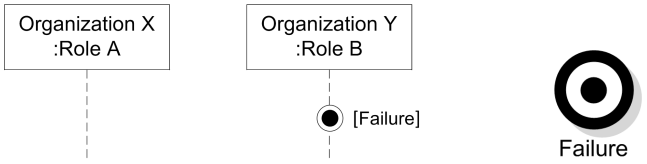


(a) Patrón origen. (b) Patrón destino.

Figura 4.16: Representación gráfica de la regla *successExplicitTermination2EndEvent*.

4.1.1.10. Regla failureExplicitTermination2EndEvent

Esta regla define que una terminación explícita UPCOLBPIP!Termination (Figura 4.17(a)), cuyo UPCOLBPIP!Termination.terminationElement es *Failure*, se transforma en un evento de fin BPMN!EndEvent con una definición de evento BPMN!EventDefiniton de tipo BPMN!TerminateEventDefinition (Figura 4.17(b)), para indicar que el proceso de interfaz finalizó con una falla. A su atributo name se le asigna el valor UPCOLBPIP!Termination.name.



(a) Patrón origen. (b) Patrón destino.

Figura 4.17: Representación gráfica de la regla *failureExplicitTermination2EndEvent*.

4.1.1.11. Regla implicitTermination2EndEvent

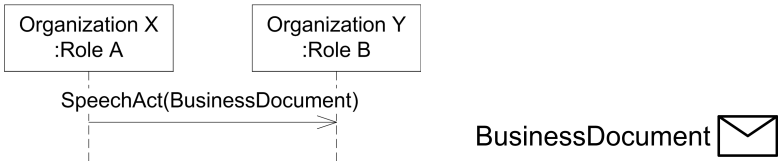
Esta regla define que un evento de fin `BPMN!EndEvent` sin una definición de evento `BPMN!EventDefiniton` asociada se agrega al proceso de interfaz para representar la terminación implícita del proceso colaborativo. A su atributo `name` se le asigna el valor *Success* para representar la terminación exitosa del proceso.

4.1.1.12. Regla condition2Expression

Esta regla define que una condición `UPCOLBPIP!Condition` se transforma en una expresión `BPMN!Expression`. A su atributo `id` se le asigna el valor `UPCOLBPIP!Condition.ConditionExpression`. Las condiciones son utilizadas en segmentos de flujos de control alternativos.

4.1.1.13. Regla businessDocument2Message

Esta regla define que un documento de negocio `UPCOLBPIP!BusinessDocument` (Figura 4.18(a)) se transforma en un mensaje `BPMN!Message` (Figura 4.18(b)), para representar la información intercambiada entre los participantes. A su atributo `name` se le asigna el valor `UPCOLBPIP!BusinessDocument.name`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.18: Representación gráfica de la regla *businessDocument2Message*.

4.1.1.14. Regla interactionPath2sequenceFlow

Para un camino de interacción `UPCOLBPIP!InteractionPath` esta regla indica que se debe generar un flujo de secuencia `BPMN!SequenceFlow`. A dicho flujo de secuencia se le asignan los siguientes valores a sus atributos: al atributo `id` se le asigna el valor `UPCOLBPIP!InteractionPath.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!InteractionPath.guard` o en caso que el guard no exista se asigna el nombre del acto de comunicación (`UPCOLBPIP!SpeechAct.name`) del primer mensaje del camino; al atributo `isInmediate` se le asigna el valor `true`; al atributo `targetRef` se le asocia el primer elemento encontrado en el camino de interacción respectivo; y a su atributo `conditionExpression` se le asigna el valor `UPCOLBPIP!InteractionPath.guard`, si existe.

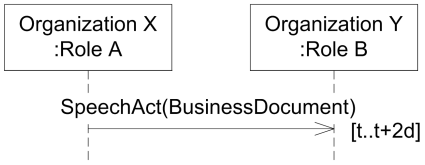
4.1.1.15. Regla timeConstraint2Timer

Una restricción de tiempo `UPCOLBPIP!TimeConstraint` se transforma de acuerdo al tipo del elemento del proceso colaborativo al cual está asociado:

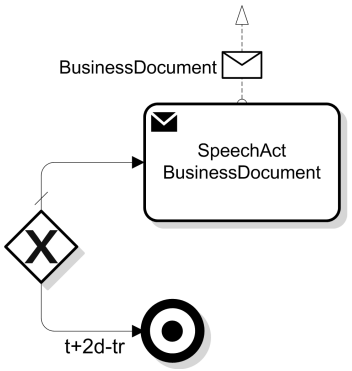
- Si la misma está asociada a un mensaje de negocio `UPCOLBPIP!BusinessMessage` (Figura 4.19(a)) se transforma de acuerdo a si el rol para el que se genera el proceso de interfaz es quien envía o recibe el mensaje. Para el rol emisor, se transforma en un gateway `BPMN!ExclusiveGateway` (Figura 4.19(b)) a cuyo atributo `gatewayDirection` se le asigna el valor `Diverging`. A dicho gateway se le asignan dos flujos de secuencia de salida `BPMN!SequenceFlow`, uno para el mensaje a enviarse `UPCOLBPIP!BusinessMessage`, el cual se transforma según la regla *sendBusinessMessage2SendTask* (Sección 4.1.1.6), este es el camino por defecto; y el otro para terminar la ejecución del proceso en caso que se alcance la restricción de tiempo `UPCOLBPIP!TimeConstraint`. En este caso, `BPMN!SequenceFlow` tiene asignada una condición que verifica la expresión de tiempo y se conecta a un evento `BPMN!EndEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TerminateEventDefinition`.

  Por otro lado, para el rol receptor, el mensaje recibido

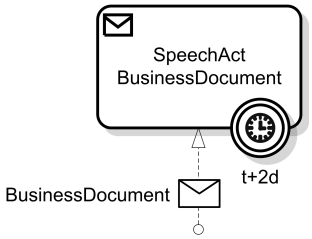
`UPCOLBPIP!BusinessMessage` se transforma de acuerdo a la regla *receiveBusinessMessage2ReceiveTask* (Sección 4.1.1.6). La restricción de tiempo `UPCOLBPIP!TimeConstraint` se transforma en un evento `BPMN!BoundaryEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TimerEventDefinition` (Figura 4.19(c)), la cual se adjunta a los límites de la tarea. A su atributo `cancelActivity` se le asigna el valor `true` y a su atributo `timeDate` se le asigna el valor `UPCOLBPIP!TimeConstraint.endTime`. El flujo de secuencia de salida del evento `BPMN!BoundaryEvent` se conecta a un evento de fin `BPMN!EndEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TerminateEventDefinition`.



(a) Patrón origen.



(b) Patrón destino del rol emisor.

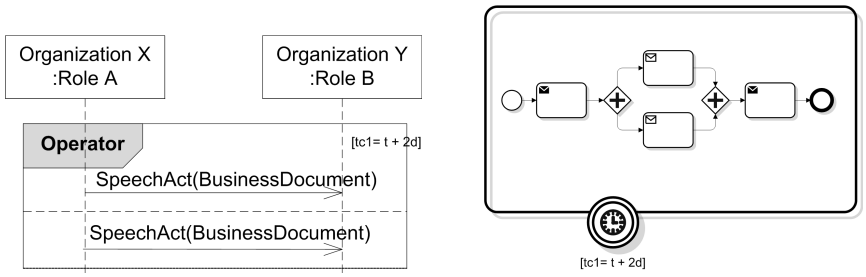


(c) Patrón destino del rol receptor.

Figura 4.19: Representación gráfica de la regla *timeConstraint2Timer* aplicada a un mensaje de negocio.

- Si la restricción de tiempo está asociada a un segmento de flujo de control `UPCOLBPIP!And`, `UPCOLBPIP!Xor` y `UPCOLBPIP!Or` (<<Multi-Merge>> y << $NoutofM$ >>) (Figura 4.20(a)), se agrega un subproceso expandido `BPMN!SubProcess`. Dentro del subproceso, el segmento de flujo de control se transforma de acuerdo a su regla

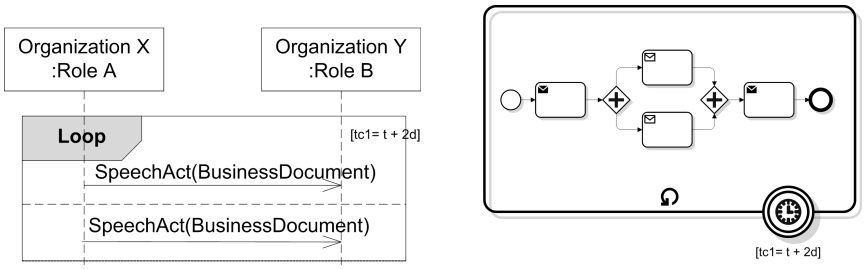
respectiva. La restricción de tiempo `UPCOLBPIP!TimeConstraint` se transforma en un evento `BPMN!BoundaryEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TimerEventDefinition`, la cual se adjunta a los límites del subproceso (Figura 4.20(b)). A su atributo `cancelActivity` se le asigna el valor *true* y a su atributo `timeDate` se le asigna el valor `UPCOLBPIP!TimeConstraint.endTime`. El flujo de secuencia de salida del evento `BPMN!BoundaryEvent` se conecta a un evento de fin `BPMN!EndEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TerminateEventDefinition`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.20: Representación gráfica de la regla *timeConstraint2Timer* aplicada a un segmento de flujo de control *And*, *Xor* y *Or*.

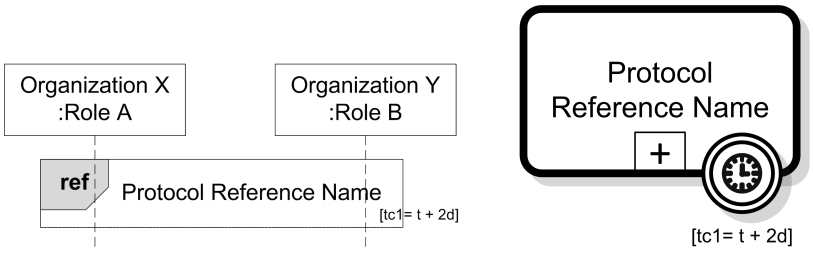
- Si la restricción de tiempo está asociada a un segmento de flujo de control `UPCOLBPIP!Loop` *Until* o *While* (Figura 4.21(a)), primero éste se transforma según la regla *cfsLoopUntil2SubProcess* (Sección 4.1.1.7) o *cfsLoopWhile2SubProcess* (Sección 4.1.1.7) respectivamente. Luego, la restricción de tiempo `UPCOLBPIP!TimeConstraint` se transforma en un evento `BPMN!BoundaryEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TimerEventDefinition`, la cual se adjunta a los límites del subproceso (Figura 4.21(b)). A su atributo `cancelActivity` se le asigna el valor *true* y a su atributo `timeDate` se le asigna el valor `UPCOLBPIP!TimeConstraint.endTime`. El flujo de secuencia de salida del evento `BPMN!BoundaryEvent` se conecta a un evento de fin `BPMN!EndEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TerminateEventDefinition`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.21: Representación gráfica de la regla *timeConstraint2Timer* aplicada a un segmento de flujo de control *Loop*.

- Si la restricción de tiempo está asociada a una referencia de protocolo `UPCOLBPIP!ProtocolReference` (Figura 4.22(a)), primero éste se transforma según la regla *protocolReference2CallActivity* (Sección 4.1.1.8). Luego, la restricción de tiempo `UPCOLBPIP!TimeConstraint` se transforma en un evento `BPMN!BoundaryEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TimerEventDefinition`, la cual se adjunta a los límites de la tarea (Figura 4.22(b)). A su atributo `cancelActivity` se le asigna el valor *true* y a su atributo `timeDate` se le asigna el valor `UPCOLBPIP!TimeConstraint.endTime`. El flujo de secuencia de salida del evento `BPMN!BoundaryEvent` se conecta a un evento de fin `BPMN!EndEvent` con una definición de evento `BPMN!EventDefiniton` de tipo `BPMN!TerminateEventDefinition`.



(a) Patrón origen. (b) Patrón destino.

Figura 4.22: Representación gráfica de la regla *timeConstraint2Timer* aplicada a una referencia de protocolo.

4.1.1.16. Regla successorSequenceFlow

Esta regla genera los flujos de secuencias en el modelo de salida que permiten vincular los elementos generados por los patrones destino de las reglas descritas anteriormente, de acuerdo al orden de los elementos del modelo de entrada definido por el atributo `successor` de los mismos. Mediante esta regla, se genera un `BPMN!SequenceFlow` que une el último elemento generado por un patrón destino de una regla con el primer elemento generado por otro patrón destino de otra regla, de acuerdo a la secuencia definida en el modelo de entrada (Figura 4.23).

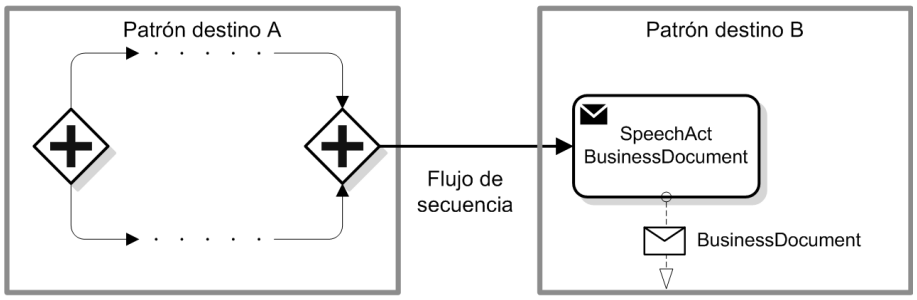


Figura 4.23: Representación gráfica de la regla *successorSequenceFlow*.

4.2. Implementación del método

El prototipo del método fue implementado como un plug-in de Eclipse (Carlson, 2005), usando Eclipse Modeling Framework (EMF) (Steinberg y otros, 2008), una herramienta de modelado conceptual y generador de código para crear herramientas y otras aplicaciones basadas en un modelo de datos estructurado; y ADT (Jouault y otros, 2008), un plug-in de Eclipse compuesto por un motor de transformación ATL y un ambiente de desarrollo integrado. Los meta-modelos usados se definieron conforme al meta-meta-modelo Ecore de EMF y la definición de transformación se implementó usando el lenguaje ATL.

Para facilitar la comprensión del funcionamiento del prototipo, se muestra el escenario de uso en el que puede emplearse (Figura 4.24): (1) el usuario selecciona el modelo de proceso colaborativo a transformar; (2) el prototipo solicita al usuario el rol a ser considerado durante la transformación; (3) el usuario selecciona el rol; (4) el prototipo genera el modelo de parámetros que contiene el rol

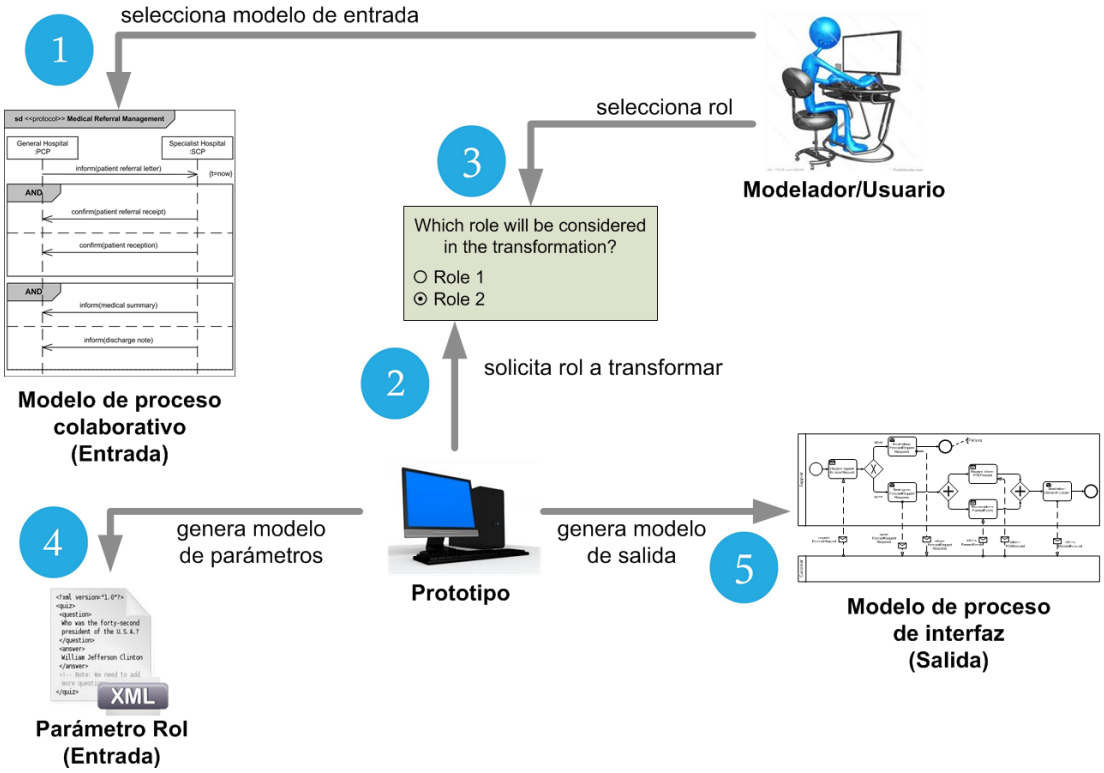


Figura 4.24: Escenario de uso del prototipo que implementa el método propuesto.

a usarse durante la transformación; finalmente, (5) el prototipo genera el modelo de proceso de interfaz para el rol seleccionado.

En la implementación de la definición de transformación del método se aplicó una técnica de composición denominada *superimposición* (Wagelaar y otros, 2010), que permite la reutilización, mantenimiento y escalabilidad de definiciones de transformación de modelos, mediante la división de dichas definiciones en varias de tamaño y alcance manejable, que luego podrán usarse en la composición de una nueva definición de transformación (Capítulo 2). Aplicando dicha técnica, se definieron dos definiciones de transformación (módulos ATL): (a) protocol2interface.atl, que contiene las reglas definidas para transformar mensajes de negocio (Apéndice B.2); (b) upcolbpip2bpmn.atl, que contiene las restantes reglas definidas para transformar los demás elementos de los modelos de procesos colaborativos (Apéndice B.1). De esta manera, la definición de transformación upcolbpip2bpmn.atl puede ser utilizada por el método presentado en este capítulo y reutilizada por el método basado en MDA para generar modelos de procesos de integración presentado en el Capítulo 5.

También se aplicaron técnicas de *refactorización* (Wimmer y otros, 2012)

(Capítulo 2) a ambas definiciones de transformación para mejorar la estructura de las mismas, pero preservando su comportamiento, y de esta manera mejorar la calidad de las definiciones y de los modelos obtenidos.

El Listado 4.1 muestra la definición de transformación protocol2interface.atl (línea 7) definida para transformar los mensajes de negocio. Dicha definición de transformación usa el meta-modelo upcolbpip.ecore (línea 3) del lenguaje UP-ColBPIP propuesto por Villarreal y otros (2010), el meta-modelo BPMN20.ecore (línea 4) del lenguaje BPMN propuesto por The Eclipse Foundation (2011) y el meta-modelo parameters.ecore (línea 5) propuesto como parte de la tesis. Estos tres meta-modelos con extensión .ecore están basados y se construyeron con EMF.

La definición protocol2interface.atl además usa las librerías speechActs.atl y parameterHelpers.atl (Capítulo 5). La librería speechActs.atl (línea 10) contiene los "helpers" o funciones (Capítulo 2) que permiten obtener la intención de cada mensaje de negocio indicada por el acto de comunicación asociado al mensaje. La librería parameterHelpers.atl (línea 11) contiene los helpers o funciones que permiten consultar los parámetros definidos en el modelo de parámetros.

Listado 4.1: Definición de transformación protocol2interface.atl

```
1 -- @description  Transforming collaborative processes into interface processes
2 -- @atlcompiler atl2010
3 -- @path MMupcolbpip=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/upcolbpip.
      ecore
4 -- @path MMbpmn2=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/BPMN20.ecore
5 -- @path MMParam=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/parameters.
      ecore
6
7 module protocol2interface;
8 create OUT: MMbpmn2 from IN: MMupcolbpip, INParam: MMParam;
9
10 uses speechActs;
11 uses parameterHelpers;
12
13 ------------------------------------------------------------------------
14 ----------- Transforming business messages sent -------------------------
15 ------------------------------------------------------------------------
16 rule sendBusinessMessage2SendTask {
17     from
18     businessMessage: MMupcolbpip!BusinessMessage (
19       businessMessage.isSenderRequiredRole(thisModule.getParameterValue('roleId',
20           'role'))
21     )
22   to
23     sendTask: MMbpmn2!SendTask (
```

```
24      id <- businessMessage.id,
25      name <- businessMessage.intention.toString() + ' ' + businessMessage.
26          information.name,
27      implementation <- '##WebService',
28      completionQuantity <- 1,
29      isForCompensation <- false,
30      startQuantity <- 1,
31      messageRef <- businessMessage.information,
32      outgoing <- if thisModule.
33          getAllipElementsForSubprocess() -> includes(businessMessage) then
34          thisModule.successorSequenceFlowForSubprocess(businessMessage)
35        else
36          thisModule.successorSequenceFlowForProcess(businessMessage)
37        endif
38    )
39 ------------------------------------------------------------------------
40 ------------- Transforming business messages sent ----------------------
41 ------------- with time constraint -------------------------------------
42 ------------------------------------------------------------------------
43 rule sendBusinessMessageWithTimeConstraint2SendTask extends
       sendBusinessMessage2SendTask{
44   from
45     businessMessage: MMupcolbpip!BusinessMessage (
46       businessMessage.hasTimeConstraint()
47     )
48   to
49     gatewayDiverging: MMbpmn2!ExclusiveGateway (
50       id <- 'ExclusiveGateway_' + businessMessage.id,
51       gatewayDirection <- #Diverging,
52       default <- internalSequenceFlow
53     ),
54     internalSequenceFlow: MMbpmn2!SequenceFlow (
55       id <- 'internalSequenceFlow_' + businessMessage.id,
56       isImmediate <- true,
57       sourceRef <- gatewayDiverging,
58       targetRef <- sendTask
59     ),
60     sendTask: MMbpmn2!SendTask,
61     timerSequenceFlow: MMbpmn2!SequenceFlow (
62       id <- 'timerSequenceFlow_' + businessMessage.id,
63       isImmediate <- true,
64       sourceRef <- gatewayDiverging,
65       name <- businessMessage.getTimeConstraint() + '-TR',
66       targetRef <- terminateEvent
67     ),
68     terminateEvent: MMbpmn2!EndEvent (
69       id <- 'terminateEvent_' + businessMessage.id,
70       eventDefinitions <- eventDefinition
71     ),
72     eventDefinition: MMbpmn2!TerminateEventDefinition (
73       id <- 'eventDefinition_' + businessMessage.id
```

```
74     )
75
76 }
77 ------------------------------------------------------------------------
78 ----------- Transforming business messages received --------------------
79 ------------------------------------------------------------------------
80 rule receiveBusinessMessage2ReceiveTask {
81   from
82     businessMessage: MMupcolbpip!BusinessMessage (
83       businessMessage.isReceiverRequiredRole(thisModule.getParameterValue('roleId
             ',
84           'role'))
85     )
86   to
87     receiveTask: MMbpmn2!ReceiveTask (
88       id <- businessMessage.id,
89       name <- businessMessage.intention.toString() + ' ' + businessMessage.
90           information.name,
91       implementation <- '##WebService',
92       completionQuantity <- 1,
93       isForCompensation <- false,
94       startQuantity <- 1,
95       messageRef <- businessMessage.information,
96       outgoing <- if thisModule.getAllipElementsForSubprocess() ->
97           includes(businessMessage) then
98           thisModule.successorSequenceFlowForSubprocess(businessMessage)
99         else
100          thisModule.successorSequenceFlowForProcess(businessMessage)
101        endif
102    )
103 }
```

La línea 16 del Listado 4.1 muestra la implementación de la regla *sendBusinessMessage2SendTask* descrita en la Sección 4.1.1.6. Dicha regla usa el "helper" *isSenderRequiredRole* para determinar si el mensaje es enviado por la organización desempeñando el rol indicado en el modelo de parámetros. La línea 80 muestra la implementación de la regla *receiveBusinessMessage2ReceiveTask* descrita en la Sección 4.1.1.6. Dicha regla usa el "helper" *isReceiverRequiredRole* para determinar si el mensaje es recibido por la organización desempeñando el rol indicado en el modelo de parámetros. En la tesis, estos tipos de reglas se denominan *reglas parametrizadas*. Una regla parametrizada es aquella que tiene una expresión booleana ("guard") que usa parámetros externos definidos en un modelo de parámetros conforme a un meta-modelo. Dichas reglas se disparan automáticamente cuando se cumple su expresión booleana, lo cual las diferencia de las reglas "called" (Jouault y otros, 2008; Kurtev y otros, 2007; Wagelaar y

otros, 2010), que son un tipo especial de reglas que pueden usar parámetros, pero que deben ser explícitamente invocadas (Capítulo 2).

El Listado 4.2 muestra un fragmento de la definición de transformación upcolbpip2bpmn.atl. La línea 5 muestra la implementación de la regla *UPColBPIPModel2Definitions* descrita en la Sección 4.1.1.1. La línea 28 muestra la implementación de la regla *tradingPartner2Participant* descrita en la Sección 4.1.1.4. Debido a que los segmentos de flujo de control *And*, *Xor* y *Or* (reglas *cfsAnd2ParallelGateway*, *cfsXor2ExclusiveGateway*, *cfsOrSynchronizingMerge2InclusiveGateway*, *cfsOrMultiMerge2InclusiveGateway* y *cfsOrNoutM2ComplexGateway*, descritas en la Sección 4.1.1.7) se transforman de manera similar, se implementó una regla abstracta, denominada *cfs2Gateway* (línea 52), la cual luego es especializada para cada uno de los segmentos de flujo de control mencionados. En la línea 69 se muestra una especialización de la regla, denominada *cfsAnd2ParallelGateway*, para el segmento de flujo de control *And*. Esto se hizo aplicando la refactorización *Extract Superrule out of Matched Rules* del catálogo de refactorizaciones propuesto por Wimmer y otros (2012). Esta refactorización se puede aplicar cuando dos o más reglas contienen una funcionalidad similar y comparten supertipos comunes para sus elementos del patrón origen/destino.

Listado 4.2: Fragmento de la definición de transformación upcolbpip2bpmn.atl

```
1 ...
2 ------------------------------------------------------------------------------
3 -------------------------- UPColBPIPModel to Definitions -------------------
4 ------------------------------------------------------------------------------
5 rule UPColBPIPModel2Definitions {
6   from
7     upcolbpipModel: MMupcolbpip!UPColBPIPModel
8   to
9     definitions: MMbpmn2!Definitions (
10      id <- 'definitionsId',
11      targetNamespace <- 'http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2',
12      name <- upcolbpipModel.name,
13      exporter <- 'Eclipse',
14      exporterVersion <- '3.6',
15      rootElements <- Sequence{thisModule.getCollaborativeProcess(),
16            thisModule.getB2BCollaboration(),
17            thisModule.getAllPartners(),
18            thisModule.getAllRoles(),
19            thisModule.getAllPartners() -> collect(i | thisModule.
20                resolveTemp(i, 'partnerEntity')),
```

```
21            thisModule.getAllDocuments()}
22    )
23 }
24 ...
25 ------------------------------------------------------------------------
26 -------------------------- TradingPartner to Participant ----------------
27 ------------------------------------------------------------------------
28 rule tradingPartner2Participant {
29   from
30     tradingPartner: MMupcolbpip!TradingPartner
31   to
32     participant: MMbpmn2!Participant (
33       id <- tradingPartner.id,
34       name <- tradingPartner.name,
35       processRef <- if tradingPartner.performsRole(thisModule.getParameterValue('
             roleId','role')) then
36           thisModule.getCollaborativeProcess()
37         else
38           OclUndefined
39         endif
40     ),
41     partnerEntity: MMbpmn2!PartnerEntity (
42       id <- 'partnerEntity' + tradingPartner.id,
43       name <- tradingPartner.name,
44       participantRef <- participant
45     )
46 }
47
48 ...
49 ------------------------------------------------------------------------------
50 ------------ abstract rule used for transforming the cfs -------------------
51 ------------------------------------------------------------------------------
52 abstract rule cfs2Gateway {
53   from
54     cfs: MMupcolbpip!ControlFlowSegment
55   to
56     gateway: MMbpmn2!Gateway (
57       id <- cfs.id,
58       name <- cfs.name,
59       gatewayDirection <- #Diverging,
60       outgoing <- cfs.interactionPath -> collect(i | thisModule.resolveTemp(i,
61           'sequenceFlows'))
62     )
63 }
64
65 ------------------------------------------------------------
66 --------------- And to ParallelGateway --------------------
67 --------------- without time constraint -------------------
68 ------------------------------------------------------------
69 rule cfsAnd2ParallelGateway extends cfs2Gateway {
70   from
```

```
71     cfs: MMupcolbpip!And
72   to
73     gateway: MMbpmn2!ParallelGateway
74 }
75 ...
```

La clase Java denominada Upcolbpip2bpmn.java (Apéndice B.4) del plug-in es la que permite lanzar el motor de transformación ATL que ejecuta automáticamente la transformación. Los métodos más importantes de esta clase son *loadModels*, *doProtocol2interface*, *saveModels*. El método *loadModels* se encarga de cargar los modelos de entrada (modelo de proceso colaborativo y el modelo de parámetros) e inicializa el modelo de salida (modelo de proceso de interfaz).

El método *doUpcolbpip2bpmn* (Listado 4.3) se encarga de transformar el modelo de proceso colaborativo en un modelo de proceso de interfaz, ejecutando las definiciones de transformaciones. Como puede verse la línea 2, la interfaz *ILauncher* define un lanzador de transformación. El método *initialize* del lanzador (línea 5) inicializa el lanzador con las opciones especificadas en *launcherOptions*. El método *addInModel* del lanzador (línea 6) agrega un modelo de entrada al contexto de transformación. Este método también se utiliza para cargar los metamodelos utilizados en esta transformación. El método *addOutModel* del lanzador (línea 8) agrega un modelo de salida al contexto de la transformación. El método *addLibrary* del lanzador (línea 11) agrega una librería ATL a la transformación. El método *launch* del lanzador (línea 17) inicia la transformación mediante los parámetros especificados y el conjunto determinado de módulos.

El método *saveModels* se encarga de almacenar el modelo de salida (proceso de interfaz) generado por el método *doProtocol2interface*.

Listado 4.3: Método *doUpcolbpip2bpmn* de la clase Upcolbpip2bpmn.java

```
1  public Object doUpcolbpip2bpmn(IProgressMonitor monitor) throws ATLCoreException
       , IOException, ATLExecutionException {
2    ILauncher launcher = new EMFVMLauncher();
3    List<InputStream> inputStreamsToClose = new ArrayList<InputStream>();
4    Map<String, Object> launcherOptions = getOptions();
5    launcher.initialize(launcherOptions);
6    launcher.addInModel(inModel, "IN", "MMupcolbpip");
7    launcher.addInModel(inparamModel, "INParam", "MMParam");
8    launcher.addOutModel(outModel, "OUT", "MMbpmn2");
9    InputStream libraryStream_speechActs = getLibraryAsStream("speechActs");
10   inputStreamsToClose.add(libraryStream_speechActs);
11   launcher.addLibrary("speechActs", getLibraryAsStream("speechActs"));
```

```
12    InputStream libraryStream_parameterHelpers = getLibraryAsStream("
          parameterHelpers");
13    inputStreamsToClose.add(libraryStream_parameterHelpers);
14    launcher.addLibrary("parameterHelpers", getLibraryAsStream("parameterHelpers")
          );
15    InputStream[] modulesStreams = getModulesList();
16    inputStreamsToClose.addAll(Arrays.asList(modulesStreams));
17    Object result = launcher.launch("run", monitor, launcherOptions, (Object[])
          modulesStreams);
18    for (InputStream inputStream : inputStreamsToClose) {
19      inputStream.close();
20    }
21    return result;
22  }
```

En el archivo de definición de propiedades Upcolbpip2bpmn.properties del plug-in (Listado 4.4) se indican los módulos a usarse, la ubicación de los metamodelos y las librerías, y las opciones de lanzamiento del motor ATL. En la línea 6 se indica que se aplicará la técnica superimposición sobre los módulos upcolbpip2bpmn.atl y protocol2interface.atl.

Listado 4.4: Archivo de propiedades Upcolbpip2bpmn.properties del plug-in

```
1 # ================================================================================
2 # Upcolbpip2bpmn properties
3 # ================================================================================
4
5 # ATL modules: if several, by order of superimposition (the latter ones overrides
      the former ones)
6 Upcolbpip2bpmn.modules = upcolbpip2bpmn.atl,protocol2interface.atl
7
8 # Metamodels paths or nsUris
9 Upcolbpip2bpmn.metamodels.MMupcolbpip = resources/metamodels/upcolbpip.ecore
10 Upcolbpip2bpmn.metamodels.MMParam = resources/metamodels/parameters.ecore
11 Upcolbpip2bpmn.metamodels.MMbpmn2 = resources/metamodels/BPMN20.ecore
12
13 # Libraries paths
14 Upcolbpip2bpmn.libraries.speechActs = speechActs.asm
15 Upcolbpip2bpmn.libraries.parameterHelpers = parameterHelpers.asm
16
17 # ATL Launching options
18 Upcolbpip2bpmn.options.supportUML2Stereotypes = false
19 Upcolbpip2bpmn.options.printExecutionTime = true
20 Upcolbpip2bpmn.options.OPTION_CONTENT_TYPE = false
21 Upcolbpip2bpmn.options.allowInterModelReferences = true
22 Upcolbpip2bpmn.options.step = false
```

Debido a que el motor de transformación ATL serializa los modelos BPMN generados en formato XMI, se realiza una transformación auxiliar (Figura 4.25) que transforma los mismos en modelos BPMN en formato XML, de acuerdo al formato de intercambio de archivos del estándar BPMN2 (OMG, 2011a). De esta manera, los modelos obtenidos pueden ser editados con diferentes herramientas de edición de modelos de procesos de negocio que soportan el lenguaje BPMN.

La transformación auxiliar se realiza aplicando plantillas (hojas de estilo) XSLT (W3C, 2007c), el cual es un lenguaje estándar para la trasformación de documentos XML. Esta transformación se realiza aplicando la plantilla XMI2XML.xslt propuesta por Hille-Doering (2010). Aunque el estándar BPMN provee una plantilla XSLT para realizar esta transformación, la misma no es completamente funcional y no cubre todos los elementos y atributos de la especificación (Hille-Doering, 2010).

Finalmente, los modelos BPMN en formato XML, obtenidos en la transformación auxiliar, son validados contra el esquema XSD BPMN20.xsd propuesto por el estándar BPMN2 (OMG, 2011a). El propósito de esta validación es determinar si el modelo está bien formado con respecto al estándar.

La Figura 4.25 muestra y resume la cadena de transformaciones realizadas para generar un modelo de proceso de interfaz bien formado a partir de un modelo de proceso colaborativo.

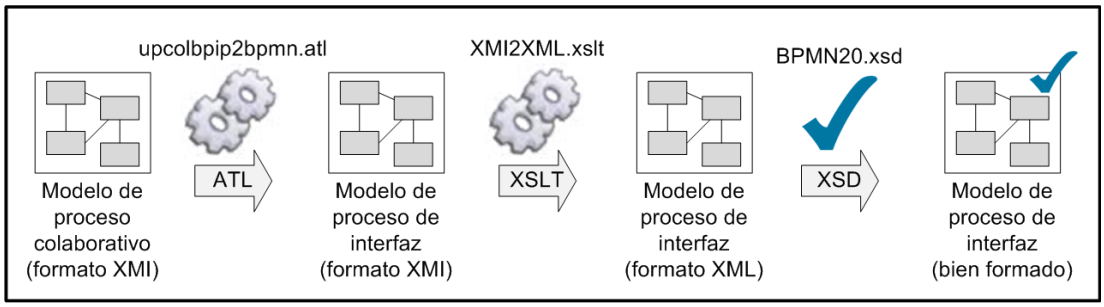


Figura 4.25: Cadena de transformaciones realizadas para obtener un modelo de proceso de interfaz válido.

4.3. Aplicación del método a un caso de estudio

En esta sección se aplica el método propuesto a un caso de estudio para demostrar la funcionalidad del mismo. Para ello se usará el mismo caso de estudio

presentado en el Capítulo 3. Para continuar con dicho caso se va a transformar el modelo de proceso colaborativo *Collaborative Replenishment Plan* mostrado en la Figura 4.26.

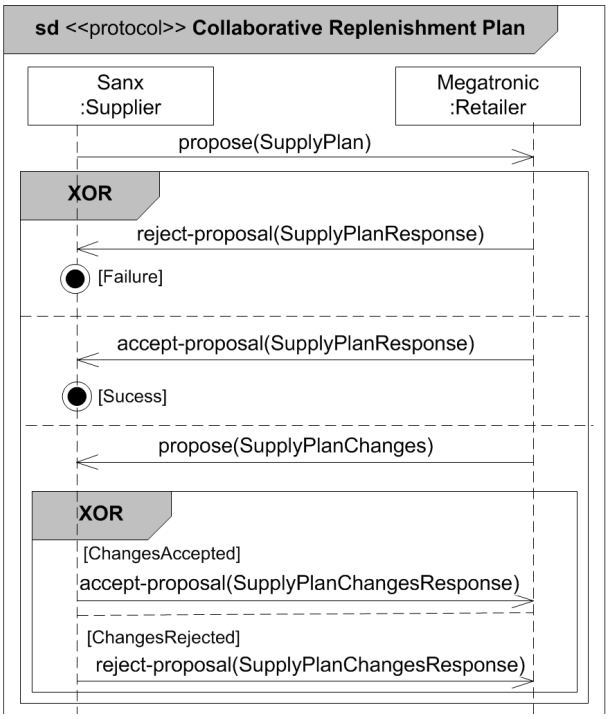


Figura 4.26: Modelo de proceso colaborativo *Collaborative Replenishment Plan*.

Primero, el usuario selecciona el archivo XMI correspondiente al modelo de proceso colaborativo *Collaborative Replenishment Plan* a transformar. En la Figura 4.27 se muestra dicho modelo (archivo CollaborativeReplenishment-Plan.upcolbpip) como una instancia (modelo EMF) del meta-modelo UP-ColBPIP basado en EMF.

Luego, el usuario debe indicar el rol a ser considerado en la transformación. En este caso de estudio se transformará el rol *Supplier* (proveedor), desempeñado por la organización Sanx. Cuando el usuario selecciona dicho rol, se genera automáticamente el modelo de parámetros mostrado en la Figura 4.28 como una instancia del meta-modelo Parameters. Dicho modelo de parámetros, junto con el modelo de proceso colaborativo, es usado como entrada por el método propuesto.

Finalmente, al ejecutar el método se genera automáticamente el modelo de proceso de interfaz correspondiente al rol *Supplier* mostrado en la Figura 4.29.

La Figura 4.30 muestra el modelo de proceso de interfaz como una instancia del meta-modelo BPMN. En dicha figura puede verse que el modelo UP-ColBPIP

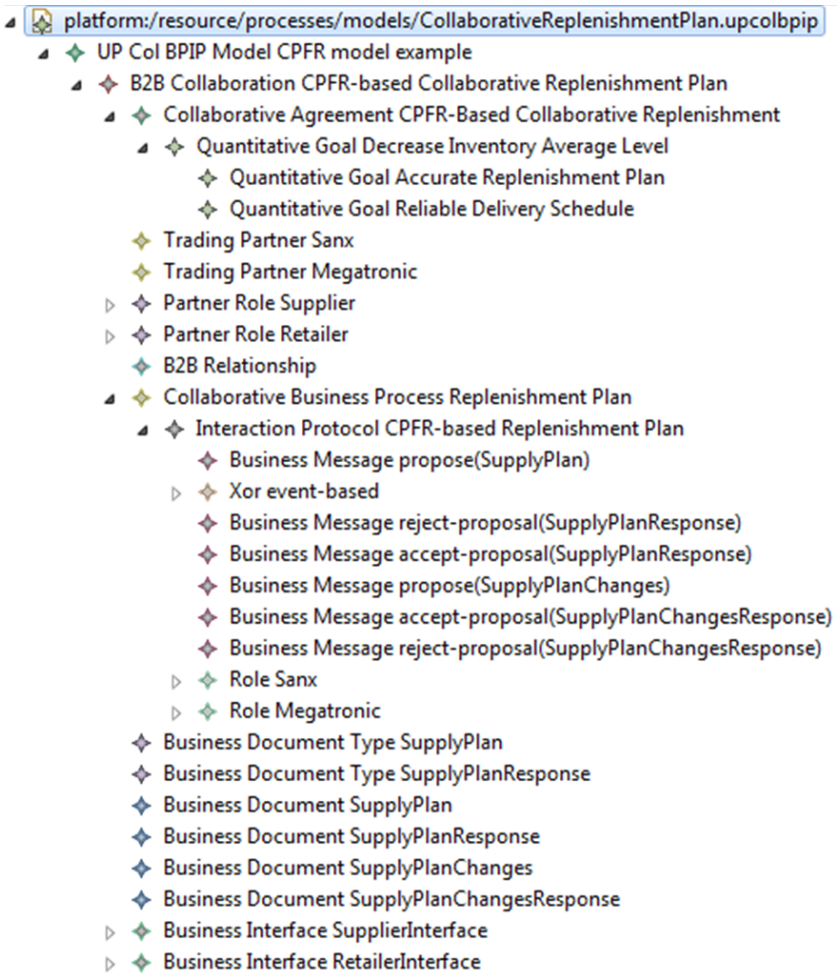


Figura 4.27: Modelo de entrada como instancia del meta-modelo UP-ColBPIP.

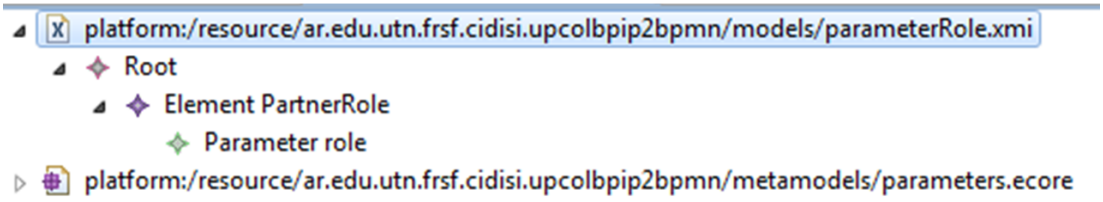


Figura 4.28: Modelo de parámetros como instancia del meta-modelo Parameters.

se transformó en una `BPMN!Definitions`, aplicando la regla *UPColBPIPModel2Definitions* (Sección 4.1.1.1)

En la Figura 4.29 puede verse que la colaboración inter-organizacional se transformó en una `BPMN!Collaboration`, aplicando la regla *b2bCollaboration2Collaboration* (Sección 4.1.1.2). Cada organización se transformó en un `BPMN!Participant`, aplicando la regla *tradingPartner2Participant* (Sección 4.1.1.4, y su respectivo rol en un `BPMN!PartnerRole`,

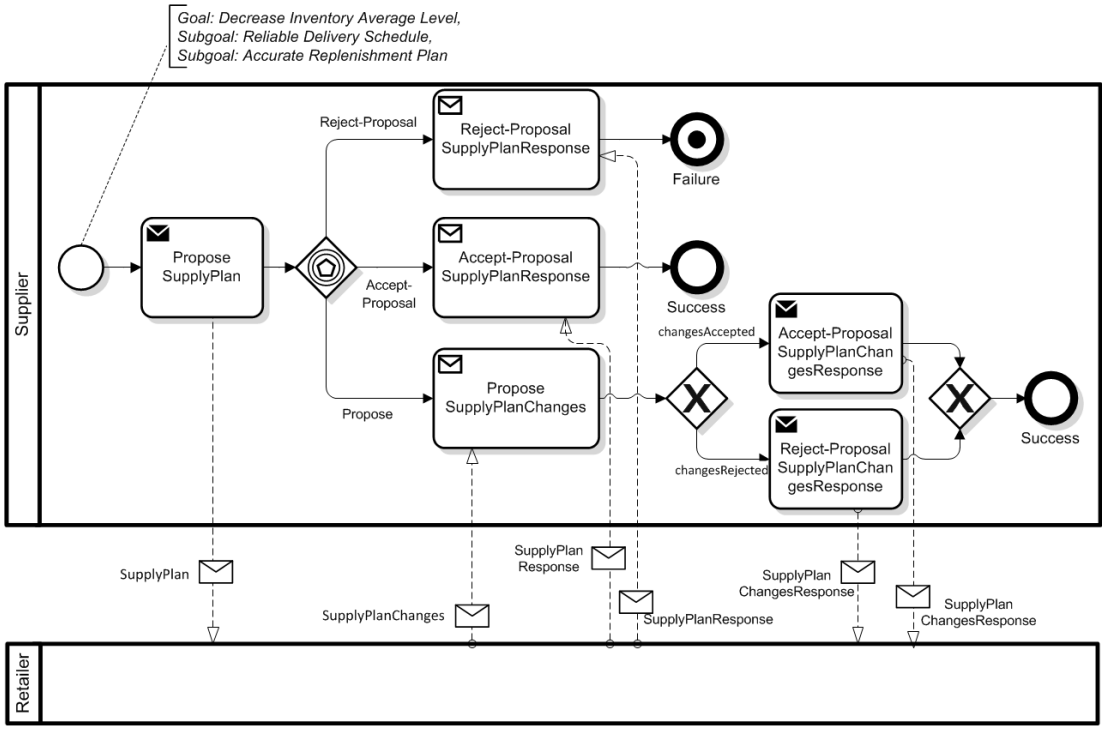


Figura 4.29: Modelo de proceso de interfaz correspondiente al rol *Supplier*.

aplicando la regla *partnerRole2PartnerRole* (Sección 4.1.1.5).

El proceso colaborativo se transformó en un `BPMN!Process`, aplicando la regla *collaborativeProcess2Process* (Sección 4.1.1.3). Como indica dicha regla, se agregó un `BPMN!StartEvent` para indicar el inicio del proceso. El mensaje de negocio *propose(SupplyPlan)* enviado al minorista se transformó en una `BPMN!SendTask`, aplicando la regla *sendBusinessMessage2SendTask* (Sección 4.1.1.6).

El segmento de flujo de control *Xor* se transformó en un `BPMN!EventBasedGateway`, aplicando la regla *cfsXor2ExclusiveGateway* (Sección 4.1.1.7). Cada camino de interacción se transformó en un `BPMN!SequenceFlow`, aplicando la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14). Luego se transformaron los elementos definidos en cada camino. El mensaje de negocio *reject-proposal(SupplyPlanResponse)* del primer camino se transformó en una `BPMN!ReceiveTask`, aplicando la regla *receiveBusinessMessage2ReceiveTask* (Sección 4.1.1.6). La terminación explícita se transformó en un `BPMN!EndEvent`, aplicando la regla *failureExplicitTermination2EndEvent* (Sección 4.1.1.10). El mensaje de negocio *accept-proposal(SupplyPlanResponse)* del segundo camino se transformó en una `BPMN!ReceiveTask`, aplicando la regla

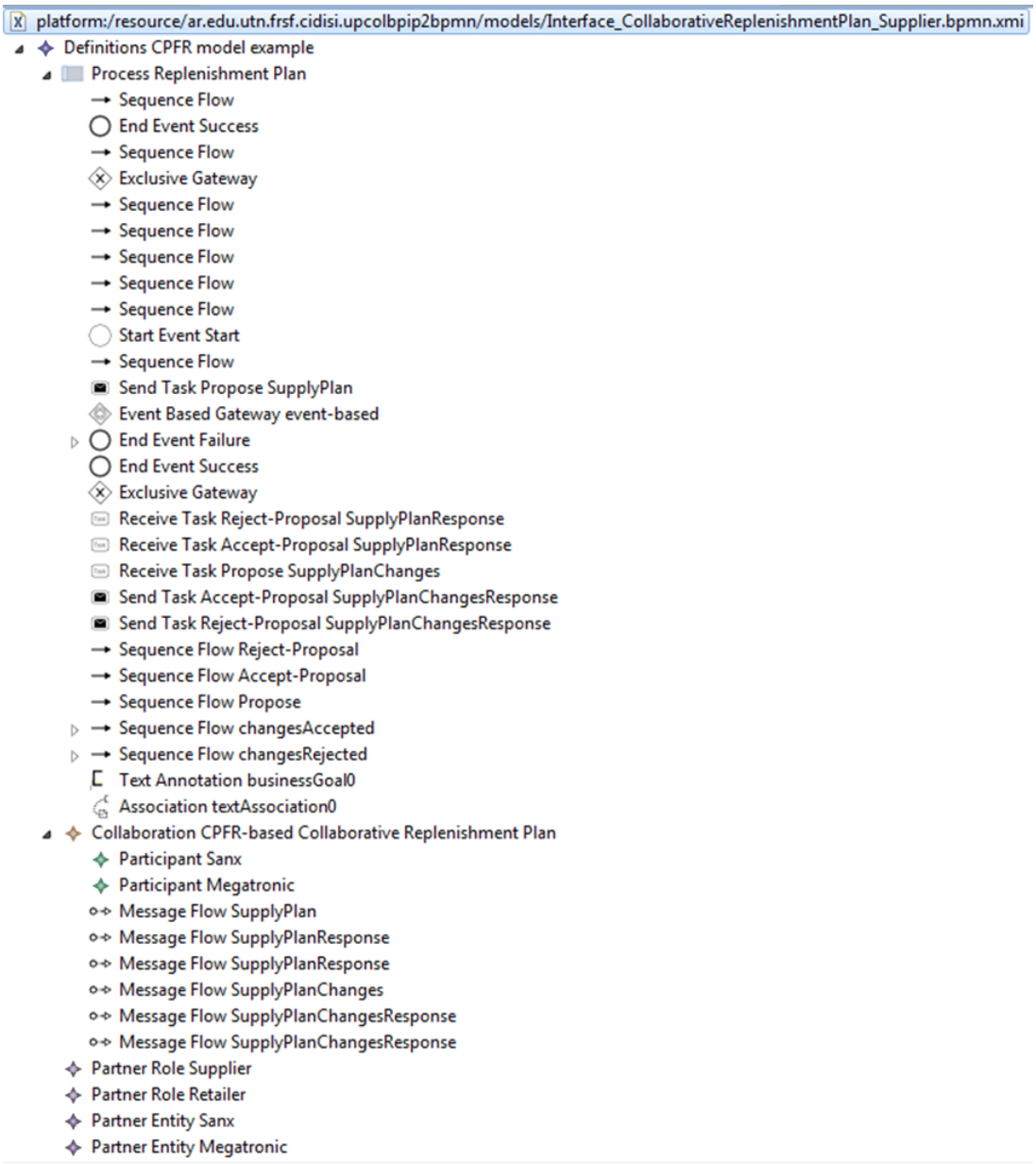


Figura 4.30: Modelo de salida como instancia del meta-modelo BPMN.

receiveBusinessMessage2ReceiveTask. La terminación explícita se transformó en un `BPMN!EndEvent`, aplicando la regla *successExplicitTermination2EndEvent* (Sección 4.1.1.9). El mensaje de negocio *propose(SupplyPlanChanges)* se transformó en una `BPMN!ReceiveTask`, aplicando la regla *receiveBusinessMessage2ReceiveTask*. Como sólo hay un camino que no tiene terminación explícita, no se requiere agregar un `BPMN!ExclusiveGateway` para sincronización, como lo indica la regla *cfsXor2ExclusiveGateway*.

El segmento de flujo de control *Xor* se transformó en un

`BPMN!ExclusiveGateway`, aplicando la regla *cfsXor2ExclusiveGateway* (Sección 4.1.1.7). Cada camino de interacción se transformó en un `BPMN!SequenceFlow`, aplicando la regla *interactionPath2sequenceFlow*. Luego, se transforman los elementos definidos en cada camino. El mensaje de negocio *accept-proposal(SupplyPlanChangesResponse)* del primer camino se transformó en una `BPMN!SendTask`, aplicando la regla *sendBusinessMessage2SendTask* (Sección 4.1.1.6). El mensaje de negocio *reject-proposal(SupplyPlanChangesResponse)* del segundo camino se transformó en una `BPMN!SendTask`, aplicando la regla *sendBusinessMessage2SendTask*. Luego, se agregó un `BPMN!ExclusiveGateway` para sincronización, como lo indica la regla *cfsXor2ExclusiveGateway*.

Todos los documentos de negocio intercambiados se transformaron en `BPMN!Message`, aplicando la regla *businessDocument2Message* (Sección 4.1.1.13).

También se agregó un `BPMN!EndEvent` para representar la terminación del proceso colaborativo, aplicando la regla *implicitTermination2EndEvent* (Sección 4.1.1.11).

Todos los elementos generados por las reglas ejecutadas se vincularon aplicando la regla *successorSequenceFlow* (Sección 4.1.1.16).

4.4. Conclusiones

El método para la generación automática de modelos de procesos de interfaz a partir de modelos de procesos colaborativos propuesto en este capítulo brinda soporte a la actividad *Generación de Procesos de Interfaz* de la fase *Diseño de la Solución Inter-organizacional* de la metodología propuesta en el capítulo anterior.

Permite a las organizaciones generar procesos de interfaz interoperables y conforme a la lógica acordada en el proceso colaborativo. Esto es garantizado debido a que los modelos de procesos de interfaz de cada organización se derivan del modelo de proceso colaborativo aplicando un enfoque “top-down” y los principios de MDA.

El método incrementa el nivel de abstracción en el diseño de la vista particular que cada organización tiene de la colaboración inter-organizacional, usando el lenguaje BPMN para definir modelos orientados a actividades de procesos de

interfaz. Esto facilita a las organizaciones entender y enfocarse en los requerimientos de negocio requeridos para desempeñar su correspondiente rol acordado en el proceso colaborativo.

La definición de transformación del método implementada mediante técnicas de herencia de reglas, refactorización y superimposición permiten lograr la escalabilidad, mantenibilidad y reusabilidad de dicha definición, incrementando su calidad.

La definición en forma estructurada de los patrones destino de las reglas de transformación correspondientes a los segmentos de flujo de control, la derivación de estas reglas aplicando la teoría de patrones de flujo de control de workflows a ambos lenguajes UP-ColBPIP y BPMN, y el etiquetado siguiendo el estilo *verbo-objeto* de todas las actividades públicas para facilitar la comunicación y el entendimiento de los modelos de procesos de negocio, permitieron incrementar la calidad de los modelos de procesos de interfaz generados.

El método mostró que es posible derivar modelos de procesos de interfaz a partir de modelos de procesos colaborativos debido a que toda la información requerida por el modelo de salida, se encuentra definida en el modelo de entrada. Sólo se requiere que el modelador o usuario indique el modelo y el rol a transformar. Esto se logró definiendo un conjunto de reglas que permiten transformar un elemento del proceso colaborativo definido con el lenguaje UP-ColBPIP en un patrón destino definido con el lenguaje BPMN.

Finalmente, la definición del conjunto de reglas de transformación permitió demostrar que cada elemento del lenguaje UP-ColBPIP puede expresarse con uno o más elementos del lenguaje BPMN.

CAPÍTULO 5

Diseño de Modelos de Procesos de Integración

El capítulo presenta un método basado en MDA para el diseño y generación automática de modelos de procesos de integración definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP (Sección 5.1). Se describen las reglas de transformación del método (Sección 5.2), la implementación del mismo, la cual permite su ejecución automática (Sección 5.3), y la aplicación a un caso de estudio (Sección 5.4). Finalmente, se presentan las conclusiones (Sección 5.5).

5.1. Método para la generación de modelos de procesos de integración

El diseño de procesos de integración es una tarea compleja, propensa a errores, que consume tiempo y requiere de la experiencia y conocimiento de los analistas de negocio y diseñadores de sistemas para identificar las actividades privadas que soportan el intercambio de mensajes inter-organizacionales. Los modelos de procesos de integración de las organizaciones deben estar alineados y ser consistentes con el modelo de proceso colaborativo, por lo cual deben ser correctamente definidos para garantizar su interoperabilidad (Lazarte y otros, 2011).

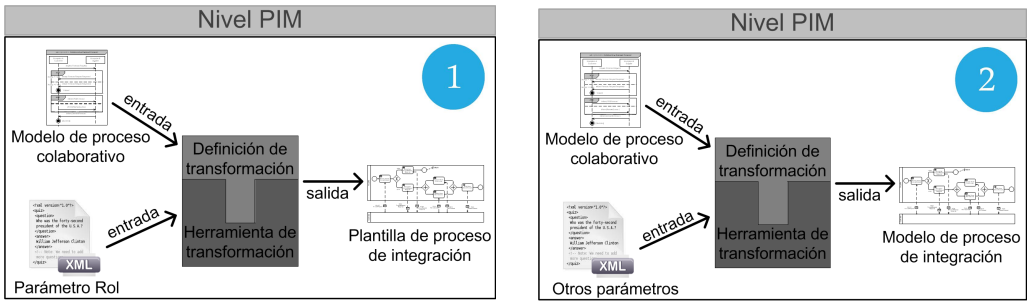
Con el propósito de asistir a las organizaciones y facilitar la tarea, se propone un método que aplica los principios de MDA para diseñar y generar en forma automática modelos de procesos de integración definidos con el lenguaje BPMN (OMG, 2011a) a partir de modelos de procesos colaborativos (representados como protocolos de interacción) definidos con el lenguaje UP-ColBPIP (Villarreal

y otros, 2010, 2007b). El método permite generar una plantilla de proceso de integración, y un modelo de proceso de integración en base a criterios de diseño definidos por el usuario. Una *plantilla de proceso de integración* es un *modelo de proceso BPMN configurable* (Gröner y otros, 2012; La Rosa y otros, 2011a), cuyas actividades privadas son abstractas, representadas por nodos configurables (puntos de variación). Dichos nodos pueden ser configurados estableciendo valores a sus atributos para definir modelos concretos. Cada nodo captura una decisión de diseño que debe ser evaluada por el usuario. Una plantilla captura el comportamiento común que debe formar parte de todo modelo de proceso de integración. De esta manera, a partir de una plantilla se pueden definir modelos con diferentes variantes.

Un *modelo de proceso de integración* BPMN es aquel que tiene al menos definida una actividad privada concreta, es decir, no abstracta. Además un modelo es consistente con el esqueleto de una plantilla. Es decir, es una *configuración* particular o una instancia de la plantilla que tiene definida las actividades públicas y en forma concreta las actividades privadas requeridas para que la organización pueda desempeñar el rol que le corresponde en un proceso colaborativo.

El método propuesto permite dar soporte a la actividad *Diseño de Procesos de Integración* de la fase *Diseño de la Solución Inter-organizacional* de la metodología presentada en el Capítulo 3.

El método tiene dos alternativas de ejecución (Figura 5.1); ambas transforman modelos de procesos de negocio en un nivel PIM. En la primera alternativa de transformación (Figura 5.1(a)), el método toma como entrada un modelo de



(a) Generación de plantilla de proceso de integración.

(b) Generación de modelo de proceso de integración.

Figura 5.1: Método basado en MDA para la generación de procesos de integración a partir de procesos colaborativos.

proceso colaborativo y un modelo de parámetros (conteniendo sólo el rol de la organización para la que se genera la plantilla) para generar como salida una plantilla de proceso de integración de la organización. En la segunda alternativa de transformación (Figura 5.1(b)), para el mismo modelo de proceso colaborativo y otro modelo de parámetros (conteniendo el rol de la organización y otros parámetros de configuración que se describen en la Sección 5.1.3), el método genera como salida un modelo de proceso de integración de la organización. En ambos casos se utiliza la misma definición de transformación. Esta definición contiene las reglas de transformación que permiten generar las actividades privadas abstractas cuando se genera una plantilla. Contiene también especializaciones de dichas reglas, que tienen en cuenta los parámetros de configuración para generar las actividades privadas concretas.

Para llevar a cabo las transformaciones, en las reglas se propone un conjunto de patrones destino definidos con el lenguaje BPMN para cada elemento del lenguaje UP-ColBPIP correspondiente a protocolos de interacción. De esta manera, la semántica de cada uno de estos elementos, se representa en términos de los elementos y semántica provista por el lenguaje BPMN, desde el punto de vista particular de una organización (indicado por el rol a transformar), de manera similar al método definido en el Capítulo 4.

El método ofrece a las organizaciones una herramienta de diseño automatizada, ya que permite generar actividades públicas y privadas en un modelo de proceso de integración a partir de un modelo de proceso colaborativo, posibilitando al usuario incorporar decisiones de diseño a través de la definición de parámetros de entrada. Dado que en un modelo de proceso colaborativo sólo se define el comportamiento común y público a las partes involucradas, para generar el comportamiento privado de cada parte se requiere de una estrategia particular. En este método se propone un enfoque novedoso para generar, a partir de los mensajes de negocio de los protocolos de interacción, las actividades públicas de envío/recepción de mensajes, y las actividades privadas que permiten generar o procesar dichos mensajes. Para ello, se hace uso de la teoría de los *actos de comunicación* en la que se basa el lenguaje UP-ColBPIP para definir los mensajes de negocio de los protocolos, y por otro lado se aplica la teoría de los *patrones de actividades de workflow* (Thom y otros, 2009), descritos en el Capítulo 2. Los patrones de actividades de workflow representan funciones de negocio recurrentes

y frecuentemente encontradas en procesos de negocio. Los patrones permiten reusar el conocimiento capturado en ellos para generar las actividades públicas y privadas requeridas por una organización para soportar el intercambio de mensajes de negocio, garantizando la interoperabilidad de los modelos obtenidos. Para cada acto de comunicación se encontró y asoció un patrón de actividad que representa su semántica, a partir del cual se definió el patrón destino de la regla que transforma un mensaje basado en dicho acto de comunicación. Por lo tanto, los patrones destino de las reglas de transformación se definieron de acuerdo a los *patrones de actividades de workflow*.

En las siguientes sub-secciones, se describe primero la generación de una plantilla de proceso de integración, luego la generación de un modelo de proceso de integración y finalmente la generación de un modelo de parámetros usado como entrada en el proceso de transformación.

5.1.1. Generación de una plantilla de proceso de integración

El Algoritmo 2 describe los pasos necesarios para generar una plantilla de proceso de integración (conforme al meta-modelo BPMN, Apéndice A.2) a partir de un modelo de proceso colaborativo (conforme al meta-modelo UP-ColBPIP, Apéndice A.1). El proceso de transformación consiste en analizar cada elemento e de un modelo de proceso colaborativo M_{PC} desde el punto de vista de una organización (indicado por el rol R seleccionado a transformar) y generar los patrones destino p, aplicando reglas de transformación. La plantilla M_{PI} se construye mediante la composición de los elementos generados a partir de los patrones p. El rol R se define en un modelo de parámetros conforme al meta-modelo Parameters presentado en el Capítulo 4. Con el propósito de preservar los aspectos privados de

Algoritmo 2 Algoritmo para generar una plantilla de proceso de integración M_{PI} a partir de un modelo de proceso colaborativo dado M_{PC} y un rol R.

Entrada: M_{PC} y R
  para cada elemento $e \in M_{PC}$ y punto de vista de R **hacer**
    transformar elemento e en patrón p
    insertar patrón p en modelo M_{PI}
  fin para
Salida: M_{PI}

cada organización, los elementos correspondientes al otro rol no se transforman, por consiguiente, el "pool" (OMG, 2011a) correspondiente a la otra organización se modela como una caja negra.

La plantilla de proceso de integración M_{PI} generada (Figura 5.1(a)) contiene: (a) las actividades públicas que permiten el envío y recepción de mensajes de negocio definidos en el modelo de proceso colaborativo; (b) las actividades privadas, definidas en forma abstracta, requeridas para procesar o generar la información a ser intercambiada en los mensajes de negocio; y (c) el flujo de control entre las actividades públicas y privadas, de acuerdo al flujo de control definido en el modelo de proceso colaborativo. Una plantilla generada para una organización es única. Es decir, el método garantiza que al aplicar una transformación para el rol de una organización siempre se va a generar la misma plantilla.

La plantilla es un modelo de proceso configurable, cuyos nodos pueden ser definidos y configurados por el usuario (Figura 5.2). Los nodos configurables representan las actividades privadas abstractas, las cuales pueden definirse como atómicas o como subprocesos, de acuerdo a los requerimientos de la organización; y los constructores de flujo de control ("gateways"), en los cuales se puede agregar el flujo de secuencia por defecto o definir explícitamente las condiciones a ser evaluadas. Las actividades públicas y los constructores de flujo de control generados deben permanecer inalterables, para preservar en el modelo de proceso de integración el comportamiento definido en el modelo de proceso colaborativo del que se deriva.

Una ventaja de la plantilla es que mediante la configuración de sus nodos, los diseñadores pueden derivar un modelo de proceso de integración, evitando el

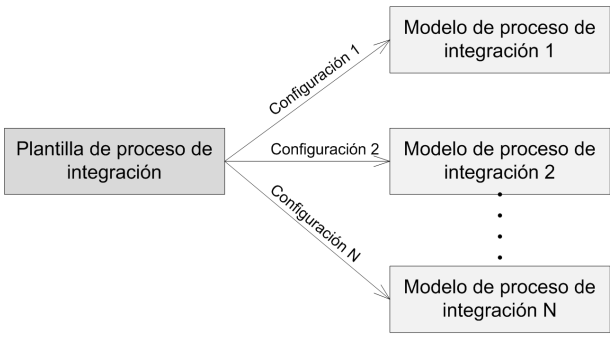
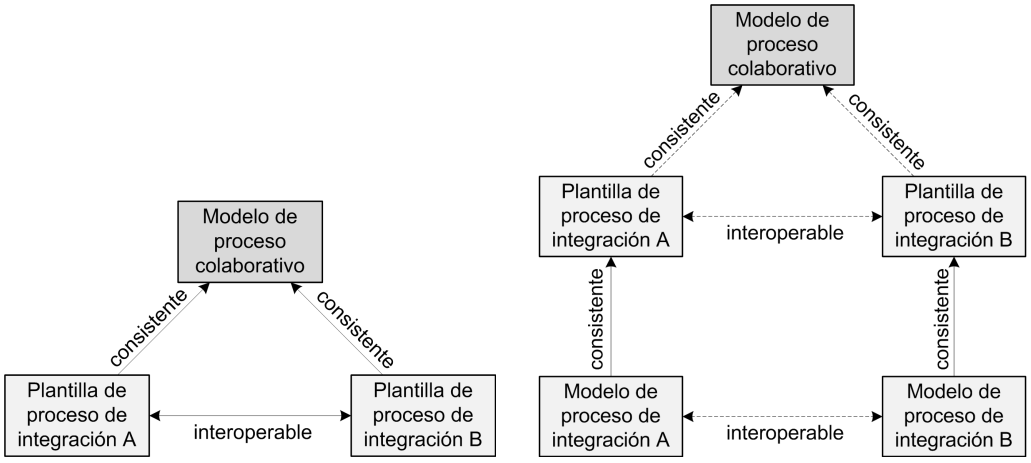


Figura 5.2: Relación entre la plantilla y los modelos de procesos de integración derivados de la misma.

diseño desde cero. Esta propiedad de ser configurable permite generar diferentes modelos de procesos de integración, lo cual la hace reusable (Figura 5.2). La reusabilidad aumenta la calidad de los modelos generados reduciendo la probabilidad de errores, permite a la organización adaptarse más rápidamente a los cambios de requerimientos en los modelos de procesos colaborativos, facilita la propagación de cambios, y reduce tiempo y costos de desarrollo. La plantilla también sirve como medio de discusión y comunicación entre los analistas de negocio y diseñadores de sistemas responsables de definir la solución inter-organizacional y la solución de arquitectura de TI (Capítulo 3).

Como se muestra en la Figura 5.3(a), la plantilla generada por el método para una organización, es consistente con el modelo de proceso colaborativo e interoperable con las plantillas de las restantes organizaciones participantes. La consistencia de comportamiento es garantizada por el proceso de transformación modelo-a-modelo, mediante el cual se genera la plantilla. La interoperabilidad en el intercambio de mensajes definido en la plantilla es garantizada mediante la aplicación de patrones de actividades de workflow (Sección 5.2.1), ya que los mismos indican actividades de envío y recepción de mensajes sincronizadas de cada una de las partes involucradas.

La plantilla puede ser utilizada para chequear la consistencia de un modelo de proceso de integración, para garantizar la interoperabilidad entre los mode-



(a) Relación de consistencia e interoperabilidad entre las plantillas y el modelo de proceso colaborativo.

(b) Relación de consistencia e interoperabilidad entre los modelos de procesos de integración.

Figura 5.3: Relaciones de consistencia e interoperabilidad entre los procesos de negocio generados por el método.

los de procesos de integración de las organizaciones con respecto al modelo de proceso colaborativo (Figura 5.3(b)). Si el modelo de proceso de integración es consistente con la plantilla, entonces también es consistente con el modelo de proceso colaborativo, y además es interoperable con los modelos de procesos de integración de las demás organizaciones involucradas.

5.1.2. Generación de un modelo de proceso de integración

La plantilla de un proceso de integración contiene actividades privadas abstractas cuyos atributos deben ser definidos para generar un modelo de proceso de integración que indique los detalles de cada una de estas actividades, tales como el tipo de actividad, quién la realiza, etc.

Con el propósito de asistir a los usuarios, se propone un método basado en un enfoque de cuestionario y parámetros que automatiza el diseño de modelos de procesos de integración. Este enfoque ha sido recientemente utilizado en otras propuestas (La Rosa y otros, 2009, 2011a, 2007). Diferentes preguntas relacionadas a los elementos de un modelo de proceso colaborativo de entrada son respondidas por el usuario a través de un cuestionario interactivo que lo guía durante el diseño, formulando sólo las preguntas relevantes en un orden consistente (Figura 5.4).

Dicho cuestionario contiene preguntas expresadas en lenguaje natural, por ejemplo, *¿Qué tipo de actividad es responsable de generar el documento de negocio X asociado al mensaje Y?*, junto con las respuestas posibles, por ejemplo, qué tipo de actividad BPMN es: *Service*, *User*, *Business Rule*, *Script*, *Manual* o *None*.

Después que el usuario ha respondido todas las preguntas del cuestionario, se genera un modelo de parámetros, con las respuestas del cuestionario contenidas en los parámetros de configuración (Sección 5.1.3), los cuales serán usados en el proceso de transformación.

Para generar un modelo de proceso de integración, el proceso de transformación se realiza de manera similar a lo expresado en el Algoritmo 2. En este caso, además del rol, se consideran los parámetros de configuración, los cuales son usados para analizar cada patrón origen y generar el correspondiente patrón destino. El modelo de proceso de integración generado (Figura 5.4) contiene: (a) las actividades públicas que permiten el envío y la recepción de mensajes de nego-

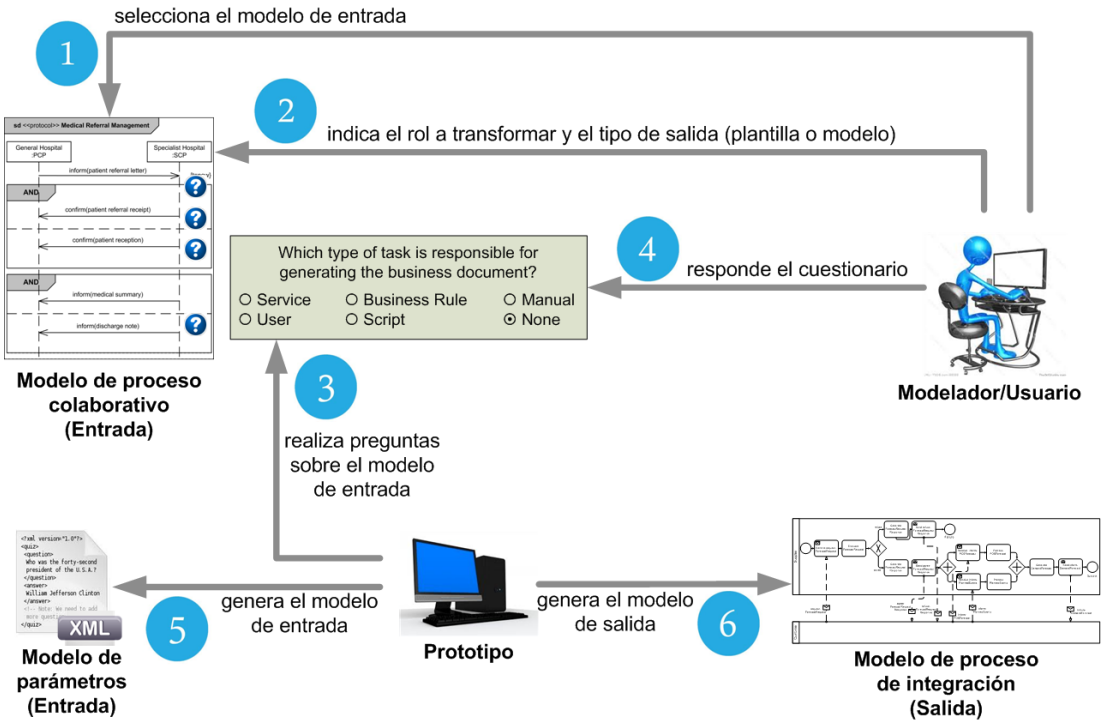


Figura 5.4: Pasos realizados para generar un modelo de proceso de integración.

cio definidos en el modelo de proceso colaborativo; (b) las actividades privadas con valores en sus atributos, obtenidos del modelo de parámetros definido por el usuario; (c) las actividades privadas abstractas en caso que no se hayan definido parámetros para alguna de las mismas, es decir, puntos de configuración no completados; y (d) el flujo de control (con detalles obtenidos del modelo de parámetros, si los hubiera) entre las actividades públicas y privadas, de acuerdo al flujo de control definido en el proceso colaborativo.

Cabe aclarar que si el usuario solo proporciona el rol de la organización, el modelo de proceso de integración generado será equivalente a la plantilla de dicho proceso.

Debido a que la plantilla y los modelos de procesos de integración son generados aplicando las mismas reglas, cada modelo generado es una configuración de una plantilla, dado que siempre tendrá el mismo comportamiento. Esto permite establecer que todo modelo de proceso de integración generado con este método es consistente con la plantilla. Por consiguiente, se puede garantizar que siempre será interoperable con los modelos de procesos de integración de las demás organizaciones que participan del proceso colaborativo (Figura 5.3(b)).

Una característica del método es su flexibilidad, ya que permite al usuario

probar diferentes configuraciones hasta generar un modelo de proceso de integración que satisfaga los requerimientos de negocio (Figura 5.2). Las diferentes configuraciones se logran asignando distintos valores a los parámetros.

Eventualmente, un modelo de proceso de integración podrá ser refinado por un usuario hasta obtener un modelo con todos los detalles que los diseñadores de sistemas necesitan para generar la solución de arquitectura de TI.

5.1.3. Generación de un modelo de parámetros

Las respuestas del cuestionario interactivo por parte del usuario generan un modelo de parámetros que contiene los parámetros de configuración. Dichos parámetros se definen analizando los atributos requeridos por los elementos del patrón destino de las reglas de transformación.

El modelo de parámetros es conforme al meta-modelo Parameters (Figura 5.5). Un modelo de parámetros `ParameterModel` se compone de al menos un elemento `Element`. Un `Element` hace referencia a un elemento del modelo de proceso colaborativo (por ejemplo, un rol o mensaje de negocio) para el cual el modelador o usuario ha ingresado información. Cada elemento tiene atributos: `ID`, `name`, `type`; y uno o más parámetros `Parameter` para representar la información ingresada. El atributo `ID` identifica el elemento del proceso colaborativo al cual se le ha establecido un parámetro; `name` indica el nombre del elemento; `type` indica el tipo del elemento. Cada `Parameter` tiene definido los atributos `name`, el cual indica el nombre del parámetro; y `value`, el cual indica el valor del parámetro. Dicho modelo contiene como mínimo el rol a transformar.

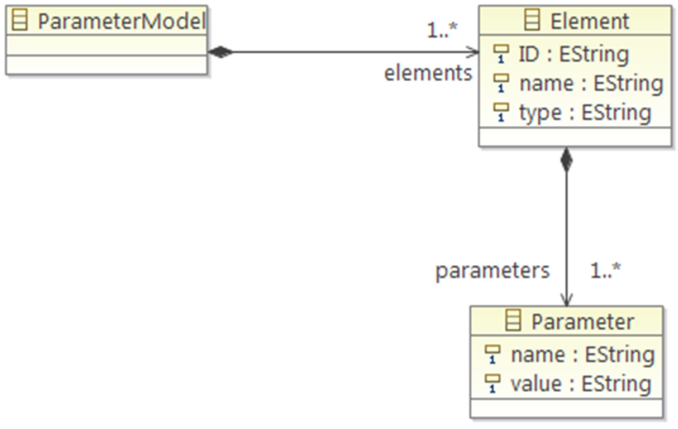


Figura 5.5: Meta-modelo Parameters.

Los parámetros de configuración que se pueden definir corresponden a parámetros asociados a elementos del modelo de proceso colaborativo de entrada, tales como mensajes de negocio y segmentos de flujo de control. Los mismos se describen a continuación.

5.1.3.1. Parámetros de configuración para los mensajes de negocio

Un mensaje de negocio de un modelo de proceso colaborativo indica la transferencia de información de una organización hacia otra. El contenido que transporta el mensaje es indicado por el documento de negocio asociado al mismo. En este caso, el usuario puede indicar valores a los diferentes parámetros que se utilizan para definir los atributos de la tarea responsable de generar o procesar dicho documento de negocio. Los parámetros que permiten definir dicha tarea son:

- *taskType*: indica si un documento de negocio a ser enviado/recibido será generado/procesado, respectivamente, por un servicio, un usuario, un "script", una regla de negocio o manualmente. Los valores posibles de este parámetro son: *ServiceTask*, *UserTask*, *BusinessRuleTask*, *ScriptTask*, *ManualTask* y *None*. El valor por defecto es *None*, en cuyo caso la tarea se genera como abstracta. Estos valores se corresponden con los tipos de tareas que permite definir el lenguaje BPMN. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Qué tipo de actividad es responsable de generar/procesar el documento de negocio X asociado al mensaje Y?*

- *implementation*: indica la tecnología que será usada para generar/procesar el documento de negocio. Los posibles valores de este parámetro son: *Unspecified*, *WebService* o una URI indicando cualquier otra tecnología. El valor por defecto es *Unspecified*. Si el valor del parámetro *taskType* es *ManualTask*, este parámetro no es requerido. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Qué tecnología se usa para generar/procesar el documento de negocio X asociado al mensaje Y?*

- *performer*: indica quién realizará o será responsable de una tarea. Se puede especificar como un individuo específico, un grupo, un rol o posición dentro la organización, o una organización. Este parámetro sólo es requerido si el

valor del parámetro *taskType* es *UserTask* o *ManualTask*. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Qué usuario o rol es responsable de generar/procesar el documento de negocio X asociado al mensaje Y?*

- *operation*: indica la operación que será invocada por una tarea de tipo servicio. Este parámetro sólo es requerido si el valor del parámetro *taskType* es *ServiceTask*. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es la operación invocada para generar/procesar el documento de negocio X asociado al mensaje Y?*

- *script*: indica el "script" que será ejecutado para generar/procesar el documento de negocio. Este parámetro sólo es requerido si el valor del parámetro *taskType* es *ScriptTask*. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es el script que será ejecutado para generar/procesar el documento de negocio X asociado al mensaje Y?*

- *scriptFormat*: indica el formato de un "script" y debe ser especificado en un formato de tipo MIME. Este parámetro sólo es requerido si el valor del parámetro *taskType* es *ScriptTask*. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es el formato del script ingresado anteriormente?*

5.1.3.2. Parámetros de configuración para los segmentos de flujo de control

Los segmentos de flujo de control representan secuencias complejas de mensajes. En el caso de segmentos en donde la ejecución de sus caminos depende de la evaluación de una condición, el usuario puede indicar cuál es el camino de interacción por defecto o la condición a ser evaluada en cada camino. Aunque en un modelo de proceso colaborativo basado en protocolo de interacción se definen las condiciones para los segmentos de flujo de control *Xor*, *Or*, *Loop*, *Exception* y *Cancel*, generalmente éstas se definen en forma conceptual usando texto en lenguaje natural, es decir, no indican las variables reales a ser evaluadas. Por lo tanto, para generar un modelo de proceso de integración que contenga condiciones en sus "gateways" o constructores que hagan referencia a atributos del proceso,

se puede especificar las mismas mediante parámetros. Los parámetros definidos son los siguientes:

- *defaultPath*: indica el camino de interacción por defecto. El valor que toma el parámetro es el nombre del camino por defecto y sólo es requerido para los segmentos de flujo de control *Xor* y *Or*. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es el camino de interacción por defecto del segmento de flujo de control X?*. En este caso, se muestra una lista conteniendo los nombres de los caminos de interacción del segmento de flujo de control respectivo.

- *expression*: indica la expresión a ser evaluada en un camino de interacción para definir el flujo del proceso colaborativo. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es la expresión a ser evaluada en el camino de interacción X?*

- *language*: indica el lenguaje en que se definió la expresión y debe ser especificado en un formato URI. Este parámetro sólo es requerido si se definió una condición a un camino de interacción. La pregunta que se realiza al usuario para obtener el valor de este parámetro es: *¿Cuál es el lenguaje usado para definir la expresión ingresada anteriormente?*

5.2. Definición de transformación del método

Las reglas de transformación para generar modelos de procesos de integración, son definidas mediante especialización de las reglas de transformación definidas para generar una plantilla (reglas base). El propósito es mantener el mismo comportamiento de una plantilla y considerar los parámetros de configuración definidos para los elementos del modelo de proceso colaborativo a efectos de generar las correspondientes actividades privadas o flujos de control concretos. En el caso en que no se hayan definido parámetros para un elemento del modelo de entrada, dicho elemento se transforma aplicando la regla base respectiva.

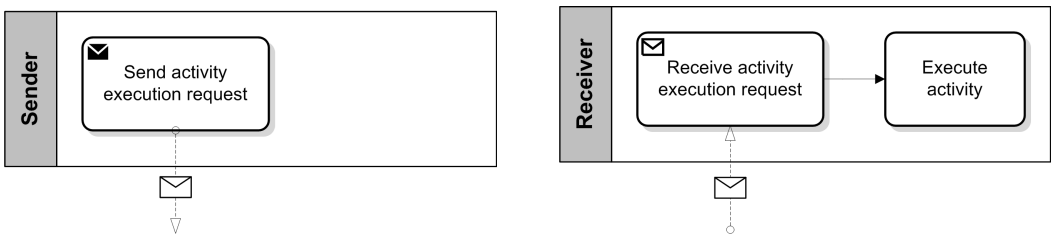
A continuación, se presentan las reglas de transformación de los mensajes de negocio y luego las correspondientes a los restantes elementos del proceso colaborativo. En cada caso, primero se presentan las reglas utilizadas para generar una plantilla, y luego las reglas especializadas para generar modelos de procesos de

integración. Las reglas se describen conceptualmente y los elementos involucrados en cada una de ellas se identifican usando la notación punto.

5.2.1. Transformación de mensajes de negocio

En los modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP, la semántica de cada mensaje de negocio está dada por el acto de comunicación asociado al mismo. Esta semántica se utilizó para determinar los patrones de actividades de workflow a usar en cada una de las reglas de transformación. Para definir los patrones origen y destino de estas reglas se analizó la similitud semántica entre los actos de comunicación usados por UP-ColBPIP y los patrones de actividades de workflow. En la Tabla 5.1 se muestra la similitud semántica de los principales actos de comunicación y el correspondiente patrón de actividades de workflow.

Cada patrón de actividades de workflow está expresado en términos de actividades que realiza un emisor y actividades que realiza un receptor. Por lo tanto, los patrones destino de las reglas de transformación se definen de acuerdo a si el rol de la organización definido en el modelo de parámetros es el de emisor o receptor del mensaje. Para el rol emisor, el patrón destino contiene todas las actividades correspondientes a dicho rol del patrón de actividades de workflow aplicado en la regla (Figura 5.6(a)). Para el rol receptor, el patrón destino contiene todas las actividades correspondientes a dicho rol del patrón de actividades de workflow aplicado en la regla (Figura 5.6(b)). De esta manera, se garantiza que existe una tarea de envío por cada tarea de recepción y viceversa, y por consiguiente se garantiza la interoperabilidad y sincronización de las partes en el intercambio de mensajes.



(a) Patrón destino para el rol emisor. (b) Patrón destino para el rol receptor.

Figura 5.6: Ejemplos de patrón destino de reglas de transformación basadas en el patrón de actividades de workflow *Unidirectional Performative*.

Tabla 5.1: Similitud semántica entre los actos de comunicación y los patrones de actividades de workflow (WAPs).

Speech acts	WAPs	Descripción
Accept-Proposal	Notification (WAP5)	El emisor notifica al receptor la aceptación de una propuesta enviada previamente. Dicha aceptación es el resultado de una tarea de decisión.
Agree	Notification (WAP5)	El emisor notifica al receptor la aceptación de la solicitud enviada previamente para realizar una acción. Dicha aceptación es el resultado de una tarea de decisión.
Call-for-Proposal	Unidirectional Performative (WAP3)	El emisor solicita al receptor que genere una propuesta. El emisor continúa la ejecución de su procesos inmediatamente después de enviar dicha solicitud.
Confirm	Notification (WAP5)	El emisor confirma al receptor que cierta proposición es verdadera.
Disconfirm	Notification (WAP5)	El emisor confirma al receptor que cierta proposición es falsa.
Inform	Unidirectional Performative (WAP3)	El emisor solicita el receptor que procese un documento de negocio particular para que éste tenga conocimiento de la información contenida en tal documento de negocio.
Propose	Unidirectional Performative (WAP3)	El emisor solicita al receptor que evalúe una propuesta para determinar cómo continuar la ejecución del proceso.
Refuse	Notification (WAP5)	El emisor notifica al receptor la negación a realizar una acción solicitada. Dicha negación o rechazo es el resultado de una tarea de decisión.
Reject-Proposal	Notification (WAP5)	El emisor notifica al receptor que rechaza la propuesta enviada anteriormente. TDicha negación o rechazo es el resultado de una tarea de decisión.
Request	Unidirectional Performative (WAP3)	El emisor solicita al receptor que realice una acción.

Los patrones de actividades de workflow proveen las actividades que permiten enviar y recibir mensajes de negocio, y las actividades que permiten procesar y evaluar los documentos de negocio de los mensajes recibidos, pero no proveen las actividades que permiten generar los documentos de negocio a ser enviados. Consecuentemente, estas actividades se han agregado a los patrones destino del rol emisor, ya que son requeridas como actividades privadas en el envío de mensajes.

En cada patrón destino, cada actividad pública que permite a una organización enviar o recibir un mensaje de negocio se define como `Send Task` o `Receive Task` (OMG, 2011a), respectivamente. A ambos tipos de tareas se asocia un `Message` (OMG, 2011a), que representa el documento de negocio intercambiado. Las actividades privadas para las cuales no se han ingresado parámetros de configuración se definen como `Task` (OMG, 2011a), para indicar que son actividades abstractas o nodos configurables del modelo de proceso.

Las actividades definidas en cada patrón destino se etiquetan siguiendo el estilo *verbo-objeto* (Mendling y otros, 2010b), en donde *verbo* es el acto de comunicación y *objeto* es el documento de negocio intercambiado.

5.2.1.1. Transformación de un mensaje de negocio con un acto de comunicación Call-for-Proposal, Inform, Propose o Request

Para el caso de mensajes de negocio basados en los actos de comunicación *call-for-proposal*, *inform*, *propose* o *request*, cuya semántica en común es que el emisor requiere que el receptor realice alguna actividad, el patrón destino de la regla de transformación se basa en el patrón de actividades de workflow *Unidirectional Performative* (WAP3), el cual también permite al emisor solicitar al receptor la ejecución de una actividad particular.

Las reglas presentadas en las siguientes sub-secciones se utilizan en el caso que se esté generando una plantilla o bien no se hayan definido parámetros para los mensajes de negocio de los actos de comunicación indicados.

Regla sendBusinessMessage2WAP3

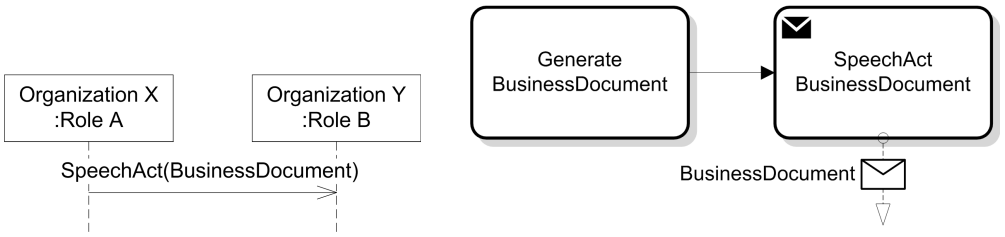
Un mensaje de negocio `UPCOLBPIP!BusinessMessage` cuya intención es `UPCOLBPIP!SpeechAct.name`, la información intercambiada es `UPCOLBPIP!BusinessDocument` y es enviada por el rol

`UPCOLBPIP!ParnerRole = PARAMETERS!role` (Figura 5.7(a)), se transforma en dos tareas; una tarea abstracta `BPMN!Task` y una tarea `BPMN!SendTask`; y un flujo de secuencia `BPMN!SequenceFlow` que vincula ambas tareas (Figura 5.7(b)).

Los atributos de `BPMN!Task` se establecen como sigue: al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor *Generate* + `UPCOLBPIP!BusinessDocument.name`; al atributo `completionQuantity` se le asigna el valor *1*, al atributo `isForCompensation` se le asigna el valor *false*; y al atributo `startQuantity` se le asigna el valor *1*.

Los atributos de `BPMN!SendTask` se establecen como sigue: al atributo `id` se le asigna el valor *sendTask_* + `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false* y al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.

Los atributos de `BPMN!SequenceFlow` se establecen como sigue: al atributo `id` se le asigna el valor *sequenceFlow_* + `UPCOLBPIP!BusinessMessage.id`; al atributo `isImmediate` se le asigna el valor *true*; al atributo `sourceRef` se le asigna el valor `BPMN!Task`; y al atributo `targetRef` se le asigna el valor `BPMN!SendTask`.



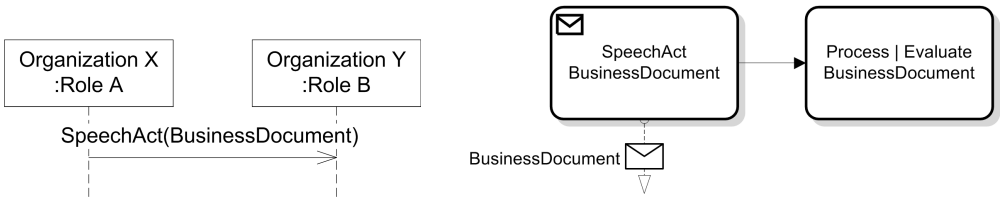
(a) Patrón origen. (b) Patrón destino.

Figura 5.7: Representación gráfica de la regla *sendBusinessMessage2WAP3*.

Regla receiveBusinessMessage2WAP3

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` cuya intención es `UPCOLBPIP!SpeechAct.name`, la información intercambiada es `UPCOLBPIP!BusinessDocument` y es recibida por el rol `UPCOLBPIP!ParnerRole = PARAMETERS!role` (Figura 5.8(a)), se transforma en dos tareas; una tarea `BPMN!ReceiveTask` y una tarea abstracta `BPMN!Task`; y un flujo de secuencia `BPMN!SequenceFlow` que vincula ambas tareas (Figura 5.8(b)).

Los atributos de `BPMN!ReceiveTask` se configuran como sigue: al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false* y al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.



(a) Patrón origen. (b) Patrón destino.

Figura 5.8: Representación gráfica de la regla *receiveBusinessMessage2WAP3*.

Los atributos de `BPMN!Task` se configuran como sigue: al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor *Process* + `UPCOLBPIP!BusinessDocument.name` si el acto de comunicación es *Inform*, caso contrario se le asigna el valor *Evaluate* + `UPCOLBPIP!BusinessDocument.name`; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false*; y al atributo `startQuantity` se le asigna el valor *1*. En el caso de los actos de comunicación *call-for-proposal*, *propose* y *request*, se etiqueta la tarea generada como *evaluate* debido a que dichos actos de comunicación implican que el receptor debe tomar una decisión.

Los atributos de `BPMN!SequenceFlow` se establecen como sigue: al atributo `id` se le asigna el valor *sequenceFlow_* + `UPCOLBPIP!BusinessMessage.id`; al atributo `isImmediate` se le asigna el valor *true*; al atributo `sourceRef` se le asigna el valor `BPMN!ReceiveTask`; y al atributo `targetRef` se le asigna el valor `BPMN!Task`.

5.2.1.2. Transformación de un mensaje de negocio con el acto de comunicación Accept-Proposal, Agree, Confirm, Disconfirm, Refuse o Reject-Proposal

Para el caso de mensajes de negocio basados en los actos de comunicación *accept-proposal*, *agree*, *confirm*, *disconfirm*, *refuse* o *reject-proposal*, el patrón destino de la regla de transformación se basa en el patrón de actividades de workflow *Notification* (WAP5), el cual permite al emisor notificar al receptor el resultado (accept-proposal, agree, confirm, disconfirm, refuse o reject-proposal) de la ejecución de una actividad. La razón de la decisión es comunicada en el documento de negocio intercambiado.

Las reglas presentadas en las siguientes sub-secciones se utilizan en el caso que se esté generando una plantilla o bien no se hayan definido parámetros para los mensajes de negocio de los actos de comunicación indicados.

Regla sendBusinessMessage2WAP5

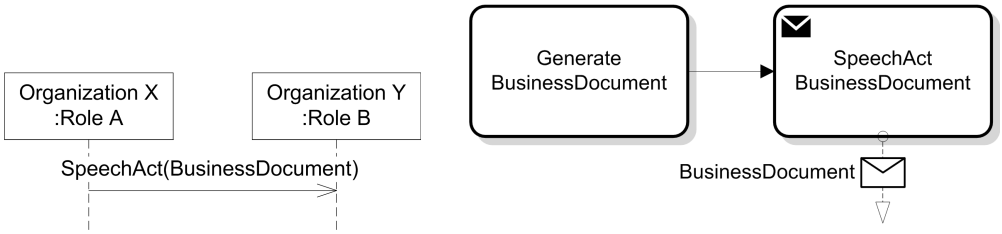
Un mensaje de negocio `UPCOLBPIP!BusinessMessage` cuya intención es `UPCOLBPIP!SpeechAct.name`, la información intercambiada es `UPCOLBPIP!BusinessDocument` y es enviada por el rol `UPCOLBPIP!ParnerRole = PARAMETERS!role` (Figura 5.9(a)), se transforma en dos tareas, una tarea abstracta `BPMN!Task` y una tarea `BPMN!SendTask`; y un flujo de secuencia `BPMN!SequenceFlow` que vincula ambas tareas (Figura 5.9(b)).

Los atributos de `BPMN!Task` se configuran como sigue: al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor *Generate* + `UPCOLBPIP!BusinessDocument.name`; al atributo `completionQuantity` se le asigna el valor *1*, al atributo `isForCompensation` se le asigna el valor *false*; y al atributo `startQuantity`

se le asigna el valor *1*.

Los atributos de `BPMN!SendTask` se configuran como sigue: al atributo `id` se le asigna el valor *sendTask_* + `UPCOLBPIP!BusinessMessage.id`, al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false* y al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.

Los atributos de `BPMN!SequenceFlow` se establecen como sigue: al atributo `id` se le asigna el valor *sequenceFlow_* + `UPCOLBPIP!BusinessMessage.id`; al atributo `isImmediate` se le asigna el valor *true*; al atributo `sourceRef` se le asigna el valor `BPMN!Task` y al atributo `targetRef` se le asigna el valor `BPMN!SendTask`.



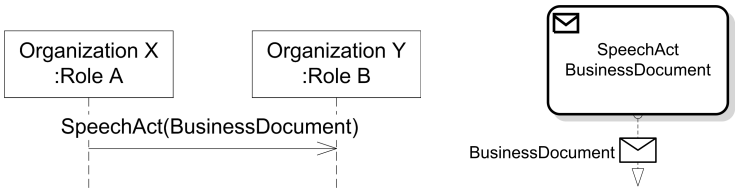
(a) Patrón origen. (b) Patrón destino.

Figura 5.9: Representación gráfica de la regla *sendBusinessMessage2WAP5*.

Regla receiveBusinessMessage2WAP5

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` cuya intención es `UPCOLBPIP!SpeechAct.name`, la información intercambiada es `UPCOLBPIP!BusinessDocument` y es recibida por el rol `UPCOLBPIP!ParnerRole = PARAMETERS!role` (Figura 5.10(a)), se transforma en una tarea `BPMN!ReceiveTask` (Figura 5.10(b)). Al atributo `id` se le asigna el valor `UPCOLBPIP!BusinessMessage.id`; al atributo `name` se le asigna el valor `UPCOLBPIP!SpeechAct.name` + `UPCOLBPIP!BusinessDocument.name`; al atributo `implementation` se le asigna el valor *##WebService*; al atributo `completionQuantity` se le asigna el valor *1*; al atributo `isForCompensation` se le asigna el valor *false* y

al atributo `startQuantity` se le asigna el valor *1*; y al atributo `messageRef` se le asigna el valor `UPCOLBPIP!BusinessDocument.information`.



(a) Patrón origen. (b) Patrón destino.

Figura 5.10: Representación gráfica de la regla *receiveBusinessMessage2WAP5*.

5.2.1.3. Especialización de las reglas de transformación para mensajes de negocio

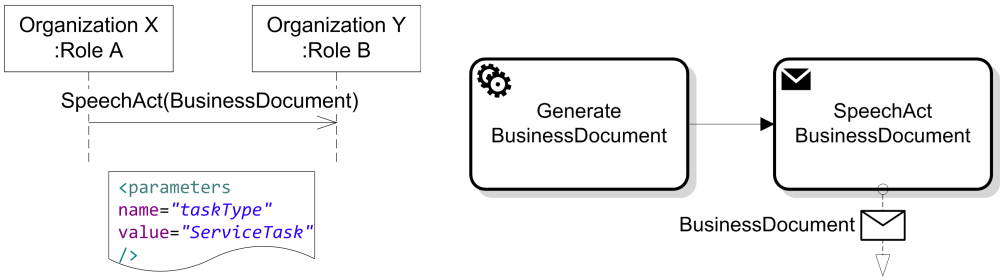
En esta sección se describen las reglas de transformación que configuran las actividades privadas de un modelo de proceso de integración de acuerdo a los parámetros definidos en un modelo de parámetros. Para ello, se especializan las reglas de transformación definidas para los mensajes de negocio agregando una condición ("guard") que evalúa los parámetros de configuración. Estas reglas se aplican en el caso en que se hayan definido parámetros de configuración para los mensajes de negocio del modelo de proceso colaborativo.

Regla sendBusinessMessage2WAP3_ServiceTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ServiceTask* (Figura 5.11(a)), se transforma de acuerdo a la regla *sendBusiness-Message2WAP3* y se genera una tarea `BPMN!ServiceTask` en lugar de una tarea abstracta `BPMN!Task` (Figura 5.11(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.

Si además se definió un parámetro `PARAMETERS!operation` para el mensaje de negocio, se genera una interface `BPMN!Interface` con el propósito de definir las operaciones que serán invocadas por la tarea `BPMN!ServiceTask`. Los atributos de `BPMN!Interface` se definen como sigue: al atributo `id` se le asigna el valor *interface_* + `UPCOLBPIP!BusinessMessage.id`, al atributo

name se le asigna el valor UPCOLBPIP!BusinessMessage.name + *Interface* y al atributo operations se le asigna una operación BPMN!Operation para especificar la operación que será definida como parte de la interface. Los atributos de BPMN!Operation se definen como sigue: al atributo id se le asigna el valor *operation_* + UPCOLBPIP!BusinessMessage.id, al atributo name se le asigna el valor de PARAMETERS!operation. El atributo operationRef de la tarea BPMN!ServiceTask hace referencia a dicha operación.



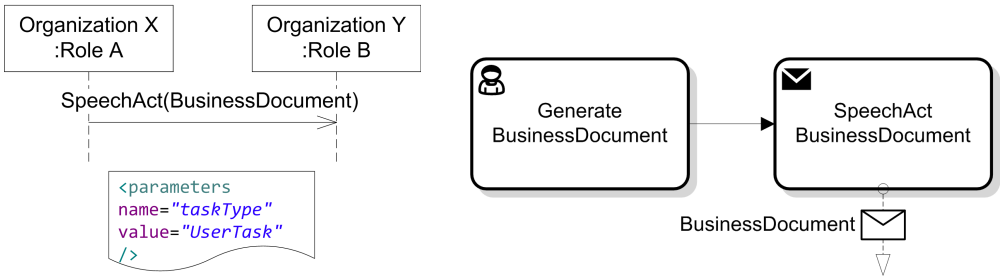
(a) Patrón origen. (b) Patrón destino.

Figura 5.11: Representación gráfica de la regla *sendBusinessMessage2WAP3_-ServiceTask*.

Regla sendBusinessMessage2WAP3_UserTask

Un mensaje de negocio UPCOLBPIP!BusinessMessage que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro PARAMETERS!taskType cuyo valor es *UserTask* (Figura 5.12(a)), se transforma de acuerdo a la regla *sendBusiness-Message2WAP3* y se genera una tarea BPMN!UserTask en lugar de una tarea abstracta (Figura 5.12(b)). A su atributo implementation se le asigna el valor de PARAMETERS!implementation.

Si además se definió un parámetro PARAMETERS!performer para el mensaje de negocio, al atributo resources de la tarea BPMN!UserTask se le asigna un BPMN!Performer para indicar quién es responsable de la tarea. Los atributos de BPMN!Performer se establecen como sigue: al atributo id se le asigna el valor *performer_* + UPCOLBPIP!BusinessMessage.id y a su atributo name se le asigna el valor de PARAMETERS!performer.

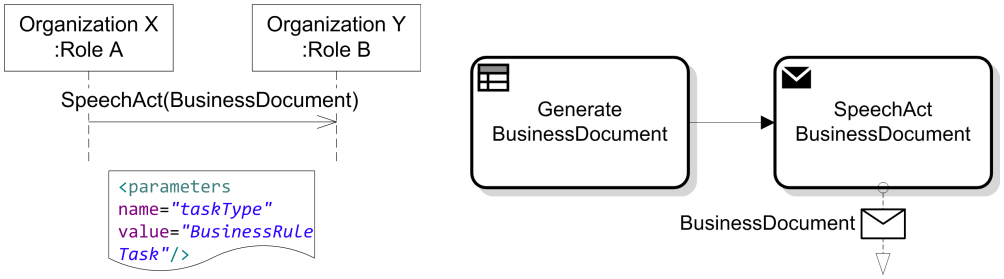


(a) Patrón origen. (b) Patrón destino.

Figura 5.12: Representación gráfica de la regla *sendBusinessMessage2WAP3_-UserTask*.

Regla sendBusinessMessage2WAP3_BusinessRuleTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *BusinessRuleTask* (Figura 5.13(a)), se transforma de acuerdo a la regla *sendBusinessMessage2WAP3* y se genera una tarea `BPMN!BusinessRuleTask` en lugar de una tarea abstracta (Figura 5.13(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.



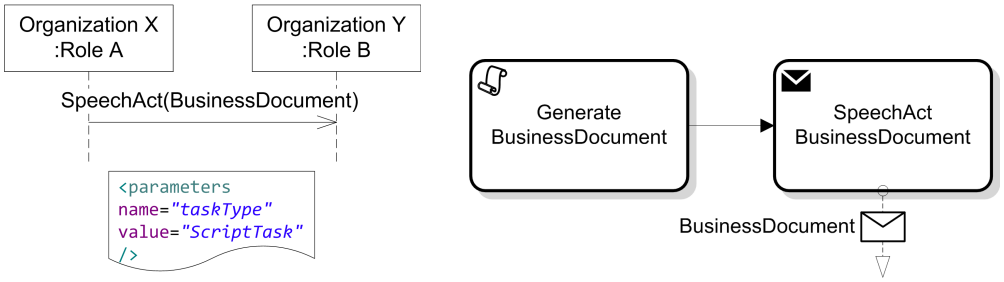
(a) Patrón origen. (b) Patrón destino.

Figura 5.13: Representación gráfica de la regla *sendBusinessMessage2WAP3_-BusinessRuleTask*.

Regla sendBusinessMessage2WAP3_ScriptTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ScriptTask* (Figura 5.14(a)), se transforma de acuerdo a la regla *sendBusiness-*

Message2WAP3 y se genera una tarea `BPMN!ScriptTask` en lugar de una tarea abstracta (Figura 5.14(b)). A su atributo `script` se le asigna el valor de `PARAMETERS!script` y a su atributo `scriptFormat` se le asigna el valor de `PARAMETERS!scriptFormat`.

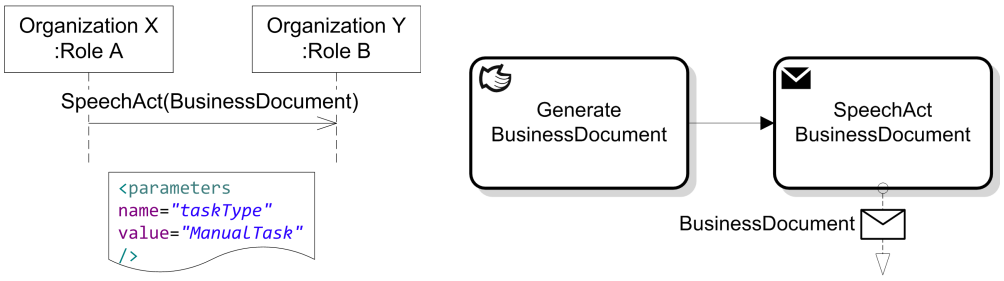


(a) Patrón origen. (b) Patrón destino.

Figura 5.14: Representación gráfica de la regla *sendBusinessMessage2WAP3_ScriptTask*.

Regla sendBusinessMessage2WAP3_ManualTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ManualTask* (Figura 5.15(a)), se transforma de acuerdo a la regla *sendBusinessMessage2WAP3* y se genera una tarea `BPMN!ManualTask` en lugar de una tarea abstracta (Figura 5.15(b)).



(a) Patrón origen. (b) Patrón destino.

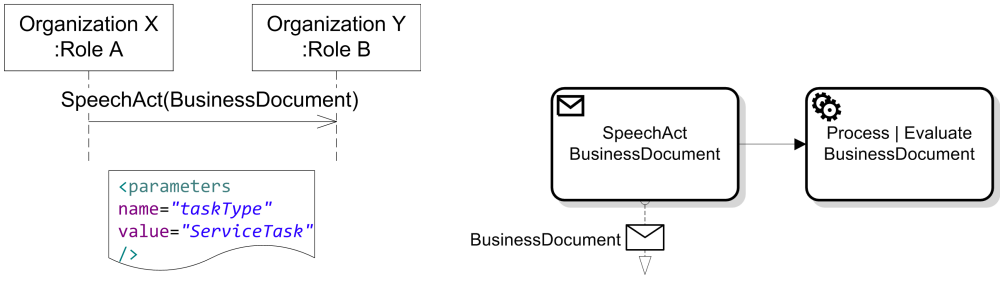
Figura 5.15: Representación gráfica de la regla *sendBusinessMessage2WAP3_ManualTask*.

Si además se definió un parámetro `PARAMETERS!performer` para el mensaje de negocio, al atributo `resources` de la tarea `BPMN!ManualTask` se le

asigna un BPMN!Performer para indicar quién es responsable de la tarea. Los atributos de BPMN!Performer se establecen como sigue: al atributo id se le asigna el valor *performer_* + UPCOLBPIP!BusinessMessage.id y a su atributo name se le asigna el valor de PARAMETERS!performer.

Regla receiveBusinessMessage2WAP3_ServiceTask

Un mensaje de negocio UPCOLBPIP!BusinessMessage que satisface las condiciones definidas para la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro PARAMETERS!taskType cuyo valor es *ServiceTask* (Figura 5.16(a)), se transforma de acuerdo a la regla *receiveBusinessMessage2WAP3* y se genera una tarea BPMN!ServiceTask en lugar de una tarea abstracta (Figura 5.16(b)). A su atributo implementation se le asigna el valor de PARAMETERS!implementation.



(a) Patrón origen. (b) Patrón destino.

Figura 5.16: Representación gráfica de la regla *receiveBusinessMessage2WAP3_ServiceTask*.

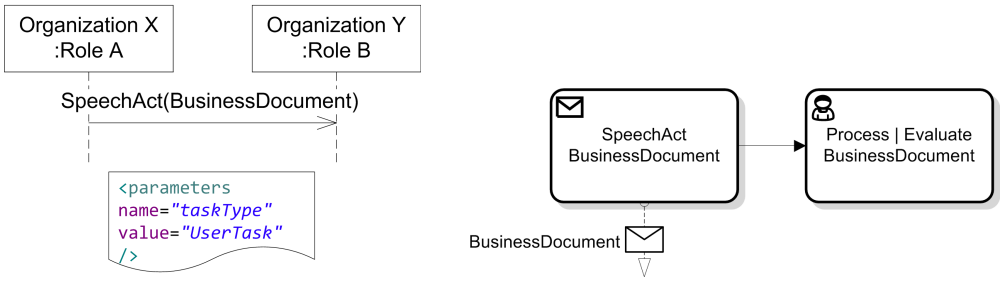
Si además se definió un parámetro PARAMETERS!operation para el mensaje de negocio, se genera una interface BPMN!Interface con el propósito de definir las operaciones que serán invocadas por la tarea BPMN!ServiceTask. Los atributos de BPMN!Interface se definen como sigue: al atributo id se le asigna el valor *interface_* + UPCOLBPIP!BusinessMessage.id, al atributo name se le asigna el valor UPCOLBPIP!BusinessMessage.name + *Interface* y al atributo operations se le asigna una operación BPMN!Operation para especificar la operación que será definida como parte de la interface. Los atributos de BPMN!Operation se definen como sigue: al atributo id se le asigna el valor *operation_* + UPCOLBPIP!BusinessMessage.id, al atributo name se le asigna el valor de PARAMETERS!operation. El atributo operationRef

de la tarea `BPMN!ServiceTask` hace referencia a dicha operación.

Regla receiveBusinessMessage2WAP3_UserTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *UserTask* (Figura 5.17(a)), se transforma de acuerdo a la regla *receiveBusinessMessage2WAP3* y se genera una tarea `BPMN!UserTask` en lugar de una tarea abstracta (Figura 5.17(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.

Si además se definió un parámetro `PARAMETERS!performer` para el mensaje de negocio, al atributo `resources` de la tarea `BPMN!UserTask` se le asigna un `BPMN!Performer` para indicar quién es responsable de la tarea. Los atributos de `BPMN!Performer` se establecen como sigue: al atributo `id` se le asigna el valor *performer_* + `UPCOLBPIP!BusinessMessage.id` y a su atributo `name` se le asigna el valor de `PARAMETERS!performer`.



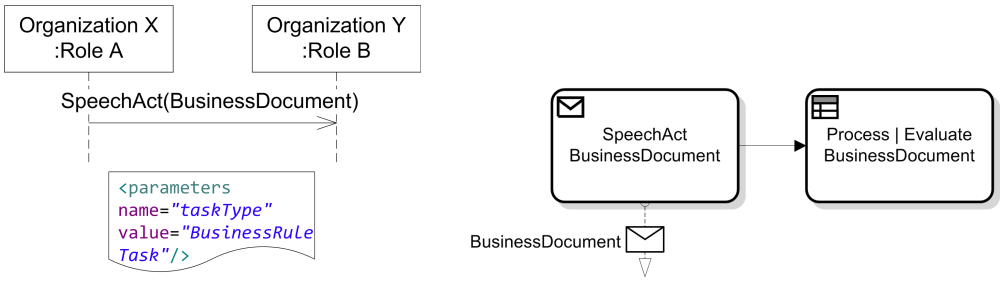
(a) Patrón origen. (b) Patrón destino.

Figura 5.17: Representación gráfica de la regla *receiveBusinessMessage2WAP3_-UserTask*.

Regla receiveBusinessMessage2WAP3_BusinessRuleTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es `BusinessRuleTask` (Figura 5.18(a)), se transforma de acuerdo a la regla *receiveBusinessMessage2WAP3* y se genera una tarea `BPMN!BusinessRuleTask` en lugar de una tarea abstracta (Figu-

ra 5.18(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.

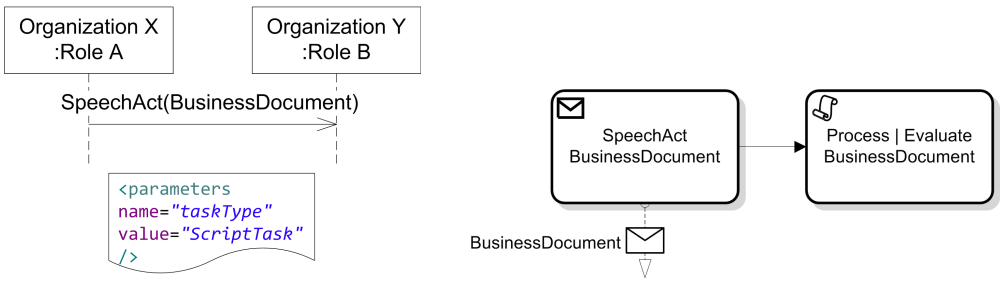


(a) Patrón origen. (b) Patrón destino.

Figura 5.18: Representación gráfica de la regla *receiveBusinessMessage2WAP3_-BusinessRuleTask*.

Regla receiveBusinessMessage2WAP3_ScriptTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ScriptTask* (Figura 5.19(a)), se transforma de acuerdo a la regla *receive-BusinessMessage2WAP3* y se genera una tarea `BPMN!ScriptTask` en lugar de una tarea abstracta (Figura 5.19(b)). A su atributo `script` se le asigna el valor de `PARAMETERS!script` y a su atributo `scriptFormat` se le asigna el valor de `PARAMETERS!scriptFormat`.



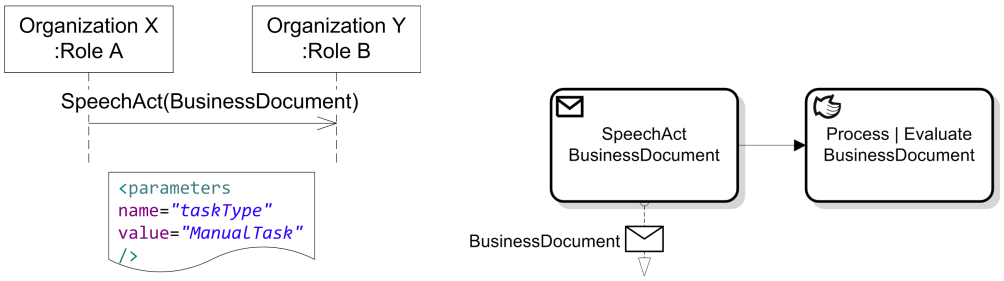
(a) Patrón origen. (b) Patrón destino.

Figura 5.19: Representación gráfica de la regla *receiveBusinessMessage2WAP3_-ScriptTask*.

Regla receiveBusinessMessage2WAP3_ManualTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ManualTask* (Figura 5.20(a)), se transforma de acuerdo a la regla *receiveBusinessMessage2WAP3* y se genera una tarea `BPMN!ManualTask` en lugar de una tarea abstracta (Figura 5.20(b)).

Si además se definió un parámetro `PARAMETERS!performer` para el mensaje de negocio, al atributo `resources` de la tarea `BPMN!ManualTask` se le asigna un `BPMN!Performer` para indicar quién es responsable de la tarea. Los atributos de `BPMN!Performer` se establecen como sigue: al atributo `id` se le asigna el valor *performer_* + `UPCOLBPIP!BusinessMessage.id` y a su atributo `name` se le asigna el valor de `PARAMETERS!performer`.



(a) Patrón origen. (b) Patrón destino.

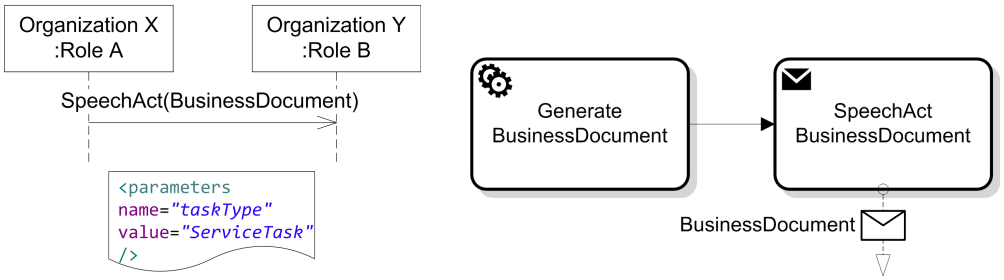
Figura 5.20: Representación gráfica de la regla *receiveBusinessMessage2WAP3_ManualTask*.

Regla sendBusinessMessage2WAP5_ServiceTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ServiceTask* (Figura 5.21(a)), se transforma de acuerdo a la regla *sendBusinessMessage2WAP5* y se genera una tarea `BPMN!ServiceTask` en lugar de una tarea abstracta (Figura 5.21(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.

Si además se definió un parámetro `PARAMETERS!operation` para el mensaje de negocio, se genera una interface `BPMN!Interface` con el propósito de

definir las operaciones que serán invocadas por la tarea `BPMN!ServiceTask`. Los atributos de `BPMN!Interface` se definen como sigue: al atributo `id` se le asigna el valor *interface_* + `UPCOLBPIP!BusinessMessage.id`, al atributo `name` se le asigna el valor `UPCOLBPIP!BusinessMessage.name` + *Interface* y al atributo `operations` se le asigna una operación `BPMN!Operation` para especificar la operación que será definida como parte de la interface. Los atributos de `BPMN!Operation` se definen como sigue: al atributo `id` se le asigna el valor *operation_* + `UPCOLBPIP!BusinessMessage.id`, al atributo `name` se le asigna el valor de `PARAMETERS!operation`. El atributo `operationRef` de la tarea `BPMN!ServiceTask` hace referencia a dicha operación.



(a) Patrón origen. (b) Patrón destino.

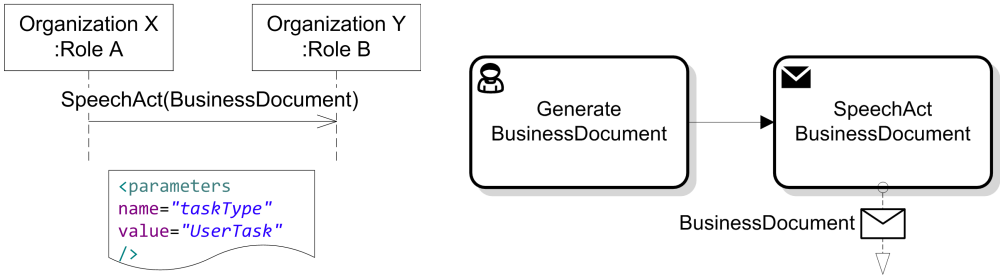
Figura 5.21: Representación gráfica de la regla *sendBusinessMessage2WAP5_ServiceTask*.

Regla sendBusinessMessage2WAP5_UserTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *UserTask* (Figura 5.22(a)), se transforma de acuerdo a la regla *sendBusinessMessage2WAP5* y se genera una tarea `BPMN!UserTask` en lugar de una tarea abstracta (Figura 5.22(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.

Si además se definió un parámetro `PARAMETERS!performer` para el mensaje de negocio, al atributo `resources` de la tarea `BPMN!UserTask` se le asigna un `BPMN!Performer` para indicar quién es responsable de la tarea. Los atributos de `BPMN!Performer` se establecen como sigue: al atributo `id` se le asigna el valor *performer_* + `UPCOLBPIP!BusinessMessage.id` y a su atributo `name`

se le asigna el valor de `PARAMETERS!performer`.

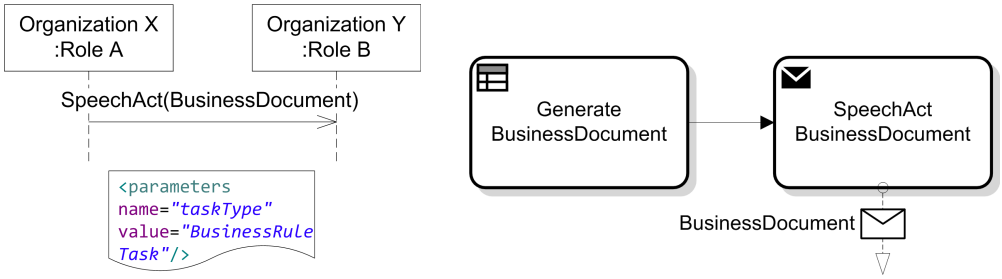


(a) Patrón origen. (b) Patrón destino.

Figura 5.22: Representación gráfica de la regla *sendBusinessMessage2WAP5_-UserTask*.

Regla sendBusinessMessage2WAP5_BusinessRuleTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *BusinessRuleTask* (Figura 5.23(a)), se transforma de acuerdo a la regla*sendBusinessMessage2WAP5* y se genera una tarea `BPMN!BusinessRuleTask` en lugar de una tarea abstracta (Figura 5.23(b)). A su atributo `implementation` se le asigna el valor de `PARAMETERS!implementation`.



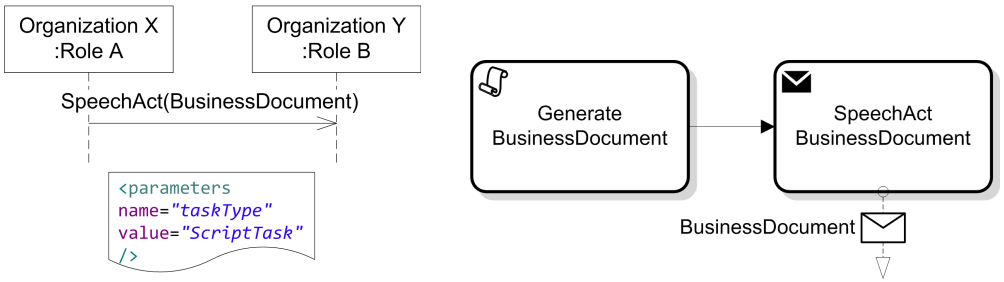
(a) Patrón origen. (b) Patrón destino.

Figura 5.23: Representación gráfica de la regla *sendBusinessMessage2WAP5_-BusinessRuleTask*.

Regla sendBusinessMessage2WAP5_ScriptTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2)

y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ScriptTask* (Figura 5.24(a)), se transforma de acuerdo a la regla *sendBusinessMessage2WAP5* y se genera una tarea `BPMN!ScriptTask` en lugar de una tarea abstracta (Figura 5.24(b)). A su atributo `script` se le asigna el valor de `PARAMETERS!script` y a su atributo `scriptFormat` se le asigna el valor de `PARAMETERS!scriptFormat`.

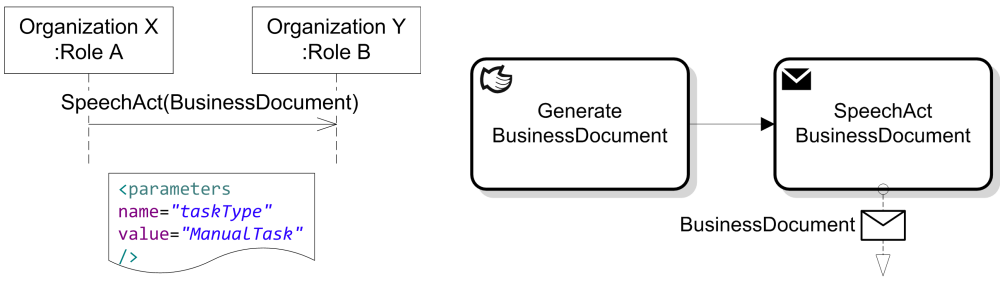


(a) Patrón origen. (b) Patrón destino.

Figura 5.24: Representación gráfica de la regla *sendBusinessMessage2WAP5_ScriptTask*.

Regla sendBusinessMessage2WAP5_ManualTask

Un mensaje de negocio `UPCOLBPIP!BusinessMessage` que satisface las condiciones definidas para la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2) y además tiene definido un parámetro `PARAMETERS!taskType` cuyo valor es *ManualTask* (Figura 5.25(a)), se transforma de acuerdo a la regla*sendBusinessMessage2WAP5* y se genera una tarea `BPMN!ManualTask` en lugar de una tarea abstracta (Figura 5.25(b)).



(a) Patrón origen. (b) Patrón destino.

Figura 5.25: Representación gráfica de la regla *sendBusinessMessage2WAP5_ManualTask*.

Si además se definió un parámetro PARAMETERS!performer para el mensaje de negocio, al atributo resources de la tarea BPMN!ManualTask se le asigna un BPMN!Performer para indicar quién es responsable de la tarea. Los atributos de BPMN!Performer se establecen como sigue: al atributo id se le asigna el valor *performer_* + UPCOLBPIP!BusinessMessage.id y a su atributo name se le asigna el valor de PARAMETERS!performer.

5.2.2. Transformación de los restantes elementos de un modelo de proceso colaborativo

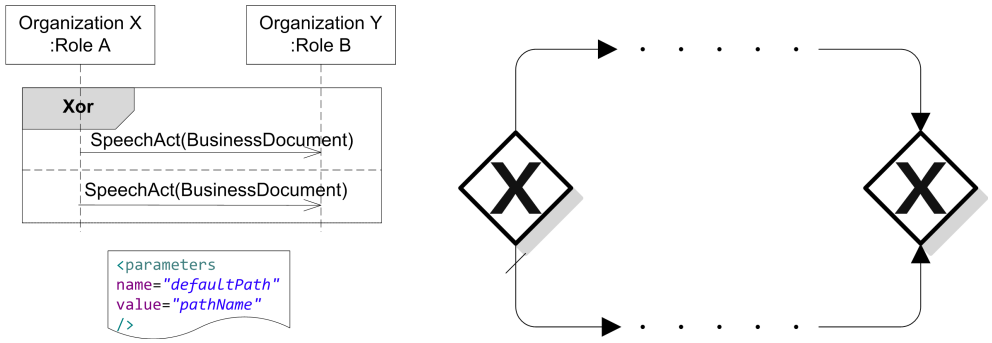
El método re-usa las reglas de transformación correspondientes a los restantes elementos del proceso colaborativo (por ejemplo, segmentos de flujo de control, protocolos de referencia, restricciones de tiempo y eventos de terminación) del método basado en MDA para la generación automática de procesos de interfaz descrito en el Capítulo 4. La reusabilidad en la ingeniería dirigida por modelos, del mismo modo que la ingeniería de software, es considerada fundamental para lograr productividad y calidad (mantenibilidad, confiabilidad y exactitud), reduciendo el esfuerzo y costo de desarrollo (Sánchez Cuadrado y García Molina, 2008; Wimmer y otros, 2012).

Para definir y configurar los segmentos de flujo de control, también se especializan las reglas de transformación correspondientes re-usando las definidas en el Capítulo 4 y agregando un "guard" que evalúa los parámetros de configuración.

Las reglas presentadas a continuación se aplican al modelo de entrada en el caso de que se hayan definido parámetros a los elementos del mismo.

5.2.2.1. Regla cfsXor2ExclusiveGateway_defaultPath

Un segmento de flujo de control UPCOLBPIP!Xor que satisface las condiciones definidas para la regla *cfsXor2ExclusiveGateway* para el rol que envía el primer mensaje de cada camino (Sección 4.1.1.7) y además tiene definido un parámetro PARAMETERS!defaultPath (Figura 5.26(a)), se transforma como se definió en dicha regla y además se asigna al atributo default el camino por defecto definido en el parámetro (Figura 5.26(b)).

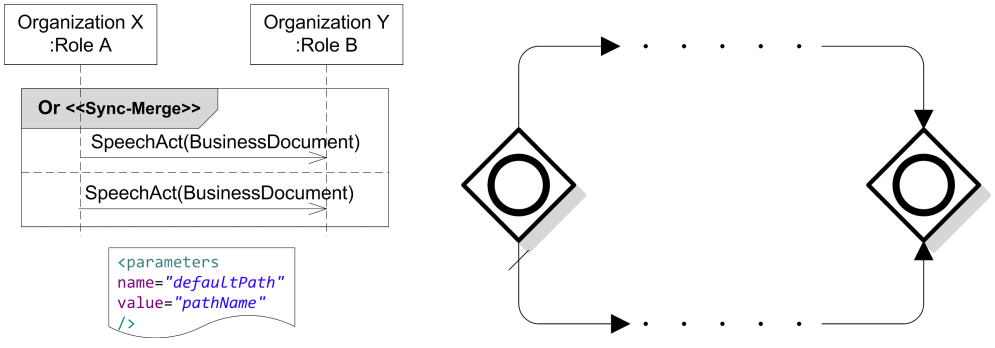


(a) Patrón origen. (b) Patrón destino.

Figura 5.26: Representación gráfica de la regla *cfsXor2ExclusiveGateway_defaultPath*.

5.2.2.2. Regla cfsOrSynchronizingMerge2InclusiveGateway_defaultPath

Un segmento de flujo de control `UPCOLBPIP!Or` que satisface las condiciones definidas para la regla *cfsOrSynchronizingMerge2InclusiveGateway* (Sección 4.1.1.7) y además tiene definido un parámetro `PARAMETERS!defaultPath` (Figura 5.27(a)), se transforma como se definió en dicha regla y además se asigna al atributo `default` el camino por defecto definido en el parámetro (Figura 5.27(b)).



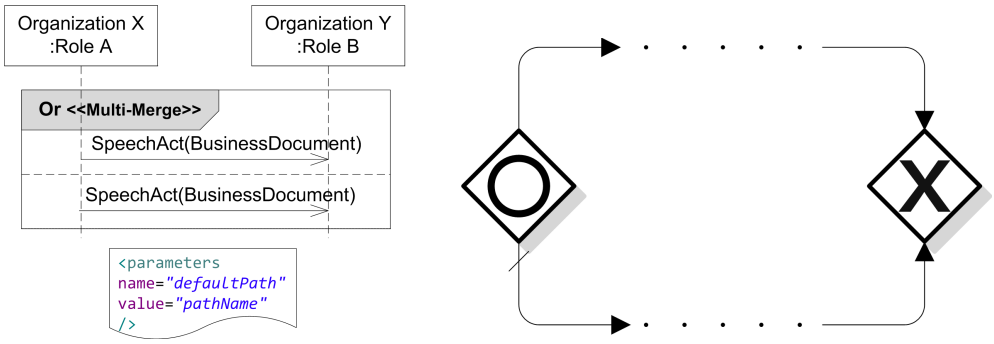
(a) Patrón origen. (b) Patrón destino.

Figura 5.27: Representación gráfica de la regla *cfsOrSynchronizingMerge2InclusiveGateway_defaultPath*.

5.2.2.3. Regla cfsOrMultiMerge2InclusiveGateway_defaultPath

Un segmento de flujo de control `UPCOLBPIP!Or` que satisface las condiciones definidas para la regla *cfsOrMultiMerge2InclusiveGateway* (Sección 4.1.1.7)

y además tiene definido un parámetro `PARAMETERS!defaultPath` (Figura 5.28(a)), se transforma como se definió en dicha regla y además se asigna al atributo `default` el camino por defecto definido en el parámetro (Figura 5.28(b)).

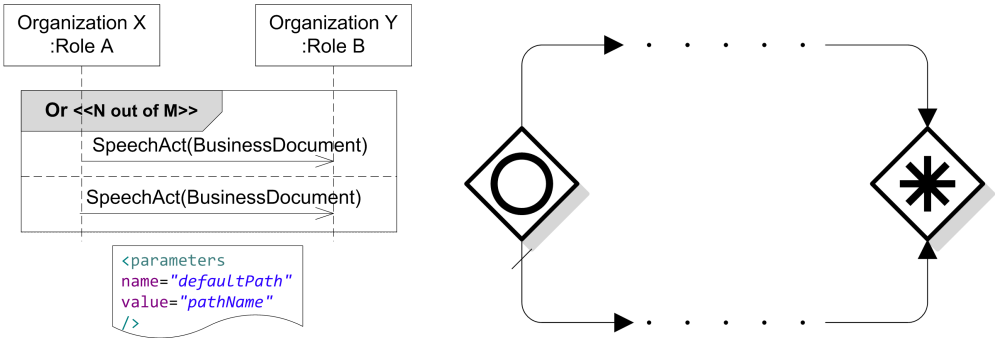


(a) Patrón origen. (b) Patrón destino.

Figura 5.28: Representación gráfica de la regla *cfsOrMultiMerge2InclusiveGateway_defaultPath*.

5.2.2.4. Regla cfsOrNoutM2ComplexGateway_defaultPath

Un segmento de flujo de control `UPCOLBPIP!Or` que satisface las condiciones definidas para la regla *cfsOrNoutM2ComplexGateway* (Sección 4.1.1.7) y además tiene definido un parámetro `PARAMETERS!defaultPath` (Figura 5.29(a)), se transforma como se definió en dicha regla y además se asigna al atributo `default` el camino por defecto definido en el parámetro (Figura 5.29(b)).



(a) Patrón origen. (b) Patrón destino.

Figura 5.29: Representación gráfica de la regla *cfsOrNoutM2ComplexGateway_defaultPath*.

5.2.2.5. Regla condition2Expression_FormalExpression

Una condición `UPCOLBPIP!Condition` que satisface las condiciones definidas para la regla *condition2Expression* (Sección 4.1.1.12) y además tiene definido un parámetro `PARAMETERS!expression` se transforma como se definió en dicha regla y se genera una expresión formal `BPMN!FormalExpression`, en lugar de una expresión en lenguaje natural `BPMN!Expression`. A su atributo `body` se le asigna el valor de `PARAMETERS!expression` y a su atributo `language` se le asigna el valor de `PARAMETERS!language`.

5.3. Implementación del método

El método presentado fue implementado utilizando la aplicación basada en Eclipse Upcolbpip2bpmn (Apéndice B.4) descrita en el Capítulo 4.

En la implementación de la definición de transformación, se aplicó la técnica de superimposición con el propósito de re-usar la definición de transformación upcolbpip2bpmn.atl (Apéndice B.1) definida para el método que genera los modelos de procesos de interfaz (Capítulo 4). En dicha definición se especializó la regla *condition2Expression* con el fin de definir la expresión que será evaluada en un camino de interacción si ésta fue definida en el modelo de parámetros (Sección 5.2.2.5). La línea 6 del Listado 5.1 muestra la implementación de la regla *condition2Expression* y la línea 19 muestra la implementación de la regla *condition2Expression_FormalExpression* que especializa la misma.

También se especializaron las reglas correspondientes a los segmentos de flujo de control *Xor* y *Or* con el propósito de definir el flujo de secuencia por defecto en caso que se hayan definido parámetros de configuración para los mismos (Secciones 5.2.2.1, 5.2.2.2, 5.2.2.3 y 5.2.2.4).

Listado 5.1: Fragmento de la definición de transformación *upcolbpip2bpmn.atl*

```
1 ...
2 ------------------------------------------------------------------------
3 -------------- Condition to Expression ---------------------------------
4 -------------- without parameters --------------------------------------
5 ------------------------------------------------------------------------
6 rule condition2Expression {
7   from
8     guard: MMupcolbpip!Condition
9   to
```

```
10     expression: MMbpmn2!Expression (
11       id <- guard.ConditionExpression
12     )
13 }
14
15 ------------------------------------------------------------------------
16 -------------- Condition to Expression ---------------------------------
17 -------------- with parameters -----------------------------------------
18 ------------------------------------------------------------------------
19 rule condition2Expression_FormalExpression extends  condition2Expression {
20   from
21     guard: MMupcolbpip!Condition
22     (thisModule.hasParameter(guard.refImmediateComposite().refGetValue('name'), '
           expression'))
23   to
24     expression: MMbpmn2!FormalExpression (
25       id <- guard.ConditionExpression,
26       language <- thisModule.getParameterValue(guard.refImmediateComposite().
             refGetValue('name'),'language'),
27       body <- thisModule.getParameterValue(guard.refImmediateComposite().
             refGetValue('name'), 'expresssion')
28     )
29 }
```

Para implementar las reglas correspondientes a los mensajes de negocio se especificó una definición de transformación denominada protocol2integration.atl (Apéndice B.3). El Listado 5.2 muestra un fragmento de dicha definición. Como ejemplo, la línea 6 muestra la implementación de la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1). La línea 49 muestra la implementación de la regla *sendBusinessMessage2WAP3_UserTask* (Sección 5.2.1.3) para el caso de que no se haya definido un parámetro `performer`. La línea 67 muestra la implementación de la misma regla para el caso de que se haya definido un parámetro `performer` al mensaje de negocio.

Listado 5.2: Fragmento de la definición de transformación protocol2integration.atl

```
1 ...
2 ------------------------------------------------------------------------
3 ---- Send Business Message to WAP3 - Sender role -----------------------
4 ---- without parameters ------------------------------------------------
5 ------------------------------------------------------------------------
6 rule sendBusinessMessage2WAP3 {
7   from
8     businessMessage: MMupcolbpip!BusinessMessage (
9       businessMessage.isSenderRequiredRole(thisModule.getParameterValue('roleId','
            role')) and
10       (businessMessage.isInform or businessMessage.isCallForProposal or
```

```
            businessMessage.isPropose or businessMessage.isRequest)
    )
  to
    Task: MMbpmn2!Task
    (
      id <- 'abstractTask_' + businessMessage.id,
      name <- 'Generate ' + businessMessage.information.name,
      completionQuantity <- 1,
      isForCompensation <- false,
      startQuantity <- 1
    ),
    sendTask: MMbpmn2!SendTask (
      id <- 'sendTask_' + businessMessage.id,
      name <- 'Send ' + businessMessage.intention + ' ' + businessMessage.
          information.name,
      implementation <- '##WebService',
      completionQuantity <- 1,
      isForCompensation <- false,
      startQuantity <- 1,
      messageRef <- businessMessage.information,
      outgoing <- if thisModule.getAllipElementsForSubprocess() ->
          includes(businessMessage) then
          thisModule.successorSequenceFlowForSubprocess(businessMessage)
        else
          thisModule.successorSequenceFlowForProcess(businessMessage)
        endif
    ),
    internalSequenceFlow: MMbpmn2!SequenceFlow (
      id <- 'sequenceFlow_' + businessMessage.id,
      isImmediate <- true,
      sourceRef <- Task,
      targetRef <- sendTask
    )
}

-------------------------------------------------------------------------
---- Send Business Message to WAP3 - Sender role ----------------------
---- with parameters:  taskType= UserTask -----------------------------
-------------------------------------------------------------------------
rule sendBusinessMessage2WAP3_UserTask extends sendBusinessMessage2WAP3 {
  from
    businessMessage: MMupcolbpip!BusinessMessage (
       thisModule.hasParameterValue(businessMessage.id, 'taskType', 'UserTask')
    )
  to
    Task: MMbpmn2!UserTask
    (
      id <- 'userTask_' + businessMessage.id,
      implementation <- thisModule.getParameterValue(businessMessage.id, '
          implementation')
    )
```

```
60 }
61
62 ------------------------------------------------------------------------
63 ---- Send Business Message to WAP3 - Sender role -----------------------
64 ---- with parameters:  taskType= UserTask ------------------------------
65 ----- and performer ----------------------------------------------------
66 ------------------------------------------------------------------------
67 rule sendBusinessMessage2WAP3_UserTaskWithPeformer extends
       sendBusinessMessage2WAP3_UserTask {
68   from
69     businessMessage: MMupcolbpip!BusinessMessage (
70        thisModule.hasParameter(businessMessage.id, 'performer')
71     )
72   to
73     Task: MMbpmn2!UserTask (
74       resources <- performer
75     ),
76     performer: MMbpmn2!Performer (
77       id <- 'performer_'  + businessMessage.id,
78       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
79     )
80 }
81 ...
```

El Listado 5.3 muestra la librería parameterHelpers.atl, la cual contiene los "helpers" que permiten consultar los parámetros definidos en un modelo de parámetros. En la línea 5 se muestra el "helper" *hasParameter*, el cual devuelve un valor booleano indicando si un elemento identificado con un `id` tiene definido un parámetro `Parameter`.

Listado 5.3: Librería parameterHelpers.atl

```
1 -- @path MMParam=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/parameters.
       ecore
2
3 library parameterHelpers;
4
5 helper def: hasParameter(id: String, parameter:String): Boolean =
6   MMParam!Parameter.allInstances() -> select (e | e.name = parameter)
7                   -> select ( e | e.refImmediateComposite() = thisModule.
                        getElementParameter(id)).notEmpty();
8
9 helper def: getElementParameter(id: String): MMParam!Element =
10  MMParam!Element.allInstances() -> any(e | e.ID = id);
11
12 helper def: hasParameterValue(id: String, parameter:String, value: String):
       Boolean =
13  MMParam!Parameter.allInstances() -> select (e | e.name = parameter and e.value =
         value)
```

```
14                    -> select( e | e.refImmediateComposite() = thisModule.
                          getElementParameter(id))
15                    -> notEmpty();
16
17 helper def: getParameterValue(id: String, parameter:String): String =
18   MMParam!Parameter.allInstances() -> select (e | e.name = parameter)
19                    -> any ( e | e.refImmediateComposite() = thisModule.
                          getElementParameter(id)).value;
```

Para ejecutar automáticamente el método, se modificó el archivo de propiedades Upcolbpip2bpmn.properties (Apéndice B.5) del plug-in, mostrado en el Listado 5.4, para indicar que se usarán los módulos upcolbpip2bpmn.atl (Apéndice B.1) y protocol2integration.atl (Apéndice B.3) en la superimposición (línea 2).

Listado 5.4: Archivo de propiedades Upcolbpip2bpmn.properties del plug-in

```
1 ...
2 Upcolbpip2bpmn.modules = upcolbpip2bpmn.atl,protocol2integration.atl
3 ...
```

El modelo BPMN serializado en formato XMI (ya sea la plantilla o un modelo de proceso de integración) se transforma en un modelo BPMN en formato XML, de acuerdo al formato de intercambio de archivos del estándar BPMN2, aplicando la plantilla XMI2XML.xslt propuesta por Hille-Doering (2010). Finalmente, este modelo BPMN en formato XML se valida contra el esquema XSD BPMN20.xsd propuesto por el estándar BPMN2 OMG (2011a) para determinar si el modelo está bien formado.

La Figura 5.30 muestra y resume la cadena de transformaciones realizadas para generar un modelo de proceso de integración bien formado a partir de un modelo de proceso colaborativo.

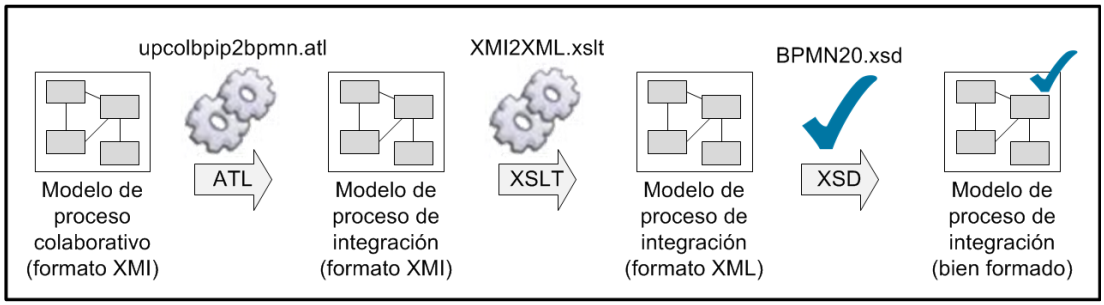


Figura 5.30: Cadena de transformaciones realizadas para obtener un modelo de proceso de integración bien formado.

5.4. Aplicación del método a un caso de estudio

Para demostrar la funcionalidad del método, en esta sección se aplica el mismo al caso de estudio desarrollado en los Capítulos 3 y 4. Se describe la transformación del modelo de proceso colaborativo *Collaborative Replenishment Plan* (Figura 5.31), considerando el rol *Supplier* desempeñado por la organización Sanx que genera el modelo de proceso de integración de dicha organización.

En las siguientes sub-secciones se detalla el proceso de transformación para generar la plantilla de proceso de integración y luego la generación del modelo de proceso de integración.

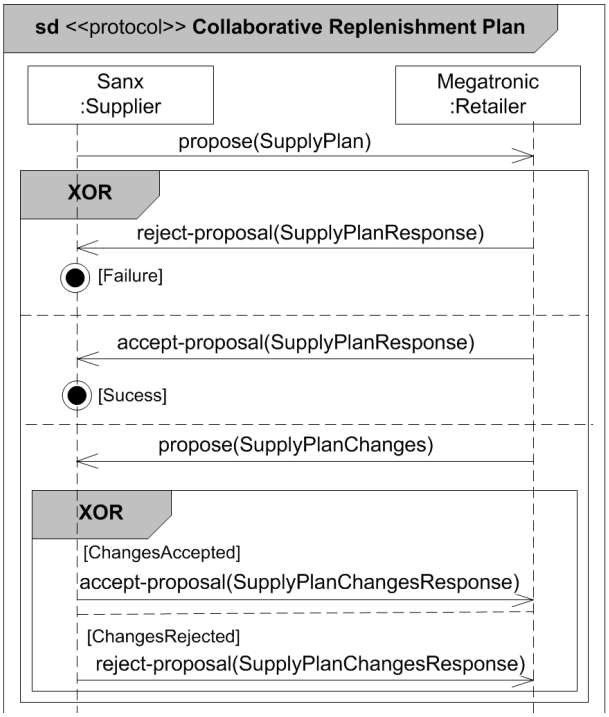


Figura 5.31: Modelo de proceso colaborativo *Collaborative Replenishment Plan* (entrada del método).

5.4.1. Generando una plantilla de proceso de integración

Para generar una plantilla del proceso de integración, el usuario selecciona el modelo de proceso colaborativo a transformar. Luego selecciona el rol a considerar en la transformación e indica el tipo de modelo que va a generar, en este caso una plantilla. Cuando el usuario selecciona dicho rol, en este caso el rol *Supplier*,

se genera automáticamente el modelo de parámetros que es usado como entrada al método propuesto, junto con el modelo de proceso colaborativo.

A continuación, el método genera la plantilla del proceso de integración del proveedor (Figura 5.32). En este caso, como el modelo de parámetros sólo contiene el rol, las actividades privadas se definen como abstractas y son nodos configurables (en el diagrama corresponde a las actividades que no tienen indicado su tipo). La Figura 5.33 muestra la plantilla como una instancia (modelo EMF) del meta-modelo BPMN basado en EMF.

El modelo UP-ColBPIP se transformó en una `BPMN!Definitions` (Figura 5.33), aplicando la regla *UPColBPIPModel2Definitions* (Sección 4.1.1.1)

La colaboración inter-organizacional se transforma en una `BPMN!Collaboration` (Figura 5.32), aplicando la regla *b2bCollaboration2Collaboration* (Sección 4.1.1.2). Cada organización se transforma en un `BPMN!Participant`, aplicando la regla *tradingPartner2Participant* (Sección 4.1.1.4), y su respectivo rol en un `BPMN!PartnerRole`, aplicando la regla *partnerRole2PartnerRole* (Sección 4.1.1.5).

El proceso colaborativo se transforma en un `BPMN!Process`, aplicando la regla *collaborativeProcess2Process* (Sección 4.1.1.3). Como indica dicha regla, se agregó un `BPMN!StartEvent` para indicar el inicio del proceso. El mensaje de negocio enviado por el proveedor se transforma en dos tareas, una tarea `BPMN!Task` y una tarea `BPMN!SendTask`, aplicando la regla *sendBusinessMessage2WAP3* (Sección 5.2.1.1).

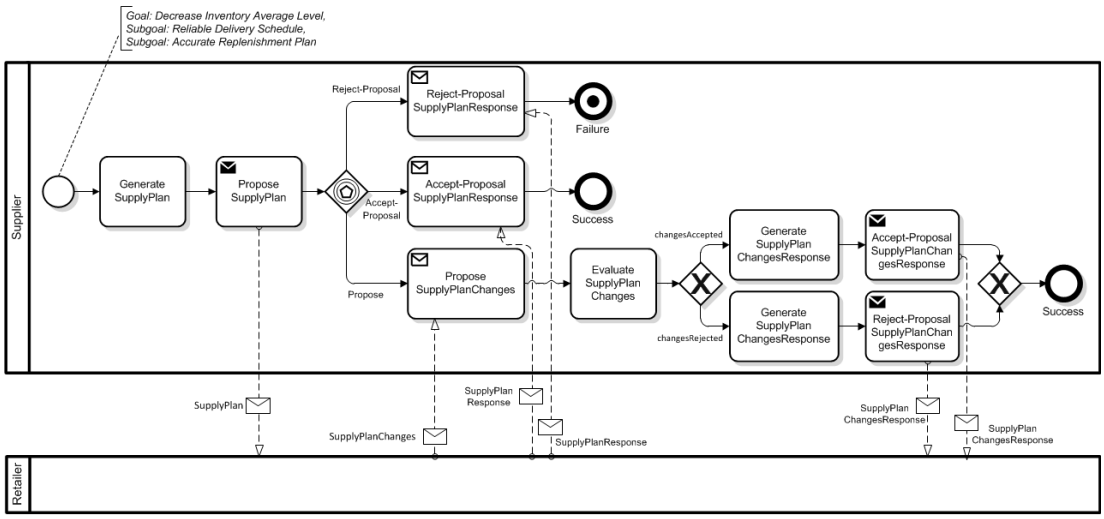


Figura 5.32: Plantilla del proceso de integración del proveedor.

Figura 5.33: Plantilla del proceso de integración del proveedor como instancia del meta-modelo BPMN.

El segmento de flujo de control *Xor* se transforma en un `BPMN!EventBasedGateway`, aplicando la regla *cfsXor2ExclusiveGateway* (Sección 4.1.1.7). Cada camino de interacción se transforma en un `BPMN!SequenceFlow`, aplicando la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14). Luego, se transforman los elementos definidos en cada camino. El mensaje de negocio *reject-proposal(SupplyPlanResponse)* del primer camino se transforma en una `BPMN!ReceiveTask`, aplicando la regla *receiveBusiness-Message2WAP5* (Sección 5.2.1.2). La terminación explícita se transforma en

un BPMN!EndEvent, aplicando la regla *failureExplicitTermination2EndEvent* (Sección 4.1.1.10). El mensaje de negocio *accept-proposal(SupplyPlanResponse)* del segundo camino se transforma en una BPMN!ReceiveTask, aplicando la regla *receiveBusinessMessage2WAP5*. La terminación explícita se transforma en un BPMN!EndEvent, aplicando la regla *successExplicitTermination2EndEvent* 4.1.1.9. El mensaje de negocio *propose(SupplyPlanChanges)* se transforma en dos tareas, una tarea BPMN!ReceiveTask y una tarea BPMN!Task, aplicando la regla *receiveBusinessMessage2WAP3* (Sección 5.2.1.1). Como solo hay un camino que no tiene terminación explícita, no se requiere agregar un BPMN!ExclusiveGateway para sincronización, como lo indica la regla *cfsXor2ExclusiveGateway*.

El segmento de flujo de control *Xor* se transforma en un BPMN!ExclusiveGateway, aplicando la regla *cfsXor2ExclusiveGateway* (Sección 4.1.1.7). Cada camino de interacción se transforma en un BPMN!SequenceFlow, aplicando la regla *interactionPath2sequenceFlow* (Sección 4.1.1.14). Luego, se transforman los elementos definidos en cada camino. El mensaje de negocio *accept-proposal(SupplyPlanChangesResponse)* del primer camino se transforma en dos tareas, una tarea BPMN!Task y una tarea BPMN!SendTask, aplicando la regla *sendBusinessMessage2WAP5* (Sección 5.2.1.2). El mensaje de negocio *reject-proposal(SupplyPlanChangesResponse)* del segundo camino se transforma en dos tareas, una tarea BPMN!Task y una tarea BPMN!SendTask, aplicando la regla *sendBusinessMessage2WAP5*. Dado que ningún camino tiene terminación explícita, se agrega un BPMN!ExclusiveGateway para sincronización, como lo indica la regla *cfsXor2ExclusiveGateway*.

Todos los documentos de negocio intercambiados se transformaron en BPMN!Message, aplicando la regla *businessDocument2Message* (Sección 4.1.1.13).

También se agregó un BPMN!EndEvent para representar la terminación del proceso colaborativo, aplicando la regla *implicitTermination2EndEvent* (Sección 4.1.1.11).

Todos los elementos generados por las reglas ejecutadas se vincularon aplicando la regla *successorSequenceFlow* (Sección 4.1.1.16).

5.4.2. Generando un modelo de proceso de integración

Para generar un modelo de proceso de integración, el usuario selecciona el modelo de proceso colaborativo a transformar. Luego selecciona el rol a considerar en la transformación e indica el tipo de modelo que va a generar, en este caso un modelo de proceso de integración. El usuario debe responder el cuestionario interactivo que recopila información sobre los elementos del modelo de proceso colaborativo *Collaborative Replenishment Plan* (Figura 5.31).

Luego que el usuario respondió las preguntas del cuestionario, con esa información se generó el modelo de parámetros (Figura 5.34).

En el modelo de parámetros se definieron parámetros a tres elementos del modelo de proceso colaborativo de entrada (además del rol): al elemento *propose(SupplyPlanChanges)*, al elemento *accept-proposal(SupplyPlanChangesResponse)* y al elemento *xor2*. El elemento *propose(SupplyPlanChanges)* es de tipo `BusinessMessage` (mensaje de negocio) y su identificador es *bm4*. Este elemento tiene definido el parámetro `taskType`, cuyo valor es *ManualTask*, y el parámetro `performer` cuyo valor es *Order Manager*. El elemento *accept-proposal(SupplyPlanChangesResponse)* es de tipo *BusinessMessage* y su identificador es *bm5*. Este elemento tiene definido el parámetro `taskType`, cuyo valor es *UserTask*, el parámetro `implementation`, cuyo valor es *##unspecified*, y el parámetro `performer` cuyo valor es *Order Management Analyst*. El elemento *xor2* es de tipo *Xor* y tiene definido el

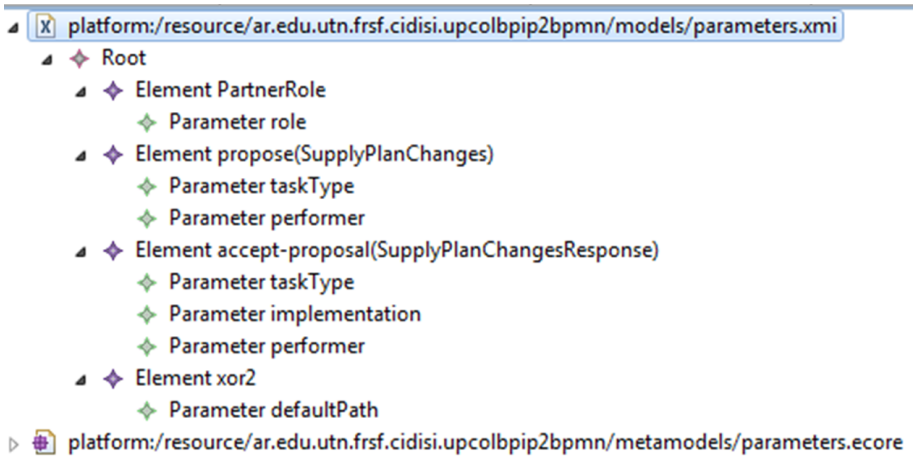


Figura 5.34: Modelo de parámetros de configuración como instancia del metamodelo Parameters.

parámetro `defaultPath` cuyo valor es *path1xor2*. En el Listado 5.5 se muestra el código del modelo de parámetros usado en este caso de estudio.

Listado 5.5: Ejemplo de modelo de parámetros

```
<?xml version="1.0" encoding="ASCII"?>
<parameters:ParameterModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:parameters="ar.edu
    .utn.frsf.cidisi.upcolbpip2bpmn.parameters" xsi:schemaLocation="ar.edu.utn.
    frsf.cidisi.upcolbpip2bpmn.parameters ../metamodels/parameters.ecore">
  <elements ID="roleId" name="PartnerRole" type="PartnerRole">
    <parameters name="role" value="Supplier"/>
  </elements>
  <elements ID="bm4" name="propose(SupplyPlanChanges)" type="BusinessMessage">
    <parameters name="taskType" value="ManualTask"/>
    <parameters name="performer" value="Order Manager"/>
  </elements>
  <elements ID="bm5" name="accept-proposal(SupplyPlanChangesResponse)" type="
      BusinessMessage">
    <parameters name="taskType" value="UserTask"/>
    <parameters name="implementation" value="##unspecified"/>
    <parameters name="performer" value="Order Management Analyst"/>
  </elements>
  <elements ID="xor2" name="xor2" type="Xor">
    <parameters name="defaultPath" value="path1xor2"/>
  </elements>
</parameters:ParameterModel>
```

A continuación, la ejecución del método genera el modelo de proceso de integración del proveedor (Figura 5.35), considerando los parámetros de configuración definidos en el modelo de parámetros de entrada.

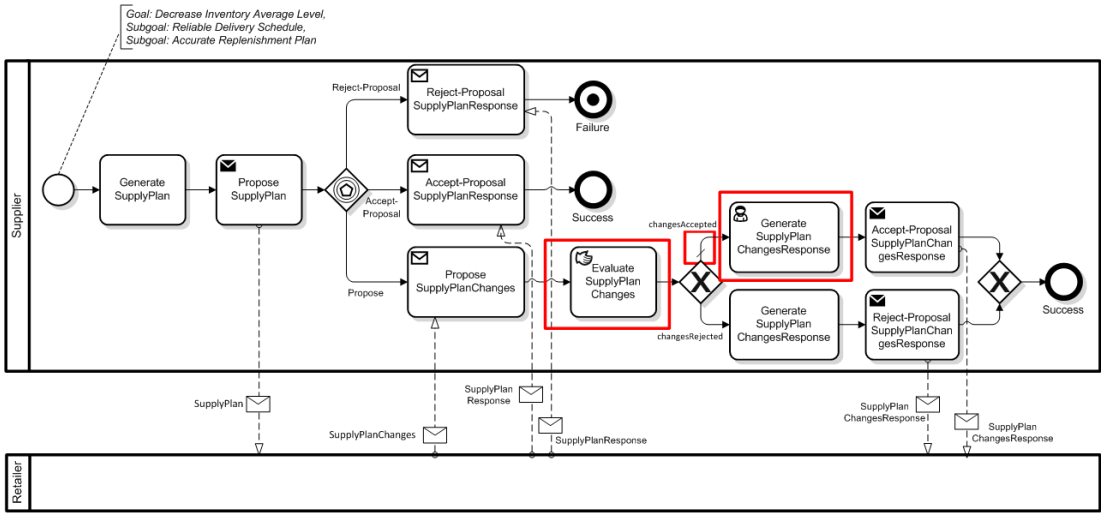


Figura 5.35: Modelo de proceso de integración del proveedor.

El modelo de proceso de integración se generó de manera similar a la plantilla, sólo se diferencia en las reglas utilizadas para transformar los mensajes de negocio *propose(SupplyPlanChanges)* y *accept-proposal(SupplyPlanChangesResponse)*, y el segundo segmento de flujo de control *Xor*, ya que dichos elementos del modelo de proceso colaborativo tienen definidos parámetros de configuración.

El mensaje de negocio *propose(SupplyPlanChanges)* tiene definido dos parámetros; `taskType` y `performer`. Por lo tanto, este mensaje se transforma aplicando la regla *receiveBusinessMessage2WAP3_ManualTask* (Sección 5.2.1.3).

El mensaje de negocio *accept-proposal(SupplyPlanChangesResponse)* tiene definido tres parámetros; `taskType`, `implementation` y `performer`. Por lo tanto, este mensaje se transforma, aplicando la regla *sendBusinessMessage2WAP5_UserTask* (Sección 5.2.1.3).

El segundo segmento de flujo de control *Xor* tiene definido un parámetro `defaultPath`. Por lo tanto, este segmento se transforma aplicando la regla *cfsXor2ExclusiveGateway_defaultPath* (Sección 5.2.2.1).

5.5. Conclusiones

El método propuesto brinda soporte a la actividad *Diseño de Procesos de Integración* de la fase *Diseño de la Solución Inter-organizacional* de la metodología presentada en el Capítulo 3. En base a los principios de MDA automatiza la generación de plantillas y modelos de procesos de integración, definidos con el lenguaje BPMN, a partir de un modelo de proceso colaborativo, definido con el lenguaje UP-ColBPIP. Una plantilla es un modelo de proceso configurable y un modelo de proceso de integración es una configuración de una determinada plantilla. Los usos de la plantilla pueden ser varios: para derivar en forma manual un modelo de proceso de integración; para verificar si un modelo de proceso de integración definido en forma manual es consistente con la plantilla y por consiguiente con el modelo de proceso colaborativo; y como medio de comunicación y discusión entre los "stakeholders", permitiendo probar diferentes opciones de diseño hasta generar una configuración que satisfaga las necesidades de negocio, facilitando y agilizando el diseño de modelos de procesos de integración. Las actividades privadas generadas para una plantilla son actividades abstractas.

En base al uso de un enfoque de cuestionario y parámetros definidos por el usuario, el método permite generar modelos de procesos de integración con actividades privadas concretas y detalladas. De esta manera, es posible incorporar en el proceso de transformación automático criterios y decisiones de diseño del usuario para generar diferentes modelos de procesos de integración.

Las plantillas y los modelos de procesos de integración generados son consistentes con el comportamiento definido y acordado en el correspondiente modelo de proceso colaborativo, y son interoperables con las plantillas y modelos de procesos de integración de las restantes organizaciones participantes. La generación de la plantilla mediante una transformación modelo-a-modelo a partir de un modelo de proceso colaborativo, garantiza su consistencia y la del modelo de proceso de integración que se genera como una configuración de la plantilla.

El uso de patrones de actividades de workflow asegura la interoperabilidad debido a que garantiza la sincronización de las actividades públicas que permiten el intercambio de mensajes. Por otra parte automatiza y facilita el diseño de modelos de procesos de integración; reduce el tiempo y costo de desarrollo; mejora la calidad de los modelos; y permite el re-uso del conocimiento capturado en ellos para generar las actividades públicas y privadas.

La implementación de la definición de transformación del método aplicando técnicas de herencia de reglas, refactorización y superimposición permite lograr escalabilidad, mantenibilidad y reusabilidad de la misma, mejorando la calidad de los modelos generados.

Finalmente, el método propuesto muestra que es posible generar plantillas y modelos de procesos de integración (que contiene aspectos privados de las organizaciones) a partir de modelos de procesos colaborativos (que sólo contienen aspectos públicos). El uso de patrones de actividades de workflow permite determinar las actividades privadas requeridas tanto en plantillas como modelos de procesos de integración. Además, sólo se requiere que el usuario indique el modelo a transformar y los parámetros de configuración a través de respuestas a preguntas en forma de cuestionario. Esto se logró definiendo un conjunto de reglas que permiten transformar un elemento del proceso colaborativo definido con el lenguaje UP-ColBPIP en un patrón destino definido con el lenguaje BPMN; y parámetros de configuración que permiten agregar más detalle a los elementos del modelo de entrada.

CAPÍTULO 6

Gestión de Modelos de Procesos de Negocio en Colaboraciones Inter-Organizacionales

Este capítulo presenta un repositorio distribuido para la gestión de modelos de procesos de negocio involucrados en colaboración inter-organizacionales (Sección 6.1). Se introduce su arquitectura y diseño, sus principales funcionalidades y las entidades que almacena. Se describe la implementación del repositorio distribuido (Sección 6.2). Finalmente, se presentan las conclusiones (Sección 6.3).

6.1. Repositorio distribuido de modelos de procesos de negocio

Los repositorios actuales de modelos de procesos de negocio (Capítulo 2) no proveen funcionalidades para dar soporte a los principales requerimientos asociados con la gestión de modelos de procesos de negocio involucrados en colaboraciones inter-organizacionales. Dichos requerimientos son: interoperabilidad, consistencia, sincronización, privacidad, asistencia en el diseño de los procesos de negocio e implementación distribuida (Capítulo 2).

Con el propósito de satisfacer estos requerimientos se propone un repositorio distribuido para la gestión de modelos conceptuales de procesos de negocio, el cual puede ser utilizado durante el desarrollo, implementación y gestión de colaboraciones inter-organizacionales (Lazarte y otros, 2013). El repositorio distribuido consiste de: un *repositorio global* público que da soporte a la gestión de modelos de procesos colaborativos y de interfaz; y *repositorios locales* privados

de las organizaciones que dan soporte a la gestión de modelos de procesos de integración. El objetivo es: permitir a las organizaciones compartir y gestionar en un mismo lugar y con la misma aplicación sus modelos de procesos colaborativos y de interfaz, usando un repositorio global público; y que tengan su repositorio local sincronizado con el repositorio global para gestionar sus modelos de procesos de integración.

La separación de modelos de procesos de negocio en diferentes repositorios permite cumplir con los requerimientos de los repositorios para ambientes inter-organizacionales. El repositorio distribuido provee las siguientes funcionalidades:

- **Sincronización de modelos de procesos de negocio distribuidos**. Posibilita sincronizar los modelos de procesos colaborativos y de interfaz almacenados en el repositorio global con los modelos de procesos de integración almacenados en los repositorios locales, para asegurar que los cambios en los modelos de procesos colaborativos (nuevas versiones) sean propagados y trasladados a cambios en los modelos de procesos de interfaz/integración existentes (nuevas versiones) o en la generación de nuevos modelos de procesos de interfaz/integración. En consecuencia, las organizaciones podrán usar la versión correcta de los modelos para evaluar, re-diseñar e implementar los modelos de procesos colaborativos y de integración.

- **Verificación de consistencia entre modelos de procesos de negocio**. Provee un servicio para chequear que la lógica de negocio de un modelo de proceso de integración definido manualmente, almacenado en un repositorio local, es consistente con la lógica del correspondiente modelo de proceso colaborativo, almacenado en el repositorio global. Esto permite a las organizaciones definir y mantener en sus repositorios locales modelos de procesos de integración en conformidad con la lógica acordada en los modelos de proceso colaborativos.

- **Garantía de interoperabilidad entre los modelos de procesos de negocio**. El repositorio global provee servicios que implementan los métodos basados en MDA propuestos (Capítulos 4 y 5), de tal manera de generar modelos de procesos de interfaz e integración interoperables y consistentes. A través de estos servicios el repositorio garantiza que los procesos de integración pueden interactuar y ejecutar en forma correcta el comportamiento

del correspondiente proceso colaborativo.

- **Preservación de los aspectos privados de la organización**. Permite a cada organización gestionar los modelos de procesos de integración en su repositorio local, manteniéndolos ocultos y protegidos del acceso de otras organizaciones.

- **Soporte en el diseño de procesos de negocio**. Provee servicios que asisten a las organizaciones en el diseño de modelos de procesos colaborativos, de interfaz y de integración. El repositorio global permite gestionar catálogos de modelos de procesos colaborativos, y provee soporte para buscar y seleccionar en los catálogos modelos de referencia de procesos colaborativos que pueden ser re-usados o adaptados para crear nuevos modelos de procesos colaborativos. También permite la generación automática de modelos de procesos de interfaz, plantillas y modelos de procesos de integración correspondientes a un modelo de proceso colaborativo. Los modelos de procesos de interfaz se mantienen en el repositorio global, en cambio las plantillas y modelos de procesos de integración se almacenan en el repositorio local de la organización. De este modo, el repositorio distribuido hace más fácil, rápido y eficiente el diseño de estos procesos de negocio.

El repositorio distribuido fue definido de acuerdo a los principios del diseño orientado a servicios (Erl, 2007) y está basado en SOA. El propósito es proveer servicios de bajo acoplamiento, reusables, interoperables y distribuidos que soporten las funcionalidades del repositorio global y los repositorios locales. El *repositorio global* provee servicios para la gestión de redes colaborativas, colaboraciones inter-organizacionales, modelos de procesos colaborativos y modelos de procesos de interfaz. Puede ser accedido y gestionado por todas las organizaciones registradas en el repositorio.

Un *repositorio local* provee servicios para la gestión de los modelos de procesos de integración que una organización requiere al momento de establecer colaboraciones inter-organizacionales. Puede ser accedido y gestionado sólo por la organización propietaria del mismo. El propósito es preservar la autonomía de las organizaciones y sus aspectos privados, y permitir la gestión descentralizada y distribuida de los modelos de procesos de negocio involucrados en las colaboraciones inter-organizacionales de las que participa.

El repositorio distribuido presenta una arquitectura de tres capas: datos, servicios y presentación (Figura 6.1). La *capa de datos* almacena los modelos de procesos de negocio y datos relacionados. La *capa de servicios* provee todos los servicios requeridos para gestionar los modelos de procesos de negocio contenidos en el repositorio distribuido. La *capa de presentación* consiste de aplicaciones cliente que proveen las interfaces de usuario que permiten el acceso al repositorio distribuido.

Con esta arquitectura, el repositorio puede ser implementado físicamente o lógicamente distribuido. En un repositorio *físicamente distribuido*, los componentes del repositorio global (Figura 6.1) son implementados y desplegados en un servidor público, accesible a través de Internet, por una tercera parte neutral o un consorcio de organizaciones. Los componentes de un repositorio local (Figura 6.1) son implementados y desplegados en un servidor privado, accesible a través de una Intranet, por cada organización que participa en las redes colaborativas definidas en el repositorio distribuido. En este caso, las organizaciones gestionan

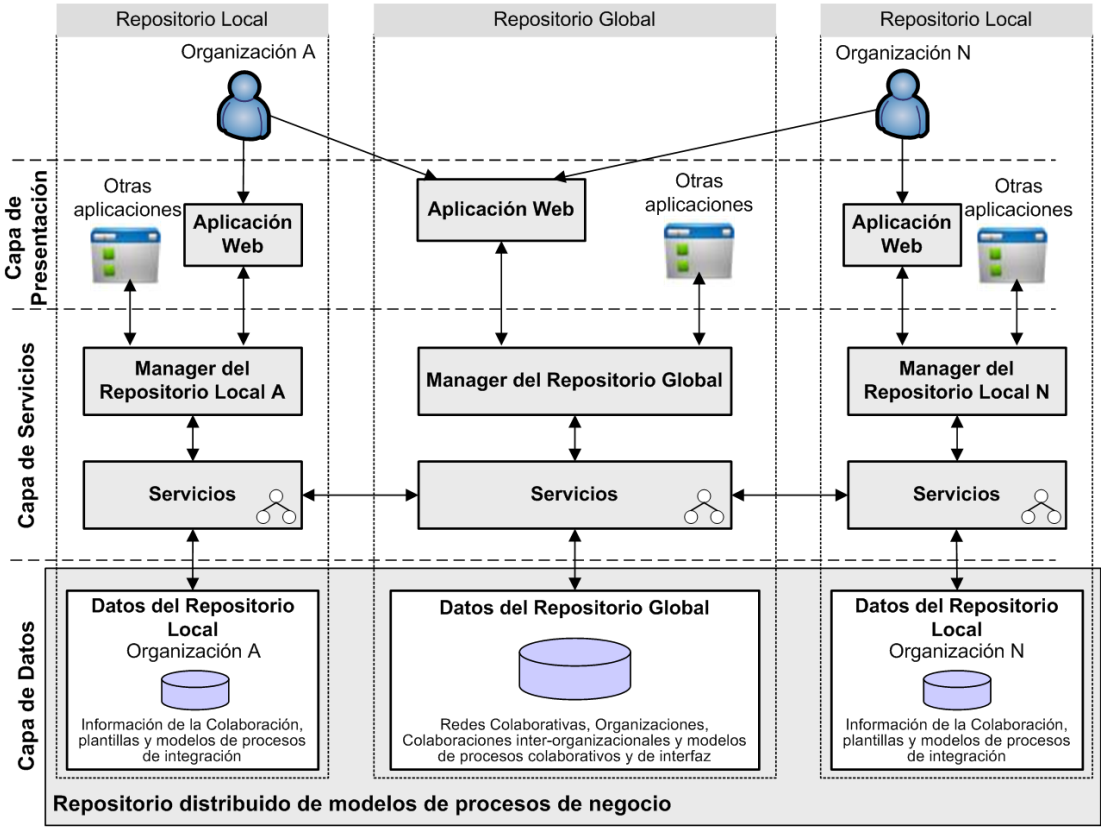


Figura 6.1: Arquitectura del repositorio distribuido de modelos de procesos de negocio.

localmente sus modelos de procesos de integración.

En un repositorio *lógicamente distribuido*, los componentes del repositorio global y los repositorios locales son implementados y desplegados en el mismo servidor público, accesible a través de Internet. En este caso, se controla y restringe el acceso al repositorio local para asegurar que cada organización acceda sólo a sus propios modelos de procesos de integración.

6.1.1. La capa de datos

En esta sección se describen las entidades almacenadas en el repositorio global y en uno local respectivamente.

El repositorio global contiene las organizaciones registradas y las redes colaborativas (Figura 6.2). Los usuarios (entidad `User`) corresponden a personas de las organizaciones que se registraron en el repositorio global.

Una red colaborativa (entidad `Collaborative Network`) es un grupo de organizaciones de un dominio específico relacionadas con el propósito de definir y participar en colaboraciones inter-organizacionales. Consiste de organizaciones (entidad `Organization`) que son miembros de la misma y está conformada por colaboraciones inter-organizacionales (entidad `Cross-organizational`

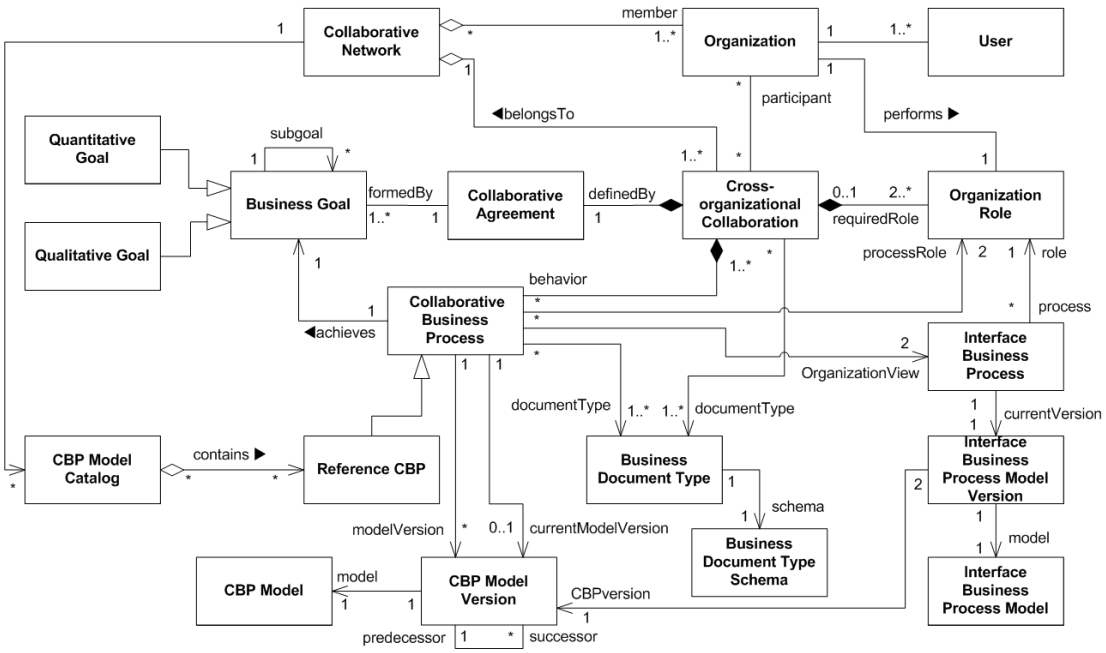


Figura 6.2: Entidades almacenadas en el repositorio global.

`Collaboration`) que dichas organizaciones han acordado llevar a cabo. Puede también tener catálogos de modelos de procesos colaborativos (entidad `CBP Model Catalog`) que contienen modelos de referencia de dichos procesos. Un proceso colaborativo de referencia (entidad `Reference CBP`) puede ser obtenido de modelos de negocio de referencia que son establecidos por iniciativas públicas o privadas de organizaciones y empresas (por ejemplo, RosettaNet, SCOR, CPFR) para un dominio en particular. Provee conocimiento explícito de cómo llevar a cabo la colaboración en dicho dominio, el cual puede ser reutilizado o adaptado para definir modelos de procesos colaborativos.

Una colaboración inter-organizacional (entidad `Cross-organizational Collaboration`) representa un conjunto de organizaciones cooperando para lograr metas de negocio comunes a través de la definición conjunta, implementación, ejecución y evaluación de procesos colaborativos. Consiste de las organizaciones participantes de la misma (entidad `Organization`), el rol que cada organización desempeña (entidad `Organization Role`), un acuerdo de colaboración (entidad `Collaborative Agreement`) y los procesos colaborativos a llevar a cabo (entidad `Collaborative Business Process`). Un acuerdo colaborativo (entidad `Collaborative Agreement`) define los parámetros que gobiernan una colaboración inter-organizacional, como por ejemplo, el modelo de negocio de referencia usado, la duración de la colaboración, etc. (Chituc y otros, 2009; Villarreal y otros, 2007b). Tiene asociado una jerarquía de metas de negocio comunes (entidad `Business Goal`) a ser alcanzadas por las organizaciones. Las metas de negocio cuantitativas (entidad `Quantitative Goal`) se definen en base a una métrica o medida de rendimiento acerca de alguna información compartida por las organizaciones, la cual puede ser expresada en alguna unidad de medida específica, tal como volumen, cantidad, tiempo, etc. Las metas de negocio cualitativas (entidad `Qualitative Goal`) se describen informalmente en lenguaje natural y el cumplimiento de la misma no depende de un valor específico sino del juicio de las organizaciones sobre si la meta ha sido alcanzada.

En una colaboración inter-organizacional también se pueden definir tipos de documentos de negocio (entidad `Business Document Type`) los cuales se refieren a la información a ser intercambiada entre las organizaciones. Un tipo de documento de negocio define la estructura general del documento y qué información será incluida. El mismo puede usarse en diferentes procesos colaborativos.

La estructura se define referenciando a un esquema definido por un estándar de intercambio de datos o B2B (por ejemplo, BODs (OAGi, 2009), UBL (OAGi, 2006)) o definido "ad-hoc" por las organizaciones. La estructura es definida en un esquema XML, el cual es asociado al tipo de documento de negocio y almacenado en el repositorio global.

Un proceso colaborativo (entidad `Collaborative Business Process`) tiene asociado los roles de las organizaciones (entidad `Organization Role`) de acuerdo a los definidos en la colaboración inter-organizacional. A través de éstos y la colaboración inter-organizacional en la que se definió el proceso colaborativo, es posible conocer las organizaciones involucradas. También tiene asociada una meta a alcanzar de acuerdo a las definidas en la colaboración inter-organizacional. Es posible también definir los tipos de documentos a ser intercambiados, los cuales deben corresponderse a los definidos en la colaboración inter-organizacional.

El comportamiento de un proceso colaborativo se define explícitamente en las versiones de modelo del mismo. La evolución de un modelo de proceso colaborativo se mantiene en el repositorio global mediante su versionamiento (entidad `CBP Model Version`). Se pueden definir diferentes versiones del mismo y sólo una versión puede ser marcada como la versión actual del modelo. El repositorio global soporta la definición de una jerarquía de versiones de modelos, lo que permite representar la huella o pista de cambios realizados en un modelo e indicar la versión origen (predecesora) y las versiones derivadas de él (sucesoras). Una versión de modelo de proceso colaborativo (entidad `CBP Model Version`) contiene los detalles de la versión tales como fecha de creación, número de versión, descripción, etc. y el modelo de proceso colaborativo que representa (entidad `CBP Model`) definido con algún lenguaje de modelado de procesos colaborativo.

Un proceso colaborativo también tiene asociado procesos de interfaz (entidad `Interface Business Process`), definidos para cada rol de las organizaciones. Cada proceso de interfaz tiene una versión actual (entidad `Interface Business Process Model Version`) la cual contiene detalles tales como fecha de creación, versión del correspondiente proceso colaborativo, y el modelo de proceso de interfaz que representa (entidad `Interface Business Process Model`). Para los procesos de interfaz no es necesario mantener una huella o pista de cambios debido a que la versión de modelo de los mismos se puede derivar

completamente a partir de la versión correspondiente de un modelo de proceso colaborativo, como se describe en la siguiente sección.

Un repositorio local de una organización (Figura 6.3) contiene información replicada del repositorio global sobre las redes colaborativas (entidad `Collaborative Network`) y las colaboraciones inter-organizacionales (entidad `Cross-organizational Collaboration`) en las que participa la organización (entidad `Organization`), el rol de la organización en cada colaboración (entidad `Organization Role`) y los procesos colaborativos que la organización soporta (entidad `Collaborative Business Process`).

En un repositorio local se mantiene un proceso de integración (entidad `Integration Business Process`) por cada proceso colaborativo. Para gestionar la evolución de los procesos de integración, el repositorio soporta el versionamiento de modelos de procesos de integración. Un repositorio local permite almacenar diferentes versiones de una plantilla de proceso de integración y de modelo de procesos de integración. Por cada versión de modelo de proceso colaborativo mantenida en el repositorio global, habrá una versión de plantilla de proceso de integración (entidad `IBP Template Model Version`) y podrá haber más de una versión de modelo de proceso de integración (entidad `IBP Model Version`), el cual a su vez tiene asociado la versión de plantilla correspondiente. Solo una versión de plantilla y de modelo puede ser marcada como la versión actual. La versión actual de una plantilla es la correspondiente a la versión actual del modelo de proceso colaborativo. Cada versión de plantilla y de modelo tienen

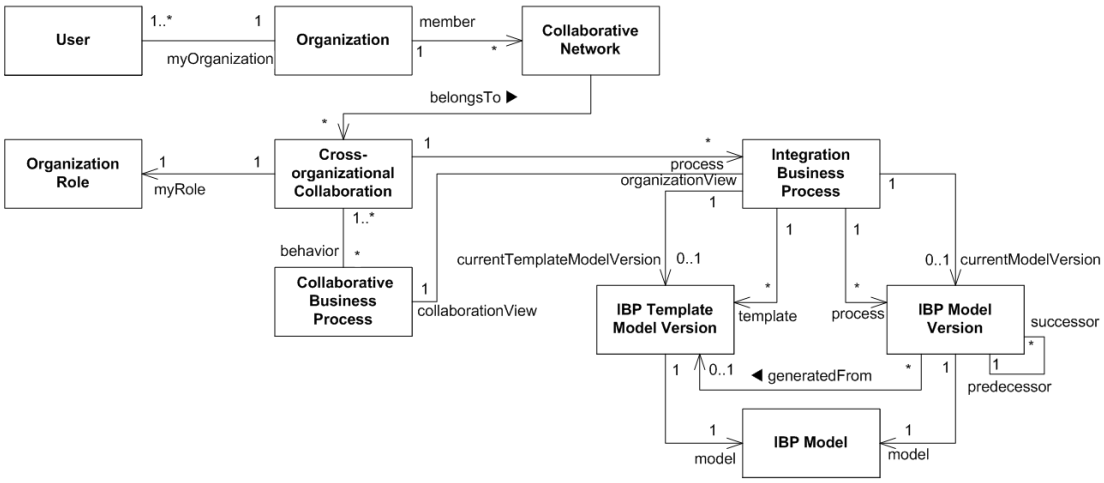


Figura 6.3: Entidades almacenadas en un repositorio local.

asociada el modelo propiamente dicho que los representa (entidad `IBP Model`), definido con algún lenguaje de modelado de procesos de negocio.

Un repositorio local soporta la definición de una jerarquía de versiones de modelos de procesos de integración, lo que permite representar la huella o pista de cambios realizados en un modelo e indicar la versión origen (predecesor) y las versiones derivadas de él (sucesoras). Para las plantillas no es necesario mantener una huella o pista de cambios debido a que la versión de las mismas se deriva a partir de la versión correspondiente de un modelo de proceso colaborativo, como se describe en la siguiente sección.

Los usuarios (entidad `User`) son las personas de la organización que tienen permisos para acceder al repositorio local.

6.1.2. La capa de servicios

La *capa de servicios* provee servicios que dan soporte a las funcionalidades del repositorio distribuido para gestionar modelos de procesos de negocio. Esta capa consiste de un *manager* del repositorio global y un *manager* del repositorio local, de cada organización (Figura 6.1). El *manager del repositorio global* actúa como una fachada ("façade") proveyendo una interfaz unificada para las aplicaciones cliente del repositorio global con el fin de gestionar modelos de procesos colaborativos y de interfaz. Un *manager del repositorio local* actúa como una fachada proveyendo una interfaz unificada para las aplicaciones cliente del repositorio local con el fin de gestionar modelos de procesos de integración. Ambos "managers" interactúan con servicios básicos y específicos que contienen la lógica de las funcionalidades del repositorio distribuido.

Los servicios básicos soportan las funciones básicas provistas por repositorios genéricos (Bernstein y Dayal, 1994) y repositorios de modelos de procesos de negocio (Yan y otros, 2012). Estos servicios son:

- *Servicio de almacenamiento*: tiene a su cargo la creación, actualización y eliminación de modelos de procesos de negocio, de acuerdo a los modelos de entidades descritos en la sección anterior.

- *Servicio de recuperación*: tiene a su cargo obtener los modelos de procesos de negocio requeridos de acuerdo a algún criterio usando métodos de navegación, consulta y búsqueda.

- *Servicio de integración*: tiene a su cargo la integración de herramientas externas con el repositorio.

- *Servicio de gestión de acceso*: tiene a su cargo asegurar que los usuarios sólo puedan acceder el repositorio para el que han sido autorizados.

- *Servicio de notificación*: tiene a su cargo generar notificaciones cuando se agregan versiones de los modelos de procesos de negocio.

- *Servicio de check-in/check-out*: tiene a su cargo desplegar ("check-out") un modelo del repositorio, bloquearlo para que otros no puedan cambiarlo, hacer los cambios deseados y liberar ("check-in") el modelo para futuros cambios.

- *Servicio de gestión de configuración*: tiene a su cargo mantener la relación entre (una versión de) un modelo de proceso de negocio y (las versiones de) los modelos de subprocesos que lo componen.

- *Servicio de gestión de ciclo de vida* ("life-cycle"): tiene a su cargo mantener el estado (obsoleto, en validación, actual) del ciclo de vida en el que se encuentra un modelo de proceso de negocio.

Por otra parte, los servicios específicos implementan las funciones particulares para dar soporte a los requerimientos identificados en las colaboraciones inter-organizacionales. Las interfaces de estos servicios se describen en las siguientes sub-secciones.

6.1.2.1. Servicios del repositorio global

En esta sección se describen los servicios específicos provistos por el repositorio global.

Servicio GlobalUserManagement

El servicio `GlobalUserManagement` tiene a su cargo gestionar los usuarios habilitados para acceder al repositorio global con el propósito de gestionar la información de su correspondiente organización. Las operaciones provistas (Figura 6.4) son las siguientes:

- La operación `addUser` agrega un usuario al repositorio global y lo vincula a una organización determinada.
- La operación `removeUser` elimina un usuario del repositorio global.
- La operación `updateUser` actualiza la información correspondiente a un usuario.
- La operación `getUser` obtiene información correspondiente a un usuario.

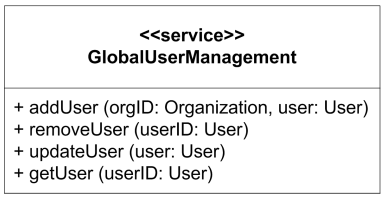


Figura 6.4: Servicio `GlobalUserManagement`.

Servicio OrganizationManagement

El servicio `OrganizationManagement` tiene a su cargo gestionar los perfiles de las organizaciones registradas en el repositorio. Las operaciones provistas (Figura 6.5) son las siguientes:

- La operación `addOrganization` carga un perfil de una organización en el repositorio global.
- La operación `getOrganization` obtiene el perfil de una organización.
- La operación `getOrganizationList` obtiene una lista de organizaciones basado en un criterio de búsqueda con el fin de invitar a una organización a formar parte de una red colaborativa o establecer una colaboración inter-organizacional.
- La operación `removeOrganization` remueve lógicamente el perfil de una organización que ya no desea usar el repositorio.
- La operación `updateOrganization` actualiza el perfil de una organización.

- La operación `getUserList` obtiene una lista de usuarios habilitados para gestionar el perfil de la organización.

Para poder participar de una red colaborativa la organización debe cargar su perfil a través de la aplicación Web. Esto deriva en una invocación a la operación `addOrganization` del servicio.

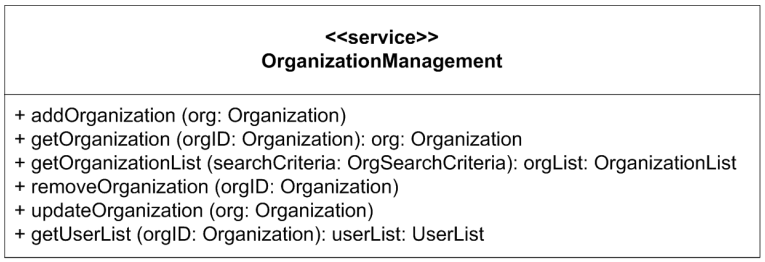


Figura 6.5: Servicio `OrganizationManagement`.

Servicio CollaborativeNetworkManagement

El servicio `CollaborativeNetworkManagement` tiene a su cargo gestionar las redes colaborativas. Las operaciones provistas (Figura 6.6) son las siguientes:

- La operación `addCollaborativeNet` crea una red colaborativa en el repositorio global.

- La operación `updateCollaborativeNet` actualiza la información correspondiente a una red colaborativa.

- La operación `removeCollaborativeNet` elimina una red colaborativa. Esta operación requiere que sus miembros no estén participando de alguna colaboración inter-organizacional en el marco de dicha red.

- La operación `getCollaborativeNet` obtiene información de una red colaborativa.

- La operación `addMember` agrega una organización como miembro de una red colaborativa. Esta operación requiere que la organización haya cargado su perfil.

- La operación `removeMember` desvincula a una organización de una red colaborativa.
- La operación `getMember` obtiene información de un miembro de la red colaborativa.
- La operación `getMemberList` obtiene una lista de miembros de una red colaborativa.
- La operación `getCOCList` obtiene una lista de colaboraciones inter-organizacionales establecidas entre los miembros de la red.
- La operación `getCBPCatalogList` obtiene una lista de catálogos de procesos de referencia definidos para una red colaborativa.
- La operación `getCNList` obtiene una lista de redes colaborativas en las que participa una organización.
- La operación `sendInvitation` envía una invitación a una organización para unirse a una red colaborativa. Dicha invitación se realiza enviando un e-mail a una organización indicando la misión de la red, sus miembros y un link al que debe acceder, en caso que desea unirse a la misma.

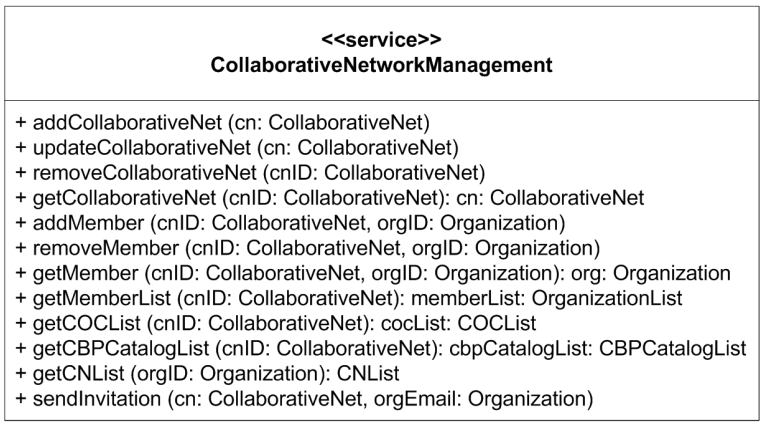


Figura 6.6: Servicio `CollaborativeNetworkManagement`.

Servicio CrossOrganizationalCollaborationManagement

El servicio `CrossOrganizationalCollaborationManagement` tiene a su cargo gestionar colaboraciones inter-organizacionales. Las operaciones provistas (Figura 6.7) son las siguientes:

- La operación `addCOC` crea una colaboración inter-organizacional en una red colaborativa.

- La operación `getCOC` obtiene información de una colaboración inter-organizacional.

- La operación `removeCOC` elimina una colaboración inter-organizacional. Esta operación requiere que el acuerdo de colaboración haya finalizado o que las organizaciones involucradas estén de acuerdo con eliminarla.

- La operación `updateCOC` actualiza la información de una colaboración inter-organizacional.

- La operación `getCOCList` obtiene una lista de colaboraciones en las que participa una organización.

- La operación `addParticipant` agrega un participante a una colaboración inter-organizacional, indicando el rol que desempeña en la misma.

- La operación `removeParticipant` desvincula un participante de una colaboración inter-organizacional. Esta operación requiere que el acuerdo de colaboración haya finalizado o las restantes organizaciones estén de acuerdo con la desvinculación.

- La operación `getParticipant` obtiene información de un participante de una colaboración inter-organizacional desempeñando un rol determinado.

- La operación `getParticipantList` obtiene una lista de participantes de una colaboración inter-organizacional.

- La operación `getOrganizationRole` obtiene el rol que una organización desempeña en una colaboración inter-organizacional.

- La operación `getOrganizationRoleList` obtiene una lista de roles involucrados en una colaboración inter-organizacional.

- La operación `getBusinessDocumentList` obtiene una lista de tipos de documentos de negocio definidos en una colaboración inter-organizacional.

- La operación `sendInvitation` envía una invitación a una organización a participar en una colaboración inter-organizacional. Dicha invitación se realiza enviando un e-mail a una organización con una descripción de la colaboración inter-organizacional y un link al que debe acceder, en caso que desea participar en la misma.

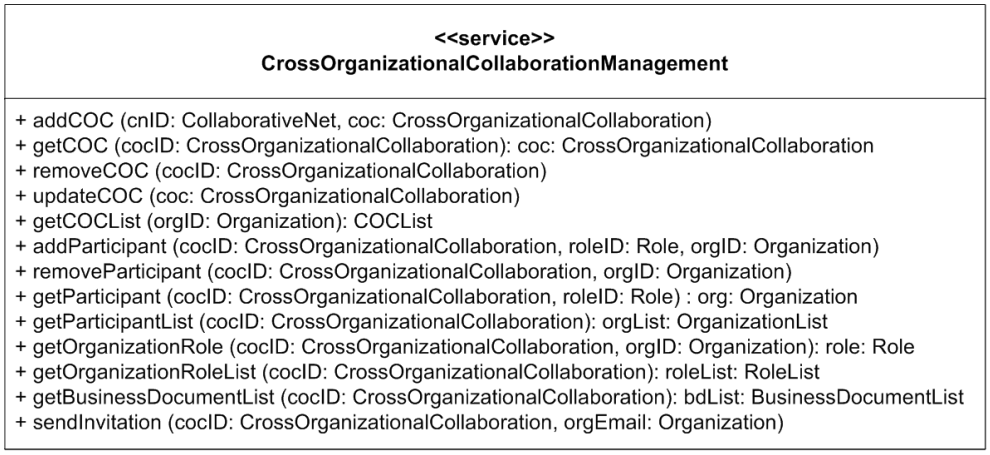


Figura 6.7: Servicio `CrossOrganizationalCollaborationManagement`.

Servicio CollaborativeAgreementManagement

El servicio `CollaborativeAgreementManagement` tiene a su cargo gestionar los acuerdos de colaboración definidos para una colaboración inter-organizacional. Las operaciones provistas (Figura 6.8) son las siguientes:

- La operación `addCollaborativeAgreement` agrega un acuerdo de colaboración de una colaboración inter-organizacional.

- La operación `updateCollaborativeAgreement` actualiza un acuerdo de colaboración determinado.

- La operación `getCollaborativeAgreement` obtiene un acuerdo de colaboración determinado.

- La operación `addBusinessGoal` agrega una meta de negocio y la vincula a un acuerdo de colaboración.

- La operación `removeBusinessGoal` elimina una meta de negocio.

- La operación `updateBusinessGoal` actualiza una meta de negocio.
- La operación `getBusinessGoal` obtiene una meta de negocio.
- La operación `getBusinessGoalList` obtiene una lista de metas de negocio definidas para un acuerdo de colaboración. Se puede indicar si se quieren obtener las metas cuantitativas, las cuantitativas o ambas.

Figura 6.8: Servicio `CollaborativeAgreementManagement`.

Servicio BusinessDocumentTypeManagement

El servicio `BusinessDocumentTypeManagement` tiene a su cargo gestionar los tipos de documentos de negocio que se definieron para cada colaboración inter-organizacional y que serán intercambiados en los modelos de procesos colaborativos de la misma. Las operaciones provistas (Figura 6.9) son las siguientes:

- La operación `addBusinessDocument` agrega un tipo de documento de negocio al repositorio y lo asocia a una colaboración inter-organizacional.
- La operación `removeBusinessDocument` elimina un tipo de documento de negocio del repositorio.
- La operación `updateBusinessDocument` actualiza un tipo de documento de negocio.
- La operación `getBusinessDocument` obtiene un tipo de documento de negocio.

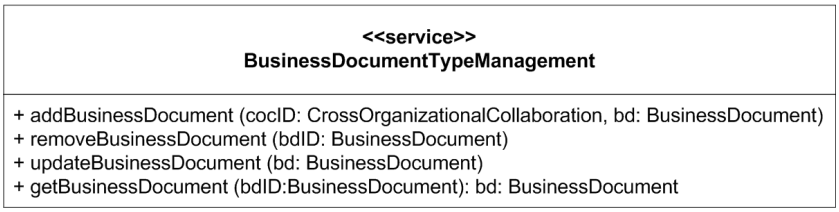


Figura 6.9: Servicio `BusinessDocumentTypeManagement`.

Servicio CBPCatalogManagement

El servicio `CBPCatalogManagement` tiene a su cargo gestionar catálogos de procesos colaborativos de referencia de una colaboración inter-organizacional, el cual generalmente corresponde a un modelo de negocio de referencia, como por ejemplo CPFR. Las operaciones provistas (Figura 6.10) son las siguientes:

- La operación `addCBPCatalog` agrega un catálogo a una red colaborativa.
- La operación `updateCBPCatalog` actualiza la información de un catálogo.
- La operación `removeCBPCatalog` elimina un catálogo de una red colaborativa.
- La operación `getCBPCatalog` obtiene información de un catálogo.
- La operación `addReferenceCBP` agrega un proceso de referencia a un catálogo.
- La operación `getReferenceCBP` obtiene un proceso de referencia de un catálogo.
- La operación `getReferenceCBPList` obtiene una lista de procesos de referencia pertenecientes a un catálogo.
- La operación `removeReferenceCBP` elimina un proceso de referencia de un catálogo.
- La operación `updateReferenceCBP` actualiza un proceso de referencia perteneciente a un catálogo.

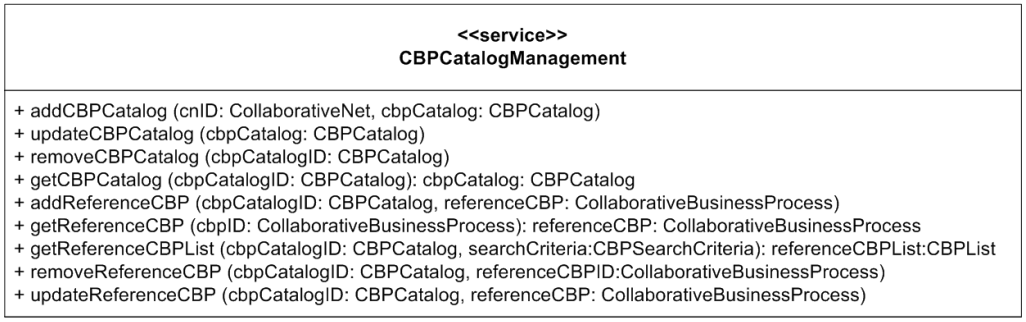


Figura 6.10: Servicio `CBPCatalogManagement`.

Servicio CollaborativeBusinessProcessManagement

El servicio `CollaborativeBusinessProcessManagement` tiene a su cargo gestionar procesos colaborativos. Las operaciones provistas (Figura 6.11) son las siguientes:

- La operación `addCBP` agrega un proceso colaborativo a una colaboración inter-organizacional.
- La operación `removeCBP` elimina un proceso colaborativo de una colaboración inter-organizacional. Esta operación requiere que no haya modelos almacenados en el repositorio global para dicho proceso colaborativo.
- La operación `updateCBP` actualiza la información de un proceso colaborativo.
- La operación `getCBP` obtiene un proceso colaborativo.
- La operación `getOrganizationRoleList` obtiene una lista de roles involucrados en un proceso colaborativo.
- La operación `assignBusinessDocument` asigna un tipo de documento de negocio a un proceso colaborativo.
- La operación `assignBusinessGoal` asigna una meta de negocio a un proceso colaborativo.
- La operación `getBusinessDocumentList` obtiene una lista de tipos de documentos de negocio asignados a un proceso colaborativo.

- La operación `getBusinessGoal` obtiene la meta de negocio asignada a un proceso colaborativo.

Figura 6.11: Servicio `CollaborativeBusinessProcessManagement`.

Servicio CBPModelVersionManagement

El servicio `CBPModelVersionManagement` tiene a su cargo gestionar modelos para los procesos colaborativos definidos en el repositorio. Las operaciones provistas (Figura 6.12) son las siguientes:

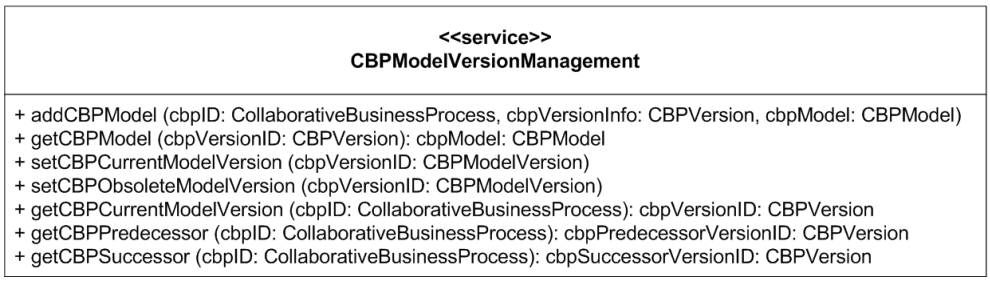


Figura 6.12: Servicio `CBPModelVersionManagement`.

- La operación `addCBPModel` agrega un modelo de proceso colaborativo, junto con la información de la versión. Esta operación invoca otros servicios (Figura 6.13).

  Primero, se invoca la operación `verifyCBPmodelCorrectness` del servicio `CBPmodelCorrectnessChecking`, con el propósito de verificar y determinar, previo a generar los procesos de interfaz e integración, si el modelo de proceso colaborativo está libre de errores. Si el resultado de verificación indica errores, se cancela el agregado del modelo y

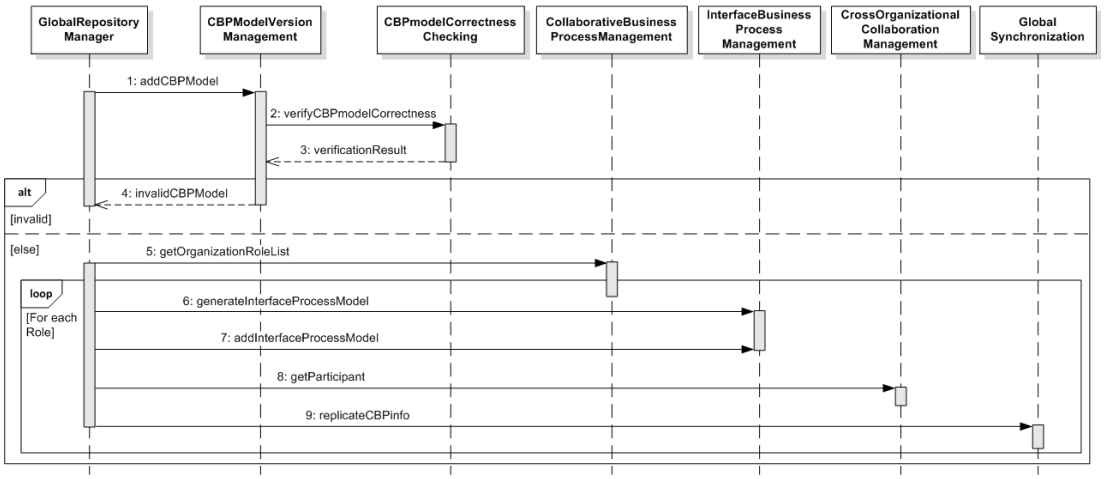


Figura 6.13: Interacción entre servicios requeridos para agregar un proceso colaborativo (Se omiten las entradas/salidas de cada operación).

se devuelve el mensaje de falla `invalidCBPmodel` indicando los errores encontrados. Si la verificación resulta exitosa, se invoca el servicio `CollaborativeBusinessProcessManagement` para obtener los roles de las organizaciones involucradas en el modelo de proceso colaborativo agregado.

Luego, en forma iterativa se realizan los siguientes pasos. Por cada rol obtenido, se invocan las operaciones del servicio `InterfaceBusinessProcessManagement` para generar y almacenar en el repositorio global el modelo de proceso de interfaz correspondiente. A continuación, se invoca el servicio `CrossOrganizationalCollaborationManagement` para obtener la organización desempeñando dicho rol con el propósito de enviarle la información correspondiente. Finalmente, se invoca la operación `replicateCBPinfo` del servicio `GlobalSynchronization` para llevar a cabo la sincronización del modelo de proceso colaborativo con la plantilla del repositorio local correspondiente a la organización obtenida.

- La operación `getCBPModel` obtiene un modelo de proceso colaborativo.
- La operación `setCBPCurrentModelVersion` establece como actual ("current") una versión de un modelo de proceso colaborativo.
- La operación `setCBPObsoleteModelVersion` establece como obsoleto

("obsolete") una versión de un modelo de proceso colaborativo.

- La operación `getCBPCurrentModelVersion` obtiene la versión actual de un modelo de proceso colaborativo.

- La operación `getCBPPredecessor` obtiene la versión predecesora de un modelo de proceso colaborativo.

- La operación `getCBPSuccessor` obtiene la versión sucesora de un modelo de proceso colaborativo.

Servicio CBPmodelCorrectnessChecking

El servicio `CBPmodelCorrectnessChecking` tiene a su cargo asegurar que el comportamiento de un modelo de proceso colaborativo está bien definido, es decir, libre de errores.

Provee la operación `verifyCBPmodelCorrectness` (Figura 6.14) que verifica la correctitud (tales como ausencia de "deadlocks" y "livelocks") de un modelo de proceso colaborativo.

La operación implementa un método para la verificación de modelos de procesos colaborativos propuesto por Roa y otros (2012). Este método formaliza la vista global de las interacciones de modelos de procesos colaborativos por medio de *Global Interaction Nets* (GI-Net), las cuales son un tipo particular de Redes de Petri Jerárquicas y Coloreadas. El método define la propiedad *Solidez* ("Soundness") para GI-Nets como el principal criterio de correctitud para verificar flujos de control avanzados en modelos de procesos colaborativos. Una característica importante de este método es la de ser independiente del lenguaje de modelado, pudiendo aplicarse para verificar modelos de procesos colaborativos definidos en cualquier lenguaje.

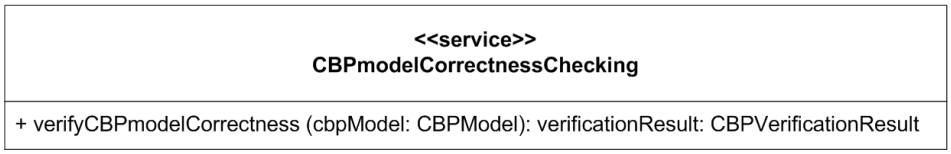


Figura 6.14: Servicio `CBPmodelCorrectnessChecking`.

Servicio InterfaceBusinessProcessManagement

El servicio `InterfaceBuninessProcessManagement` tiene a su cargo gestionar los procesos de interfaz correspondientes a cada rol que interviene en un modelo de proceso colaborativo. Las operaciones provistas (Figura 6.15) son las siguientes:

- La operación `addInterfaceBP` agrega un proceso de interfaz al repositorio global.
- La operación `removeInterfaceBP` elimina un proceso de interfaz del repositorio global. Esta operación requiere que no existan modelos almacenados en el repositorio global para dicho proceso.
- La operación `updateInterfaceBP` actualiza la información de un proceso de interfaz.
- La operación `getInterfaceBP` obtiene un proceso de interfaz.

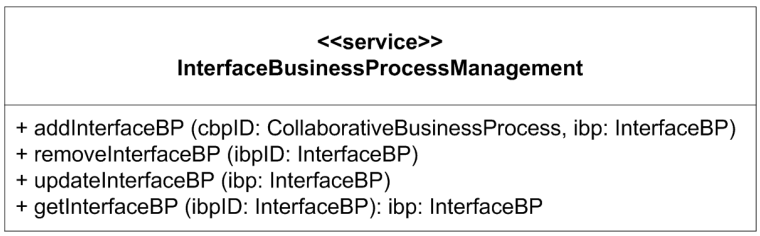


Figura 6.15: Servicio `InterfaceBusinessProcessManagement`.

Servicio InterfaceBPModelVersionManagement

El servicio `InterfaceBPModelVersionManagement` tiene a su cargo gestionar los modelos de procesos de interfaz. Las operaciones provistas (Figura 6.16) son las siguientes:

- La operación `addInterfaceBPModel` agrega un modelo de proceso de interfaz correspondiente a una versión de un modelo de proceso colaborativo al repositorio.
- La operación `removeInterfaceBPModel` elimina un modelo de proceso de interfaz correspondiente a una versión obsoleta de un modelo de proceso colaborativo.

- La operación `getInterfaceBPModel` obtiene un modelo de proceso de interfaz.

- La operación `generateInterfaceBPModel` genera un modelo de proceso de interfaz a partir de un modelo de proceso colaborativo y un rol. Para ello, esta operación implementa el método presentado en el Capítulo 4.

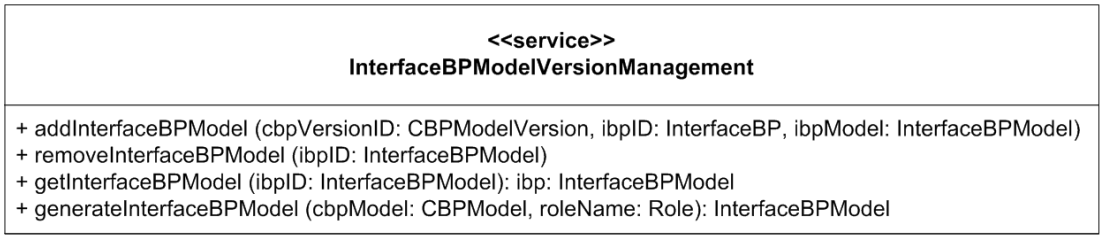


Figura 6.16: Servicio `InterfaceBPModelVersionManagement`.

Servicio IntegrationBPTemplateGeneration

El servicio `IntegrationBPTemplateGeneration` tiene a su cargo generar las plantillas de procesos de integración para facilitar el diseño de modelos de procesos de integración.

Provee la operación `generateTemplate` (Figura 6.17) que genera una plantilla de proceso de integración a partir de un modelo de proceso colaborativo y un rol. Para ello, esta operación implementa el método presentado en el Capítulo 5.

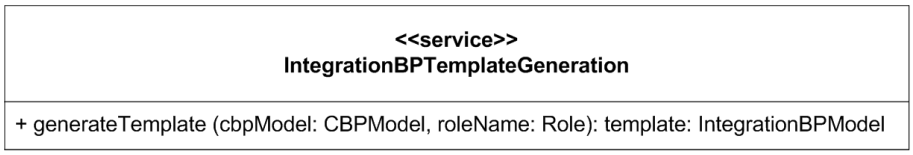


Figura 6.17: Servicio `IntegrationBPTemplateGeneration`.

Servicio GlobalSynchronization

El servicio `GlobalSynchronization` tiene a su cargo replicar la información del repositorio global a un repositorio local. Las operaciones provistas (Figura 6.18) son las siguientes:

Figura 6.18: Servicio `GlobalSynchronization`.

- La operación `replicateCOCinfo` replica en el repositorio local información sobre una colaboración inter-organizacional cuando se agrega una organización como participante de la misma en el repositorio global.

- La operación `replicateCBPinfo` replica en el repositorio local de una organización la información acerca de un modelo de proceso colaborativo y su respectiva plantilla de proceso de integración. Esta operación se invoca cuando se agrega un modelo de proceso colaborativo al repositorio global (Figura 6.13). En el comportamiento de esta operación (Figura 6.19), primero se invoca el servicio `IntegrationBPTemplateGeneration` para generar la plantilla de proceso de integración de la organización. Luego se invoca al servicio `LocalSynchronization` del repositorio local de la organización el cual tiene a su cargo almacenar en el repositorio local la información sobre el proceso colaborativo y la plantilla de proceso de integración generada.

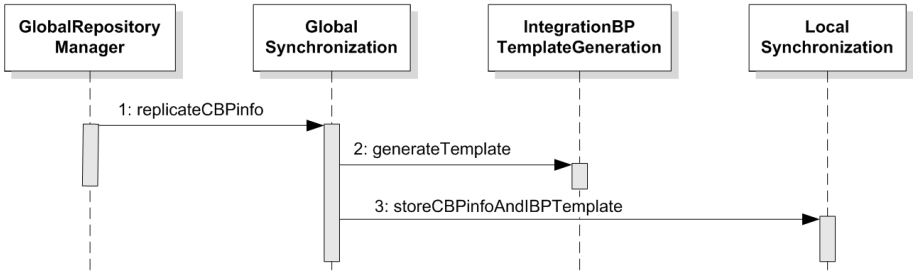


Figura 6.19: Interacción entre servicios requeridos para sincronizar los modelos cuando se agrega un nuevo modelo de proceso colaborativo (Se omiten las entradas/salidas de cada operación).

- La operación `notifyObsoleteCBP` notifica al servicio `LocalSynchronization` de las organizaciones participantes que un modelo de proceso colaborativo quedó obsoleto. El propósito es que cada una de ellas cambie a `obsolete` el estado de las respectivas plantillas

y modelos de proceso de integración .

6.1.2.2. Servicios de un repositorio local

En esta sección se describen los servicios provistos por un repositorio local.

Servicio IntegrationBusinessProcessManagement

El servicio `IntegrationBusinessProcessManagement` tiene a su cargo gestionar los procesos de integración correspondientes a una colaboración inter-organizacional. Las operaciones provistas (Figura 6.20) son las siguientes:

- La operación `addIntegrationBP` agrega un proceso de integración en un repositorio local y lo vincula a la correspondiente colaboración inter-organizacional.
- La operación `updateIntegrationBP` actualiza la información de un proceso de integración.
- La operación `getIntegrationBP` obtiene un proceso de integración de un repositorio local.
- La operación `getIntegrationBPList` obtiene una lista de procesos de integración correspondientes a una colaboración inter-organizacional.

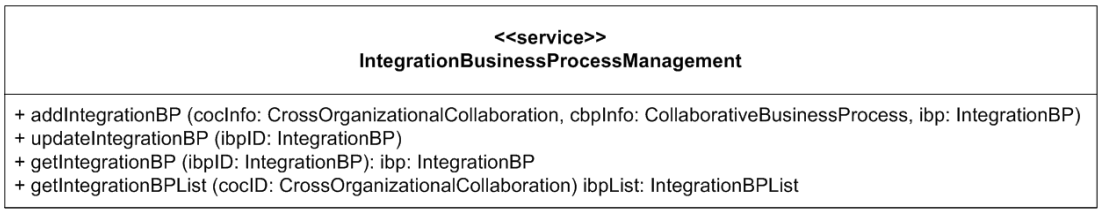


Figura 6.20: Servicio `IntegrationBusinessProcessManagement`.

Servicio TemplateModelVersionManagement

El servicio `TemplateModelVersionManagement` tiene a su cargo gestionar versiones de plantillas de procesos de integración. Las operaciones provistas (Figura 6.21) son las siguientes:

- La operación `addTemplateModel` agrega una versión de una plantilla al repositorio local.
- La operación `getTemplateModel` obtiene una versión de una plantilla.
- La operación `setCurrentTemplateModelVersion` establece como actual una versión de una plantilla.
- La operación `getCurrentTemplateModelVersion` obtiene la versión actual de una plantilla.
- La operación `setObsoleteTemplate` establece como obsoleta una versión de una plantilla correspondiente a una versión obsoleta de un modelo de proceso colaborativo.

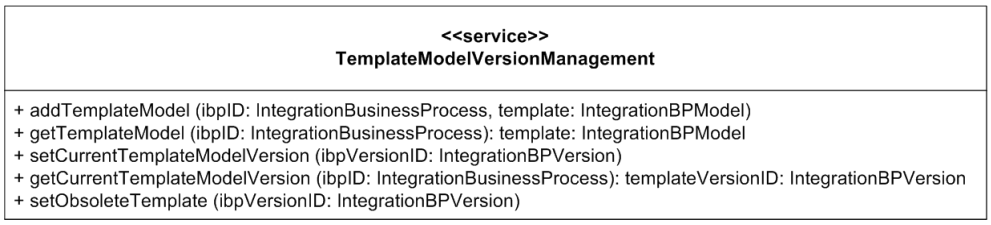


Figura 6.21: Servicio `TemplateModelVersionManagement`.

Servicio IntegrationBPModelVersionManagement

El servicio `IntegrationBPModelVersionManagement` tiene a su cargo gestionar versiones de modelos de procesos de integración. Las operaciones provistas (Figura 6.22) son las siguientes:

- La operación `addntegrationIBPModel` agrega una versión de un modelo de proceso de integración al repositorio local, indicando la versión predecesora.
- La operación `removeIntegrationBPModel` remueve una versión de un modelo de proceso de integración del repositorio local.
- La operación `getIntegrationBPModel` obtiene una versión de un modelo de proceso de integración.

- La operación `setCurrentIntegrationBPModelVersion` establece como actual una versión de un modelo de proceso de integración.
- La operación `getCurrentIntegrationBPModelVersion` obtiene la versión actual de un modelo de proceso de integración.
- La operación `getIntegrationBPPredecessor` obtiene la versión predecesora de un modelo de proceso de integración.
- La operación `getIntegrationBPSuccessor` obtiene la versión sucesora de un modelo de proceso de integración.
- La operación `setObsoleteIntegrationBP` establece como obsoleta una versión de un modelo de proceso de integración correspondiente a una versión obsoleta de un modelo de proceso colaborativo (en caso que se haya generado a partir de un modelo de proceso colaborativo) o una versión obsoleta de una plantilla (en caso que se haya generado a partir de una plantilla).

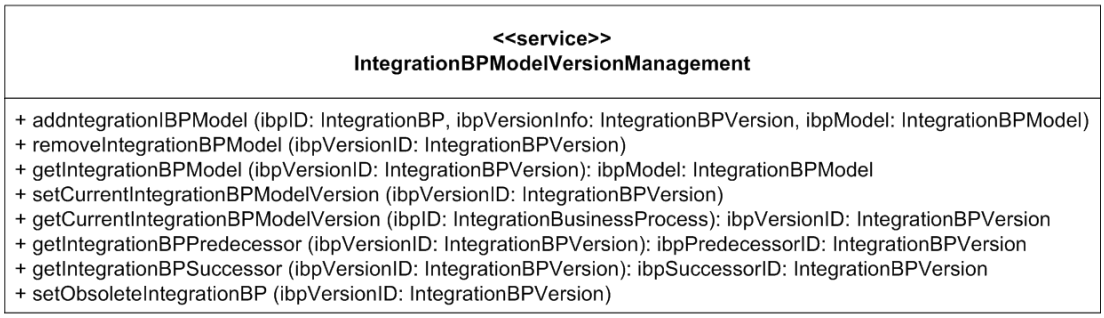


Figura 6.22: Servicio `IntegrationBPModelVersionManagement`.

Servicio IBPModelGeneration

El servicio `IBPModelGeneration` tiene a su cargo generar modelos de procesos de integración. Provee la operación `generateIBPModel` (Figura 6.23) que genera un modelo de proceso de integración a partir de un modelo de proceso colaborativo obtenido del repositorio global y un modelo de parámetros generado a partir de un cuestionario interactivo. Para ello, esta operación implementa el método presentado en el Capítulo 5.

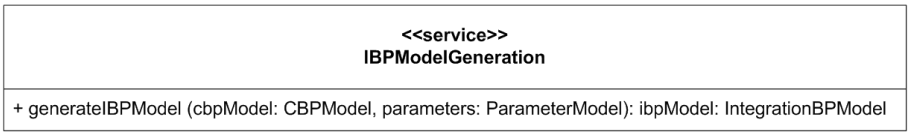


Figura 6.23: Servicio `IBPModelGeneration`.

Servicio IBPModelConsistencyChecking

Una organización puede usar las plantillas de procesos de integración para definir en forma manual sus modelos de procesos de integración (Capítulo 5). En este caso requiere verificar la consistencia de dichos modelos para garantizar su interoperabilidad con los modelos de procesos de integración de las otras organizaciones participantes y, consecuentemente, la correcta terminación de un proceso colaborativo.

Debido a que interoperabilidad y consistencia son nociones relacionadas (Decker y Weske, 2007; Weske, 2007), verificando si un modelo de proceso de integración es consistente con su correspondiente plantilla, se asegura la interoperabilidad con los modelos de procesos de integración de las otras organizaciones, siempre que estos últimos sean consistentes con su correspondiente plantilla (Capítulo 5). De esta manera, cada organización puede verificar localmente si un modelo de proceso de integración es consistente con el modelo de proceso colaborativo. No se requiere la verificación de consistencia entre plantillas de procesos de integración y su correspondiente modelo de proceso colaborativo debido a que estas plantillas se generan automáticamente a partir de modelos de procesos colaborativos aplicando una transformación modelo-a-modelo (Capítulo 5).

El servicio `IBPModelConsistencyChecking` realiza la verificación de la consistencia entre un modelo de proceso de integración y su correspondiente plantilla comparando sus comportamientos.

Provee la operación `verifyIBPModelConsistency` (Figura 6.24), la cual implementa el método de verificación de consistencia de comportamientos propuesto por Martens (2005). Este método fue propuesto para obtener la consistencia entre dos tipos diferentes de procesos de negocio, por ejemplo entre procesos abstractos y procesos ejecutables modelados con el lenguaje WS-BPEL. El servicio `IBPModelConsistencyChecking` re-usa la lógica de este método para comparar dos modelos de procesos de integración, uno es la plantilla y el otro es un modelo cuyo esqueleto es la plantilla. Por consiguiente, aplicando este

método, se dice que un modelo de proceso de integración es consistente si y sólo si el comportamiento del modelo simula el comportamiento de la plantilla.

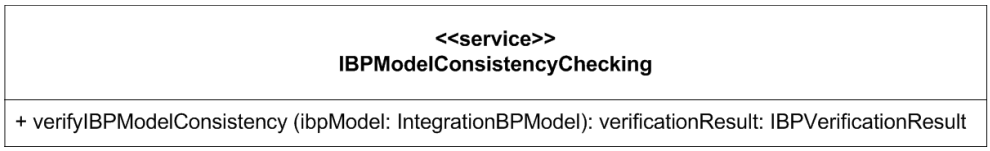


Figura 6.24: Servicio `IBPModelConsistencyChecking`.

El método de verificación analiza la consistencia entre una plantilla de proceso de integración y un modelo de proceso de integración derivado de la misma en tres pasos (Martens, 2005):

1. Transformar la plantilla y el modelo a redes de Petri. Debido a que en la tesis se usa el lenguaje Business Process Model and Notation (BPMN) para modelar procesos de integración, en lugar de WS-BPEL, se ajustó este paso del método original para aceptar modelos de procesos definidos con el lenguaje BPMN. El mapeo de un modelo de proceso en BPMN a redes de Petri se realiza aplicando las reglas de transformación propuesta por Dijkman y otros (2008).

2. Extraer la información relevante y generar el *communication graph* (Martens, 2005; Martens y otros, 2006) para ambos, plantilla y modelo de proceso de integración.

3. Verificar la simulación entre los comportamientos comparando los correspondientes *communication graphs*

Un modelo de proceso de integración simula el comportamiento de la plantilla si y sólo si el *communication graph* del modelo simula el *communication graph* de la plantilla.

Servicio LocalSynchronization

El servicio `LocalSynchronization` tiene a su cargo mantener la sincronización entre las plantillas de proceso de integración y los modelos de procesos colaborativos almacenados en el repositorio global. El objetivo es asegurar que la

versión del modelo de proceso de integración usado por las organizaciones corresponde con la versión correcta del respectivo modelo de proceso colaborativo. Las operaciones provistas (Figura 6.25) son las siguientes:

- La operación `storeCOCinfo` almacena en el repositorio local la información recibida del repositorio global sobre la colaboración inter-organizacional en la que la organización está siendo agregada como participante.

- La operación `storeCBPinfoAndIBPTemplate` almacena la información recibida sobre el proceso de integración y del proceso colaborativo correspondiente. Para ello, este servicio invoca los servicios `IntegrationBusinessProcessManagement` y `TemplateModelVersionManagement` para agregar la plantilla al repositorio local.

- La operación `deprecateIBP` define como obsoletos a las plantillas y a los modelos de procesos de integración generados a partir de éstas, correspondientes a procesos colaborativos obsoletos. El propósito es que dichas plantillas no puedan usarse para generar modelos de procesos de integración y además dejen de usarse los respectivos modelos de procesos de integración generados a partir de las plantillas. Para ello, este servicio invoca los servicios `TemplateModelVersionManagement` y `IntegrationBusinessProcessModelVersionManagement`.

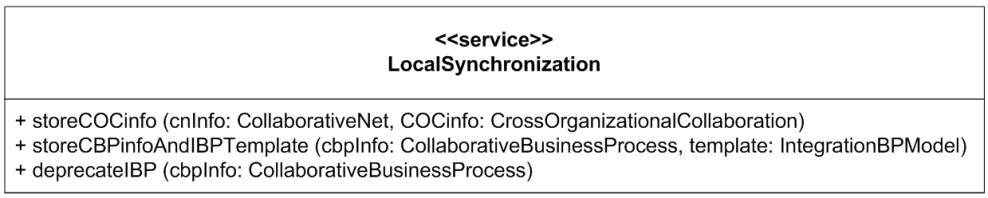


Figura 6.25: Servicio `LocalSynchronization`.

Servicio LocalUserManagement

El servicio `LocalUserManagement` tiene a su cargo gestionar los usuarios habilitados para acceder al repositorio local. Las operaciones provistas (Figura 6.26) son las siguientes:

- La operación `addUser` agrega un usuario al repositorio local.
- La operación `removeUser` elimina un usuario del repositorio local.
- La operación `updateUser` actualiza la información correspondiente a un usuario.
- La operación `getUser` obtiene información correspondiente a un usuario.

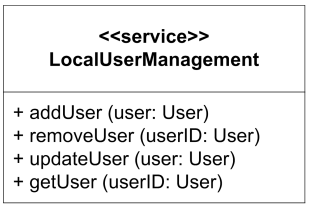


Figura 6.26: Servicio `LocalUserManagement`.

6.1.3. La capa de presentación

La *capa de presentación* se refiere a las aplicaciones cliente que proveen las interfaces de usuario que permiten a las organizaciones acceder e interactuar con el repositorio distribuido y hacer uso de sus funcionalidades. En caso que el repositorio se implemente como un repositorio lógicamente distribuido, se provee la misma aplicación Web para acceder tanto al repositorio global como a los repositorios locales a través de Internet. En caso que el repositorio se implemente como un repositorio físicamente distribuido, se provee una aplicación Web para acceder al repositorio global a través de Internet y otra aplicación Web para que cada organización acceda a su repositorio local a través de Internet/Intranet.

Las organizaciones pueden implementar sus propias aplicaciones clientes para acceder a su repositorio local. Otras aplicaciones cliente, tales como herramientas de modelado de procesos o agentes de software, pueden también interactuar con el repositorio distribuido. Una organización puede delegar en agentes de software la responsabilidad de establecer acuerdos dinámicos con otras organizaciones como propone Tello-Leal y otros (2011) para colaborar y ejecutar procesos colaborativos mediante la creación, búsqueda, selección y recuperación de colaboraciones inter-organizacionales y modelos de procesos colaborativos en el repositorio global.

6.2. Implementación del repositorio distribuido

El repositorio fue implementado usando las siguientes tecnologías: el plug-in de Eclipse EMF para almacenar modelos de procesos en formato XMI, Servicios Web basados en los estándares WSDL y SOAP para implementar los servicios definidos, J2EE para implementar el servidor de aplicaciones, el servidor de base de datos MySQL (Oracle, 2013), el servidor Web Apache Tomcat (Apache, 2013b) y el motor de servicios Web Apache Axis2/Java (Apache, 2013a).

La arquitectura del repositorio fue implementada siguiendo el patrón de diseño *Modelo - Vista - Controlador (MVC)*, que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. Este patrón fue aplicado usando el "framework" *Spring MVC* (Walls, 2011). Mediante este "framework", todas las solicitudes se canalizan a un único controlador central, denominado *DispatcherServlet*, el cual es un "servlet" que delega la responsabilidad de una solicitud a otros componentes de la aplicación Web que la procesan.

La capa de datos de la arquitectura del repositorio representa la capa *Modelo* del patrón MVC. Esta capa fue implementada aplicando el patrón de diseño Objeto de Acceso a Datos - Data Access Object (DAO) para aislar la aplicación de la tecnología de persistencia Java subyacente. En esta capa se utilizó el "framework" *Hibernate* (Bauer y King, 2005), el cual es una herramienta de Mapeo Objeto-Relacional - Object-Relational Mapping (ORM), que facilita el mapeo de atributos entre la base de datos relacional y el modelo de objetos de la aplicación. El mapeo se realizó usando "annotations" (metadatos) que se agregan directamente a las clases Java. El Listado 6.1 muestra un fragmento de la clase Java CrossOrganizationalCollaboration.java encargada de persistir la entidad `Cross-OrganizationalCollaboration`. También se aplicó la técnica *Generics* que permite la reutilización de código, facilitando la creación de las clases DAO.

Listado 6.1: Fragmento de la Clase Java CrossOrganizationalCollaboration.java

```
1 package com.globalRepository.model;
2
3 import java.util.Date;
4 import java.util.List;
5
```

```
 6 import javax.persistence.CascadeType;
 7 import javax.persistence.Column;
 8 import javax.persistence.Entity;
 9 import javax.persistence.FetchType;
10 import javax.persistence.GeneratedValue;
11 import javax.persistence.GenerationType;
12 import javax.persistence.Id;
13 import javax.persistence.ManyToMany;
14 import javax.persistence.ManyToOne;
15 import javax.persistence.OneToMany;
16 import javax.persistence.OneToOne;
17 import javax.persistence.Table;
18
19 /**
20  * Information about collaborative association between organizations
21  */
22
23 @Entity
24 @Table(name = "CrossOrganizationalCollaboration")
25 public class CrossOrganizationalCollaboration {
26
27   @Id
28   @GeneratedValue(strategy = GenerationType.IDENTITY)
29   @Column(name = "idCOC")
30   private Integer idCOC;
31
32   // Date the collaboration was established.
33   @Column(name = "creationDate")
34   private Date creationDate;
35
36   @Column(name = "name")
37   private String name;
38
39   // A CrossOrganizationalCollaboration defines a CollaborativeAgreement. If
40   // the CrossOrganizationalCollaboration is eliminates,
41   // the CollaborativeAgreement must be deleted too
42   @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
43   private CollaborativeAgreement collaborativeAgreement;
44
45   // A CrossOrganizationalCollaboration behaves according to one or more
46   // CollaborativeBusinessProcess.
47   @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "
         collaboration")
48   private List<CollaborativeBusinessProcess> cbProcesses;
49
50   // A CrossOrganizationalCollaboration contain two Organizations and theirs
51   // roles.
52   @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
53   private OrganizationAndRole organizationRole1;
54
55   @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
```

```
56  private OrganizationAndRole organizationRole2;
57
58  // The Collaboration can contain one or more BusinessDocumentTypes.
59  @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
60  private List<BusinessDocumentType> documentsTypes;
61
62  // This attribute specify the CollaborativeBussinessProcess that is actually
63  // been used.
64  @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
65  private CollaborativeBusinessProcess actualProcess;
66
67  // This attribute is used to identify the collaborative Network where this
68  // Collaboration
69  // takes place.
70  @ManyToOne
71  private CollaborativeNetwork network;
72
73  // CONSTRUCTOR
74
75  public CrossOrganizationalCollaboration() {
76    super();
77  }
78  // SETTERS AND GETTERS
79 ...
```

La capa de servicios de la arquitectura del repositorio representa la capa *Controlador* del patrón MVC. Esta capa contiene toda la lógica de la aplicación, principalmente los servicios definidos que soportan las funcionalidades del repositorio global y los repositorios locales. En esta capa, los servicios se implementaron como servicios Web basados en WSDL, cuyas invocaciones se gestionan usando el motor de servicios Web *Apache Axis2/Java*. Como ejemplo de definición de servicio Web, en la Figura 6.27 se muestra la interfaz de Servicio Web del servicio `IntegrationBPTemplateGeneration` (Sección 6.1.2.1) y en el Listado 6.2 se muestra un extracto del documento WSDL correspondiente a dicho servicio.

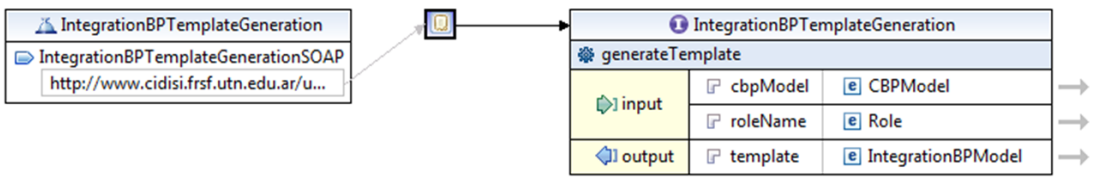


Figura 6.27: Servicio Web IntegrationBPTemplateGeneration.

Listado 6.2: Fragmento del documento WSDL IntegrationBPTemplateGeneration

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="
      http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2/
      IntegrationBPTemplateGeneration/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl
      /" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
      IntegrationBPTemplateGeneration" targetNamespace="http://www.cidisi.frsf.utn.
      edu.ar/upcolpip2bpmn2/IntegrationBPTemplateGeneration/">
3   <wsdl:types>
4     <xsd:schema
5         ...
6         </xsd:schema>
7   </wsdl:types>
8   <wsdl:message name="generateTemplateRequest">
9     <wsdl:part element="tns:CBPModel" name="cbpModel" />
10    <wsdl:part name="roleName" element="tns:Role"></wsdl:part>
11  </wsdl:message>
12  <wsdl:message name="generateTemplateResponse">
13    <wsdl:part element="tns:IntegrationBPModel" name="template" />
14  </wsdl:message>
15  <wsdl:portType name="IntegrationBPTemplateGeneration">
16    <wsdl:operation name="generateTemplate">
17      <wsdl:input message="tns:generateTemplateRequest"/>
18      <wsdl:output message="tns:generateTemplateResponse"/>
19    </wsdl:operation>
20  </wsdl:portType>
21  <wsdl:binding name="IntegrationBPTemplateGenerationSOAP"
22    type="tns:IntegrationBPTemplateGeneration">
23    <soap:binding style="document"
24      transport="http://schemas.xmlsoap.org/soap/http" />
25    <wsdl:operation name="generateTemplate">
26      <soap:operation
27         soapAction="http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2/
               IntegrationBPTemplateGeneration/generateTemplate" />
28      <wsdl:input>
29        <soap:body use="literal" />
30      </wsdl:input>
31      <wsdl:output>
32        <soap:body use="literal" />
33      </wsdl:output>
34    </wsdl:operation>
35  </wsdl:binding>
36  <wsdl:service name="IntegrationBPTemplateGeneration">
37    <wsdl:port binding="tns:IntegrationBPTemplateGenerationSOAP" name="
          IntegrationBPTemplateGenerationSOAP">
38      <soap:address location="http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2"/>
39    </wsdl:port>
40  </wsdl:service>
41 </wsdl:definitions>
```

La lógica de los servicios `InterfaceBPModelVersionManagement` y `IntegrationBPTemplateGeneration` fue implementada haciendo uso de los plug-in de Eclipse basado en ATL descritos en los Capítulos 4 y 5, los cuales implementan los métodos de transformación de modelos propuestos. La lógica del servicio `CBPmodelCorrectnessChecking` se implementó haciendo uso del plug-in de Eclipse para verificación presentado en (Roa y otros, 2012).

La capa de presentación de la arquitectura del repositorio representa la capa *Vista* del patrón MVC. Esta capa contiene todas las interfaces gráficas que serán accesibles a los usuarios que interactúan con el repositorio. Esta capa se implementó usando *Wicket* (Dashorst y Hillenius, 2008), el cual es un "framework" basado en componentes codificados enteramente en Java y mapeados a código HTML. Por lo tanto, cada página de la aplicación Web es implementada por dos archivos: un archivo `.java` (por ejemplo, Home.java) y un archivo `.html` (por ejemplo, Home.html). Como ejemplo, la Figura 6.28 muestra la página Web Home del repositorio global.

El repositorio distribuido soporta la gestión de modelos de procesos colaborativos basados en el lenguaje UP-ColBPIP en formato XMI, de acuerdo al plug-in EMF de Eclipse, y modelos de procesos de interfaz e integración basados en el lenguaje BPMN en formato XML, de acuerdo al estándar BPMN. Por consiguiente, los modelos de procesos colaborativos pueden ser creados y editados

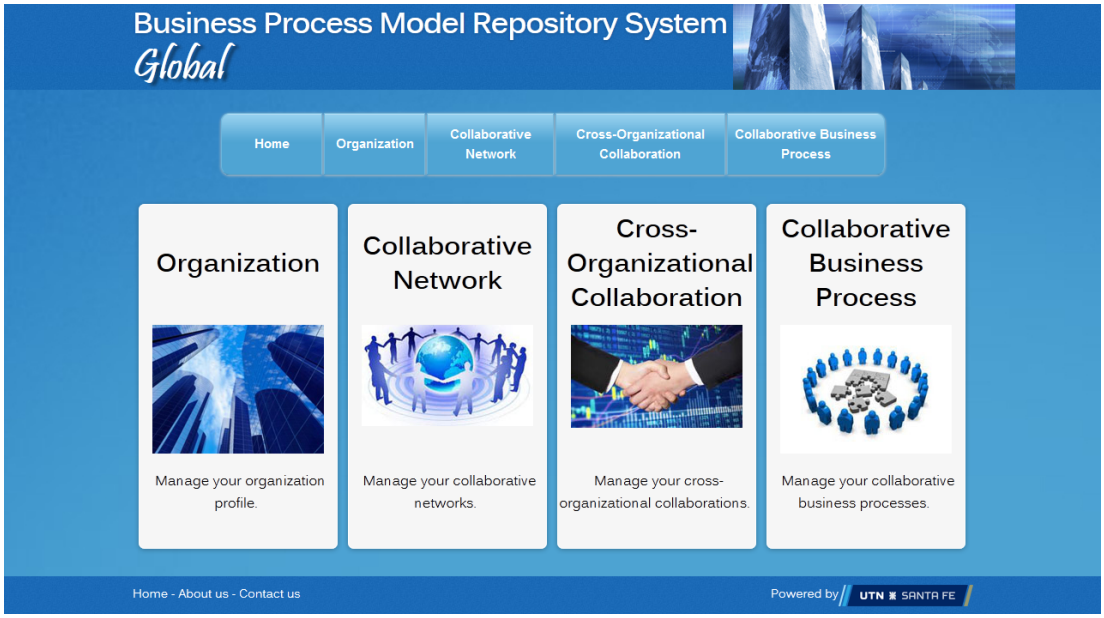


Figura 6.28: Página Home.html del repositorio global.

usando el plug-in UP-ColBPIP de Eclipse (Villarreal y otros, 2010). Por otro lado, los modelos de procesos de interfaz e integración pueden editarse con cualquier herramienta de modelado que soporte el estándar BPMN, como por ejemplo, el plug-in BPMN 2.0 Modeler de Eclipse (The Eclipse Foundation, 2011).

6.3. Conclusiones

El sistema de repositorio distribuido propuesto posibilita gestionar modelos de procesos de negocio para colaboraciones inter-organizacionales. En base a los principios de diseño orientado a servicios y SOA fue posible definir servicios interoperables, reusables, distribuidos y débilmente acoplados.

La arquitectura del repositorio permite a las organizaciones: acceder a un repositorio global para gestionar redes colaborativas, colaboraciones inter-organizacionales, modelos de procesos colaborativos y procesos de interfaz; y que cada una implemente su repositorio local para gestionar modelos de procesos de integración interoperables. El repositorio distribuido permite mantener dichos modelos sincronizados y consistentes con los correspondientes modelos de procesos colaborativos preservando los aspectos privados.

Las funcionalidades disponibles en el repositorio distribuido hacen del mismo una herramienta de ayuda en el desarrollo e implementación de colaboraciones inter-organizacionales, en particular aplicando la metodología presentada en el Capítulo 3. Brinda soporte para la fase *Análisis Inter-organizacional* permitiendo almacenar y gestionar la información de los artefactos que se definen en dicha fase (acuerdos de colaboración y metas de negocio); y para la fase de *Diseño de la Solución Inter-organizacional* permitiendo diseñar y gestionar los modelos de procesos colaborativos, de interfaz y de integración.

El repositorio provee las funcionalidades necesarias para gestionar la evolución de los modelos a través del manejo y sincronización de versiones de los mismos.

Mediante los catálogos de modelos de referencia de procesos colaborativos mantenidos en el repositorio global se da soporte a la tarea de diseño, ya que estos catálogos pueden ser re-usados por las organizaciones para definir modelos de procesos colaborativos y agregarlos al repositorio global.

La implementación e integración en el repositorio de los métodos basados

en MDA propuestos en los capítulos anteriores permite proveer servicios para generar automáticamente modelos de procesos de interfaz, plantillas y modelos de procesos de integración a partir de modelos de procesos colaborativos. Las plantillas pueden ser usadas por las organizaciones como un esqueleto para definir modelos de procesos de integración y agregarlos a sus repositorios locales.

El repositorio provee servicios de verificación de correctitud de modelos de procesos colaborativos. Esto permite detectar errores en las primeras etapas del desarrollo donde se toman las principales decisiones de diseño generando así modelos de procesos de interfaz y de integración libres de errores lógicos. El servicio de chequeo de consistencia de modelos de procesos de integración da a las organizaciones la alternativa de diseñar manualmente modelos de procesos de integración a partir de plantillas y verificar luego la consistencia de dichos modelos con la lógica definida en los procesos colaborativos.

CAPÍTULO 7

Caso de Estudio

El capítulo presenta un ejemplo de prueba de concepto y validación, describiendo el uso de un repositorio distribuido para gestionar los modelos de procesos de negocio que intervienen en el desarrollo y gestión de colaboraciones inter-organizacionales. Las colaboraciones definidas están basadas en estándares de referencia internacionales de gestión de cadenas de suministro colaborativas. Se describe el contexto del caso de estudio (Sección 7.1); y la aplicación del repositorio para dicho caso de estudio (Sección 7.2). Finalmente se presentan las conclusiones (Sección 7.3).

7.1. Introducción

El caso de estudio refiere al dominio de colaboraciones inter-organizacionales dirigidas a la gestión integrada de cadenas de suministro. La cadena de suministro consiste en una red de distribución de productos de la industria alimenticia, conformada por tres organizaciones. Un fabricante, denominado Nabco, y dos minoristas (cadenas de supermercados al por menor o "Retailers"), denominados WemarFood y MartStores. El propósito de la colaboración es mejorar el desempeño de la cadena, disminuir inventarios y mejorar el nivel de servicio a los consumidores.

La colaboración inter-organizacional se plantea entre el fabricante y cada minorista por separado mediante interacciones par a par ("peer-to-peer"), siguiendo los estándares de referencia de modelos de negocio colaborativos CPFR y VMI. Esta forma de interacción tiene como principales beneficios: la ejecución descentralizada del proceso de colaboración y la preservación de la autonomía de las organizaciones (Villarreal y otros, 2003). Nabco establece una colaboración con WemarFood basada en el modelo CPFR, y con MartStores establece otra colabo-

ración basada en el modelo VMI. Ambos modelos de gestión colaborativos están enfocados en cadenas de suministro dirigidas por demanda, orientados a alinear las metas de negocio y coordinar operaciones entre fabricantes y distribuidores o minoristas.

Se implementó un repositorio distribuido prototipo para crear un escenario en el que estas organizaciones conforman una red de distribución colaborativa, y almacenan y comparten los detalles de las colaboraciones inter-organizacionales. El propósito principal es mostrar cómo las organizaciones pueden gestionar los respectivos modelos de procesos de negocio, tanto en la etapa de desarrollo de la solución inter-organizacional de las colaboraciones que acuerdan, como así también a posteriori durante la gestión del ciclo de vida de dichas colaboraciones.

El prototipo fue implementado y desplegado como un repositorio distribuido físicamente. El repositorio global fue desplegado en un servidor público accesible a través de Internet, administrado por el centro de investigación CIDISI desempeñando el rol de una tercera parte neutral que provee el servicio. Para cada organización se desplegó un servidor con un repositorio local privado. Las organizaciones acceden al repositorio global para definir redes de colaboración, colaboraciones inter-organizacionales, y gestionar los modelos de procesos colaborativos y de interfaz, mientras que para gestionar los modelos de procesos de integración acceden al repositorio local.

7.2. Uso del repositorio distribuido

Para usar el repositorio distribuido, las organizaciones deben cargar su perfil en el repositorio global mediante la página Web principal del servicio (Figura 7.1). Si el usuario ya está registrado puede iniciar sesión, si no debe cargar el perfil de la organización y registrarse en el repositorio. En éste último caso, el usuario debe ingresar datos relativos a la organización. Cuando el usuario selecciona el botón *Register* se invoca la operación `addOrganization` del servicio `OrganizationManagement` del repositorio global.

Ingresado el perfil, se solicitan datos de la persona de contacto y usuario "manager" (responsable o referente de la organización en el repositorio). La operación invocada es `addUser` del servicio `GlobalUserManagement` del repositorio global. En el caso de estudio, se cargaron los perfiles de las organizaciones

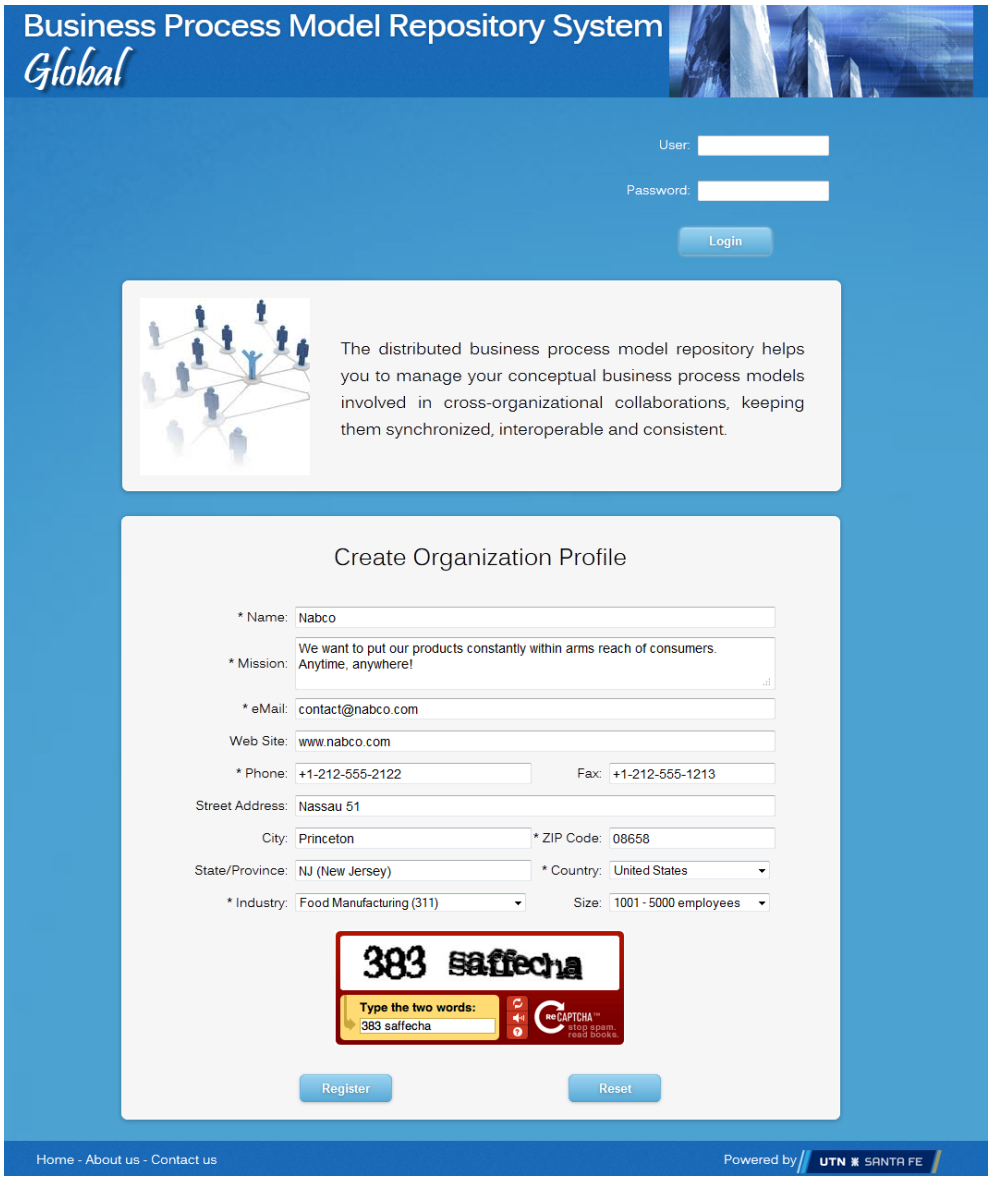


Figura 7.1: Página index.html del repositorio global con información de la organización Nabco.

y se registraron los respectivos usuarios.

7.2.1. Creación de la red colaborativa

Una organización registrada en el repositorio global puede crear una red colaborativa e invitar a otras a integrarla. La creación de una red dispara la operación `addCollaborativeNet` del servicio `CollaborativeNetworkManagement` del repositorio global.

Para el caso de estudio, el usuario de Nabco creó la red *Collaborative Distri-*

bution Network of Foodstuffs. La página Web CollaborativeNetwork del repositorio global (Figura 7.2) muestra las redes colaborativas de las cuales Nabco es miembro.

En la pestaña *Data* de dicha página se muestra información de la red seleccionada. En las restantes pestañas se puede consultar y gestionar sus miembros, las colaboraciones establecidas en dicha red y los catálogos de procesos colaborativos de referencia.

En la pestaña *Members*, el usuario tiene la opción de invitar a otras organizaciones a formar parte de la red. Al seleccionar el botón *Invite Another Organization*, se abre una página (tipo "modal box") en la cual puede seleccionar una organización registrada en el repositorio o ingresar el e-mail de una organización externa al repositorio y realizar la invitación. Al confirmar la in-

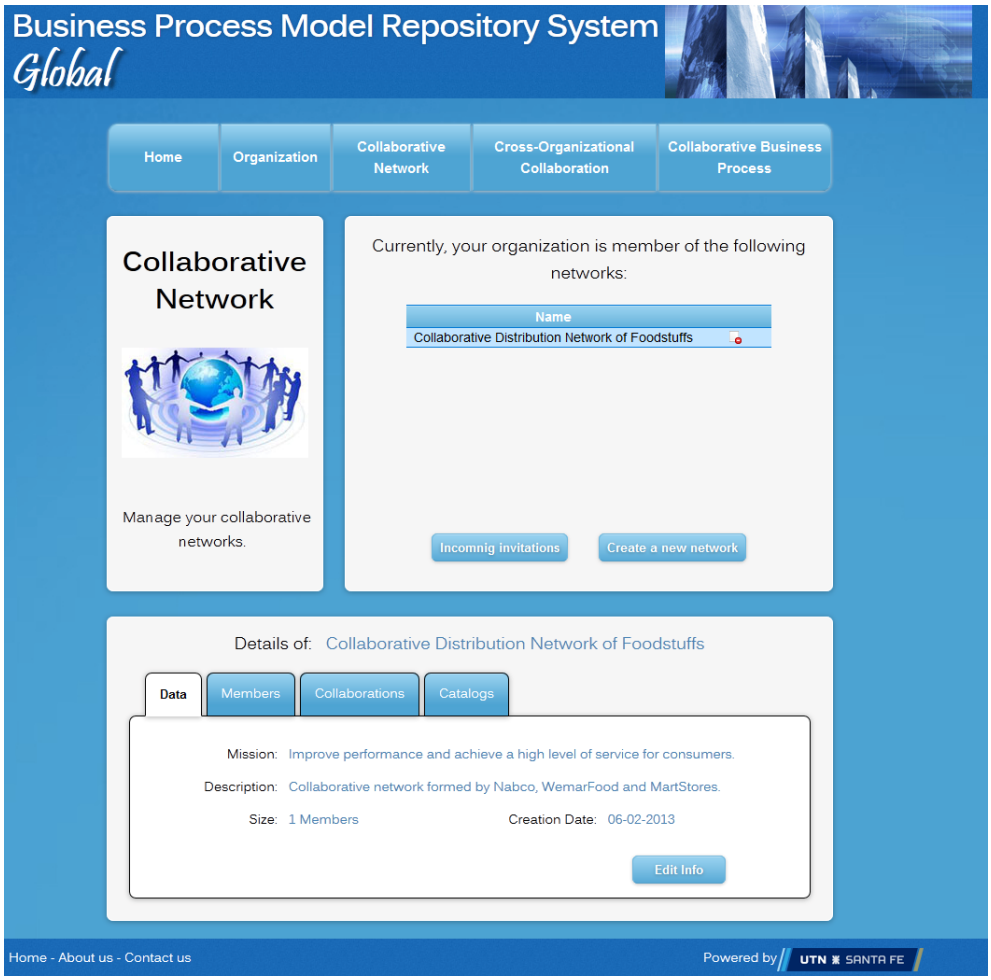


Figura 7.2: Página CollaborativeNetwork.html con información de la red *Collaborative Distribution Network of Foodstuffs*.

vitación, la aplicación Web invoca la operación `sendInvitation` del servicio `CollaborativeNetworkManagement`, la cual genera un e-mail de invitación. En dicho e-mail se indica la misión de la red, sus miembros y un "link" a la página Web del repositorio global a la cual la organización puede acceder para aceptar o rechazar la invitación. En el caso de aceptación, se invoca la operación `addMember` del servicio `CollaborativeNetworkManagement`, que permite agregar la organización a la red. En esta pestaña *Members* (Figura 7.3) se muestra la lista de organizaciones miembros invitadas a formar parte de la red.

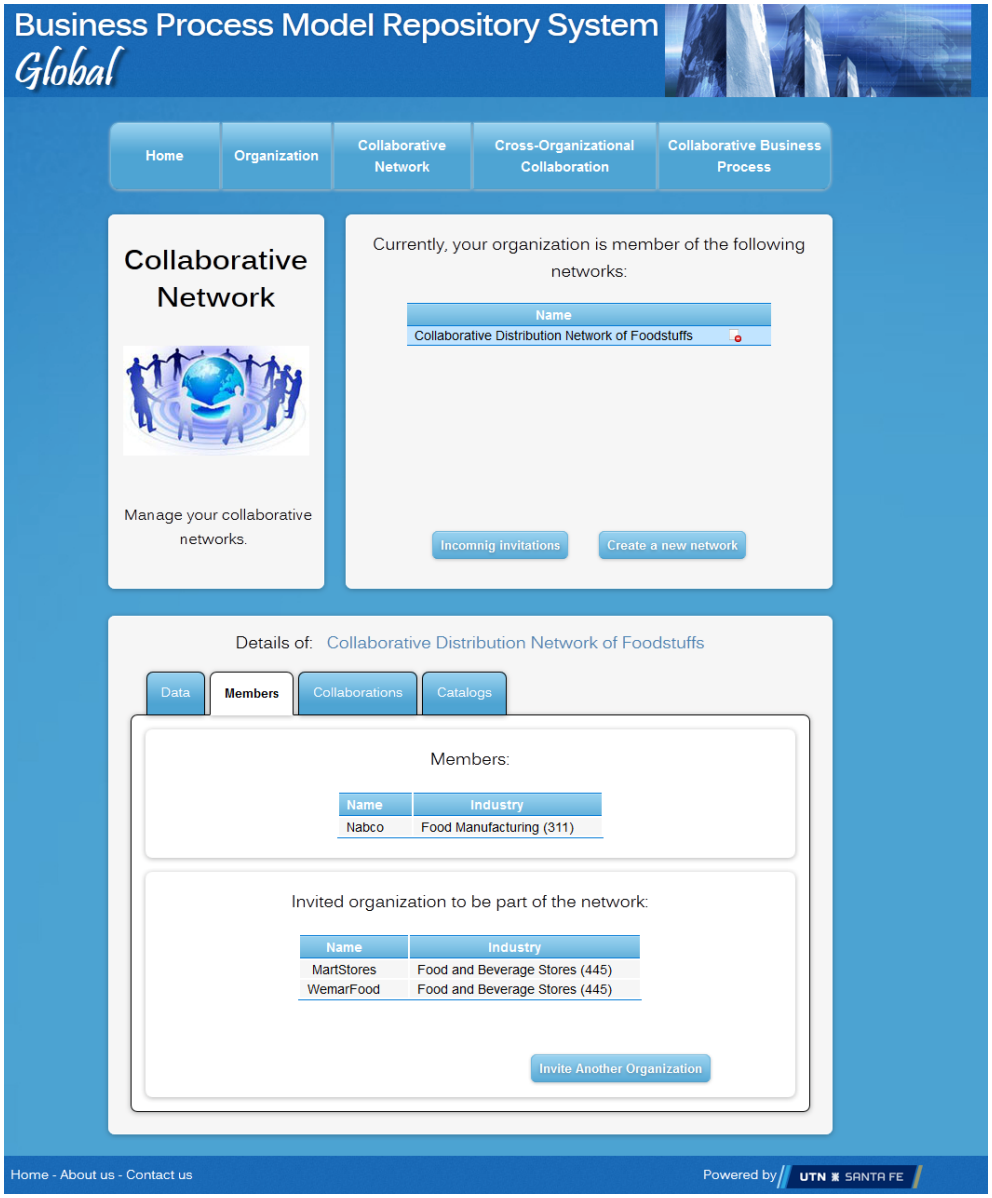


Figura 7.3: Pestaña *Members* de la página CollaborativeNetwork.html con información de las invitaciones enviadas.

En el caso de estudio, el usuario de Nabco envió un e-mail a WemarFood y MartStores invitándolas a unirse a la red creada. Al aceptar la invitación las organizaciones fueron automáticamente incorporadas como miembro de la red. Una vez conformada la red colaborativa, los miembros pueden enviar invitaciones para establecer colaboraciones inter-organizacionales entre ellos.

7.2.2. Definición de colaboraciones inter-organizacionales

En esta sección se describen las colaboraciones inter-organizacionales definidas entre los miembros de la red *Collaborative Distribution Network of Foodstuffs*.

Nabco acordó con WemarFood una colaboración inter-organizacional en base al modelo CPFR (VICS, 2004), el cual propone que los planes de negocio y los pronósticos sean monitoreados y actualizados por ambas organizaciones. Esto se consigue a través de procesos colaborativos que se enfocan en coordinar la negociación y la transferencia de planes de promociones y pronósticos de ventas entre un fabricante y un minorista.

La página Web CrossOrganizationalCollaboration muestra la colaboración creada por el usuario de Nabco, denominada *CPFR-based Collaboration Nabco-WemarFood* (Figura 7.4). Se puede observar que se ha definido un acuerdo de colaboración y las metas de negocio: reducir los niveles de inventario; mejorar la precisión de los pronósticos; reducir el desperdicio; incrementar las ventas; y reducir o eliminar ineficiencias de la cadena de suministro. Las operaciones invocadas son `addCollaborativeAgreement` y `addBusinessGoal` del servicio `CollaborativeAgreementManagement`.

La pestaña *Business Documents* (Figura 7.5) muestra los documentos de negocio definidos para dicha colaboración. Éstos son: *BusinessPlan*; *SalesForecast*; *SalesForecastRequest*; *SalesForecastResponse*; *SalesForecastExceptionCriteria*; *OrderForecast*; *OrderForecastRequest*; *OrderForecastResponse*; *POSdata*; *PlannedEventsData*; *InventoryStrategiesData*; *CurrentInventoryPositionData*; *OrderForecastExceptionCriteria*; y *Order*. La operación invocada es `addBusinessDocument` del servicio `BusinessDocumentTypeManagement`.

La pestaña *Collaborative Business Processes* (Figura 7.6) muestra los procesos colaborativos que el usuario de Nabco definió para alcan-

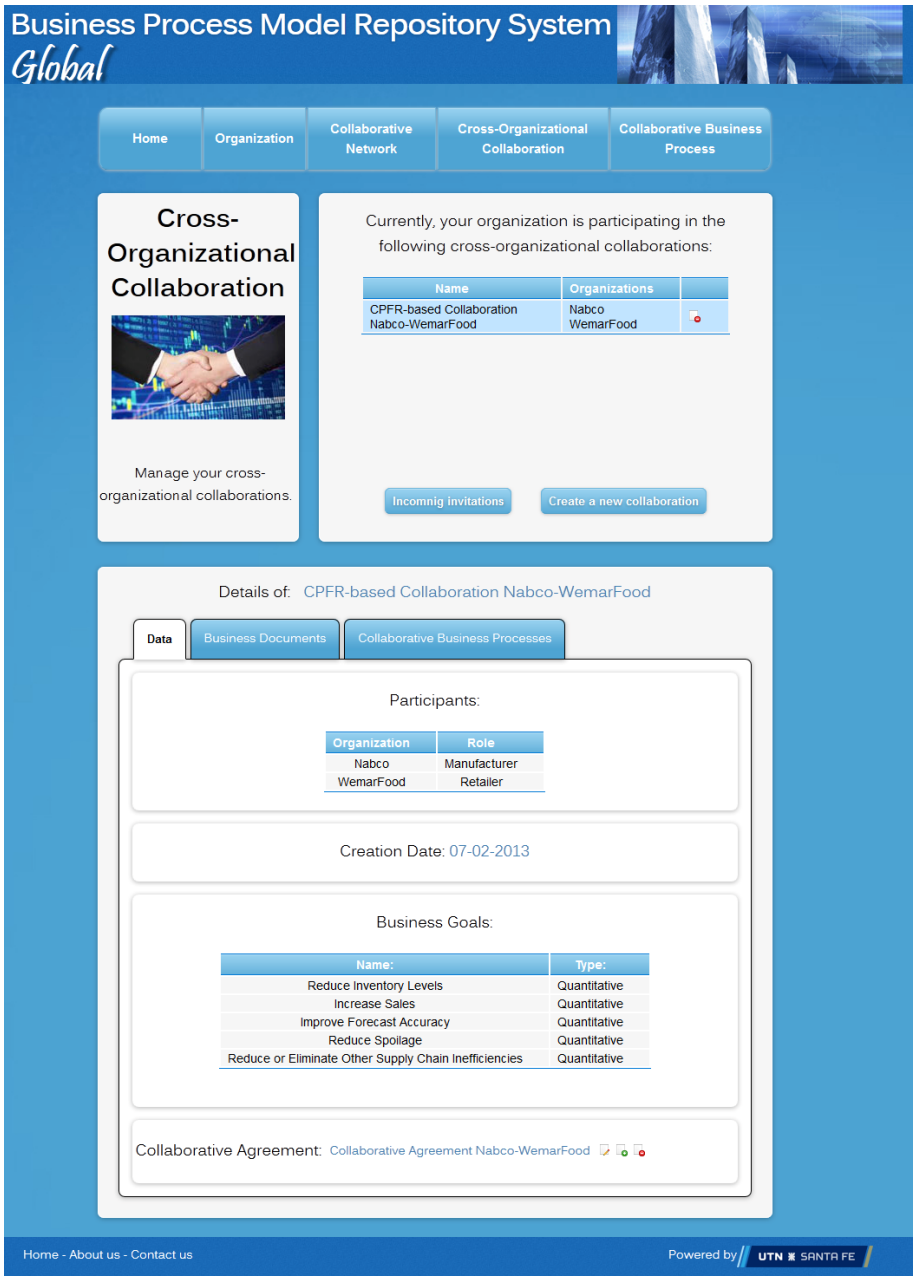


Figura 7.4: Página CrossOrganizationalCollaboration.html con información de la colaboración *CPFR-based Collaboration Nabco-WemarFood*.

zar las metas definidas para dicha colaboración: *CPFR Process*, *Sales Forecasting*, *Order Forecasting*, *Order Generation* y *Delivery Execution*. Para cada uno se cargó sus atributos, nombre, descripción, escenarios de ejecución, pre-condiciones, post-condiciones y la meta asignada. Las operaciones invocadas son `addCBP` y `assignBusinessGoal` del servicio `CollaborativeBusinessProcessManagement`.

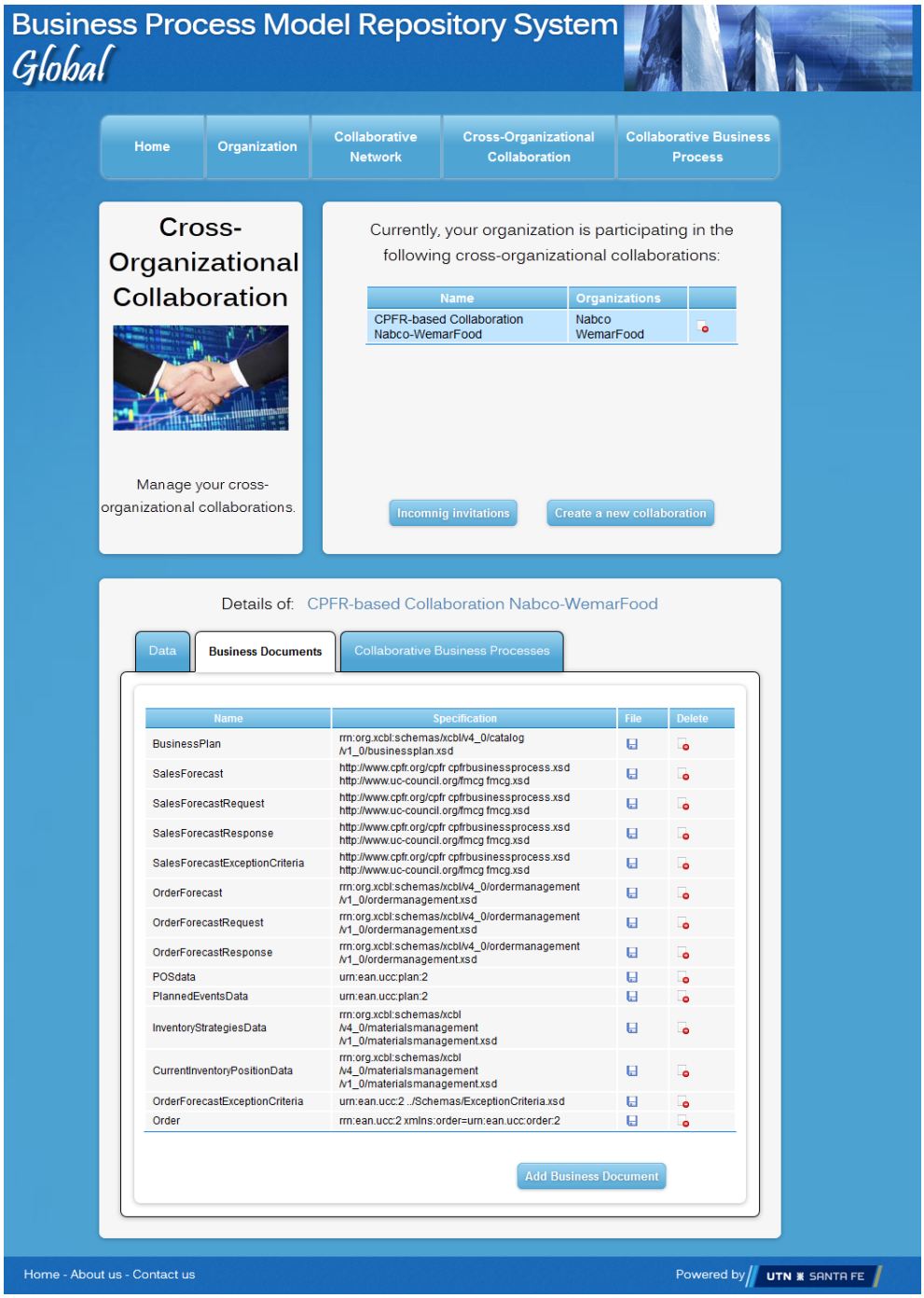


Figura 7.5: Pestaña *Business Documents* de la página CrossOrganizationalCollaboration.html.

Creada la colaboración, el usuario de Nabco invitó a participar de la misma a WemarFood. A través de la ejecución de la operación `sendInvitation` del servicio `CrossOrganizationalCollaborationManagement` se envió un e-mail al usuario de WemarFood conteniendo un "link" a la pági-

Figura 7.6: Pestaña *Collaborative Business Processes* de la página CrossOrganizationalCollaboration.html con los procesos colaborativos definidos para *CPFR-based Collaboration Nabco-WemarFood*.

na Web del repositorio global a la cual puede acceder y aceptar la invitación. Al aceptar, se invocó la operación `addParticipant` del servicio `CrossOrganizationalCollaborationManagement` que agregó a WemarFood como participante de esta colaboración.

De manera similar se creó la colaboración inter-organizacional *VMI-based Collaboration Nabco-MartStores*, cuyos miembros son Nabco y MartStores. En este caso la colaboración está basada en el modelo de negocio colaborativo VMI (Franke, 2010). En este tipo de colaboración, el fabricante es el responsable de la tarea de planificar el aprovisionamiento de productos y controlar los niveles de inventario de sus productos en el minorista. El aprovisionamiento de productos

se realiza en forma colaborativa.

En la página Web CrossOrganizationalCollaboration (Figura 7.7) pueden verse las dos colaboraciones de las que participa Nabco, y los detalles de la colaboración seleccionada *VMI-based Collaboration Nabco-MartStores*. Para esta colaboración se definieron las metas: optimizar el nivel de inventario, incrementar la productividad, incrementar la satisfacción al cliente y reducir costos. Para alcanzar estas metas se definieron los procesos colaborativos: *Forecast-based VMI*, *Blanket Pur-*

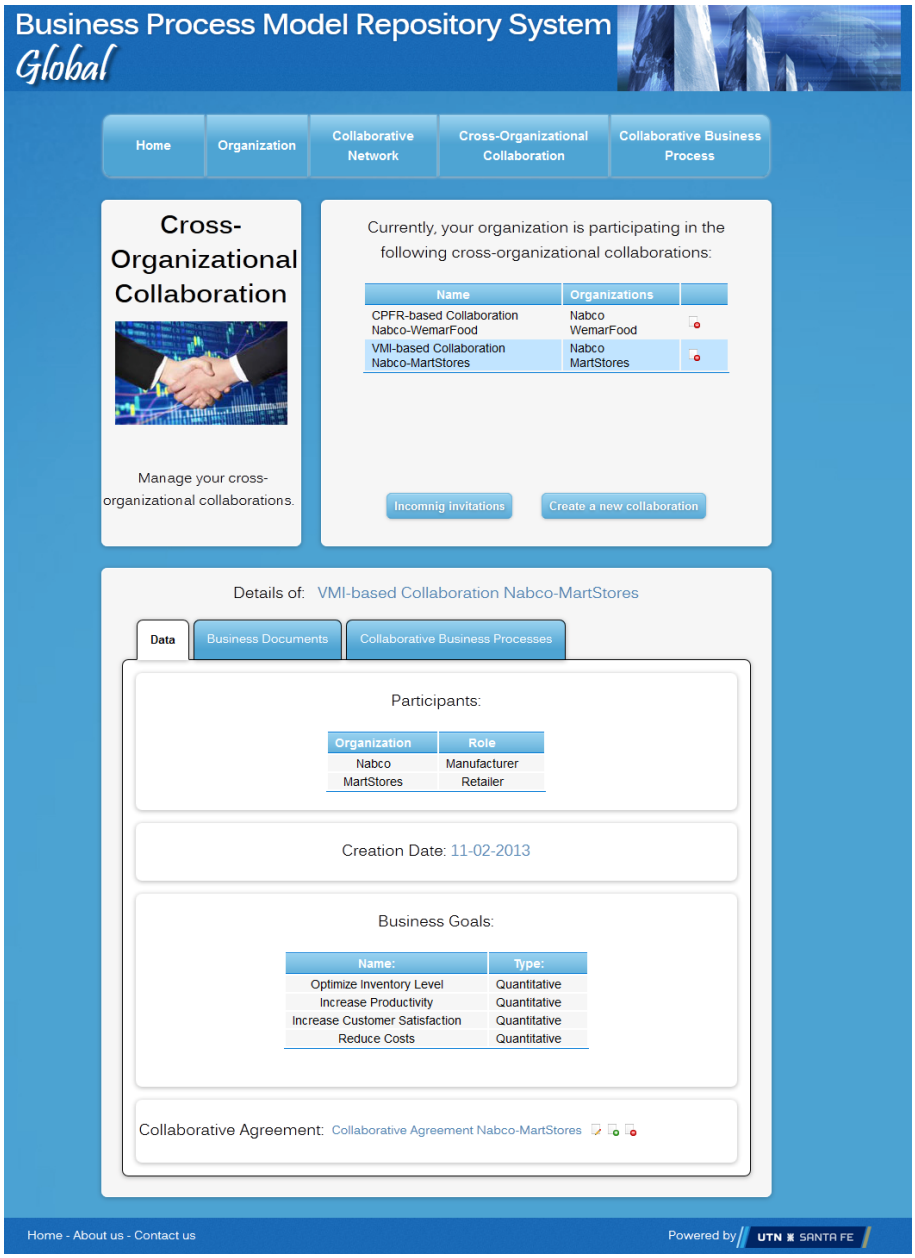


Figura 7.7: Página CrossOrganizationalCollaboration.html con información de la colaboración *VMI-based Collaboration Nabco-MartStores*.

chase Order, *Consumption Schedule*, *Release Against Blanket Purchase Order* y *Replenishment Schedule*.

Creada una colaboración, con sus miembros, el acuerdo de colaboración, las metas y al menos un proceso colaborativo, cada organización participante puede cargar y gestionar modelos para los procesos colaborativos ya definidos.

7.2.3. Gestión de modelos de procesos de negocio

Como ejemplo, en esta sección se describe la gestión de los modelos de procesos colaborativos *Order Forecasting* y *Sales Forecasting* correspondientes a la colaboración *CPFR-based Collaboration Nabco-WemarFood*. También, se describe la gestión de las plantillas y de los modelos de procesos de integración generados a partir de estos dos modelos. Los modelos de los restantes procesos colaborativos fueron gestionados de manera similar.

7.2.3.1. Gestión de nuevos modelos de procesos colaborativos

El repositorio distribuido no brinda soporte directo al diseño de modelos de procesos colaborativos. Se debe utilizar un lenguaje y una herramienta externa. En particular, el prototipo solo permite almacenar modelos de procesos colaborativos definidos con el lenguaje UP-ColBPIP. Por lo cual, las organizaciones debieron definir estos modelos usando dicho lenguaje y la herramienta plug-in de Eclipse para UP-ColBPIP (Villarreal y otros, 2010).

La Figura 7.8 muestra la representación gráfica de la primer versión del de modelo del proceso *Order Forecasting* definido con el plug-in de Eclipse para UP-ColBPIP. El modelo es almacenado en el repositorio global en formato XMI (Capítulo 6).

Es proceso *Order Forecasting* fue creado para que los participantes acuerden un pronóstico de órdenes a corto/mediano plazo. La versión del modelo, expresada en un protocolo de interacción, indica que el proceso comienza con el minorista quien solicita un pronóstico de órdenes *OrderForecastRequest* al fabricante.

El fabricante evalúa dicha solicitud y puede responder de dos maneras diferentes: (1) envía una respuesta de aceptación con el mensaje *agree* conteniendo el documento de negocio *OrderForecastRequestResponse*; o (2) envía una respuesta de rechazo con el mensaje *refuse* conteniendo el documento de negocio *OrderFo-*

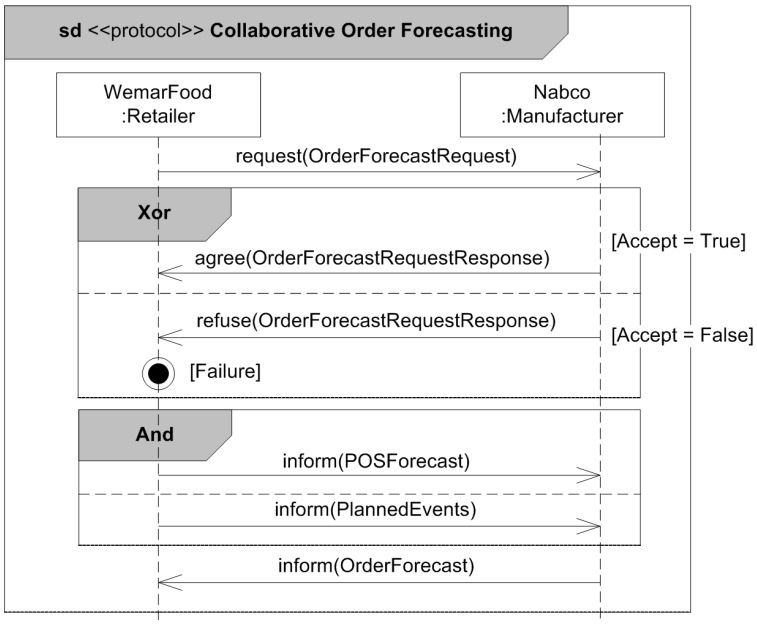


Figura 7.8: Versión del modelo del proceso colaborativo *Order Forecasting*.

recastRequestResponse, y en esta alternativa el proceso finaliza.

Si el fabricante acepta la solicitud, el minorista debe enviarle en forma independiente: el pronóstico de ventas de cada uno de sus puntos de ventas, mediante el mensaje *inform* con el documento de negocio *POSForecast*; y los eventos planificados (promociones y estrategias de ventas), mediante el documento de negocio *PlannedEvents*, para un horizonte de tres meses.

Con la información recibida, el fabricante genera un pronóstico de órdenes *OrderForecast* y lo envía al minorista. Luego, el proceso finaliza.

En la pestaña *Model Version* de la página Web CollaborativeBusinessProcess (Figura 7.9) se puede ver que el usuario de WemarFood cargó la versión de modelo explicada del proceso colaborativo *Order Forecasting*.

Al seleccionar el botón *Add Model* se abre una página (tipo "modal box") en la cual puede indicar datos de la versión de modelo, como por ejemplo, fecha de creación, versión predecesora, y cargar el modelo en formato XMI. Al seleccionar el botón *Save Model* para confirmar la carga de la versión del modelo, se invoca la operación `addCBPModel` del servicio `CBPModelVersionManagement` del repositorio global.

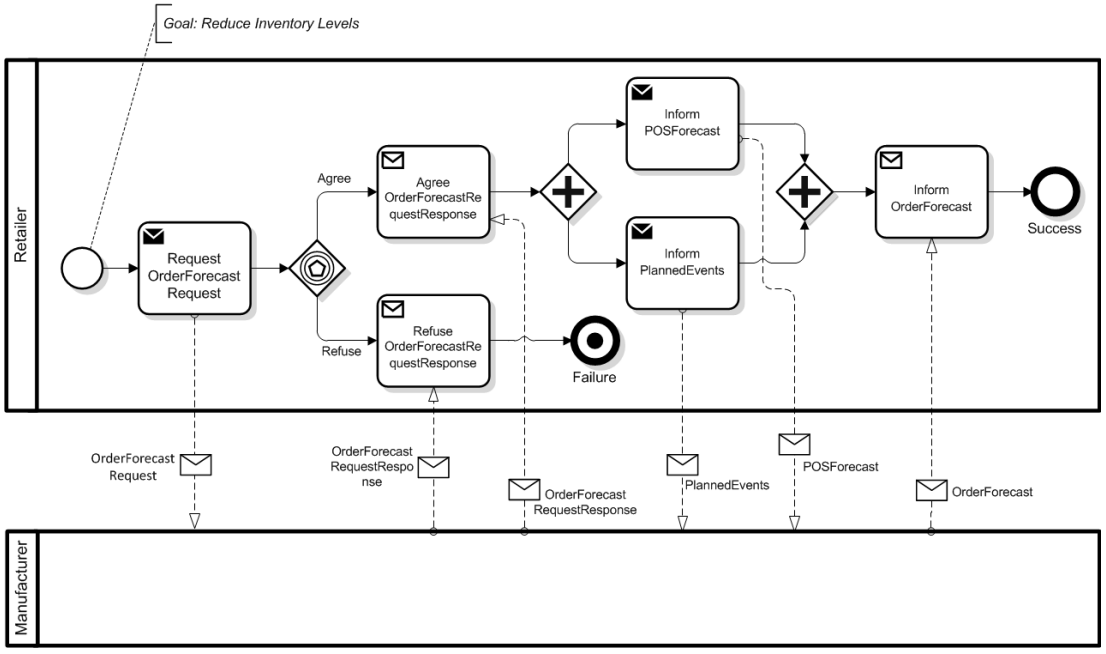
Esta operación disparó una secuencia de interacciones entre los servicios del repositorio global y los repositorios locales de Nabco y WemarFood (Capítulo 6).

La Figura 7.10 muestra la representación gráfica de las versio-

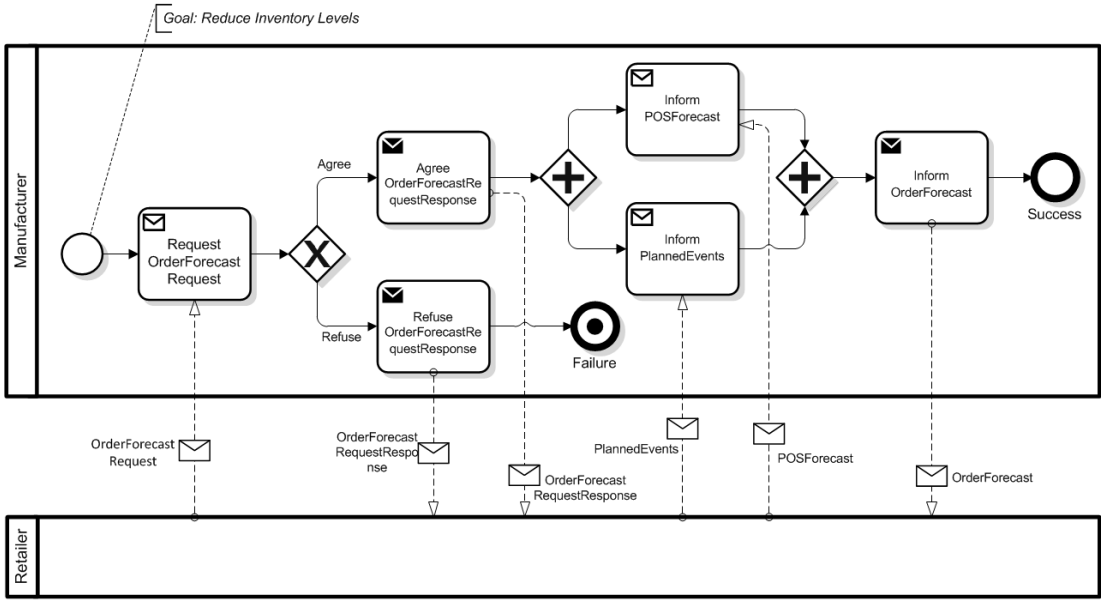
Figura 7.9: Pestaña *Model Version* de la página CollaborativeBusinessProcess.html con información del modelo *Order Forecasting*.

nes de modelos de los procesos de interfaz generados por el servicio `InterfaceBPModelVersionManagement` para los roles *Retailer* y *Manufacturer* correspondientes a la versión de modelo del proceso colaborativo *Order Forecasting*. Éstos modelos son almacenados en el repositorio global en formato BPMN XML (Capítulo 6).

Por cada rol, la información de la colaboración y del proceso colaborativo fue replicada en los correspondientes repositorios locales, a través de la operación `replicateCBPinfo`, del servicio `GlobalSynchronization` del repositorio global. Esto a su vez desencadenó la invocación de la operación `getParticipantList` del servicio `CrossOrganizationalCollaborationManagement` que devolvió *Nabco* y *WemarFood* como las organizaciones a quienes enviar las plantillas



(a) Rol *Retailer* desempeñado por la organización WemarFood.



(b) Rol *Manufacturer* desempeñado por la organización Nabco.

Figura 7.10: Modelos de procesos de interfaz generados por el servicio `InterfaceBPModelVersionManagement` correspondientes al modelo de proceso colaborativo *Order Forecasting*.

correspondientes.

Luego, para cada rol, se generaron las versiones de plantillas de los procesos de integración de las organizaciones, a través del servicio

`IntegrationBPTemplateGeneration` del repositorio global. Luego, en cada caso, se almacenó la versión de plantilla y la información de la colaboración y del proceso colaborativo en los repositorios locales. Esto se realizó a través de la operación `storeCBPinfoAndIBPTemplate` del servicio `LocalSynchronization` del repositorio local de cada organización.

La Figura 7.11 muestra la representación gráfica de la versión de plantilla generada por el servicio `IntegrationBusinessProcessTemplateGeneration` para el rol *Retailer* correspondiente la primera versión de modelo del proceso colaborativo *Order Forecasting*.

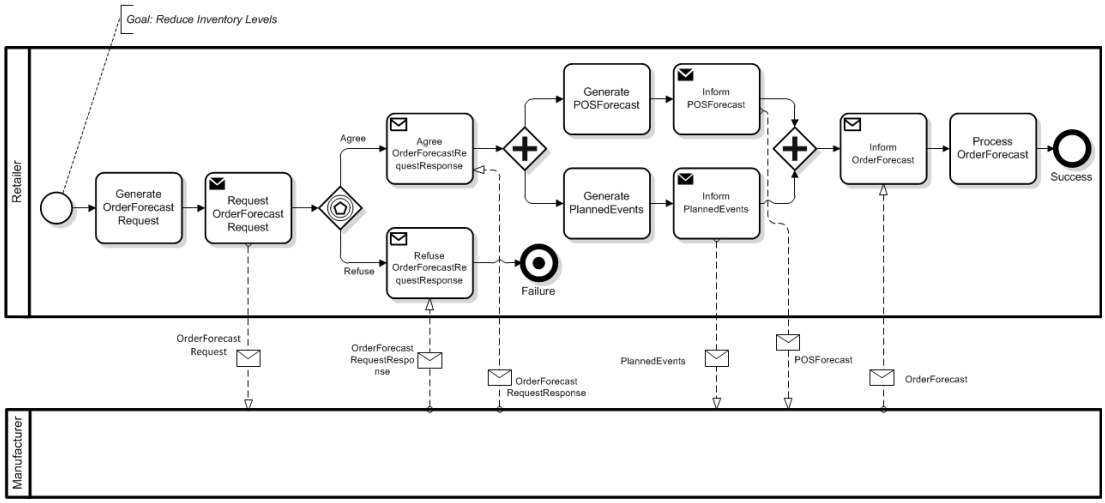


Figura 7.11: Plantilla del proceso de integración generada por el servicio `IntegrationBPTemplateGeneration` para el rol *Retailer* desempeñado por WemarFood.

En la página Web IntegrationBusinessProcess del repositorio local de WemarFood (Figura 7.12) puede verse información del proceso de integración *IBP Collaborative Order Forecasting* correspondiente al proceso colaborativo *Order Forecasting* junto con la versión de plantilla generada *Template Collaborative Order Forecasting*. La misma puede ser descargada por el usuario de la organización con el propósito de usarla como esqueleto para definir una versión de modelo del proceso de integración.

La Figura 7.13 muestra la representación gráfica de la primera versión de modelo del proceso colaborativo *Sales Forecasting* cargada en el repositorio global.

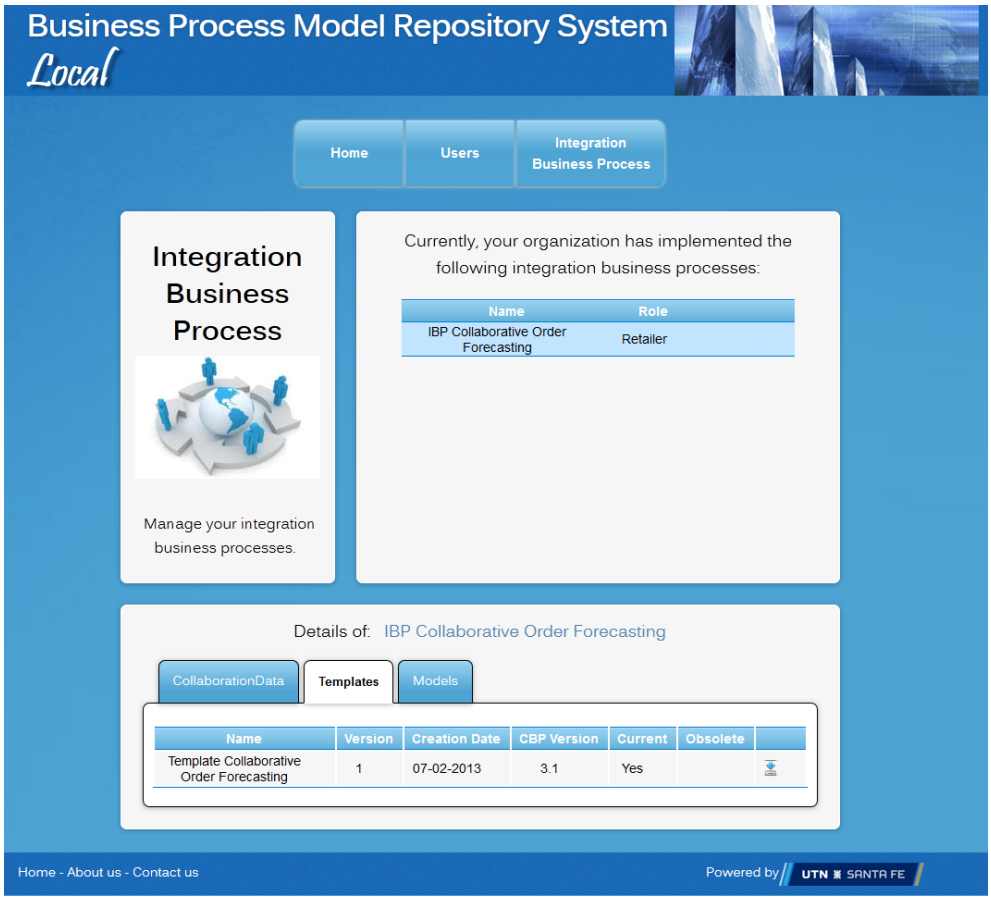


Figura 7.12: Página IntegrationBusinessProcess.html del repositorio local con información del proceso de integración correspondiente al proceso colaborativo *Order Forecasting*.

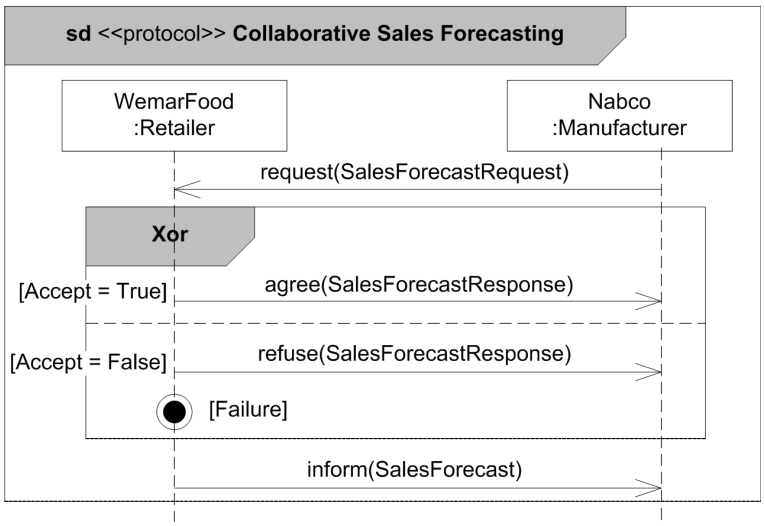


Figura 7.13: Modelo de proceso colaborativo *Sales Forecasting*.

El propósito de este proceso colaborativo es generar un pronóstico de ventas a corto/mediano plazo. El proceso es iniciado por el fabricante solicitando un pronóstico de ventas *SalesForecastRequest* al minorista. En la solicitud indica los datos a considerar en el pronóstico, como por ejemplo, los productos sobre los cuales realizar el pronóstico, el horizonte de tiempo, los períodos, etc.

El minorista evalúa la solicitud y puede responder de dos maneras diferentes: (1) acepta la solicitud y se compromete a generar el pronóstico de ventas, comunicando dicha decisión a través del mensaje *agree* con el documento de negocio *SalesForecastResponse*; (2) rechaza la solicitud a través del mensaje *refuse*, indicando las causas en el documento de negocio *SalesForecastResponse*. Esto significa que el proceso finaliza con una falla.

Si el minorista acepta la solicitud, basado en el plan de negocio acordado y datos internos (información de sus puntos de ventas, promociones, etc.) genera un pronóstico de ventas *SalesForecast* y lo envía al fabricante.

Cuando se cargó la versión de modelo del proceso colaborativo *Sales Forecasting* al repositorio global se invocaron los servicios para verificarlo, como así también aquellos que generaron los modelos de procesos de interfaz y las plantillas correspondientes y replicaron la información en los repositorios locales de las organizaciones Nabco y WemarFood.

En la página Web IntegrationBusinessProcess muestra los procesos de integración *IBP Collaborative Order Forecasting* y *IBP Collaborative Sales Forecasting* generados en el repositorio local de WemarFood (Figura 7.14). También se muestra información de la primera versión de plantilla *Template Collaborative Sales Forecasting* correspondiente al proceso de integración *IBP Collaborative Sales Forecasting* seleccionado en la lista de procesos de integración.

7.2.3.2. Gestión de modelos de procesos de integración

El prototipo del repositorio da soporte al almacenamiento en formato XML de modelos de procesos de integración definidos con el lenguaje BPMN. Los mismos puedes ser editados con una herramienta externa que soporte el estándar BPMN, como por ejemplo, el plug-in BPMN 2.0 Modeler de Eclipse (The Eclipse Foundation, 2011).

Una vez generadas las versiones de plantillas de procesos de integración y almacenadas en los repositorios locales, cada organización puede usarlas para

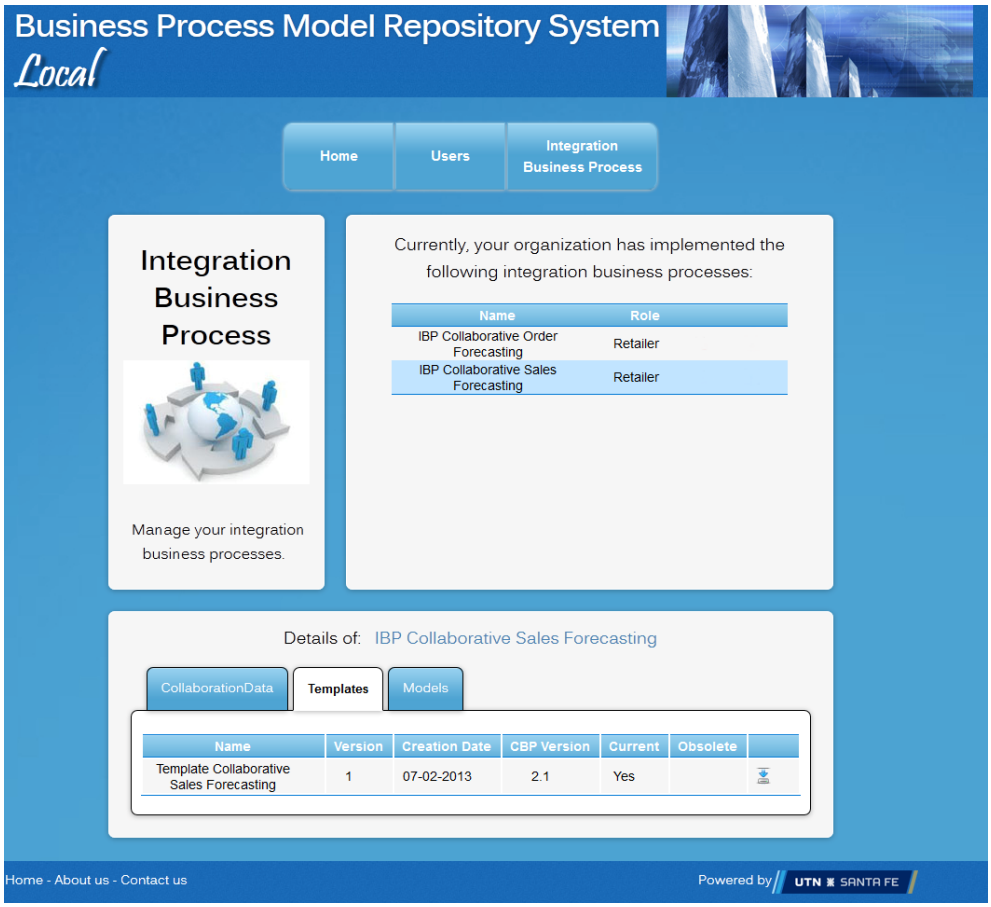


Figura 7.14: Página IntegrationBusinessProcess.html del repositorio local con información de los procesos de integración correspondientes a los modelos de procesos colaborativos cargados en el repositorio global.

definir versiones de modelos de los procesos de integración.

En este caso, para generar una versión de modelo del proceso de integración *IBP Model Collaborative Order Forecasting* a partir de la versión 1 de la plantilla *Template Collaborative Order Forecasting* (Figura 7.11), un usuario de la organización WemarFood descargó dicha plantilla y la editó usando el plug-in BPMN 2.0 Modeler. En el diseño, el usuario redefinió las actividades abstractas generando una primer versión de modelo (Figura 7.15). Dicha versión de modelo luego cargada en el repositorio local.

En la página Web IntegrationBusinessProcess de un repositorio local (Figura 7.16), cuando el usuario selecciona el botón *Add Model* se abre una página (tipo "modal box") en la cual el usuario puede indicar datos de la versión de modelo a agregar, como por ejemplo, fecha de creación, versión de la plantilla usada, una descripción y cargar el modelo en formato XML. Al seleccionar

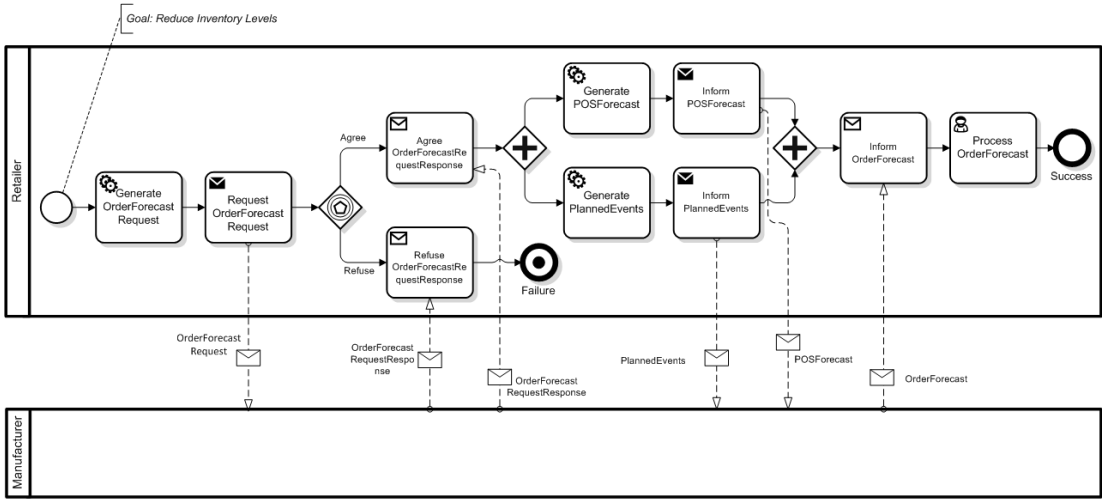


Figura 7.15: Modelo de proceso de integración *IBP Model Collaborative Order Forecasting*.

el botón *Save Model* se invoca la operación `addntegrationIBPModel` del servicio `IntegrationBPModelVersionManagement`, que invoca la invocación la operación `verifyIBPModelConsistency` del servicio `IBPModelConsistencyChecking`. La pestaña *Models* de dicha página para el repositorio local de WemarFood (Figura 7.16) se puede ver que el usuario agregó la versión de modelo del proceso de integración *IBP Model Collaborative Order Forecasting*. Al momento de ser agregada, el resultado de la verificación fue que ambos modelos son consistentes, lo cual indica que la versión de modelo del proceso de integración definida es consistente con la versión de modelo del proceso colaborativo correspondiente.

7.2.3.3. Gestión de una nueva versión de un modelo de proceso colaborativo

Es posible que con el paso del tiempo la solución inter-organizacional de la colaboración *CPFR-based Collaboration Nabco-WemarFood* definida deba adaptarse a nuevos requerimientos de negocio u organizacionales, lo cual implica gestionar nuevas versiones de modelos de procesos.

Como ejemplo, a efectos de mostrar cómo las organizaciones pueden modificar sus procesos, se ha supuesto como escenario que las partes decidieron llevar a cabo más de un ciclo de negociación para definir un pronóstico de órdenes. Esto implicó gestionar una nueva versión del proceso colaborativo *Order Forecasting*.

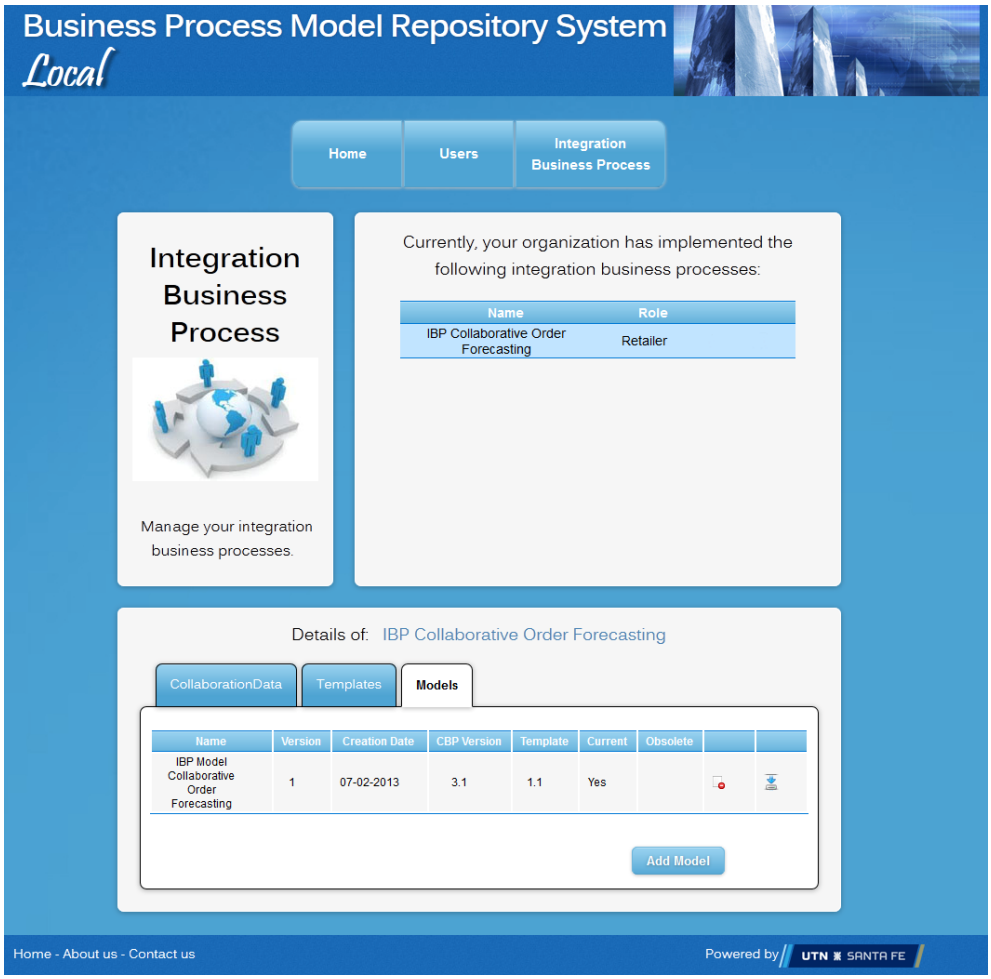


Figura 7.16: Pestaña *Models* de la página IntegrationBusinessProcess.html con información del proceso de integración *IBP Collaborative Order Forecasting*.

En este escenario, el usuario de WemarFood cargó una nueva versión de modelo de dicho proceso (Figura 7.17). En esta versión se puede ver que se agregó un segmento de flujo de control *Loop* para representar un ciclo de negociación que puede repetirse hasta 3 veces en caso de ser necesario para arribar a un acuerdo entre las partes.

Al cargarse la nueva versión de modelo del proceso,la operación `verifyCBPmodelCorrectness` invocada del servicio `CBPmodelCorrectnessChecking` generó un mensaje de error (Figura 7.18), indicando que la versión de modelo agregada no está correctamente definida. El error se debe a que el segundo camino de interacción del segundo segmento de flujo de control *Xor* tiene definida una terminación explícita, y por lo tanto, luego de un rechazo por parte del del minorista a la propuesta del

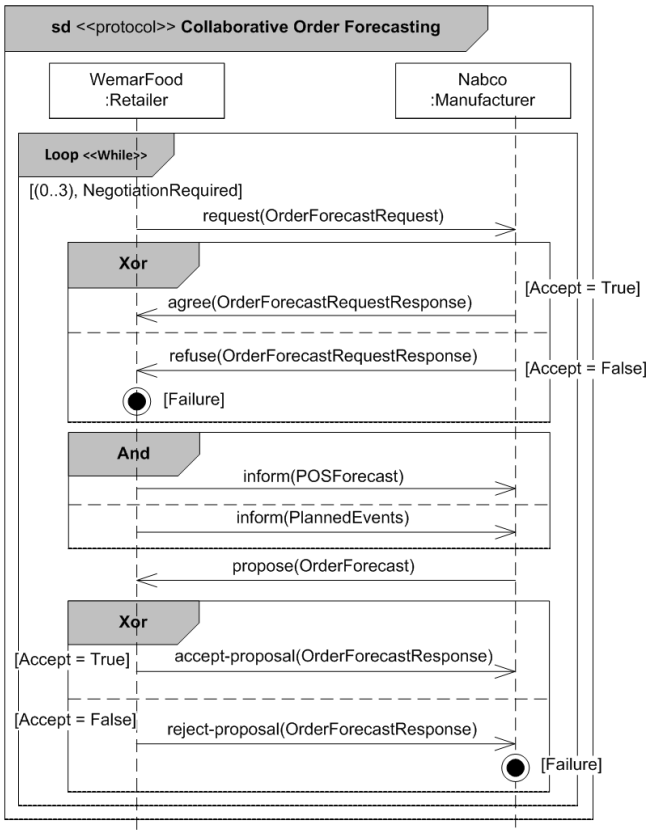


Figura 7.17: Modelo de proceso colaborativo *Order Forecasting* inválido.

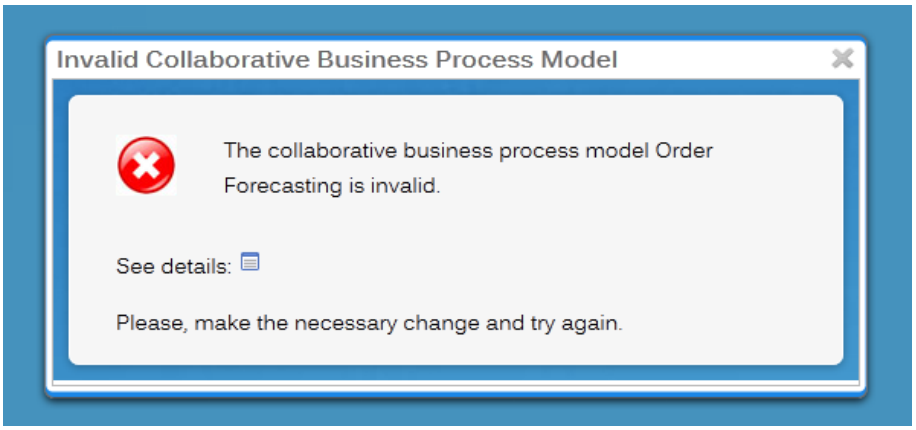


Figura 7.18: Advertencia indicando que el nuevo modelo del proceso colaborativo *Order Forecasting* cargado es inválido.

fabricante no es posible iniciar un nuevo ciclo de negociación para solicitar un nuevo pronóstico de órdenes.

Se realizaron los cambios necesarios empleando la herramienta plug-in de Eclipse para UP-ColBPIP (Figura 7.19).

La nueva versión del modelo de proceso colaborativo *Order Forecasting* se

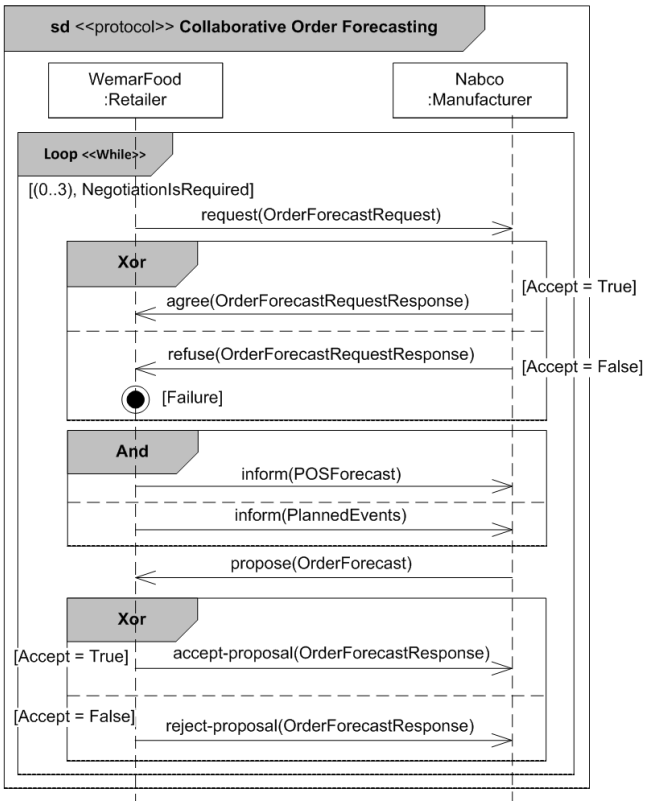


Figura 7.19: Nueva versión del modelo de proceso colaborativo *Order Forecasting*.

cargó al repositorio global. La operación `verifyCBPmodelCorrectness` invocada nuevamente del servicio `CBPmodelCorrectnessChecking` retornó como resultado que el modelo es correcto. Por lo tanto, se generaron luego las correspondientes versiones de modelos de los procesos de interfaz, las nuevas versiones de plantillas de los procesos de integración y se replicaron estas últimas en los repositorios locales de Nabco y WemarFood.

La Figura 7.20 muestra la nueva versión de plantilla generada del proceso de integración *IBP Collaborative Order Forecasting* correspondiente al rol *Retailer* desempeñado por WemarFood.

La Figura 7.21 muestra el repositorio local de la organización WemarFood, indicando que para el proceso de integración *IBP Collaborative Order Forecasting* existen dos versiones de plantillas, de la cual la última es la versión actual.

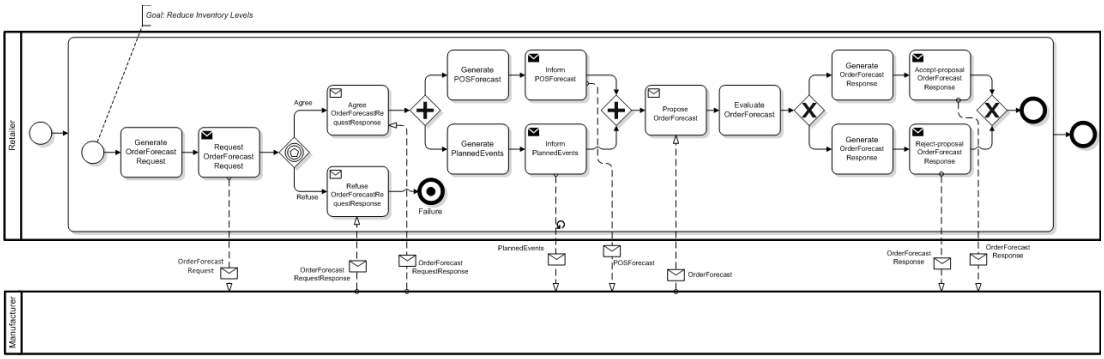


Figura 7.20: Plantilla del rol *Retailer* correspondiente a la nueva versión del modelo de proceso colaborativo *Order Forecasting*.

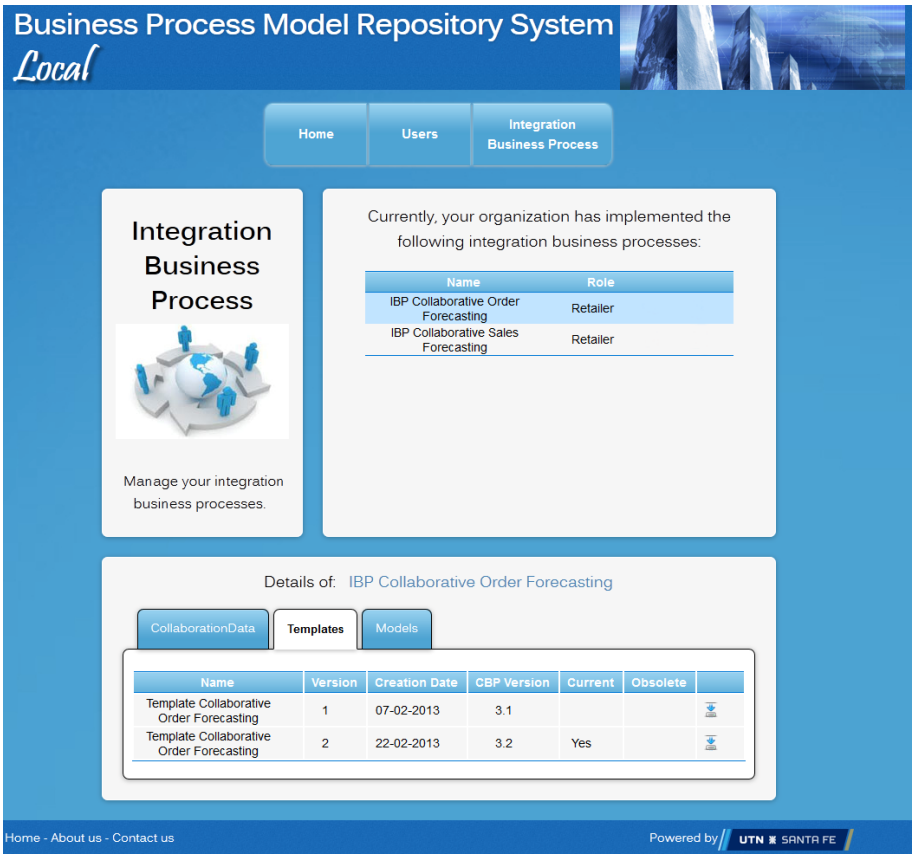


Figura 7.21: Página IntegrationBusinessProcess.html con información de las dos plantillas correspondientes al proceso de integración *IBP Collaborative Order Forecasting*.

7.3. Conclusiones

En este capítulo se presentó la aplicación del repositorio distribuido a un caso de estudio que implica colaboraciones inter-organizacionales basadas en

estándares de referencia internacionales de gestión de cadenas de suministro, en donde se mostraron diferentes escenarios de gestión de modelos de procesos y uso del repositorio. A través de este caso de estudio se pudo mostrar cómo las principales funcionalidades del repositorio dan soporte a los aspectos de gestión de modelos. Se mostró cómo es posible gestionar modelos de procesos tanto en la etapa de desarrollo de colaboraciones inter-organizacionales, como en la etapa de cambios de requerimientos en las colaboraciones.

La implementación del repositorio global a nivel prototipo permitió probar que las organizaciones pueden crear y compartir en un sitio Web público una red de colaboración y colaboraciones inter-organizacionales, pudiendo gestionar modelos de procesos colaborativos y de interfaz. El almacenamiento de versiones de modelos de procesos colaborativos en el repositorio global permitió validar las interacciones entre los servicios del repositorio que dan soporte a la verificación de dichas versiones de modelos y aquellos que mantienen la sincronización entre estas versiones de modelos con las correspondientes versiones de modelos de procesos de integración almacenadas en los repositorios locales.

La generación automática de los modelos de procesos de interfaz y las plantillas de procesos de integración permitió probar el diseño e implementación de los servicios que implementan los métodos propuestos en la tesis (Capítulos 4 y 5). También permitió probar la replicación de la información a los correspondientes repositorios locales para mantener la sincronización entre los modelos.

La implementación de los repositorios locales posibilitó mostrar cómo las organizaciones pueden gestionar en forma privada sus modelos de procesos de integración, y verificar localmente la consistencia de sus versiones de modelos de procesos de integración derivados de una plantilla y almacenados en el repositorio local.

Si bien el prototipo de repositorio distribuido desarrollado implementa el método propuesto en el Capítulo 5, sólo permite la generación automática de plantillas de procesos de integración, no así la generación automática de modelos de procesos de integración basados en un modelo de parámetros de entrada. No obstante, esta funcionalidad es factible de ser implementada, ya que el prototipo posibilita almacenar plantillas y modelos de procesos de integración. Sólo se requiere el soporte para la carga del cuestionario para definir los parámetros de entrada, los cuales se podrían generar a partir de los modelos de procesos

colaborativos almacenados en el repositorio global como archivos XMI.

A través del caso de estudio se mostró cómo el repositorio puede ser una herramienta de ayuda a las organizaciones para llevar a cabo las fases *Análisis Inter-organizacional* y *Diseño de la Solución Inter-organizacional* al momento de aplicar la metodología descrita en el Capítulo 3. A través de los servicios de verificación, generación automática de versiones de modelos de procesos de interfaz y plantillas de procesos de integración, y sincronización de modelos, el repositorio permitió a las organizaciones llevar a cabo la tarea en tiempos reducidos y con bajos costos de desarrollo, además de mantener versiones de modelos, libres de errores y consistentes entre sí.

CAPÍTULO 8

Conclusiones y Trabajos Futuros

El capítulo presenta las principales contribuciones de la tesis y describe los aspectos que constituyen el punto de partida para el desarrollo de trabajos futuros.

8.1. Principales Contribuciones

El trabajo desarrollado en la tesis ha realizado contribuciones en tres áreas de interés: gestión de procesos de negocios, ingeniería de sistemas de información e ingeniería de software. En particular las contribuciones se enmarcan en los siguientes dominios de estas áreas: diseño de modelos de procesos de negocio, metodología y métodos de desarrollo dirigido por modelos, transformaciones de modelos y repositorios de modelos de procesos de negocio.

El trabajo de investigación comenzó con la premisa de que las colaboraciones inter-organizacionales son el principal factor para mejorar la competitividad de las organizaciones. Una colaboración inter-organizacional implica integrar las organizaciones mediante la definición y ejecución conjunta de procesos de negocio colaborativos.

A partir de esta premisa se ha identificado que, para dar soporte al desarrollo y gestión de colaboraciones inter-organizacionales, las organizaciones deben diseñar los procesos colaborativos y sus procesos de negocio internos (de interfaz y de integración), y gestionar conjuntamente los modelos definidos.

El desarrollo de colaboraciones inter-organizacionales es complejo y demanda mucho tiempo, lo cual está limitando su aplicación. Las contribuciones de la tesis permiten a las organizaciones desarrollar y definir de manera eficiente y eficaz la solución inter-organizacional de las colaboraciones, expresada a través de modelos de procesos de negocio.

En base al desarrollo dirigido por modelos se proponen métodos que permiten reducir tiempo y costos de desarrollo mediante la generación y el diseño en forma automática de modelos de procesos de interfaz y de integración. La calidad de los modelos generados es garantizada por la aplicación de técnicas de herencia de reglas, refactorización y superimposición aplicadas a las definiciones de transformación de cada método, logrando la escalabilidad, mantenibilidad y reusabilidad de las mismas.

Estos métodos y el repositorio distribuido garantizan la generación y gestión de modelos de procesos de interfaz y de integración libres de errores, que satisfacen los requerimientos de consistencia, sincronización e interoperabilidad.

A continuación se detallan las principales contribuciones de la tesis.

8.1.1. Metodología para el desarrollo de colaboraciones inter-organizacionales

En primer lugar se ha destacado la importancia de definir una metodología que identifique las fases y los artefactos de desarrollo requeridos para desarrollar e implementar una colaboración inter-organizacional. Aplicando los conceptos del desarrollo dirigido por modelos, se representan los artefactos mediante modelos. Esto permite elevar el nivel de abstracción de los modelos de procesos de negocio y, mediante transformaciones de modelos, evolucionarlos en forma consistente a través de las diferentes fases hasta generar la solución tecnológica que soporta la ejecución descentralizada de procesos colaborativos mediante PAISs. De esta manera, aplicando la metodología se garantiza la consistencia y alineación de la solución tecnológica con la solución inter-organizacional.

La metodología define los modelos de procesos de negocio a diseñar y gestionar en el desarrollo de colaboraciones inter-organizacionales, y los métodos de transformación de modelos a emplear.

8.1.2. Método para la generación de modelos de procesos de interfaz

Como se ha discutido en el Capítulo 4, los modelos de procesos de interfaz pueden generarse completamente a partir de modelos de procesos colaborativos.

Esto se debe a que ambos tipos de procesos expresan los aspectos públicos de las colaboraciones, donde la información necesaria está contenida en los modelos de procesos colaborativos.

El método propuesto está basado en los principios, componentes y estándares propuestos por MDA. Permite a las organizaciones generar procesos de interfaz interoperables y consistentes con la lógica acordada en los procesos colaborativos. Esto es garantizado debido a que los modelos de procesos de interfaz (definidos con el lenguaje BPMN) de cada organización se derivan de un modelo de proceso colaborativo (definido con el lenguaje UP-ColBPIP) aplicando un enfoque "top-down".

Los principales beneficios de este método para el dominio de las colaboraciones inter-organizacionales son:

- Incremento del nivel de abstracción en el diseño de la vista particular que cada organización tiene de la colaboración inter-organizacional.

- Disminución del tiempo de diseño, ya que la generación de los modelos de proceso de interfaz es realizada en forma automática a partir de modelos de procesos colaborativos. Sólo se requiere que el modelador o usuario indique el modelo y el rol a transformar.

- Disminución de los costos como consecuencia de la automatización.

- Garantía de que el comportamiento definido en los modelos de procesos de interfaz se corresponde con el comportamiento definido en los modelos de procesos colaborativos. Para esto, teniendo en cuenta que el comportamiento de un modelo de proceso se expresa a través de la secuencia de actividades y de los constructores de flujo de control, las reglas del método fueron definidas a partir de un mapeo semántico entre los constructores de flujo de control del lenguaje UP-ColBPIP y los constructores de flujo de control del lenguaje BPMN, utilizando la teoría de los patrones de flujo de control de workflows.

8.1.3. Método para la generación de modelos de procesos de integración

Como se discutió en el Capítulo 5, el diseño de procesos de integración es clave para implementar colaboraciones inter-organizacionales. Se requiere del conocimiento y la experiencia de los analistas de negocio o diseñadores de las organizaciones para identificar y agregar las actividades privadas necesarias para generar y procesar la información intercambiada entre las organizaciones.

El método propuesto está basado en MDA y permite generar plantillas y modelos de procesos de integración (que contienen aspectos privados de las organizaciones) definidos con el lenguaje BPMN a partir de modelos de procesos colaborativos (que sólo contienen aspectos públicos) definidos con el lenguaje UP-ColBPIP. Tanto las plantillas como los modelos son interoperables y consistentes con la lógica acordada en los modelos de procesos colaborativos.

Se puede afirmar que este método da soporte al diseño de modelos de procesos de integración, porque para generar los mismos, permite que el usuario ingrese parámetros de entrada. Estos parámetros representan criterios de diseño del usuario o modelador, los cuales son tenidos en cuenta por las reglas del método para generar modelos de procesos de integración de acuerdo a las preferencias del usuario. De esta manera, se contribuye con una nueva forma de definir métodos de desarrollo dirigido por modelos que asistan al diseño, debido a que la mayoría de los métodos de desarrollo dirigido por modelos, para un modelo de entrada dado, siempre generan un mismo modelo de salida.

Los principales beneficios de este método para el dominio de las colaboraciones inter-organizacionales son:

- Disminución del tiempo de diseño de los modelos de procesos de integración. Esto puede realizarse aplicando cualquiera de las dos estrategias que posibilita el método: usando una plantilla como bosquejo o esqueleto inicial y redefiniendo la misma para diseñar un modelo de proceso de integración; o generando en forma automática un modelo de proceso de integración ingresando parámetros de entrada por el diseñador, mediante un cuestionario con preguntas referentes al modelo de proceso colaborativo de entrada de la transformación.

- Disminución de los costos como consecuencia de la automatización en la generación de las plantillas y los modelos de procesos de integración.

- Garantía de consistencia, es decir que el comportamiento definido en las plantillas y los modelos de procesos de integración se corresponde con el comportamiento definido en los modelos de procesos colaborativos a partir del cual son generados. Esto se logra mediante la definición de transformación de modelo a modelo y el re-uso en la misma de las reglas de transformación de constructores de flujo de control definidas en el método de generación de procesos de interfaz.

- Generación de las actividades tanto privadas como públicas requeridas en los procesos de integración, a través de la aplicación de la teoría de patrones de actividades. El conocimiento capturado en estos patrones fue aplicado para derivar las actividades públicas y principalmente privadas que una organización requiere realizar para dar soporte al intercambio de mensajes definidos en un proceso colaborativo

- Garantía de que las plantillas y los modelos de procesos de integración son interoperables con las plantillas y modelos de procesos de integración de las restantes organizaciones que participan en un modelo de proceso colaborativo. Esto es también alcanzado a través del uso de patrones de actividades de workflow, los cuales definen interacciones simples entre dos partes asegurando la sincronización de las actividades públicas y privadas que permiten el intercambio de mensajes.

8.1.4. Repositorio distribuido de modelos de procesos de negocio

Las organizaciones que participan de colaboraciones inter-organizacionales deben gestionar grandes colecciones de modelos de procesos de negocio, lo cual es una tarea compleja debido a que deben mantenerlos consistentes, interoperables, sincronizados y libres de errores lógicos.

El repositorio distribuido propuesto permite gestionar modelos de procesos de negocio para colaboraciones inter-organizacionales. Este repositorio consiste de un *repositorio global* público que permite a las organizaciones gestionar redes

colaborativas, colaboraciones inter-organizacionales, modelos de procesos colaborativos y modelos de procesos de interfaz; y *repositorios locales* privados que permiten a cada organización gestionar sus modelos de procesos de integración, preservando los aspectos privados.

Los principales beneficios del repositorio distribuido para el dominio de las colaboraciones inter-organizacionales son:

- Aseguramiento de la calidad de los modelos de procesos de negocio internos, ya que los servicios que los generan implementan los métodos propuestos en la tesis, garantizando que sean consistentes e interoperables. Esto también se alcanza para los modelos de procesos colaborativos, a través del servicio que implementa un método de verificación que, al momento de ser almacenados en el repositorio global, determina si están libres de errores.

- Soporte al diseño de los modelos de procesos colaborativos, ya que los mismos pueden definirse re-usando o adaptando modelos de referencia de procesos colaborativos obtenidos de catálogos almacenados en el repositorio global.

- Disminución del tiempo de diseño de los modelos de procesos de interfaz e integración, ya que los mismos se generan mediante servicios que implementan e integran los métodos de desarrollo dirigido por modelos propuestos.

- Garantía de que los modelos de procesos colaborativos están sincronizados con los modelos de procesos de interfaz almacenados en el repositorio global y con los modelos de procesos de integración almacenados en los repositorios locales de las organizaciones. La sincronización es garantizada mediante servicios que propagan y trasladan los cambios entre el repositorio global y los repositorios locales. Esta sincronización también es asegurada durante la evolución de los modelos a través de la gestión sincronizada de las versiones de modelos de los procesos.

- Garantía de que los modelos de procesos de integración almacenados en los repositorios locales mantienen su consistencia, ya que los mismos se verifican mediante un servicio que provee el chequeo de consistencia al momento de ser almacenados en los repositorios locales .

8.1.5. Validación del repositorio distribuido y de los métodos propuestos mediante casos de estudio

En el Capítulo 7 se mostró la aplicabilidad del repositorio a un caso de estudio de colaboraciones inter-organizacionales basadas en estándares de referencia internacionales de gestión de cadenas de suministro colaborativas. A través de la implementación del prototipo desarrollado se pudo verificar y mostrar la funcionalidad, factibilidad y utilidad del mismo y de los métodos propuestos para gestionar modelos de procesos de negocio en el desarrollo y gestión de colaboraciones inter-organizacionales.

El repositorio también se validó gestionando la colaboración inter-organizacional del caso de estudio presentado en el Capítulo 3.

8.2. Trabajos Futuros

A partir de la presente tesis, los trabajos futuros de investigación se enmarcan en los puntos que se detallan a continuación:

- Proveer una herramienta de diseño que asista al usuario en el refinamiento de las plantillas de procesos de integración. El propósito es que la herramienta provea y recomiende fragmentos de procesos de negocio aplicables a un determinado contexto.

- Extender el repositorio distribuido para gestionar los artefactos correspondientes a las fases *Diseño de la Arquitectura de TI* y *Desarrollo de la Solución Tecnológica* de la metodología propuesta, es decir, que gestiona no sólo modelos de procesos sino modelos de sistemas y código ejecutable. El propósito es que el repositorio permita que las organizaciones gestionen todos los artefactos requeridos en el desarrollo e implementación de una colaboración inter-organizacional usando una única herramienta.

- Extender el repositorio distribuido para gestionar modelos de procesos de negocio definidos con otros lenguajes de modelado, tales como UML, EPC y WS-BPEL. El propósito es que el repositorio provea a las organizaciones la opción de elegir el lenguaje al que están más familiarizados, reduciendo

la curva de aprendizaje al momento de definir los modelos de procesos de negocio colaborativos e internos.

- Implementar en el repositorio distribuido la funcionalidad que permite generar un modelo de proceso de integración a partir de un modelo de proceso colaborativo aplicando un cuestionario interactivo. El propósito es que el repositorio permita a las organizaciones generar automáticamente los modelos de procesos de integración usando el mismo.

APÉNDICE A

Meta-modelos usados en los métodos de transformación propuestos

A.1. Meta-modelo del lenguaje UP-ColBPIP

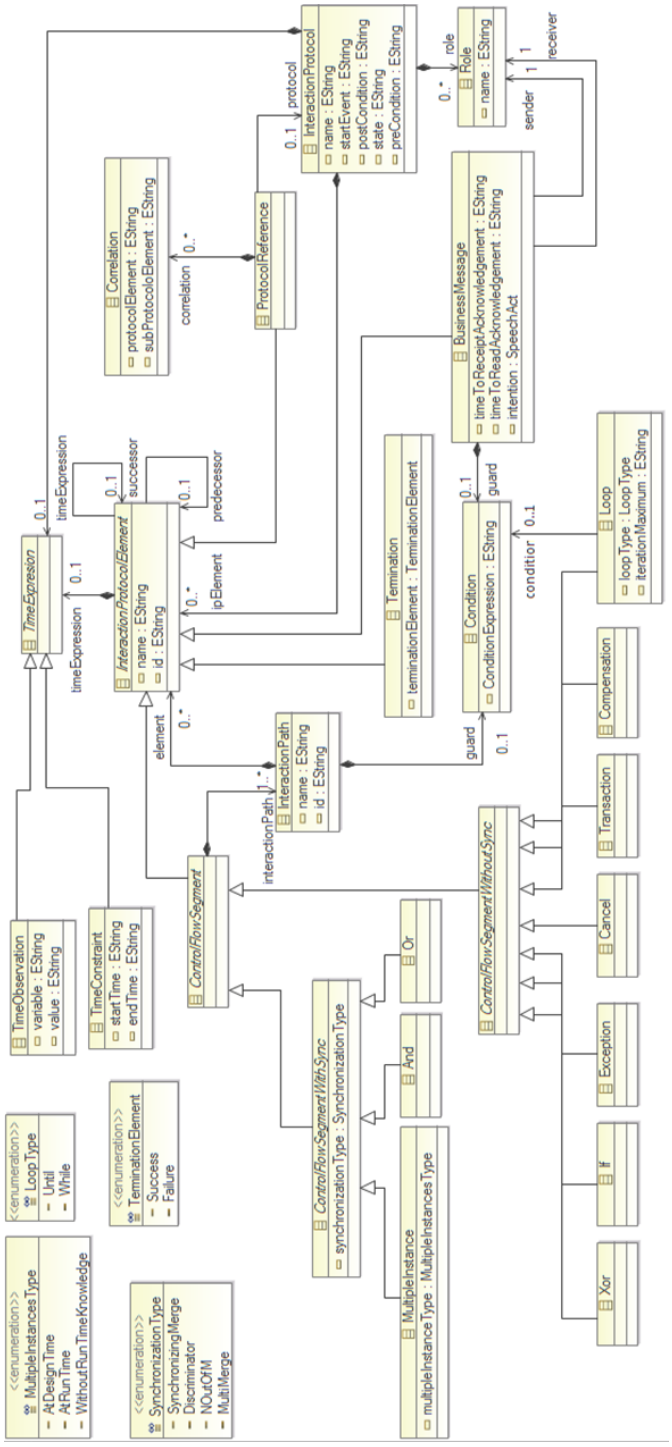


Figura A.1: Extracto del meta-modelo del lenguaje UP-ColBPIP basado en EMF.

A.2. Meta-modelo del lenguaje BPMN

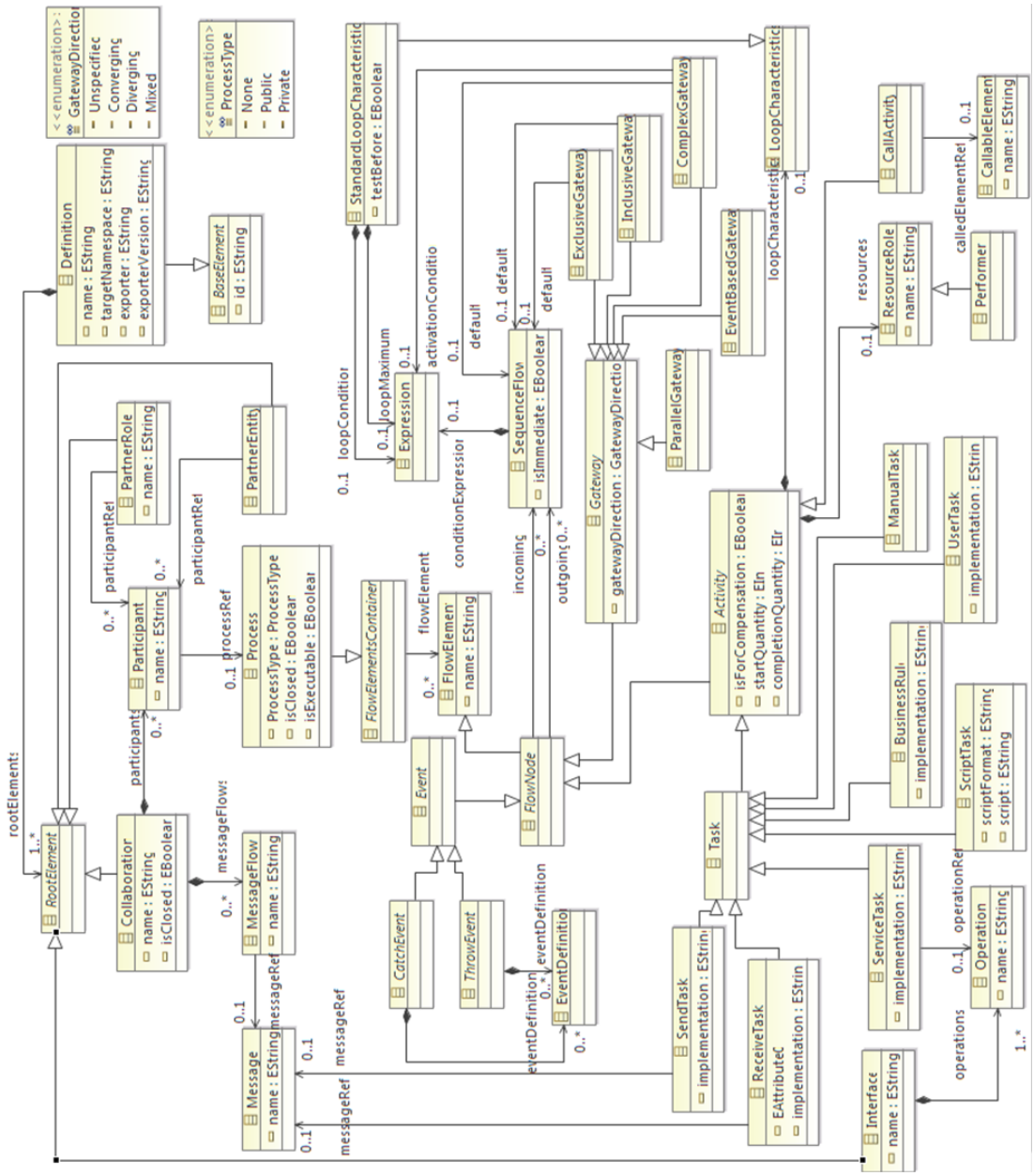


Figura A.2: Extracto del meta-modelo del lenguaje BPMN basado en EMF.

APÉNDICE B

Código de los métodos de transformación propuestos

B.1. Módulo ATL upcolbpip2bpmn

Listado B.1: Módulo upcolbpip2bpmn.atl

```
1 -- @name UP-ColBPIP to BPMN 2
2 -- @version 1.0
3 -- @domains business processes, business-to-business
4 -- @author Ivanna Lazarte
5 -- @collaborator Ramiro Savoie
6 -- @date 22/03/2012
7 -- @description  Transformation of collaborative processes into integration
      processes
8 -- @atlcompiler atl2010
9 -- @path MMupcolbpip=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/upcolbpip.
      ecore
10 -- @path MMbpmn2=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/BPMN20.ecore
11 -- @path MMParam=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/parameters.
      ecore
12
13
14 module upcolbpip2bpmn;
15 create OUT: MMbpmn2 from IN: MMupcolbpip, INParam: MMParam;
16
17 uses speechActs;
18 uses parameterHelpers;
19
20 ------------------------------------------------------------------------------
21 -- HELPERS ---------------------------------------------------------------------
22 ------------------------------------------------------------------------------
23
24 helper def: getCollaborativeProcess(): MMupcolbpip!CollaborativeBusinessProcess =
25   MMupcolbpip!CollaborativeBusinessProcess.allInstances().first();
26
27 helper def: getB2BCollaboration(): MMupcolbpip!B2BCollaboration =
```

```
28  MMupcolbpip!B2BCollaboration.allInstances().first();
29
30 helper def: getAllPartners(): Sequence(MMupcolbpip!TradingPartner) =
31  MMupcolbpip!TradingPartner.allInstances().asSequence();
32
33 helper def: getAllRoles(): Sequence(MMupcolbpip!PartnerRole) =
34  MMupcolbpip!PartnerRole.allInstances().asSequence();
35
36 helper context MMupcolbpip!TradingPartner def: performsRole(role: String): Boolean
       =
37  self.performs -> select(i | i.name = role) -> notEmpty();
38
39 helper def: getOtherPartner(role: String): MMupcolbpip!TradingPartner =
40  thisModule.getAllPartners() -> any(i | not i.performsRole(role));
41
42 helper def: getPartnerPerformingRole(role: String): MMupcolbpip!TradingPartner =
43  thisModule.getAllPartners() -> any(i | i.performsRole(role));
44
45 helper context MMupcolbpip!CollaborativeBusinessProcess def: getAllGoalstoFulfill
      ():
46    String =
47  MMupcolbpip!BusinessGoal.allInstances() -> select(e | e.superGoal = self.
48      goalToFulfill) -> collect(i | i.name) -> asSet() -> iterate(goal; acc:
            String
49      = '' | acc + if acc = '' then
50      'Goal: ' + self.goalToFulfill.name + ', Subgoal: ' + goal
51    else
52      ', Subgoal: ' + goal
53    endif);
54
55 helper def: getAllDocuments(): Sequence(MMupcolbpip!BusinessDocument) =
56  MMupcolbpip!BusinessDocument.allInstances().asSequence();
57
58 helper def: getAllMessages(): Sequence(MMupcolbpip!BusinessMessage) =
59  MMupcolbpip!BusinessMessage.allInstances().asSequence();
60
61 helper def: getExchangedDocuments(): Sequence(MMupcolbpip!BusinessDocument) =
62  thisModule.getAllMessages() -> collect(e | e.information) -> asSequence();
63
64 helper def: getAllTimeConstraints(): Sequence(MMupcolbpip!TimeConstraint) =
65  MMupcolbpip!TimeConstraint.allInstances().asSequence();
66
67 helper def: getInteractionProtocol(): MMupcolbpip!InteractionProtocol =
68  MMupcolbpip!InteractionProtocol.allInstances().first();
69
70 helper def: getProtocolName(): String =
71  MMupcolbpip!InteractionProtocol.allInstances().first().name;
72
73 helper context MMupcolbpip!BusinessMessage def: isSenderRequiredRole(role: String)
      :
74    Boolean =
```

```
75   self.sender.partnerRole.name = role;
76
77 helper context MMupcolbpip!BusinessMessage def: isReceiverRequiredRole(role:
       String):
78     Boolean =
79   self.receiver.partnerRole.name = role;
80
81 helper def: getAllIpElements(): Sequence(MMupcolbpip!InteractionProtocolElement) =
82   MMupcolbpip!InteractionProtocolElement.allInstances() -> asSequence();
83
84 helper def: getAllInterationPaths(): Sequence(MMupcolbpip!InteractionPath) =
85   MMupcolbpip!InteractionPath.allInstances().asSequence();
86
87 helper def: getAllIpElementsForProcess():
88     Sequence(MMupcolbpip!InteractionProtocolElement) =
89   let ip: Sequence(OclAny) =
90     thisModule.getAllipElementsForSubprocess()
91   in
92     thisModule.getAllIpElements() -> reject(e | ip -> includes(e)) -> flatten();
93
94 helper context MMupcolbpip!InteractionPath def: getFirstElementInInterationPath():
95     MMupcolbpip!InteractionProtocolElement =
96   if self.interactionPathBusinessMessages.oclIsUndefined() then
97     self.element -> any(e | e.predecessor = OclUndefined)
98   else
99     self.interactionPathBusinessMessages -> any(e | e.predecessor = OclUndefined)
100  endif;
101
102 helper def: getFirstElementInProcess(): MMupcolbpip!InteractionProtocolElement =
103   thisModule.getAllIpElementsForProcess() -> any(e | e.predecessor = OclUndefined)
        ;
104
105 helper context MMupcolbpip!ControlFlowSegment def: getAmountPaths(): Integer =
106   self.interactionPath.size();
107
108 helper context MMupcolbpip!ControlFlowSegment def: getAmountTerminations():
       Integer =
109   self.interactionPath -> collect(e | e.element) -> flatten() -> select(t | t.
110       oclIsTypeOf(MMupcolbpip!Termination)) -> size();
111
112 helper context MMupcolbpip!ControlFlowSegment def: getAmountPathsForSync():
       Integer =
113   self.getAmountPaths() - self.getAmountTerminations();
114
115 helper context MMupcolbpip!InteractionProtocolElement def: synchronizePaths():
       Boolean =
116   let path: MMupcolbpip!InteractionPath =
117     self.getPath()
118   in
119     if not path.oclIsUndefined() then
120       let cfs: MMupcolbpip!ControlFlowSegment =
```

```
121         self.getCFS(path)
122       in
123         if cfs.oclIsTypeOf(MMupcolbpip!CollaborativeBusinessProcess) or cfs.
124             oclIsTypeOf(MMupcolbpip!InteractionProtocol) then
125           false
126         else
127           if (cfs.getAmountPathsForSync() >= 2) then
128             true
129           else
130             false
131           endif
132         endif
133     else
134       false
135     endif;
136
137 helper context MMupcolbpip!InteractionProtocolElement def: getCFS(path:
138     MMupcolbpip!InteractionPath): MMupcolbpip!ControlFlowSegment =
139   if not path.oclIsUndefined() then
140     path.refImmediateComposite()
141   else
142     OclUndefined
143   endif;
144
145 helper context MMupcolbpip!InteractionProtocolElement def: getPath():
146     MMupcolbpip!InteractionPath =
147   if self.oclIsTypeOf(MMupcolbpip!BusinessMessage) then
148     thisModule.getAllInterationPaths() -> select(a | a.
149         interactionPathBusinessMessages -> includes(self)) -> first()
150   else
151     self.refImmediateComposite()
152   endif;
153
154 helper context MMupcolbpip!InteractionProtocolElement def: isInCFS(): Boolean =
155   thisModule.getAllInterationPaths() -> select(e | e.element -> includes(self) or
          e.
156       interactionPathBusinessMessages -> includes(self)) -> notEmpty();
157
158 helper context MMupcolbpip!InteractionProtocolElement def: isCFS(): Boolean =
159   self.oclIsKindOf(MMupcolbpip!ControlFlowSegment);
160
161 helper context MMupcolbpip!InteractionPath def: isLoop(): Boolean =
162   let cfs: MMupcolbpip!ControlFlowSegment =
163     self.refImmediateComposite()
164   in
165     cfs.oclIsTypeOf(MMupcolbpip!Loop);
166
167 helper context MMupcolbpip!InteractionPath def: getAllipElementsInPath():
168     Sequence(MMupcolbpip!InteractionProtocolElement) =
169   self.element -> union(self.interactionPathBusinessMessages);
170
```

```
171 helper context MMupcolbpip!ControlFlowSegment def: getAllipElementsInCFS():
172     Sequence(OclAny) =
173   self.interactionPath -> collect(e | e.getAllipElementsInPath()) -> flatten() ->
174       iterate(i; elements: Sequence(OclAny) = Sequence{} | if i.
175       oclIsKindOf(MMupcolbpip!ControlFlowSegment) then
176           Sequence{elements -> including(i),
177                 elements -> union(i.getAllipElementsInCFS())} -> flatten()
178         else
179           elements -> including(i)
180         endif) -> flatten() -> union(self.interactionPath);
181
182 helper def: getAllipElementsForSubprocess():
183     Sequence(MMupcolbpip!InteractionProtocolElement) =
184   MMupcolbpip!Loop.allInstances().asSequence() -> collect(e | e.
185       getAllipElementsInCFS()) -> flatten();
186
187 helper context MMupcolbpip!InteractionProtocolElement def: getSuccessor():
188     MMupcolbpip!InteractionProtocolElement =
189   if self.isInCFS() then
190     let path: MMupcolbpip!InteractionPath =
191       self.getPath()
192     in
193       let element: MMupcolbpip!InteractionProtocolElement =
194         self.getCFS(path)
195       in
196         let elementSucesor: MMupcolbpip!InteractionProtocolElement = element.
                successor
197         in
198         if (elementSucesor.oclIsKindOf(MMupcolbpip!InteractionProtocolElement))
                then
199         elementSucesor
200       else
201         OclUndefined
202       endif
203   else
204     let elementContainer: MMupcolbpip!InteractionProtocolElement =
205       self.refImmediateComposite()
206     in
207       if elementContainer.oclIsKindOf(MMupcolbpip!InteractionProtocolElement) then
208         elementContainer.successor
209       else
210         OclUndefined
211       endif
212   endif;
213
214 helper context MMupcolbpip!InteractionProtocolElement def: getSiblingSuccessor():
215     MMupcolbpip!InteractionProtocolElement =
216   let path: MMupcolbpip!InteractionPath =
217     self.getPath()
218   in
219     if not path.oclIsUndefined() then
```

```
220       let cfs: MMupcolbpip!ControlFlowSegment =
221         self.getCFS(path)
222       in
223         let path2: MMupcolbpip!InteractionPath =
224           cfs.getPath()
225         in
226           if not path2.oclIsUndefined() then
227             let cfs2: MMupcolbpip!ControlFlowSegment =
228               self.getCFS(path2)
229             in
230               if not cfs2.
231                   oclIsTypeOf(MMupcolbpip!CollaborativeBusinessProcess)
232                   then
233                 cfs2.interactionPath.excluding(path2).first().
234                     getAllipElementsInPath() -> any(e | e.successor.
235                     oclIsUndefined())
236               else
237                 OclUndefined
238               endif
239           else
240             OclUndefined
241           endif
242     else
243       OclUndefined
244     endif;
245
246 helper context MMupcolbpip!InteractionPath def: getPathName(): String =
247   if self.guard = OclUndefined then
248     self.interactionPathBusinessMessages.first().intention.toString()
249   else
250     self.guard.ConditionExpression
251   endif;
252
253 helper context MMupcolbpip!InteractionProtocolElement def: hasSuccessor(): Boolean
        =
254   not self.successor.oclIsUndefined();
255
256 helper context MMupcolbpip!ControlFlowSegmentWithoutSync def: isXorData(): Boolean
        =
257   let allGuard: Sequence(MMupcolbpip!Condition) =
258     self.interactionPath -> select(e | e.guard <> OclUndefined) -> asSequence()
259   in
260     allGuard.notEmpty();
261
262 helper context MMupcolbpip!ControlFlowSegmentWithoutSync def: isXorReceiverRole():
263     Boolean =
264   self.interactionPath -> collect(i | i.getAllMessagesReceivedByRole(thisModule.
265       getParameterValue('roleId', 'role'))) -> flatten() -> notEmpty();
266
267 helper context MMupcolbpip!InteractionPath def: getAllMessagesReceivedByRole(role:
268     String): Sequence(MMupcolbpip!BusinessMessage) =
```

```
269   self.interactionPathBusinessMessages -> select(e | e.isReceiverRequiredRole(role
          )) ->
270       asSequence();
271
272 helper context MMupcolbpip!Loop def: isLoopUntil(): Boolean =
273   if self.loopType = #Until then
274           true
275         else
276           false
277         endif;
278
279 helper context MMupcolbpip!InteractionProtocolElement def: hasTimeConstraint():
        Boolean =
280   self.timeExpression <> OclUndefined;
281
282 helper context MMupcolbpip!InteractionProtocolElement def: getTimeConstraint():
        String =
283   MMupcolbpip!TimeConstraint.allInstances() -> any(e | e.refImmediateComposite() =
284       self).endTime;
285
286 helper context MMupcolbpip!InteractionProtocolElement def: hasParentTimeConstraint
        (): Boolean =
287   if self.isInCFS() then
288     let parent: MMupcolbpip!InteractionProtocolElement = self.getPath().
            refImmediateComposite() in
289     parent.hasTimeConstraint()
290   else
291     false
292   endif;
293
294 helper context MMupcolbpip!TimeConstraint def: attachToBoundary(): Boolean =
295   let element: MMupcolbpip!InteractionProtocolElement = self.refImmediateComposite
          () in
296   if element.oclIsKindOf(MMupcolbpip!ControlFlowSegment) then
297     true
298   else
299     if element.oclIsTypeOf(MMupcolbpip!BusinessMessage) then
300       if element.isReceiverRequiredRole(thisModule.getParameterValue('roleId', '
              role')) then
301         true
302       else
303         false
304       endif
305     else
306       false
307     endif
308   endif;
309
310 ------------------------------------------------------------------------
311 ------------------------------ RULES -----------------------------------
312 ------------------------------------------------------------------------
```

```
313
314 ------------------------------------------------------------------------
315 --------------------- UPColBPIPModel to Definitions ------------------
316 ------------------------------------------------------------------------
317 rule UPColBPIPModel2Definitions {
318   from
319     upcolbpipModel: MMupcolbpip!UPColBPIPModel
320   to
321     definitions: MMbpmn2!Definitions (
322       id <- 'definitionsId',
323       targetNamespace <- 'http://www.cidisi.frsf.utn.edu.ar/upcolpip2bpmn2',
324       name <- upcolbpipModel.name,
325       exporter <- 'Eclipse',
326       exporterVersion <- '3.6',
327       rootElements <- Sequence{thisModule.getCollaborativeProcess(),
328             thisModule.getB2BCollaboration(),
329             thisModule.getAllPartners(),
330             thisModule.getAllRoles(),
331             thisModule.getAllPartners() -> collect(i | thisModule.
332                 resolveTemp(i, 'partnerEntity')),
333             thisModule.getAllDocuments()}
334     )
335 }
336
337 ------------------------------------------------------------------------
338 -------------------- B2BCollaboration to Collaboration ------------------
339 ------------------------------------------------------------------------
340 rule b2bCollaboration2Collaboration {
341   from
342     b2bCollaboration: MMupcolbpip!B2BCollaboration
343   to
344     collaboration: MMbpmn2!Collaboration (
345       id <- 'collaborationId',
346       isClosed <- true,
347       name <- b2bCollaboration.name,
348       participants <- b2bCollaboration.partners,
349       messageFlows <- thisModule.getAllMessages() -> collect(e | thisModule.
350           messageFlow(e))
351     )
352 }
353
354 ------------------------------------------------------------------------
355 ------------ CollaborativeBusinessProcess to Process -------------------
356 ------------------------------------------------------------------------
357 rule collaborativeProcess2Process {
358   from
359     collaborativeProcess: MMupcolbpip!CollaborativeBusinessProcess
360   to
361     process: MMbpmn2!Process (
362       id <- 'processId',
363       isExecutable <- false,
```

```
364       processType <- thisModule.getParameterValue('processId', 'processType'),
365       name <- collaborativeProcess.name,
366       artifacts <- Sequence{businessGoals,
367             association},
368       flowElements <- Sequence{startEvent,
369             startSequenceFlow,
370             thisModule.getAllIpElements(),
371             thisModule.getAllInterationPaths(),
372             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
373                 resolveTemp(i, 'Task')),
374             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
375                 resolveTemp(i, 'internalSequenceFlow')),
376             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
377                 resolveTemp(i, 'sendTask')),
378             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
379                 resolveTemp(i, 'receiveTask')),
380             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
381                 resolveTemp(i, 'terminateEvent')),
382             thisModule.getAllIpElementsForProcess() -> collect(i | thisModule.
383                 resolveTemp(i, 'timerSequenceFlow')),
384             thisModule.getAllTimeConstraints(),
385             thisModule.getAllTimeConstraints() -> collect(i | thisModule.
386                 resolveTemp(i, 'timerSequenceFlow')),
387             thisModule.getAllTimeConstraints() -> collect(i | thisModule.
388                 resolveTemp(i, 'terminateEvent')),
389             thisModule.getAllIpElements() -> collect(i | thisModule.
390                 resolveTemp(i, 'operation'))
391             }
392     ),
393     startEvent: MMbpmn2!StartEvent (
394       id <- 'startEvent0',
395       name <- 'Start',
396       outgoing <- startSequenceFlow
397     ),
398     startSequenceFlow: MMbpmn2!SequenceFlow (
399       id <- 'sequenceFlow0',
400       isImmediate <- true,
401       sourceRef <- startEvent,
402       targetRef <- thisModule.getFirstElementInProcess()
403     ),
404     businessGoals: MMbpmn2!TextAnnotation (
405       id <- 'businessGoal0',
406       text <- collaborativeProcess.getAllGoalstoFulfill()
407     ),
408     association: MMbpmn2!Association (
409       id <- 'textAssociation0',
410       associationDirection <- 'None',
411       sourceRef <- startEvent,
412       targetRef <- businessGoals
413     )
414 }
```

```
415
416 ------------------------------------------------------------------------
417 --------------- TradingPartner to Participant -------------------------
418 ------------------------------------------------------------------------
419 rule tradingPartner2Participant {
420   from
421     tradingPartner: MMupcolbpip!TradingPartner
422   to
423     participant: MMbpmn2!Participant (
424       id <- 'participant' + thisModule.getAllPartners().indexOf(tradingPartner).
425           toString(),
426       name <- tradingPartner.name,
427       processRef <- if tradingPartner.performsRole(thisModule.
428           getParameterValue('roleId', 'role')) then
429           thisModule.getCollaborativeProcess()
430         else
431           OclUndefined
432         endif
433     ),
434     partnerEntity: MMbpmn2!PartnerEntity (
435       id <- 'partnerEntity' + thisModule.getAllPartners().indexOf(tradingPartner).
436           toString(),
437       name <- tradingPartner.name,
438       participantRef <- participant
439     )
440 }
441
442 ------------------------------------------------------------------------
443 ------------ PartnerRole to PartnerRole ------------------------------
444 ------------------------------------------------------------------------
445 rule partnerRole2PartnerRole {
446   from
447     role: MMupcolbpip!PartnerRole
448   to
449     partnerRole: MMbpmn2!PartnerRole (
450       id <- 'partnerRole' + thisModule.getAllRoles().indexOf(role).toString(),
451       name <- role.name,
452       participantRef <- thisModule.getPartnerPerformingRole(role.name)
453     )
454 }
455
456 ------------------------------------------------------------------------
457 ----- abstract rule used for transforming the cfs ----------------------
458 ------------------------------------------------------------------------
459 abstract rule cfs2Gateway {
460   from
461     cfs: MMupcolbpip!ControlFlowSegment
462   to
463     gateway: MMbpmn2!Gateway (
464       id <- cfs.id,
465       name <- cfs.name,
```

```
466       gatewayDirection <- #Diverging,
467       outgoing <- cfs.interactionPath -> collect(i | thisModule.resolveTemp(i,
468           'sequenceFlows'))
469     )
470 }
471
472 ------------------------------------------------------------
473 -------- Data-based Xor to ExclusiveGateway ----------------
474 ------------------------------------------------------------
475 rule cfsXorData2ExclusiveGateway extends cfs2Gateway {
476   from
477     cfs: MMupcolbpip!Xor (
478       not cfs.isXorReceiverRole()
479     )
480   to
481     gateway: MMbpmn2!ExclusiveGateway
482 }
483
484
485 ------------------------------------------------------------
486 -------- Data-based Xor to ExclusiveGateway ----------------
487 -------- with time constraint ------------------------------
488 ------------------------------------------------------------
489 rule cfsXorData2ExclusiveGatewayWithTimeConstraint extends
       cfsXorData2ExclusiveGateway {
490   from
491     cfs: MMupcolbpip!Xor (
492       cfs.hasTimeConstraint()
493     )
494   to
495     subProcess: MMbpmn2!SubProcess (
496       id <- 'subProcess_' + cfs.id,
497       triggeredByEvent <- false,
498       completionQuantity <- 1,
499       isForCompensation <- false,
500       startQuantity <- 1,
501       outgoing <- thisModule.successorSequenceFlowForProcess(cfs),
502       flowElements <- Sequence{startEvent,
503             startSequenceFlow,
504             cfs.getAllipElementsInCFS(),
505             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
506                 resolveTemp(i, 'Task')),
507             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
508                 resolveTemp(i, 'internalSequenceFlow')),
509             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
510                 resolveTemp(i, 'receiveTask')),
511             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
512                 resolveTemp(i, 'sendTask')),
513             gateway}
514     ),
515
```

```
516     startEvent: MMbpmn2!StartEvent (
517       id <- 'startEvent_' + cfs.id,
518       name <- 'Start',
519       outgoing <- startSequenceFlow
520     ),
521     startSequenceFlow: MMbpmn2!SequenceFlow (
522       id <- 'sequenceFlow_' + cfs.id,
523       isImmediate <- true,
524       sourceRef <- startEvent,
525       targetRef <- gateway
526     )
527
528 }
529
530 ------------------------------------------------------------
531 -------- Event-based Xor to EventBasedGateway -------------
532 ------------------------------------------------------------
533 rule cfsXorEvent2EventBasedGateway extends cfs2Gateway {
534   from
535     cfs: MMupcolbpip!Xor (
536        cfs.isXorReceiverRole()
537     )
538   to
539     gateway: MMbpmn2!EventBasedGateway (
540       eventGatewayType <- 'Exclusive'
541     )
542 }
543
544 ------------------------------------------------------------
545 -------- Event-based Xor to EventBasedGateway ---------------
546 -------- with time constraint --------------------------
547 ------------------------------------------------------------
548 rule cfsXorEvent2EventBasedGatewayWithTimeConstraint extends
       cfsXorEvent2EventBasedGateway {
549   from
550     cfs: MMupcolbpip!Xor (
551       cfs.hasTimeConstraint()
552     )
553   to
554     subProcess: MMbpmn2!SubProcess (
555       id <- 'subProcess_' + cfs.id,
556       triggeredByEvent <- false,
557       completionQuantity <- 1,
558       isForCompensation <- false,
559       startQuantity <- 1,
560       outgoing <- thisModule.successorSequenceFlowForProcess(cfs),
561       flowElements <- Sequence{startEvent,
562             startSequenceFlow,
563             cfs.getAllipElementsInCFS(),
564             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
565                 resolveTemp(i, 'Task')),
```

```
566             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
567                 resolveTemp(i, 'internalSequenceFlow')),
568             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
569                 resolveTemp(i, 'receiveTask')),
570             cfs.getAllipElementsInCFS() -> collect(i | thisModule.
571                 resolveTemp(i, 'sendTask')),
572             gateway}
573     ),
574
575     startEvent: MMbpmn2!StartEvent (
576       id <- 'startEvent_' + cfs.id,
577       name <- 'Start',
578       outgoing <- startSequenceFlow
579     ),
580     startSequenceFlow: MMbpmn2!SequenceFlow (
581       id <- 'sequenceFlow_' + cfs.id,
582       isImmediate <- true,
583       sourceRef <- startEvent,
584       targetRef <- gateway
585     )
586 }
587
588 ------------------------------------------------------------
589 -------- Data-based Xor to ExclusiveGateway ----------------
590 -------- with default path ---------------------------------
591 ------------------------------------------------------------
592 rule cfsXorData2ExclusiveGateway_defaultPath extends cfsXorData2ExclusiveGateway {
593   from
594     cfs: MMupcolbpip!Xor (
595      thisModule.hasParameter(cfs.id, 'defaultPath')
596     )
597   to
598     gateway: MMbpmn2!ExclusiveGateway
599     (
600       default <- cfs.interactionPath -> any (e | e.name = thisModule.
              getParameterValue(cfs.id, 'defaultPath'))
601     )
602 }
603
604
605 ------------------------------------------------------------
606 --------------- And to ParallelGateway ---------------------
607 ------------------------------------------------------------
608 rule cfsAnd2ParallelGateway extends cfs2Gateway {
609   from
610     cfs: MMupcolbpip!And
611   to
612     gateway: MMbpmn2!ParallelGateway
613 }
614
615 ------------------------------------------------------------
```

```
616 -------------- Or to InclusiveGateway ---------
617 -----------------------------------------------------------
618 rule cfsOr2InclusiveGateway extends cfs2Gateway{
619   from
620     cfs: MMupcolbpip!Or
621   to
622     gateway: MMbpmn2!InclusiveGateway
623 }
624
625 -----------------------------------------------------------
626 -------- Or to InclusiveGateway ---------------
627 -------- with default path --------------------------------
628 -----------------------------------------------------------
629 rule cfsOr2InclusiveGateway_defaultPath extends cfsOr2InclusiveGateway {
630   from
631     cfs: MMupcolbpip!Or (
632      thisModule.hasParameter(cfs.id, 'defaultPath')
633     )
634   to
635     gateway: MMbpmn2!InclusiveGateway
636     (
637       default <- cfs.interactionPath -> any (e | e.name = thisModule.
              getParameterValue(cfs.id, 'defaultPath'))
638     )
639 }
640
641 ------------------------------------------------------------------------
642 ---------------------------- Loop to SubProcess ---------------------
643 ------------------------------------------------------------------------
644 rule cfsLoop2SubProcess {
645   from
646     loop: MMupcolbpip!Loop
647   to
648     subProcess: MMbpmn2!SubProcess (
649       id <- loop.id,
650       --name <- loop.name,
651       triggeredByEvent <- false,
652       completionQuantity <- 1,
653       isForCompensation <- false,
654       startQuantity <- 1,
655       outgoing <- thisModule.successorSequenceFlowForProcess(loop),
656       flowElements <- Sequence{startEvent,
657             startSequenceFlow,
658             loop.getAllipElementsInCFS(),
659             loop.getAllipElementsInCFS() -> collect(i | thisModule.
660                 resolveTemp(i, 'Task')),
661             loop.getAllipElementsInCFS() -> collect(i | thisModule.
662                 resolveTemp(i, 'internalSequenceFlow')),
663             loop.getAllipElementsInCFS() -> collect(i | thisModule.
664                 resolveTemp(i, 'receiveTask')),
665             loop.getAllipElementsInCFS() -> collect(i | thisModule.
```

```
666                 resolveTemp(i, 'sendTask'))},
667       loopCharacteristics <- loopCharacteristics
668     ),
669     loopCharacteristics: MMbpmn2!StandardLoopCharacteristics (
670       id <- 'loopCharacteristics_' + loop.id,
671       --loopMaximum <- loopMaximum,
672       --loopCondition <- loopCondition,
673       testBefore <- loop.isLoopUntil()
674     ),
675       loopMaximum: MMbpmn2!Expression (
676       id <- 'loopMaximum' + loop.id
677     ),
678     loopCondition: MMbpmn2!Expression (
679       id <- 'NegotiationRequired'
680     ),
681     startEvent: MMbpmn2!StartEvent (
682       id <- 'startEvent_' + loop.id,
683       name <- 'Start',
684       outgoing <- startSequenceFlow
685     ),
686     startSequenceFlow: MMbpmn2!SequenceFlow (
687       id <- 'sequenceFlow_' + loop.id,
688       isImmediate <- true,
689       sourceRef <- startEvent,
690       targetRef <- loop.getAllipElementsInCFS().at(1)
691     )
692 }
693
694 ------------------------------------------------------------------------
695 ------- protocolReference to CallActivity ------------------------------
696 ------------------------------------------------------------------------
697 rule protocolReference2CallActivity {
698   from
699     protocolRef: MMupcolbpip!ProtocolReference
700   to
701     callActivity: MMbpmn2!CallActivity (
702       id <- protocolRef.id,
703       name <- protocolRef.name,
704       completionQuantity <- 1,
705       isForCompensation <- false,
706       startQuantity <- 1,
707       calledElementRef <- protocolRef.protocol,
708       outgoing <- if thisModule.getAllipElementsForSubprocess() ->
709               includes(protocolRef) then
710                 thisModule.successorSequenceFlowForSubprocess(protocolRef)
711             else
712               thisModule.successorSequenceFlowForProcess(protocolRef)
713             endif
714     )
715 }
716
```

```
717 ------------------------------------------------------------------------
718 ---------- interactionPath to sequenceFlow -----------------------------
719 ------------------------------------------------------------------------
720 rule interactionPath2sequenceFlow {
721   from
722     interactionPath: MMupcolbpip!InteractionPath (
723       not interactionPath.isLoop()
724     )
725   to
726     sequenceFlows: MMbpmn2!SequenceFlow (
727       id <- interactionPath.name,
728       isImmediate <- true,
729       name <- interactionPath.getPathName(),
730       targetRef <- interactionPath.getFirstElementInInterationPath(),
731       conditionExpression <- interactionPath.guard
732     )
733 }
734
735 ------------------------------------------------------------------------
736 ------------------- Success Explicit Termination to End Event ----------
737 ------------------------------------------------------------------------
738 rule successExplicitTermination2EndEvent {
739   from
740     termination: MMupcolbpip!Termination
741     (termination.terminationElement= #Success)
742   to
743     endEvent: MMbpmn2!EndEvent (
744       id <- termination.id,
745       name <- termination.terminationElement.toString()
746     )
747 }
748
749 ------------------------------------------------------------------------
750 ---------- Failure Explicit Termination to End Event -------------------
751 ------------------------------------------------------------------------
752 rule failureExplicitTermination2EndEvent {
753   from
754     termination: MMupcolbpip!Termination
755     (termination.terminationElement= #Failure)
756   to
757     endEvent: MMbpmn2!EndEvent (
758       id <- termination.id,
759       name <- termination.terminationElement.toString(),
760       eventDefinitions <- eventDefinitionTerminate
761     ),
762
763     eventDefinitionTerminate: MMbpmn2!TerminateEventDefinition (
764     id <- 'eventDefinitionTerminate_' +  termination.id
765     )
766 }
767
```

```
768 ------------------------------------------------------------------------------
769 ---------------- Business Documento to Message ------------------------------
770 ------------------------------------------------------------------------------
771 rule businessDocument2Message {
772   from
773     businessDocument: MMupcolbpip!BusinessDocument (
774       thisModule.getExchangedDocuments() -> includes(businessDocument)
775     )
776   to
777     message: MMbpmn2!Message (
778       id <- businessDocument.name,
779       name <- businessDocument.name
780     )
781 }
782
783 ------------------------------------------------------------------------------
784 ---------------- Time Constraint to Timer ------------------------------------
785 ------------------------------------------------------------------------------
786 rule timeConstraint2Timer {
787   from
788     timeConstraint: MMupcolbpip!TimeConstraint
789     (
790       timeConstraint.attachToBoundary()
791       )
792   using {
793       element: MMupcolbpip!InteractionProtocolElement = timeConstraint.
              refImmediateComposite();
794     }
795   to
796     timer: MMbpmn2!BoundaryEvent (
797     id <-   'timer_' + element.id,
798     cancelActivity <- true,
799     name <- element.getTimeConstraint(),
800     attachedToRef <- element,
801     eventDefinitions <- eventDefinitionTimer
802     ),
803
804     eventDefinitionTimer: MMbpmn2!TimerEventDefinition (
805     id <- 'eventDefinitionTimer_' + element.id
806     --timeDate <- businessMessage.getTimeConstraint()
807     ),
808
809     timerSequenceFlow: MMbpmn2!SequenceFlow (
810       id <- 'internalSequenceFlow_' + element.id,
811       isImmediate <- true,
812       sourceRef <- timer,
813       targetRef <- terminateEvent
814     ),
815
816     terminateEvent: MMbpmn2!EndEvent (
817     id <- 'terminateEvent_' +  element.id,
```

```
818     eventDefinitions <- eventDefinitionTerminate
819     ),
820
821     eventDefinitionTerminate: MMbpmn2!TerminateEventDefinition (
822     id <- 'eventDefinitionTerminate_' +  element.id
823     )
824 }
825
826 ------------------------------------------------------------------------
827 -------------- Condition to Expression ---------------------------------
828 -------------- without parameters --------------------------------------
829 ------------------------------------------------------------------------
830 rule condition2Expression {
831   from
832     guard: MMupcolbpip!Condition
833   to
834     expression: MMbpmn2!Expression (
835       id <- guard.ConditionExpression
836     )
837 }
838
839 ------------------------------------------------------------------------
840 -------------- Condition to Expression ---------------------------------
841 -------------- with parameters -----------------------------------------
842 ------------------------------------------------------------------------
843 rule condition2Expression_FormalExpression extends  condition2Expression {
844   from
845     guard: MMupcolbpip!Condition
846     (thisModule.hasParameter(guard.refImmediateComposite().refGetValue('name'), '
          expression'))
847   to
848     expression: MMbpmn2!FormalExpression (
849       id <- guard.ConditionExpression,
850       language <- thisModule.getParameterValue(guard.refImmediateComposite().
              refGetValue('name'),'language'),
851       body <- thisModule.getParameterValue(guard.refImmediateComposite().
              refGetValue('name'), 'expresssion')
852     )
853 }
854
855 ------------------------------------------------------------------------
856 --------------------------- LAZY RULES ---------------------------------
857 ------------------------------------------------------------------------
858 lazy rule messageFlow {
859   from
860     businessMessage: MMupcolbpip!BusinessMessage
861   to
862     messageFlow: MMbpmn2!MessageFlow (
863       id <- 'messageFlow' + thisModule.getAllMessages().indexOf(businessMessage).
              toString(),
864       name <- businessMessage.information.name,
865
```

```
866       sourceRef <- if businessMessage.isSenderRequiredRole(thisModule.
867           getParameterValue('roleId', 'role')) then
868           thisModule.resolveTemp(businessMessage, 'sendTask')
869         else
870           thisModule.getOtherPartner(thisModule.getParameterValue('roleId',
871               'role'))
872         endif,
873       targetRef <- if businessMessage.isSenderRequiredRole(thisModule.
874           getParameterValue('roleId', 'role')) then
875           thisModule.getOtherPartner(thisModule.getParameterValue('roleId',
876               'role'))
877         else
878           businessMessage
879         endif,
880       messageRef <- businessMessage.information
881     )
882 }
883
884 -------------------------------------------------------------------------
885 ------- Abstract rule for -----------------------------------------------
886 ------- Synchronize interaction paths -----------------------------------
887 -------------------------------------------------------------------------
888 unique lazy abstract rule syncCFS
889 {
890   from
891     cfs: MMupcolbpip!ControlFlowSegment
892
893   to
894     gateway: MMbpmn2!Gateway (
895       id <- 'converging' + gateway.oclType().refGetValue('name'),
896       gatewayDirection <- #Converging,
897       outgoing <- if thisModule.getAllipElementsForSubprocess() -> includes(cfs)
898           then
899           thisModule.successorSequenceFlowForSubprocess(cfs)
900         else
901           thisModule.successorSequenceFlowForProcess(cfs)
902         endif
903     )
904 }
905
906 -------------------------------------------------------------------------
907 ------- Abstract rule for -----------------------------------------------
908 ------- Synchronize CFS AND ---------------------------------------------
909 -------------------------------------------------------------------------
910 lazy rule syncAnd extends syncCFS
911 {
912   from
913     cfs: MMupcolbpip!And
914   using {
915     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
916   }
```

```
917   to
918     gateway: MMbpmn2!ParallelGateway
919   do
920   {
921     process.flowElements <- process.flowElements.append(gateway);
922
923   }
924 }
925
926 ------------------------------------------------------------------------
927 ------- Abstract rule for ------------------------------------------------
928 ------- Synchronize CFS XOR ----------------------------------------------
929 ------------------------------------------------------------------------
930 lazy rule syncXor extends syncCFS
931 {
932   from
933     cfs: MMupcolbpip!Xor
934   using {
935     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
936   }
937   to
938     gateway: MMbpmn2!ExclusiveGateway
939   do
940   {
941     process.flowElements <- process.flowElements.append(gateway);
942   }
943 }
944
945 ------------------------------------------------------------------------
946 ------- Abstract rule for ------------------------------------------------
947 ------- Synchronize CFS OR Merge -----------------------------------------
948 ------------------------------------------------------------------------
949 lazy rule syncOrSyncMerge extends syncCFS
950 {
951   from
952     cfs: MMupcolbpip!Or (
953       cfs.synchronizationType = #"<<Sync-Merge>>"
954     )
955   using {
956     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
957   }
958   to
959     gateway: MMbpmn2!InclusiveGateway
960   do
961   {
962     process.flowElements <- process.flowElements.append(gateway);
963   }
964 }
965
966 ------------------------------------------------------------------------
967 ------- Abstract rule for ------------------------------------------------
```

```
968 ------- Synchronize CFS OR MultiMerge ------------------------------------
969 -----------------------------------------------------------------------------
970 lazy rule syncOrMultiMerge extends syncCFS
971 {
972   from
973     cfs: MMupcolbpip!Or (
974       cfs.synchronizationType = #"<<Multi-Merge>>"
975     )
976   using {
977     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
978   }
979   to
980     gateway: MMbpmn2!ExclusiveGateway
981   do
982   {
983     process.flowElements <- process.flowElements.append(gateway);
984   }
985 }
986
987 -----------------------------------------------------------------------------
988 ------- Abstract rule for -------------------------------------------------
989 ------- Synchronize CFS OR NoutofM ----------------------------------------
990 -----------------------------------------------------------------------------
991 lazy rule syncOrNoutofM extends syncCFS
992 {
993   from
994     cfs: MMupcolbpip!Or (
995       cfs.synchronizationType = #"<<N-out-of-M>>"
996     )
997   using {
998     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
999   }
1000  to
1001    gateway: MMbpmn2!ComplexGateway (
1002      activationCondition <- cfs.synchronizationType.toString()
1003    )
1004  do
1005  {
1006    process.flowElements <- process.flowElements.append(gateway);
1007  }
1008 }
1009
1010 -----------------------------------------------------------------------------
1011 ------- Lazy rule for -----------------------------------------------------
1012 ------- generate successor SequenceFlow -----------------------------------
1013 -----------------------------------------------------------------------------
1014 lazy rule successorSequenceFlowForProcess {
1015   from
1016     element: MMupcolbpip!InteractionProtocolElement
1017   using {
1018     process: MMbpmn2!Process = MMbpmn2!Process.allInstances().last();
```

```
1019  }
1020  to
1021    SequenceFlow: MMbpmn2!SequenceFlow (
1022      id <- 'successorSequenceFlow_' + element.id,
1023      isImmediate <- true,
1024      targetRef <- if element.hasSuccessor() then
1025            element.successor
1026          else
1027            let ipSuccessor: MMupcolbpip!InteractionProtocolElement =
1028            element.getSuccessor()
1029            in
1030            let path: MMupcolbpip!InteractionPath =
1031              element.getPath()
1032            in
1033            let cfs: MMupcolbpip!ControlFlowSegment =
1034                element.getCFS(path)
1035              in
1036            if element.synchronizePaths() then
1037              if ipSuccessor.oclType() = cfs.oclType() then
1038                ipSuccessor
1039              else
1040                thisModule.syncCFS(cfs)
1041              endif
1042            else
1043              if ipSuccessor.oclIsUndefined() then
1044                if element.hasParentTimeConstraint() or thisModule.
                       getAllipElementsForSubprocess() ->
1045                        includes(element) then
1046                  thisModule.implicitTermination2EndEventForSubprocess()
1047                else
1048                    thisModule.implicitTermination2EndEventForProcess()
1049                endif
1050              else
1051                if element.hasParentTimeConstraint() then
1052                  thisModule.implicitTermination2EndEventForSubprocess()
1053                else
1054                  ipSuccessor
1055                endif
1056              endif
1057            endif
1058          endif
1059    )
1060  do
1061  {
1062    process.flowElements <- process.flowElements.append(SequenceFlow);
1063
1064  }
1065 }
1066
1067 lazy rule successorSequenceFlowForSubprocess {
1068  from
```

```
1069    element: MMupcolbpip!InteractionProtocolElement
1070  using {
1071    subProcess: MMbpmn2!SubProcess = MMbpmn2!SubProcess.allInstances().last();
1072  }
1073  to
1074    sequenceFlow: MMbpmn2!SequenceFlow (
1075      id <- 'successorSequenceFlow_' + element.id,
1076      isImmediate <- true,
1077      targetRef <- if element.hasSuccessor() then
1078              element.successor
1079            else
1080              let ipSuccessor: MMupcolbpip!InteractionProtocolElement =
1081                element.getSuccessor()
1082              in
1083              let path: MMupcolbpip!InteractionPath =
1084                element.getPath()
1085              in
1086              let cfs: MMupcolbpip!ControlFlowSegment =
1087                element.getCFS(path)
1088              in
1089              if element.synchronizePaths() then
1090                if ipSuccessor.oclType() = cfs.oclType() then
1091                  ipSuccessor
1092                else
1093                  thisModule.syncCFS(cfs)
1094                endif
1095              else
1096                if ipSuccessor.oclIsUndefined() then
1097                  if element.hasParentTimeConstraint() or thisModule.
                          getAllipElementsForSubprocess() ->
1098                        includes(element) then
1099                      thisModule.implicitTermination2EndEventForSubprocess()
1100                  else
1101                      thisModule.implicitTermination2EndEventForProcess()
1102                  endif
1103                else
1104                  if element.hasParentTimeConstraint() then
1105                    thisModule.implicitTermination2EndEventForSubprocess()
1106                  else
1107                    ipSuccessor
1108                  endif
1109                endif
1110              endif
1111            endif
1112    )
1113  do
1114  {
1115    subProcess.flowElements <- subProcess.flowElements.append(sequenceFlow);
1116  }
1117 }
1118
```

```
1119 ------------------------------------------------------------------------
1120 ------- Lazy rule for ------------------------------------------------
1121 ------- generate implicit termination within process ------------------
1122 ------------------------------------------------------------------------
1123 lazy rule implicitTermination2EndEventForProcess {
1124   from
1125     o: OclAny
1126   using {
1127     process: MMbpmn2!Process = MMbpmn2!Process.allInstances() -> last();
1128   }
1129   to
1130     implicitEndEvent: MMbpmn2!EndEvent (
1131       id <- 'implicitEndEventProcess',
1132       name <- 'Success'
1133     )
1134   do
1135   {
1136       process.flowElements <- process.flowElements.append(implicitEndEvent);
1137
1138   }
1139 }
1140
1141 ------------------------------------------------------------------------
1142 ------- Lazy rule for ------------------------------------------------
1143 ------- generate implicit termination within subprocess ---------------
1144 ------------------------------------------------------------------------
1145 lazy rule implicitTermination2EndEventForSubprocess {
1146   from
1147     o: OclAny
1148   using {
1149     subProcess: MMbpmn2!SubProcess = MMbpmn2!SubProcess.allInstances() -> last();
1150   }
1151   to
1152     implicitEndEvent: MMbpmn2!EndEvent (
1153       id <- 'implicitEndEventSubProcess',
1154       name <- 'Success'
1155     )
1156   do
1157   {
1158       subProcess.flowElements <- subProcess.flowElements.
1159           append(implicitEndEvent);
1160   }
1161 }
```

B.2. Módulo ATL protocol2interface

Listado B.2: Módulo protocol2interface.atl

```
1 -- @name Interaction Protocol to Interface Process
2 -- @version 1.0
3 -- @domains business processes, business-to-business
4 -- @author Ivanna Lazarte
5 -- @collaborator Ramiro Savoie
6 -- @date 22/03/2012
7 -- @description  Transforming collaborative processes into interface processes
8 -- @atlcompiler atl2010
9 -- @path MMupcolbpip=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/upcolbpip.
      ecore
10 -- @path MMbpmn2=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/BPMN20.ecore
11 -- @path MMParam=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/parameters.
      ecore
12
13
14 module protocol2interface;
15 create OUT: MMbpmn2 from IN: MMupcolbpip, INParam: MMParam;
16
17 uses speechActs;
18 uses parameterHelpers;
19
20 ------------------------------------------------------------------------
21 ----------- Transforming business messages sent --------------------
22 ------------------------------------------------------------------------
23 rule sendBusinessMessage2SendTask {
24   from
25     businessMessage: MMupcolbpip!BusinessMessage (
26       businessMessage.isSenderRequiredRole(thisModule.getParameterValue('roleId',
27           'role'))
28     )
29   to
30     sendTask: MMbpmn2!SendTask (
31       id <- businessMessage.id,
32       name <- businessMessage.intention.toString() + ' ' + businessMessage.
33           information.name,
34       implementation <- '##WebService',
35       completionQuantity <- 1,
36       isForCompensation <- false,
37       startQuantity <- 1,
38       messageRef <- businessMessage.information,
39       outgoing <- if thisModule.
40           getAllipElementsForSubprocess() -> includes(businessMessage) then
41           thisModule.successorSequenceFlowForSubprocess(businessMessage)
42         else
43           thisModule.successorSequenceFlowForProcess(businessMessage)
44         endif
```

```
45     )
46 }
47
48 ------------------------------------------------------------------------
49 ------------- Transforming business messages sent --------------------
50 ------------- with time constraint -----------------------------------
51 ------------------------------------------------------------------------
52 rule sendBusinessMessageWithTimeConstraint2SendTask extends
       sendBusinessMessage2SendTask{
53   from
54     businessMessage: MMupcolbpip!BusinessMessage (
55       businessMessage.hasTimeConstraint()
56     )
57   to
58     gatewayDiverging: MMbpmn2!ExclusiveGateway (
59       id <- 'ExclusiveGateway_' + businessMessage.id,
60       gatewayDirection <- #Diverging,
61       default <- internalSequenceFlow
62     ),
63     internalSequenceFlow: MMbpmn2!SequenceFlow (
64       id <- 'internalSequenceFlow_' + businessMessage.id,
65       isImmediate <- true,
66       sourceRef <- gatewayDiverging,
67       targetRef <- sendTask
68     ),
69     sendTask: MMbpmn2!SendTask,
70     timerSequenceFlow: MMbpmn2!SequenceFlow (
71       id <- 'timerSequenceFlow_' + businessMessage.id,
72       isImmediate <- true,
73       sourceRef <- gatewayDiverging,
74       name <- businessMessage.getTimeConstraint() + '-TR',
75       targetRef <- terminateEvent
76     ),
77     terminateEvent: MMbpmn2!EndEvent (
78       id <- 'terminateEvent_' + businessMessage.id,
79       eventDefinitions <- eventDefinition
80     ),
81     eventDefinition: MMbpmn2!TerminateEventDefinition (
82       id <- 'eventDefinition_' + businessMessage.id
83     )
84
85 }
86
87 ------------------------------------------------------------------------
88 ----------- Transforming business messages received -----------------
89 ------------------------------------------------------------------------
90 rule receiveBusinessMessage2ReceiveTask {
91   from
92     businessMessage: MMupcolbpip!BusinessMessage (
93       businessMessage.isReceiverRequiredRole(thisModule.getParameterValue('roleId
           ',
```

```
94          'role'))
95    )
96  to
97    receiveTask: MMbpmn2!ReceiveTask (
98      id <- businessMessage.id,
99      name <- businessMessage.intention.toString() + ' ' + businessMessage.
100         information.name,
101     implementation <- '##WebService',
102     completionQuantity <- 1,
103     isForCompensation <- false,
104     startQuantity <- 1,
105     messageRef <- businessMessage.information,
106     outgoing <- if thisModule.getAllipElementsForSubprocess() ->
107         includes(businessMessage) then
108         thisModule.successorSequenceFlowForSubprocess(businessMessage)
109       else
110         thisModule.successorSequenceFlowForProcess(businessMessage)
111       endif
112   )
113 }
```

B.3. Módulo ATL protocol2integration

Listado B.3: Módulo protocol2integration.atl

```
1 -- @name Interaction Protocol to Integration Process
2 -- @version 1.0
3 -- @domains business processes, business-to-business
4 -- @author Ivanna Lazarte
5 -- @collaborator Ramiro Savoie
6 -- @date 22/03/2012
7 -- @description  Transforming collaborative processes into integration processes
8 -- @atlcompiler atl2010
9 -- @path MMupcolbpip=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/upcolbpip.
      ecore
10 -- @path MMbpmn2=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/BPMN20.ecore
11 -- @path MMParam=/ar.edu.utn.frsf.cidisi.upcolbpip2bpmn/metamodels/parameters.
      ecore
12
13 module protocol2integration;
14 create OUT : MMbpmn2 from IN: MMupcolbpip, INParam: MMParam;
15
16 uses speechActs;
17 uses parameterHelpers;
18
19 helper context MMupcolbpip!BusinessMessage def:getScript(bmID: String): String =
20   if thisModule.hasParameter(bmID, 'script') then
21     thisModule.getParameterValue(bmID, 'script')
22   else
23     OclUndefined
24   endif;
25
26 helper context MMupcolbpip!BusinessMessage def:getScriptFormat(bmID: String):
      String =
27   if thisModule.hasParameter(bmID, 'scriptFormat') then
28     thisModule.getParameterValue(bmID, 'scriptFormat')
29   else
30     OclUndefined
31   endif;
32
33
34 helper context MMupcolbpip!BusinessMessage def: getTaskLabel(): String =
35   if (self.isInform) then
36     'Process '
37   else
38     'Evaluate '
39   endif;
40
41 ------------------------------------------------------------------------
42 ---- Send Business Message to WAP3 - Sender role -----------------------
43 ---- without parameters ------------------------------------------------
```

```
44 ------------------------------------------------------------------------
45 rule sendBusinessMessage2WAP3 {
46   from
47     businessMessage: MMupcolbpip!BusinessMessage (
48       businessMessage.isSenderRequiredRole(thisModule.getParameterValue('roleId','
             role')) and
49        (businessMessage.isInform or businessMessage.isCallForProposal or
              businessMessage.isPropose or businessMessage.isRequest)
50     )
51   to
52     Task: MMbpmn2!Task
53     (
54       id <- 'abstractTask_' + businessMessage.id,
55       name <- 'Generate ' + businessMessage.information.name,
56       completionQuantity <- 1,
57       isForCompensation <- false,
58       startQuantity <- 1
59     ),
60     sendTask: MMbpmn2!SendTask (
61       id <- 'sendTask_' + businessMessage.id,
62       name <- businessMessage.intention.toString() + ' ' + businessMessage.
63           information.name,
64       implementation <- '##WebService',
65       completionQuantity <- 1,
66       isForCompensation <- false,
67       startQuantity <- 1,
68       messageRef <- businessMessage.information,
69       outgoing <- if thisModule.getAllipElementsForSubprocess() ->
70           includes(businessMessage) then
71           thisModule.successorSequenceFlowForSubprocess(businessMessage)
72         else
73           thisModule.successorSequenceFlowForProcess(businessMessage)
74         endif
75     ),
76     internalSequenceFlow: MMbpmn2!SequenceFlow (
77       id <- 'sequenceFlow_' + businessMessage.id,
78       isImmediate <- true,
79       sourceRef <- Task,
80       targetRef <- sendTask
81     )
82 }
83
84 ------------------------------------------------------------------------
85 ---- Send Business Message to WAP3 - Sender role -----------------------
86 ---- with parameters:  taskType= ServiceTask ---------------------------
87 ------------------------------------------------------------------------
88 rule sendBusinessMessage2WAP3_ServiceTask extends sendBusinessMessage2WAP3 {
89   from
90     businessMessage: MMupcolbpip!BusinessMessage (
91        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ServiceTask')
92     )
```

```
93  to
94    Task: MMbpmn2!ServiceTask
95    (
96      id <- 'serviceTask_' + businessMessage.id,
97      implementation <- '##WebService',
98      operationRef <- if thisModule.hasParameter(businessMessage.id, 'operation')
            then
99              thisModule.serviceOperation(businessMessage)
100           else
101             OclUndefined
102           endif
103   )
104 }
105
106 ------------------------------------------------------------------------
107 ---- Send Business Message to WAP3 - Sender role -----------------------
108 ---- with parameters:  taskType= UserTask ------------------------------
109 ------------------------------------------------------------------------
110 rule sendBusinessMessage2WAP3_UserTask extends sendBusinessMessage2WAP3 {
111   from
112     businessMessage: MMupcolbpip!BusinessMessage (
113        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'UserTask')
114     )
115   to
116     Task: MMbpmn2!UserTask
117     (
118       id <- 'userTask_' + businessMessage.id,
119       implementation <- thisModule.getParameterValue(businessMessage.id, '
            implementation')
120     )
121 }
122
123 ------------------------------------------------------------------------
124 ---- Send Business Message to WAP3 - Sender role -----------------------
125 ---- with parameters:  taskType= UserTask ------------------------------
126 ----- and performer ----------------------------------------------------
127 ------------------------------------------------------------------------
128 rule sendBusinessMessage2WAP3_UserTaskWithPeformer extends
      sendBusinessMessage2WAP3_UserTask {
129   from
130     businessMessage: MMupcolbpip!BusinessMessage (
131        thisModule.hasParameter(businessMessage.id, 'performer')
132     )
133   to
134     Task: MMbpmn2!UserTask (
135       resources <- performer
136     ),
137     performer: MMbpmn2!Performer (
138       id <- 'performer_'  + businessMessage.id,
139       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
140     )
```

```
141 }
142
143 ------------------------------------------------------------------------
144 ---- Send Business Message to WAP3 - Sender role -----------------------
145 ---- with parameters:  taskType= BusinessRuleTask ----------------------
146 ------------------------------------------------------------------------
147 rule sendBusinessMessage2WAP3_BusinessRuleTask extends sendBusinessMessage2WAP3 {
148   from
149     businessMessage: MMupcolbpip!BusinessMessage (
150        thisModule.hasParameterValue(businessMessage.id, 'taskType', '
              BusinessRuleTask')
151     )
152   to
153     Task: MMbpmn2!BusinessRuleTask
154     (
155       id <- 'businessRuleTask_' + businessMessage.id,
156       implementation <- thisModule.getParameterValue(businessMessage.id, '
              implementation')
157     )
158 }
159
160
161 ------------------------------------------------------------------------
162 ---- Send Business Message to WAP3 - Sender role -----------------------
163 ---- with parameters:  taskType= ScriptTask ----------------------------
164 ------------------------------------------------------------------------
165 rule sendBusinessMessage2WAP3_ScriptTask extends sendBusinessMessage2WAP3 {
166   from
167     businessMessage: MMupcolbpip!BusinessMessage (
168        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ScriptTask')
169     )
170   to
171     Task: MMbpmn2!ScriptTask
172     (
173       id <- 'scriptTask_' + businessMessage.id,
174       script <- businessMessage.getScript(businessMessage.id),
175       scriptFormat <- businessMessage.getScriptFormat(businessMessage.id)
176     )
177 }
178
179
180 ------------------------------------------------------------------------
181 ---- Send Business Message to WAP3 - Sender role -----------------------
182 ---- with parameters:  taskType= ManualTask ----------------------------
183 ------------------------------------------------------------------------
184 rule sendBusinessMessage2WAP3_ManualTask extends sendBusinessMessage2WAP3 {
185   from
186     businessMessage: MMupcolbpip!BusinessMessage (
187        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ManualTask')
188     )
189   to
```

```
190     Task: MMbpmn2!ManualTask(
191       id <- 'manualTask_' + businessMessage.id
192       )
193 }
194
195 ------------------------------------------------------------------------
196 ---- Send Business Message to WAP3 - Sender role -----------------------
197 ---- with parameters:  taskType= ManualTask ----------------------------
198 ----- and performer ----------------------------------------------------
199 ------------------------------------------------------------------------
200 rule sendBusinessMessage2WAP3_ManualTaskWithPerformer extends
        sendBusinessMessage2WAP3_ManualTask {
201   from
202     businessMessage: MMupcolbpip!BusinessMessage (
203       thisModule.hasParameter(businessMessage.id, 'performer')
204     )
205   to
206     Task: MMbpmn2!ManualTask (
207       resources <- performer
208     ),
209
210     performer: MMbpmn2!Performer (
211       id <- 'performer_'  + businessMessage.id,
212       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
213     )
214 }
215
216 ------------------------------------------------------------------------
217 ---- Receive Business Message to WAP3 - Receiver role -----------------
218 ---- without parameters -----------------------------------------------
219 ------------------------------------------------------------------------
220 rule receiveBusinessMessage2WAP3 {
221   from
222     businessMessage: MMupcolbpip!BusinessMessage (
223       businessMessage.isReceiverRequiredRole(thisModule.getParameterValue('roleId
              ', 'role')) and  (businessMessage.
224           isInform  or businessMessage.isCallForProposal or businessMessage.
                isPropose or businessMessage.
225           isRequest)
226     )
227   to
228     receiveTask: MMbpmn2!ReceiveTask (
229       id <- 'receiveTask_' + businessMessage.id,
230       name <- businessMessage.intention.toString() + ' ' + businessMessage.
              information.name,
231       implementation <- '##WebService',
232       completionQuantity <- 1,
233       isForCompensation <- false,
234       startQuantity <- 1,
235       messageRef <- businessMessage.information
236     ),
237
```

```
238     internalSequenceFlow: MMbpmn2!SequenceFlow (
239       id <- 'sequenceFlow_' + businessMessage.id,
240       isImmediate <- true,
241       sourceRef <- receiveTask,
242       targetRef <- Task
243     ),
244     Task: MMbpmn2!Task (
245       id <- 'abstractTask_' + businessMessage.id,
246       name <- businessMessage.getTaskLabel() + businessMessage.information.name,
247       completionQuantity <- 1,
248       isForCompensation <- false,
249       startQuantity <- 1,
250       outgoing <- if thisModule.getAllipElementsForSubprocess() ->
251           includes(businessMessage) then
252           thisModule.successorSequenceFlowForSubprocess(businessMessage)
253         else
254           thisModule.successorSequenceFlowForProcess(businessMessage)
255         endif
256     )
257 }
258
259 ------------------------------------------------------------------------
260 ---- Receive Business Message to WAP3 - Receiver role ------------------
261 ---- with parameter: taskType= ServiceTask -----------------------------
262 ------------------------------------------------------------------------
263 rule receiveBusinessMessage2WAP3_ServiceTask extends receiveBusinessMessage2WAP3 {
264   from
265     businessMessage: MMupcolbpip!BusinessMessage (
266        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ServiceTask')
267     )
268   to
269     receiveTask: MMbpmn2!ReceiveTask,
270     Task: MMbpmn2!ServiceTask (
271       id <- 'serviceTask_' + businessMessage.id,
272       implementation <- '##WebService',
273       operationRef <- if thisModule.hasParameter(businessMessage.id, 'operation')
              then
274               thisModule.serviceOperation(businessMessage)
275             else
276               OclUndefined
277             endif
278     )
279 }
280
281
282 ------------------------------------------------------------------------
283 ---- Receive Business Message to WAP3 - Receiver role ------------------
284 ---- with parameter: taskType= UserTask --------------------------------
285 ------------------------------------------------------------------------
286 rule receiveBusinessMessage2WAP3_UserTask extends receiveBusinessMessage2WAP3 {
287   from
```

```
288     businessMessage: MMupcolbpip!BusinessMessage (
289        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'UserTask')
290     )
291   to
292     receiveTask: MMbpmn2!ReceiveTask,
293     Task: MMbpmn2!UserTask (
294       id <- 'userTask_' + businessMessage.id,
295       implementation <- thisModule.getParameterValue(businessMessage.id, '
              implementation')
296   )
297 }
298
299 ------------------------------------------------------------------------
300 ---- Receive Business Message to WAP3 - Receiver role ------------------
301 ---- with parameter: taskType= UserTask ---------------------------------
302 ---- and performer -----------------------------------------------------
303 ------------------------------------------------------------------------
304 rule receiveBusinessMessage2WAP3_UserTaskWithPerformer extends
       receiveBusinessMessage2WAP3_UserTask {
305   from
306     businessMessage: MMupcolbpip!BusinessMessage (
307       thisModule.hasParameter(businessMessage.id, 'performer')
308     )
309   to
310     receiveTask: MMbpmn2!ReceiveTask,
311     Task: MMbpmn2!UserTask (
312       resources <- performer
313     ),
314
315     performer: MMbpmn2!Performer (
316       id <- 'performer_'  + businessMessage.id,
317       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
318     )
319 }
320
321 ------------------------------------------------------------------------
322 ---- Receive Business Message to WAP3 - Receiver role ------------------
323 ---- with parameter: taskType= BusinessRuleTask ------------------------
324 ------------------------------------------------------------------------
325 rule receiveBusinessMessage2WAP3_BusinessRuleTask extends
       receiveBusinessMessage2WAP3 {
326   from
327     businessMessage: MMupcolbpip!BusinessMessage (
328        thisModule.hasParameterValue(businessMessage.id, 'taskType', '
              BusinessRuleTask')
329     )
330   to
331     receiveTask: MMbpmn2!ReceiveTask,
332     Task: MMbpmn2!BusinessRuleTask (
333       id <- 'businessRuleTask_' + businessMessage.id,
```

```
334       implementation <- thisModule.getParameterValue(businessMessage.id, '
              implementation')
335     )
336 }
337
338
339 -------------------------------------------------------------------------
340 ---- Receive Business Message to WAP3 - Receiver role -------------------
341 ---- with parameter: taskType= ScriptTask -------------------------------
342 -------------------------------------------------------------------------
343 rule receiveBusinessMessage2WAP3_ScriptTask extends receiveBusinessMessage2WAP3 {
344   from
345     businessMessage: MMupcolbpip!BusinessMessage (
346        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ScriptTask')
347     )
348   to
349     receiveTask: MMbpmn2!ReceiveTask,
350     Task: MMbpmn2!ScriptTask (
351       id <- 'scriptTask_' + businessMessage.id,
352       script <- businessMessage.getScript(businessMessage.id),
353       scriptFormat <- businessMessage.getScriptFormat(businessMessage.id)
354     )
355 }
356
357 -------------------------------------------------------------------------
358 ---- Receive Business Message to WAP3 - Receiver role -------------------
359 ---- with parameter: taskType= ManualTask -------------------------------
360 -------------------------------------------------------------------------
361 rule receiveBusinessMessage2WAP3_ManualTask extends receiveBusinessMessage2WAP3 {
362   from
363     businessMessage: MMupcolbpip!BusinessMessage (
364        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ManualTask')
365     )
366   to
367     receiveTask: MMbpmn2!ReceiveTask,
368     Task: MMbpmn2!ManualTask (
369       id <- 'manualTask_' + businessMessage.id
370     )
371 }
372
373 -------------------------------------------------------------------------
374 ---- Receive Business Message to WAP3 - Receiver role -------------------
375 ---- with parameter: taskType= ManualTask -------------------------------
376 ---- and performer ------------------------------------------------------
377 -------------------------------------------------------------------------
378 rule receiveBusinessMessage2WAP3_ManualTaskWithPerformer extends
       receiveBusinessMessage2WAP3_ManualTask {
379   from
380     businessMessage: MMupcolbpip!BusinessMessage (
381       thisModule.hasParameter(businessMessage.id, 'performer')
382     )
```

```
383   to
384     receiveTask: MMbpmn2!ReceiveTask,
385     Task: MMbpmn2!ManualTask (
386       resources <- performer
387     ),
388
389     performer: MMbpmn2!Performer (
390       id <- 'performer_'  + businessMessage.id,
391       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
392     )
393 }
394
395 ------------------------------------------------------------------------
396 ---- Rule Send Business Message to WAP5 - Sender role ------------------
397 ---- without parameters ------------------------------------------------
398 ------------------------------------------------------------------------
399 rule sendBusinessMessage2WAP5 {
400   from
401     businessMessage: MMupcolbpip!BusinessMessage (
402       businessMessage.isSenderRequiredRole(thisModule.getParameterValue('roleId',
              'role')) and (businessMessage.isAcceptProposal or businessMessage.
              isAgree or businessMessage.isConfirm or
403  businessMessage.isDisconfirm or businessMessage.isRefuse or businessMessage.
         isRejectProposal)
404     )
405   to
406     Task: MMbpmn2!Task (
407       id <- 'abstractTask_' + businessMessage.id,
408       name <- 'Generate ' + businessMessage.information.name,
409       completionQuantity <- 1,
410       isForCompensation <- false,
411       startQuantity <- 1
412     ),
413     internalSequenceFlow: MMbpmn2!SequenceFlow (
414       id <- 'sequenceFlow_' + businessMessage.id,
415       isImmediate <- true,
416       sourceRef <- Task,
417       targetRef <- sendTask
418     ),
419     sendTask: MMbpmn2!SendTask (
420       id <- 'sendTask_' + businessMessage.id,
421       name <- businessMessage.intention.toString() + ' ' + businessMessage.
              information.name,
422       implementation <- '##WebService',
423       completionQuantity <- 1,
424       isForCompensation <- false,
425       startQuantity <- 1,
426       messageRef <- businessMessage.information,
427       outgoing <- if thisModule.getAllipElementsForSubprocess() ->
428           includes(businessMessage) then
429           thisModule.successorSequenceFlowForSubprocess(businessMessage)
430
```

```
431         else
432           thisModule.successorSequenceFlowForProcess(businessMessage)
433         endif
434     )
435 }
436
437 -------------------------------------------------------------------------
438 ---- Rule Send Business Message to WAP5 - Sender role ------------------
439 ---- with parameter: taskType= ServiceTask ------------------------------
440 -------------------------------------------------------------------------
441 rule sendBusinessMessage2WAP5_ServiceTask extends sendBusinessMessage2WAP5 {
442   from
443     businessMessage: MMupcolbpip!BusinessMessage (
444        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ServiceTask')
445     )
446   to
447     Task: MMbpmn2!ServiceTask (
448       id <- 'serviceTask_' + businessMessage.id,
449       implementation <- '##WebService',
450       operationRef <- if thisModule.hasParameter(businessMessage.id, 'operation')
              then
451               thisModule.serviceOperation(businessMessage)
452             else
453               OclUndefined
454             endif
455     )
456 }
457
458 -------------------------------------------------------------------------
459 ---- Rule Send Business Message to WAP5 - Sender role ------------------
460 ---- with parameter: taskType= UserTask ---------------------------------
461 -------------------------------------------------------------------------
462 rule sendBusinessMessage2WAP5_UserTask extends sendBusinessMessage2WAP5 {
463   from
464     businessMessage: MMupcolbpip!BusinessMessage (
465        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'UserTask')
466     )
467   to
468     Task: MMbpmn2!UserTask (
469       id <- 'userTask_' + businessMessage.id,
470       implementation <- thisModule.getParameterValue(businessMessage.id, '
              implementation')
471     )
472 }
473
474 -------------------------------------------------------------------------
475 ---- Rule Send Business Message to WAP5 - Sender role ------------------
476 ---- with parameter: taskType= UserTask ---------------------------------
477 ---- and performer ------------------------------------------------------
478 -------------------------------------------------------------------------
```

```
479 rule sendBusinessMessage2WAP5_UserTaskWithPerformer extends
       sendBusinessMessage2WAP5_UserTask {
480   from
481     businessMessage: MMupcolbpip!BusinessMessage (
482       thisModule.hasParameter(businessMessage.id, 'performer')
483     )
484   to
485     Task: MMbpmn2!UserTask (
486       resources <- performer
487     ),
488
489     performer: MMbpmn2!Performer (
490       id <- 'performer_'  + businessMessage.id,
491       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
492     )
493 }
494
495 ------------------------------------------------------------------------
496 ---- Rule Send Business Message to WAP5 - Sender role ------------------
497 ---- with parameter: taskType= BusinessRuleTask ------------------------
498 ------------------------------------------------------------------------
499 rule sendBusinessMessage2WAP5_BusinessRuleTask extends sendBusinessMessage2WAP5 {
500   from
501     businessMessage: MMupcolbpip!BusinessMessage (
502        thisModule.hasParameterValue(businessMessage.id, 'taskType', '
               BusinessRuleTask')
503     )
504   to
505     Task: MMbpmn2!BusinessRuleTask (
506       id <- 'businessRuleTask_' + businessMessage.id,
507       implementation <- thisModule.getParameterValue(businessMessage.id, '
              implementation')
508     )
509 }
510
511
512
513 ------------------------------------------------------------------------
514 ---- Rule Send Business Message to WAP5 - Sender role ------------------
515 ---- with parameter: taskType= ScriptTask ------------------------------
516 ------------------------------------------------------------------------
517 rule sendBusinessMessage2WAP5_ScriptTask extends sendBusinessMessage2WAP5 {
518   from
519     businessMessage: MMupcolbpip!BusinessMessage (
520        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ScriptTask')
521     )
522   to
523     Task: MMbpmn2!ScriptTask (
524       id <- 'scriptTask_' + businessMessage.id,
525       script <- businessMessage.getScript(businessMessage.id),
526       scriptFormat <- businessMessage.getScriptFormat(businessMessage.id)
```

```
527     )
528 }
529
530
531 ------------------------------------------------------------------------
532 ---- Rule Send Business Message to WAP5 - Sender role ------------------
533 ---- with parameter: taskType= ManualTask -------------------------------
534 ------------------------------------------------------------------------
535 rule sendBusinessMessage2WAP5_ManualTask extends sendBusinessMessage2WAP5 {
536   from
537     businessMessage: MMupcolbpip!BusinessMessage (
538        thisModule.hasParameterValue(businessMessage.id, 'taskType', 'ManualTask')
539     )
540   to
541     Task: MMbpmn2!ManualTask (
542       id <- 'manualTask_' + businessMessage.id
543     )
544 }
545
546 ------------------------------------------------------------------------
547 ---- Rule Send Business Message to WAP5 - Sender role ------------------
548 ---- with parameter: taskType= ManualTask -------------------------------
549 ---- and performer -----------------------------------------------------
550 ------------------------------------------------------------------------
551 rule sendBusinessMessage2WAP5_ManualTaskWithPerformer extends
       sendBusinessMessage2WAP5_ManualTask {
552   from
553     businessMessage: MMupcolbpip!BusinessMessage (
554       thisModule.hasParameter(businessMessage.id, 'performer')
555     )
556   to
557     Task: MMbpmn2!ManualTask (
558       resources <- performer
559     ),
560
561     performer: MMbpmn2!Performer (
562       id <- 'performer_'  + businessMessage.id,
563       name <- thisModule.getParameterValue(businessMessage.id, 'performer')
564     )
565 }
566
567 ------------------------------------------------------------------------
568 ---- Receive Business Message to WAP5 - Receiver role ------------------
569 ---- without parameters ------------------------------------------------
570 ------------------------------------------------------------------------
571 rule receiveBusinessMessage2WAP5 {
572   from
573     businessMessage: MMupcolbpip!BusinessMessage (
574       businessMessage.isReceiverRequiredRole(thisModule.getParameterValue('roleId
            ', 'role')) and (businessMessage.isAcceptProposal or businessMessage.
            isAgree or businessMessage.isConfirm or businessMessage.isDisconfirm or
```

```
            businessMessage.isRefuse or businessMessage.isRejectProposal)
    )
  to
    receiveTask: MMbpmn2!ReceiveTask (
      id <- 'receiveTask_' + businessMessage.id,
      name <- businessMessage.intention.toString() + ' ' + businessMessage.
          information.name,
      implementation <- '##WebService',
      completionQuantity <- 1,
      isForCompensation <- false,
      startQuantity <- 1,
      messageRef <- businessMessage.information,
      outgoing <- if thisModule.getAllipElementsForSubprocess() ->
          includes(businessMessage) then
          thisModule.successorSequenceFlowForSubprocess(businessMessage)
        else
          thisModule.successorSequenceFlowForProcess(businessMessage)
        endif
    )
}

-------------------------------------------------------------------------------
-------- LAZY RULES -----------------------------------------------------------
-------------------------------------------------------------------------------
lazy rule serviceOperation {
  from
    businessMessage: MMupcolbpip!BusinessMessage
  using {
    definitions: MMbpmn2!Definitions = MMbpmn2!Definitions.allInstances().last();
  }
  to
    operation: MMbpmn2!Operation (
      id <- 'operation_' + businessMessage.id,
      name <- thisModule.getParameterValue(businessMessage.id, 'operation'),
      inMessageRef <- businessMessage.information
    ),

    interface: MMbpmn2!Interface (
      id <- 'interface_' + businessMessage.id,
      name <- businessMessage.name + ' Interface',
      operations <- operation
      )
  do
  {
    definitions.rootElements <- definitions.rootElements.append(interface);
  }
}
```

B.4. Plug-in ATL Upcolbpip2bpmn para lanzar el motor de transformación

Listado B.4: Plug-in Upcolbpip2bpmn.java

```
1 /*******************************************************************************
2  * Copyright (c) 2010, 2011 Obeo.
3  * All rights reserved. This program and the accompanying materials
4  * are made available under the terms of the Eclipse Public License v1.0
5  * which accompanies this distribution, and is available at
6  * http://www.eclipse.org/legal/epl-v10.html
7  *
8  * Contributors:
9  *     Obeo - initial API and implementation
10 *******************************************************************************/
11 package ar.edu.utn.frsf.cidisi.upcolbpip2bpmn.tool.files;
12
13 import java.io.IOException;
14 import java.io.InputStream;
15 import java.net.URL;
16 import java.util.ArrayList;
17 import java.util.Arrays;
18 import java.util.HashMap;
19 import java.util.List;
20 import java.util.Map;
21 import java.util.Properties;
22 import java.util.Map.Entry;
23
24 import org.eclipse.core.runtime.FileLocator;
25 import org.eclipse.core.runtime.IProgressMonitor;
26 import org.eclipse.core.runtime.NullProgressMonitor;
27 import org.eclipse.core.runtime.Path;
28 import org.eclipse.core.runtime.Platform;
29 import org.eclipse.emf.ecore.resource.Resource;
30 import org.eclipse.emf.ecore.xmi.impl.EcoreResourceFactoryImpl;
31 import org.eclipse.m2m.atl.common.ATLExecutionException;
32 import org.eclipse.m2m.atl.core.ATLCoreException;
33 import org.eclipse.m2m.atl.core.IExtractor;
34 import org.eclipse.m2m.atl.core.IInjector;
35 import org.eclipse.m2m.atl.core.IModel;
36 import org.eclipse.m2m.atl.core.IReferenceModel;
37 import org.eclipse.m2m.atl.core.ModelFactory;
38 import org.eclipse.m2m.atl.core.emf.EMFExtractor;
39 import org.eclipse.m2m.atl.core.emf.EMFInjector;
40 import org.eclipse.m2m.atl.core.emf.EMFModelFactory;
41 import org.eclipse.m2m.atl.core.launch.ILauncher;
42 import org.eclipse.m2m.atl.engine.emfvm.launch.EMFVMLauncher;
43
```

```
/**
 * Entry point of the 'Upcolbpip2bpmn' transformation module.
 */
public class Upcolbpip2bpmn {

  /**
   * The property file. Stores module list, the metamodel and library locations.
   * @generated
   */
  private Properties properties;

  /**
   * The IN model.
   * @generated
   */
  protected IModel inModel;

  /**
   * The INParam model.
   * @generated
   */
  protected IModel inparamModel;

  /**
   * The OUT model.
   * @generated
   */
  protected IModel outModel;

  /**
   * The main method.
   *
   * @param args
   *            are the arguments
   * @generated
   */
  public static void main(String[] args) {
    try {
      if (args.length < 3) {
        System.out.println("Arguments not valid : {IN_model_path,
            INParam_model_path, OUT_model_path}.");
      } else {
        Upcolbpip2bpmn runner = new Upcolbpip2bpmn();
        runner.loadModels(args[0], args[1]);
        runner.doUpcolbpip2bpmn(new NullProgressMonitor());
        runner.saveModels(args[2]);
      }
    } catch (ATLCoreException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
```

```
94     } catch (ATLExecutionException e) {
95       e.printStackTrace();
96     }
97   }
98
99   /**
100   * Constructor.
101   *
102   * @generated
103   */
104  public Upcolbpip2bpmn() throws IOException {
105    properties = new Properties();
106    InputStream propertiesInputStream = getFileURL("Upcolbpip2bpmn.properties").
           openStream();
107    properties.load(propertiesInputStream);
108    propertiesInputStream.close();
109    Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put("ecore", new
           EcoreResourceFactoryImpl());
110  }
111
112  /**
113   * Load the input and input/output models, initialize output models.
114   *
115   * @param inModelPath
116   *            the IN model path
117   * @param inparamModelPath
118   *            the INParam model path
119   * @throws ATLCoreException
120   *             if a problem occurs while loading models
121   *
122   * @generated
123   */
124  public void loadModels(String inModelPath, String inparamModelPath) throws
         ATLCoreException {
125    ModelFactory factory = new EMFModelFactory();
126    IInjector injector = new EMFInjector();
127    IReferenceModel mmupcolbpipMetamodel = factory.newReferenceModel();
128    injector.inject(mmupcolbpipMetamodel, getMetamodelUri("MMupcolbpip"));
129    IReferenceModel mmparamMetamodel = factory.newReferenceModel();
130    injector.inject(mmparamMetamodel, getMetamodelUri("MMParam"));
131    IReferenceModel mmbpmn2Metamodel = factory.newReferenceModel();
132    injector.inject(mmbpmn2Metamodel, getMetamodelUri("MMbpmn2"));
133    this.inModel = factory.newModel(mmupcolbpipMetamodel);
134    injector.inject(inModel, inModelPath);
135    this.inparamModel = factory.newModel(mmparamMetamodel);
136    injector.inject(inparamModel, inparamModelPath);
137    this.outModel = factory.newModel(mmbpmn2Metamodel);
138  }
139
140  /**
141   * Save the output and input/output models.
```

```
142   *
143   * @param outModelPath
144   *            the OUT model path
145   * @throws ATLCoreException
146   *             if a problem occurs while saving models
147   *
148   * @generated
149   */
150  public void saveModels(String outModelPath) throws ATLCoreException {
151    IExtractor extractor = new EMFExtractor();
152    extractor.extract(outModel, outModelPath);
153  }
154
155  /**
156   * Transform the models.
157   *
158   * @param monitor
159   *            the progress monitor
160   * @throws ATLCoreException
161   *             if an error occurs during models handling
162   * @throws IOException
163   *             if a module cannot be read
164   * @throws ATLExecutionException
165   *             if an error occurs during the execution
166   *
167   * @generated
168   */
169  public Object doUpcolbpip2bpmn(IProgressMonitor monitor) throws ATLCoreException
        , IOException, ATLExecutionException {
170    ILauncher launcher = new EMFVMLauncher();
171    List<InputStream> inputStreamsToClose = new ArrayList<InputStream>();
172    Map<String, Object> launcherOptions = getOptions();
173    launcher.initialize(launcherOptions);
174    launcher.addInModel(inModel, "IN", "MMupcolbpip");
175    launcher.addInModel(inparamModel, "INParam", "MMParam");
176    launcher.addOutModel(outModel, "OUT", "MMbpmn2");
177    InputStream libraryStream_speechActs = getLibraryAsStream("speechActs");
178    inputStreamsToClose.add(libraryStream_speechActs);
179    launcher.addLibrary("speechActs", getLibraryAsStream("speechActs"));
180    InputStream libraryStream_parameterHelpers = getLibraryAsStream("
           parameterHelpers");
181    inputStreamsToClose.add(libraryStream_parameterHelpers);
182    launcher.addLibrary("parameterHelpers", getLibraryAsStream("parameterHelpers")
           );
183    InputStream[] modulesStreams = getModulesList();
184    inputStreamsToClose.addAll(Arrays.asList(modulesStreams));
185    Object result = launcher.launch("run", monitor, launcherOptions, (Object[])
           modulesStreams);
186    for (InputStream inputStream : inputStreamsToClose) {
187      inputStream.close();
188    }
```

```
189     return result;
190   }
191
192   /**
193    * Returns an Array of the module input streams, parameterized by the
194    * property file.
195    *
196    * @return an Array of the module input streams
197    * @throws IOException
198    *             if a module cannot be read
199    *
200    * @generated
201    */
202   protected InputStream[] getModulesList() throws IOException {
203     InputStream[] modules = null;
204     String modulesList = properties.getProperty("Upcolbpip2bpmn.modules");
205     if (modulesList != null) {
206       String[] moduleNames = modulesList.split(",");
207       modules = new InputStream[moduleNames.length];
208       for (int i = 0; i < moduleNames.length; i++) {
209         String asmModulePath = new Path(moduleNames[i].trim()).removeFileExtension
                ().addFileExtension("asm").toString();
210         modules[i] = getFileURL(asmModulePath).openStream();
211       }
212     }
213     return modules;
214   }
215
216   /**
217    * Returns the URI of the given metamodel, parameterized from the property file.
218    *
219    * @param metamodelName
220    *             the metamodel name
221    * @return the metamodel URI
222    *
223    * @generated
224    */
225   protected String getMetamodelUri(String metamodelName) {
226     return properties.getProperty("Upcolbpip2bpmn.metamodels." + metamodelName);
227   }
228
229   /**
230    * Returns the file name of the given library, parameterized from the property
           file.
231    *
232    * @param libraryName
233    *             the library name
234    * @return the library file name
235    *
236    * @generated
237    */
```

```
238  protected InputStream getLibraryAsStream(String libraryName) throws IOException
         {
239    return getFileURL(properties.getProperty("Upcolbpip2bpmn.libraries." +
           libraryName)).openStream();
240  }
241
242  /**
243   * Returns the options map, parameterized from the property file.
244   *
245   * @return the options map
246   *
247   * @generated
248   */
249  protected Map<String, Object> getOptions() {
250    Map<String, Object> options = new HashMap<String, Object>();
251    for (Entry<Object, Object> entry : properties.entrySet()) {
252      if (entry.getKey().toString().startsWith("Upcolbpip2bpmn.options.")) {
253        options.put(entry.getKey().toString().replaceFirst("Upcolbpip2bpmn.options
               .", ""),
254        entry.getValue().toString());
255      }
256    }
257    return options;
258  }
259
260  /**
261   * Finds the file in the plug-in. Returns the file URL.
262   *
263   * @param fileName
264   *            the file name
265   * @return the file URL
266   * @throws IOException
267   *             if the file doesn't exist
268   *
269   * @generated
270   */
271  protected static URL getFileURL(String fileName) throws IOException {
272    final URL fileURL;
273    if (isEclipseRunning()) {
274      URL resourceURL = Upcolbpip2bpmn.class.getResource(fileName);
275      if (resourceURL != null) {
276        fileURL = FileLocator.toFileURL(resourceURL);
277      } else {
278        fileURL = null;
279      }
280    } else {
281      fileURL = Upcolbpip2bpmn.class.getResource(fileName);
282    }
283    if (fileURL == null) {
284      throw new IOException("'" + fileName + "' not found");
285    } else {
```

```
286       return fileURL;
287     }
288   }
289
290   /**
291    * Tests if eclipse is running.
292    *
293    * @return <code>true</code> if eclipse is running
294    *
295    * @generated
296    */
297   public static boolean isEclipseRunning() {
298     try {
299       return Platform.isRunning();
300     } catch (Throwable exception) {
301       // Assume that we aren't running.
302     }
303     return false;
304   }
305 }
```

B.5. Archivo de definición de propiedades del motor de transformación Upcolbpip2bpmn

Listado B.5: Archivo de definición de propiedades Upcolbpip2bpmn

```
# ================================================================================
# Upcolbpip2bpmn properties
# ================================================================================

# ATL modules: if several, by order of superimposition (the latter ones overrides
    the former ones)
Upcolbpip2bpmn.modules = upcolbpip2bpmn.atl,protocol2integration.atl

# Metamodels paths or nsUris
Upcolbpip2bpmn.metamodels.MMupcolbpip = resources/metamodels/upcolbpip.ecore
Upcolbpip2bpmn.metamodels.MMParam = resources/metamodels/parameters.ecore
Upcolbpip2bpmn.metamodels.MMbpmn2 = resources/metamodels/BPMN20.ecore

# Libraries paths
Upcolbpip2bpmn.libraries.speechActs = speechActs.asm
Upcolbpip2bpmn.libraries.parameterHelpers = parameterHelpers.asm

# ATL Launching options
Upcolbpip2bpmn.options.supportUML2Stereotypes = false
Upcolbpip2bpmn.options.printExecutionTime = true
Upcolbpip2bpmn.options.OPTION_CONTENT_TYPE = false
Upcolbpip2bpmn.options.allowInterModelReferences = true
Upcolbpip2bpmn.options.step = false
```

Lista de Abreviaciones

ADT ATL Development Tools.

APROMORE Advanced PROcess MOdel REpository.

ATL ATLAS Transformation Language.

B2B Business-to-Business.

BOD Business Object Document.

BPEL4Chor BPEL for Choreographies.

BPM Gestión de Procesos de Negocio - Business Process Management.

BPMN Business Process Model and Notation.

BPMS Sistemas de Gestión de Procesos de Negocio - Business Process Management Systems.

CIbFw Collaborative Interoperability Framework.

CIM Modelo Independiente de la Computación - Computation-Independent Model.

CPFR Collaborative Planning Forecasting and Replenishment.

CWM Common Warehouse Metamodel.

DAO Objeto de Acceso a Datos - Data Access Object.

ebXML Electronic Business using eXtensible Markup Language.

EMF Eclipse Modeling Framework.

EPC Event-driven Process Chain.

HTML HyperText Markup Language.

IOM Modelo de Orquestación Intermedio - Intermediate Orchestration Model.

J2EE Java Platform, Enterprise Edition.

MAS Multi-Agent System.

MDA Arquitectura Dirigida por Modelos - Model-Driven Architecture.

MDD Desarrollo Dirigido por Modelos - Model-Driven Development.

MDD4SOA Model-Driven Development for Service Oriented Architectures.

MOF Meta-Object Facility.

MOFM2T MOF Model to Text Transformation Language.

MVC Modelo - Vista - Controlador.

OCL Object Constraint Language.

ORM Mapeo Objeto-Relacional - Object-Relational Mapping.

PAISs Sistemas de Información Orientados a Procesos - Process-Aware Information Systems.

PIM Modelo Independiente de la Plataforma - Platform-Independent Model.

PSM Modelo Específico de la Plataforma - Platform-Specific Model.

QVT Query/View/Transformation.

SBPR Semantic Business Process Repository.

SOA Arquitectura Orientada a Servicios - Service-Oriented Architecture.

SOAP Simple Object Access Protocol.

TI Tecnologías de la Información.

TICs Tecnologías de Información y Comunicación.

UBL Universal Business Language.

UDDI Universal Description, Discovery and Integration.

UML Unified Modeling Language.

UML4SOA UML extension for SOA.

UMM UN/CEFACT Modeling Methodology.

UP-ColBPIP UML Profile for Collaborative Business Processes based on Interaction Protocols.

VMI Vendor-Managed Inventory.

WS-BPEL Web Services Business Process Execution Language.

WS-CDL Web Services Choreography Description Language.

WSDL Web Services Description Language.

XMI XML Metadata Interchange.

XML Extensible Markup Language.

XSD XML Schema Definition Language.

XSLT Extensible Stylesheet Language Transformation.

Bibliografía

Aalst, Wil M.P. der (2004). «Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management». En: Jörg Desel; Wolfgang Reisig y Grzegorz Rozenberg (Eds.), *Lectures on Concurrency and Petri Nets*, volumen 3098 de *Lecture Notes in Computer Science*, pp. 1–65. Springer Berlin Heidelberg.

Aalst, Wil M.P. der; Hofstede, Arthur H.M. ter; Kiepuszewski, B. y Barros, A.P. (2003a). «Workflow patterns». *Distributed and parallel databases*, **14(1)**, pp. 5–51.

Aalst, Wil M.P. der; Hofstede, Arthur H.M. ter y Weske, Mathias (2003b). «Business Process Management: A Survey». En: Wil M.P. der Aalst y Mathias Weske (Eds.), *Business Process Management*, volumen 2678 de *Lecture Notes in Computer Science*, pp. 1–12. Springer Berlin Heidelberg.

Anneke G. Kleppe, Wim Bast, Jos B. Warmer (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional.

Apache (2013a). «Apache Axis2/Java». `http://axis.apache.org/axis2/java/core/`

Apache (2013b). «Apache Tomcat». `http://tomcat.apache.org/`

Barros, Alistair; Dumas, Marlon y Hofstede, Arthur H.M. ter (2005). «Service Interaction Patterns». En: Wil M.P. der Aalst; Boualem Benatallah; Fabio Casati y Francisco Curbera (Eds.), *Business Process Management*, volumen 3649 de *Lecture Notes in Computer Science*, pp. 302–318. Springer Berlin Heidelberg.

Bauer, Bernhard; Roser, Stephan y Müller, Jörg (2005). «Adaptive Design of Cross-Organizational Business Processes Using a Model-Driven Architecture». En: Otto K. Ferstl; Elmar J. Sinz; Sven Eckert y Tilman Isselhorst (Eds.), *Wirtschaftsinformatik 2005*, pp. 103–121. Springer.

Bauer, Christian y King, Gavin (2005). *Hibernate in Action*. Manning Publications Co..

Bellifemine, Fabio Luigi; Caire, Giovanni y Greenwood, Dominic (2007). *Developing Multi-Agent Systems with JADE*. Wiley.

Bernauer, Martin; Kappel, Gerti y Kramler, Gerhard (2003). «Comparing WSDL-based and ebXML-based Approaches for B2B Protocol Specification». En: Maria Orlowska; Sanjiva Weerawarana; Michael Papazoglou y Jian Yang (Eds.), *Service-Oriented Computing - ICSOC 2003*, volumen 2910 de *Lecture Notes in Computer Science*, pp. 225–240. Springer.

Bernstein, Philip A. y Dayal, Umeshwar (1994). «An Overview of Repository Technology». En: *International Conference on Very Large Data Bases*, pp. 705–705. IEEE Computer Society.

Brown, Alan; Conallen, Jim y Tropeano, Dave (2005). «Introduction: Models, Modeling, and Model-Driven Architecture (MDA)». En: Sami Beydeda; Matthias Book y Volker Gruhn (Eds.), *Model-Driven Software Development*, capítulo 1, pp. 1–16. Springer-Verlag.

Camarinha-Matos, Luis M.; Afsarmanesh, Hamideh; Galeano, Nathalie y Molina, Arturo (2009). «Collaborative networked organizations - Concepts and practice in manufacturing enterprises». *Computers & Industrial Engineering*, **57(1)**, pp. 46–60.

Carlson, David (2005). *Eclipse Distilled*. Addison-Wesley Professional.

Chituc, Claudia-Melania; Azevedo, Américo y Toscano, César (2009). «A framework proposal for seamless interoperability in a collaborative networked environment». *Computers in Industry*, **60(5)**, pp. 317–338.

Dashorst, Martijn y Hillenius, Eelco (2008). *Wicket in Action*. Manning Publications Co..

Decker, Gero y Weske, Mathias (2007). «Behavioral Consistency for B2B Process Integration». En: John Krogstie; Andreas Opdahl y Guttorm Sindre (Eds.), *Advanced Information Systems Engineering*, volumen 4495 de *Lecture Notes in Computer Science*, pp. 81–95. Springer.

Dijkman, Remco; La Rosa, Marcello y Reijers, Hajo A. (2012). «Managing large collections of business process models - Current techniques and challenges». *Computers in Industry*, **63(2)**, pp. 91–97.

Dijkman, Remco M.; Dumas, Marlon y Ouyang, Chun (2008). «Semantics and analysis of business process models in BPMN». *Information and Software Technology*, **50(12)**, pp. 1281–1294.

Dumas, Marlon; der Aalst, Wil M.P. y ter Hofstede, Arthur H.M. (2005). *Process-aware information systems: bridging people and software through process technology*. John Wiley and Sons.

Eriksson, Hans-Erik y Penker, Magnus (2000). *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, Inc..

Erl, Thomas (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR.

Erl, Thomas (2007). *SOA: Principles of Service Design*. Prentice Hall.

Franke, Peter D. (2010). *Vendor-managed inventory for high value parts: results from a survey among leading international manufacturing firms*. volumen 3. Univ.-Verl. der TU.

Gherbi, Tahar; Meslati, Djamel y Borne, Isabelle (2009). «MDE between Promises and Challenges». En: *UKSim 2009: 11th International Conference on Computer Modelling and Simulation*, pp. 152–155. IEEE Computer Society.

Gröner, Gerd; Bošković, Marko; Parreiras, Fernando Silva y Gašević, Dragan (2012). «Modeling and validation of business process families». *Information Systems*. In Press.

Gschwind, Thomas; Koehler, Jana y Wong, Janette (2008). «Applying Patterns during Business Process Modeling». En: Marlon Dumas; Manfred Reichert y Ming-Chien Shan (Eds.), *Business Process Management*, volumen 5240 de *Lecture Notes in Computer Science*, pp. 4–19. Springer.

Hille-Doering, Reiner (2010). «BPMN 2.0 File Formats: With a Transformation to a Transformation». *Informe técnico 27 Octubre 2010*, SAP Community Network.
`http://scn.sap.com/docs/DOC-3076`

Hofreiter, Birgit (2008). «Registering UML Models for Global and Local Choreographies». En: *International Conference on Electronic Commerce*, ACM.

Hofreiter, Birgit (2009). «Extending UN/CEFACT's modeling methodology by a UML profile for local choreographies». *Inf. Syst. E-Business Management*, **7(2)**, pp. 251–271.

Huemer, Christian; Liegl, Philipp; Motal, Thomas; Schuster, Rainer y Zapletal, Marco (2008). «The development process of the UN/CEFACT modeling methodology». En: *International Conference on Electronic Commerce*, volumen 342 de *ICEC '08*. ACM.

Indulska, Marta; Green, Peter; Recker, Jan y Rosemann, Michael (2009a). «Business Process Modeling: Perceived Benefits». En: Alberto Laender; Silvana Castano; Umeshwar Dayal; Fabio Casati y José de Oliveira (Eds.), *Conceptual Modeling - ER 2009*, volumen 5829 de *Lecture Notes in Computer Science*, pp. 458–471. Springer Berlin / Heidelberg.

Indulska, Marta; Recker, Jan; Rosemann, Michael y Green, Peter (2009b). «Business Process Modeling: Current Issues and Future Challenges». En: Pascal van Eck; Jaap Gordijn y Roel Wieringa (Eds.), *Advanced Information Systems Engineering*, volumen 5565 de *Lecture Notes in Computer Science*, pp. 501–514. Springer Berlin / Heidelberg.

Jiménez, Guillermo; Galeano, Nathalie; Nájera, Teresa; Aguirre, José; Rodríguez, Ciprian y Molina, Arturo (2005). «Methodology for business model definition of collaborative networked organizations». En: Luis

Camarinha-Matos; Hamideh Afsarmanesh y Angel Ortiz (Eds.), *Collaborative Networks and Their Breeding Environments*, volumen 186 de *IFIP International Federation for Information Processing*, pp. 347–354. Springer.

Jouault, Frédéric; Allilaire, Freddy; Bézivin, Jean y Kurtev, Ivan (2008). «ATL: A model transformation tool». *Science of Computer Programming*, **72(1-2)**, pp. 31–39.

Jouault, Frédéric y Kurtev, Ivan (2006). «Transforming Models with ATL». En: Jean-Michel Bruel (Ed.), *Satellite Events at the MoDELS 2005 Conference*, volumen 3844 de *Lecture Notes in Computer Science*, pp. 128–138.

Karagiannis, Dimitris (1995). «BPMS: business process management systems». *ACM SIGOIS Bulletin*, **16(1)**, pp. 10–13.

Kurtev, Ivan; van den Berg, Klaas y Jouault, Frédéric (2007). «Rule-based modularization in model transformation languages illustrated with ATL». *Science of Computer Programming*, **68(3)**, pp. 138–154.

La Rosa, Marcello; Aalst, WilM.P.; Dumas, Marlon y ter Hofstede, ArthurH.M. (2009). «Questionnaire-based variability modeling for system configuration». *Software & Systems Modeling*, **8**, pp. 251–274.

La Rosa, Marcello; Dumas, Marlon; Hofstede, Arthur H.M ter y Mendling, Jan (2011a). «Configurable Multi-Perspective Business Process Models». *Information Systems*, **36(2)**, pp. 313–340.

La Rosa, Marcello; Dumas, Marlon; Uba, Reina y Dijkman, Remco M. (2012). «Business Process Model Merging : An Approach to Business Process Consolidation». *ACM Transactions on Software Engineering and Methodology (TOSEM)*. In Press.

La Rosa, Marcello; Lux, Johannes; Seidel, Stefan; Dumas, Marlon y Hofstede, Arthur H.M. ter (2007). «Questionnaire-driven Configuration of Reference Process Models». En: John Krogstie; Andreas Opdahl y Guttorm Sindre (Eds.), *Advanced Information Systems Engineering*, volumen 4495 de *Lecture Notes in Computer Science*, pp. 424–438. Springer Berlin Heidelberg.

La Rosa, Marcello; Reijers, Hajo A.; der Aalst, Wil M.P.; Dijkman, Remco M.; Mendling, Jan; Dumas, Marlon y García-Bañuelos, Luciano (2011b). «APROMORE: An advanced process model repository». En: *Expert Systems with Applications (ESWA)*, volumen 38, pp. 7029–7040.

Lazarte, Ivanna M.; Chiotti, Omar y Villarreal, Pablo D. (2009). «Transforming Collaborative Process Models into Interface Process Models by Applying an MDA Approach». En: Claude Godart; Norbert Gronau; Sushil Sharma y Gérôme Canals (Eds.), *Software Services for e-Business and e-Society*, volumen 305 de *IFIP Advances in Information and Communication Technology*, pp. 301–315. Springer.

Lazarte, Ivanna M.; Tello-Leal, Edgar; Roa, Jorge; Chiotti, Omar y Villarreal, Pablo D. (2010). «Model-Driven Development Methodology for B2B Collaborations». En: *Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pp. 69–78. IEEE Computer Society.

Lazarte, Ivanna M.; Thom, Lucinéia Heloisa; Iochpe, Cirano; Chiotti, Omar y Villarreal, Pablo D. (2013). «A distributed repository for managing business process models in cross-organizational collaborations». *Computers in Industry*, **64**, pp. 252–267.

Lazarte, Ivanna M.; Villarreal, Pablo D.; Chiotti, Omar; Thom, Lucinéia H. y Iochpe, Cirano (2011). «An MDA-based Method for Designing Integration Process Models in B2B Collaborations». En: *International Conference on Enterprise Information Systems (ICEIS)*, volumen 3, pp. 55–65. INSTICC, SciTePress.

Li, Qing; Zhou, Jian; Peng, Qi-Rui; Li, Can-Qiang; Wang, Cheng; Wu, Jing y Shao, Bei-En (2010). «Business processes oriented heterogeneous systems integration platform for networked enterprises». *Computers in Industry*, **61(2)**, pp. 127–144.

Lu, Ruopeng y Sadiq, Shazia (2007). «A Survey of Comparative Business Process Modeling Approaches». En: Witold Abramowicz (Ed.), *Business Information Systems*, volumen 4439 de *Lecture Notes in Computer Science*, pp. 82–94. Springer Berlin Heidelberg.

Ma, Zhilei; Wetzstein, Branimir; Anicic, Darko; Heymans, Stijn y Leymann, Frank (2007). «Semantic business process repository». En: *Semantic Business Process Management Workshop (SBPM 2007)*, volumen 251, pp. 92–100. CEUR-WS.org.

Martens, Axel (2005). «Consistency between Executable and Abstract Processes». En: *International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005)*, pp. 60–67. IEEE Computer Society.

Martens, Axel; Moser, Simon; Gerhardt, Achim y Funk, Karoline (2006). «Analyzing Compatibility of BPEL Processes». En: *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, pp. 147–147. IEEE.

Mayer, Philip; Schroeder, Andreas y Koch, Nora (2008). «MDD4SOA: Model-Driven Service Orchestration». En: *International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008)*, pp. 203–212. IEEE Computer Society.

Mendling, Jan; Neumann, Gustaf y Aalst, Wil M.P. der (2007). «Understanding the Occurrence of Errors in Process Models Based on Metrics». En: Robert Meersman y Zahir Tari (Eds.), *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volumen 4803 de *Lecture Notes in Computer Science*, pp. 113–130. Springer.

Mendling, Jan; Reijers, Hajo A. y Aalst, Wil M.P. der (2010a). «Seven process modeling guidelines (7PMG)». *Information and Software Technology*, **52(2)**, pp. 127–136.

Mendling, Jan; Reijers, Hajo A. y Recker, Jan (2010b). «Activity labeling in process modeling: empirical insights and recommendations». *Information Systems*, **35(4)**, pp. 467–482.

Nikiforova, Oksana (2009). «Two Hemisphere Model Driven Approach for Generation of UML Class Diagram in the Context of MDA». *e-Informatica Software Engineering Journal*, **3(1)**, pp. 59–72.

OAGi (2006). «Universal Business Language (UBL), Version 2.0». `http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html`

OAGI (2009). «Business Object Document (BOD) Message Architecture, Version 9.0 ».
http://www.oagi.org/oagis/9.0/Documentation/Architecture.html

OASIS (2004). «Universal Description Discovery & Integration (UDDI), Version 3.0.2».
http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

OASIS (2007). «Web Services Business Process Execution Language (WSBPEL), Version 2.0».
http://www.oasis-open.org/committees/download.php/23964/-wsbpel-v2.0-primer.htm

OMG (2003a). «Common Warehouse Metamodel (CWM): Specification, Version 1.1».
http://www.omg.org/spec/CWM/1.1/

OMG (2003b). «Model-Driven Architecture (MDA) Guide, Version 1.0.1».
http://www.omg.org/cgi-bin/doc?omg/03-06-01

OMG (2008). «MOF Model to Text Transformation Language (MOFM2T), Version 1.0».
http://www.omg.org/spec/MOFM2T/1.0/

OMG (2011a). «Business Process Model and Notation (BPMN), Version 2.0».
http://www.omg.org/spec/BPMN/2.0/

OMG (2011b). «Meta Object Facility (MOF): Core Specification, Version 2.4.1».
http://www.omg.org/spec/MOF/2.4.1/

OMG (2011c). «Query/View/Transformation Specification (QVT), Version 1.1».
http://www.omg.org/spec/QVT/1.1

OMG (2011d). «Unified Modeling Language (UML): Infrastructure, Version 2.4.1».
http://www.omg.org/spec/UML/2.4.1/

OMG (2011e). «XMI Mapping Specification».
http://www.omg.org/spec/XMI/2.4.1/

OMG (2012). «Object Constraint Language (OCL)».
`http://http://www.omg.org/spec/OCL/2.3.1/`

Oracle (2013). «MySQL».
`http://dev.mysql.com/`

Ouyang, Chun; Dumas, Marlon; Aalst, Wil M.P. der; Hofstede, Arthur H.M. ter y Mendling, Jan (2009). «From business process models to process-oriented software systems». *ACM transactions on software engineering and methodology (TOSEM)*, **19(1)**, pp. 2:1–2:37.

Papazoglou, Mike P. (2003). «Service-Oriented Computing: Concepts, Characteristics and Directions». En: *International Conference on Web Information Systems Engineering (WISE 2003)*, pp. 3–12. IEEE Computer Society.

Papazoglou, Mike P. y Heuvel, Willem-Jan (2007). «Service oriented architectures: approaches, technologies and research issues». *The VLDB Journal*, **16**, pp. 389–415.

Pokahr, Alexander; Braubach, Lars y Lamersdorf, Winfried (2005). «Jadex: A BDI Reasoning Engine». En: RafaelH. Bordini; Mehdi Dastani; Jürgen Dix y Amal Fallah Seghrouchni (Eds.), *Multi-Agent Programming*, volumen 15 de *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pp. 149–174. Springer.

Pons, Claudia; Giandini, Roxana y Pérez, Gabriela (2010). *Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica*. EDULP & McGraw-Hill Educación.

Reijers, Hajo; van Wijk, Sander; Mutschler, Bela y Leurs, Maarten (2010). «BPM in Practice: Who Is Doing What?» En: Richard Hull; Jan Mendling y Stefan Tai (Eds.), *Business Process Management*, volumen 6336 de *Lecture Notes in Computer Science*, pp. 45–60. Springer.

Roa, Jorge; Chiotti, Omar y Villarreal, Pablo D. (2012). «A Verification Method for Collaborative Business Processes». En: Florian Daniel; Kamel Barkaoui y Schahram Dustdar (Eds.), *Business Process Management Workshops*, volumen 99 de *Lecture Notes in Business Information Processing*, pp. 293–305. Springer.

Roser, Stephan y Bauer, Bernhard (2005). «A Categorization of Collaborative Business Process Modeling Techniques». En: *International Conference on E-Commerce Technology Workshops*, pp. 43–51. IEEE Computer Society.

Roser, Stephan; Bauer, Bernhard y Müller, Jörg P. (2006). «Model- and architecture-driven development in the context of cross-enterprise business process engineering». En: *IEEE International Conference on Services Computing (SCC'06)*, pp. 119–126.

Roser, Stephan; Müller, Jörg P. y Bauer, Bernhard (2011). «An evaluation and decision method for ICT architectures for cross-organizational business process coordination». *Information Systems and E-Business Management*, **9(1)**, pp. 51–88.

Russell, Nick; Hofstede, Arthur H.M. ter; Edmond, David y Aalst, Wil M.P. der (2005). «Workflow Data Patterns: Identification, Representation and Tool Support». En: Lois Delcambre; Christian Kop; HeinrichC. Mayr; John Mylopoulos y Oscar Pastor (Eds.), *Conceptual Modeling ER 2005*, volumen 3716 de *Lecture Notes in Computer Science*, pp. 353–368. Springer Berlin Heidelberg.

Selic, Bran (2003). «The Pragmatics of Model-Driven Development». *IEEE Software*, **20(5)**, pp. 19–25.

Selic, Bran (2006). «Model-Driven Development: Its Essence and Opportunities». En: *Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006)*, pp. 313–319. IEEE Computer Society.

Sánchez Cuadrado, Jesús y García Molina, Jesús (2008). «Approaches for Model Transformation Reuse: Factorization and Composition». En: Antonio Vallecillo; Jeff Gray y Alfonso Pierantonio (Eds.), *Theory and Practice of Model Transformations*, volumen 5063 de *Lecture Notes in Computer Science*, pp. 168–182. Springer Berlin Heidelberg.

Steinberg, Dave; Budinsky, Frank; Paternostro, Marcelo y Merks, Ed (2008). *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2ª edición.

Tello-Leal, Edgar (2012). *Agentes de Software para la Gestión de Colaboraciones Inter-Organizacionales Dinámicas*. Tesis doctoral, Universidad Tecnológica Nacional - Facultad Regional Santa Fe.

Tello-Leal, Edgar; Chiotti, Omar y Villarreal, Pablo D. (2011). «Agents for Managing Business-to-Business Interactions». En: *International Conference on Agents and Artificial Intelligence (ICAART 2011)*, pp. 238–244.

The Eclipse Foundation (2011). «BPMN 2.0 Modeler». `http://www.eclipse.org/proposals/soa.bpmn2-modeler/`

Theling, Thomas; Zwicker, Jörg; Loos, Peter y Vanderhaeghen, Dominik (2005). «An Architecture for Collaborative Scenarios Applying a Common BPMN-Repository». En: Lea Kutvonen y Nancy Alonistioti (Eds.), *Distributed Applications and Interoperable Systems*, volumen 3543 de *Lecture Notes in Computer Science*, pp. 1073–1075. Springer.

Thom, Lucinéia; Reichert, Manfred y Iochpe, Cirano (2009). «Activity patterns in process-aware information systems: basic concepts and empirical evidence». En: *International Journal of Business Process Integration and Management*, volumen 4, pp. 93–110. Inderscience.

Thom, Lucinéia Heloisa; Lazarte, Ivanna M.; Iochpe, Cirano; Priego, Luz-Maria; Verdier, Christine; Chiotti, Omar y Villarreal, Pablo D. (2011). «On the Capabilities of BPMN for Workflow Activity Patterns Representation». En: Remco Dijkman; Jörg Hofstetter y Jana Koehler (Eds.), *Business Process Model and Notation*, volumen 95 de *Lecture Notes in Business Information Processing*, pp. 172–177. Springer.

Vanhatalo, Jussi; Koehler, Jana y Leymann, Frank (2006). «Repository for business processes and arbitrary associated metadata». *International Conference on Business Process Management, (BPM 2006)*, pp. 25–31.

VICS (2004). «Collaborative Planning, Forecasting and Replenishment (CPFR)». `http://www.vics.org/committees/cpfr/`

Villarreal, Pablo D.; Caliusco, Ma. Laura; Galli, Ma. Rosa; Salomone, Enrique y Chiotti, Omar (2003). «Decentralized Process Management for Inter-Enterprise Collaboration». En: B. S. Sahay (Ed.), *Emerging Issues in Supply Chain Management*, pp. 293–310. MacMillan India Limited.

Villarreal, Pablo D.; Lazarte, Ivanna M.; Roa, Jorge y Chiotti, Omar (2010). «A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language». En: Stefanie Rinderle-Ma; Shazia Sadiq y Frank Leymann (Eds.), *Business Process Management Workshops*, volumen 43 de *Lecture Notes in Business Information Processing*, pp. 318–329. Springer.

Villarreal, Pablo D.; Roa, Jorge; Salomone, Enrique y Chiotti, Omar (2007a). «Verification of Models in a MDA Approach for Collaborative Business Processes». *X Conferencia Iberoamericana de Software Engineering (CIbSE 2007)*, pp. 179–192.

Villarreal, Pablo D.; Salomone, Enrique y Chiotti, Omar (2006a). «MDA Approach for Collaborative Business Processes: Generating Technological Solutions based on Web Services Composition». En: *IX Conferencia Iberoamericana de Software Engineering (CIbSE 2006)*, pp. 287–300.

Villarreal, Pablo D.; Salomone, Enrique y Chiotti, Omar (2006b). «Transforming Collaborative Business Process Models into Web Services Choreography Specifications». En: Juhnyoung Lee; Junho Shim; Sang-goo Lee; Christoph Bussler y Simon Shim (Eds.), *Data Engineering Issues in E-Commerce and Services*, volumen 4055 de *Lecture Notes in Computer Science*, pp. 50–65. Springer.

Villarreal, Pablo D.; Salomone, Enrique y Chiotti, Omar (2007b). «Modeling and specifications of collaborative business processes using a MDA approach and a UML profile». En: Peter Rittgen (Ed.), *Enterprise modeling and computing with UML*, capítulo 2, pp. 13–45. Idea Group Inc.

W3C (2004). «Web Services Architecture (WSA)».
`http://www.w3.org/TR/ws-arch/`

W3C (2007a). «Simple Object Access Protocol (SOAP), Version 1.2. Part 1: Messaging Framework. Second Edition».
`http://www.w3.org/TR/soap12-part1/`

W3C (2007b). «Web Services Description Language (WSDL), Version 2.0. Part 1: Core Language».
`http://www.w3.org/TR/wsdl20/`

W3C (2007c). «XSL Transformations (XSLT), Version 2.0».
`http://www.w3.org/TR/xslt20/`

W3C (2008). «Extensible Markup Language (XML), Version 1.0. Fifth Edition».
`http://www.w3.org/TR/xml/`

Wagelaar, Dennis; Straeten, Ragnhild Van Der y Deridder, Dirk (2010). «Module superimposition: a composition technique for rule-based model transformation languages». *Software and System Modeling*, **9(3)**, pp. 285–309.

Walls, Craig (2011). *Spring in Action*. Manning Publications Co., 3ª edición.

Weske, Mathias (2007). *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York Inc.

Wimmer, Manuel; Perez, Salvador Martínez; Jouault, Frédéric y Cabot, Jordi (2012). «A Catalogue of Refactorings for Model-to-Model Transformations». *Journal of Object Technology*, **11(2)**, pp. 2: 1–40.

Xu, Lai; de Vrieze, Paul; Bouguettaya, Athman; Liang, Peng; Phalp, Keith y Jeary, Sherry (2011). «Service-Oriented Collaborative Business Processes». En: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, capítulo 5, pp. 116–132. IGI Global.

Yan, Zhiqiang; Dijkman, Remco y Grefen, Paul (2012). «Business process model repositories - Framework and survey». *Information and Software Technology*, **54(4)**, pp. 380–395.

Zaha, Johannes Maria; Dumas, Marlon; ter Hofstede, Arthur H.M.; Barros, Alistair y Decker, Gero (2006). «Service Interaction Modeling: Bridging Global and Local Views». En: *International Enterprise Distributed Object Computing Conference*, pp. 45–55. IEEE Computer Society.

Zhang, Liang-Jie; Zhang, Jia y Cai, Hong (2007). *Services Computing*. Springer.

Zinnikus, Ingo; Hahn, Christian y Fischer, Klaus (2008). «A model-driven, agent-based approach for the integration of services into a collaborative business process». En: Lin Padgham; David C. Parkes; Jörg P. Müller y Simon Parsons (Eds.), *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, volumen 1, pp. 241–248. IFAAMAS.

zur Muehlen, Michael y Indulska, Marta (2010). «Modeling languages for business processes and business rules: A representational analysis». *Information Systems*, **35(4)**, pp. 379–390.