# METODOLOGÍA BASADA EN EL DESARROLLO DIRIGIDO POR MODELOS PARA LA EJECUCIÓN DE PROCESOS COLABORATIVOS MEDIANTE AGENTES DE SOFTWARE

EDGAR TELLO LEAL
ANA BERTHA RÍOS ALVARADO
IVÁN LÓPEZ ARÉVALO
OMAR CHIOTTI
PABLO DAVID VILLAREAL

La GENERACIÓN del CONOCIMIENTO con VALORES

# Metodología basada en el desarrollo dirigido por modelos para la ejecución de procesos colaborativos mediante agentes de software

Tamaulipas
GOBIERNO DEL ESTADO

Consejo de publicaciones UAT

PLAZA Y VALDES
PYV
EDITORES


C. P. Enrique C. Etienne Pérez del Río
Presidente

Dr. José Luis Pariente Fragoso
Vicepresidente

Dr. Héctor Capello García
Secretario técnico

C. P. Guillermo Mendoza Cavazos
Vocal

Dr. Marco Aurelio Navarro Leal
Vocal

Mtro. Luis Alonso Sánchez Fernández
Vocal

Mtro. José David Vallejo Manzur
Vocal

# Metodología basada en el desarrollo dirigido por modelos para la ejecución de procesos colaborativos mediante agentes de software

Edgar Tello Leal, Ana Bertha Ríos Alvarado,
Iván López Arévalo, Omar Chiotti,
Pablo David Villarreal

Tamaulipas
GOBIERNO DEL ESTADO

Consejo de publicaciones UAT

PLAZA Y VALDES
PYV
EDITORES

Primera edición, 2015
Edgar Tello Leal, Ana Bertha Ríos Alvarado, Iván López Arévalo, Omar Chiotti, Pablo David Villarreal

Metodología basada en el desarrollo dirigido por modelos para la ejecución de procesos colaborativos mediante agentes de software / Edgar Tello Leal… et al. – México: UAT ; Plaza y Valdés 2015

pp. 214; 17 x 23 cm

1. Alianzas estratégicas 2. Relaciones interorganizacionales
3. Sistemas de información en administración I. Tello Leal, Edgar, aut.

LC (HD69.S8 M47) Dewey 658.044

Metodología basada en el desarrollo dirigido por modelos para la ejecución de procesos colaborativos mediante agentes de software
ISBN 978-607-7654-78-0

Serie
La Generación del Conocimiento con Valores

Diseño de serie
Universidad Autónoma de Tamaulipas

Diseño de portada
César Susano

D. R. © 2015, Universidad Autónoma de Tamaulipas

Matamoros, s. n., Zona Centro, Ciudad Victoria, C. P. 87000, México, Tamaulipas

Ediciones UAT
Tel. (52) 834 3181-800, ext. 1140
www.libros.uat.edu.mx

Plaza y Valdés, S. A. de C. V.
Manuel María Contreras 73, Colonia San Rafael
México, D. F., 06470 Tel. (55) 5097-2070
editorial@plazayvaldes.com • www.plazayvaldes.com

Se prohíbe la reproducción total o parcial de esta obra
—incluido el diseño tipográfico y de portada—,
sea cual fuere el medio, electrónico o mecánico,
sin el consentimiento por escrito del Consejo de Publicaciones UAT

Una Edición del Departamento de Fomento Editorial
de la Universidad Autónoma de Tamaulipas

Fomento Editorial

# ÍNDICE

## PARTE II
## METODOLOGÍA E IMPLEMENTACIÓN

### 3. Metodología de desarrollo de soluciones tecnológicas basadas en agentes de software



### 4. Plataforma de gestión de colaboraciones interorganizacionales dinámicas

*A Gary, Clau y Gegy*

# Prefacio

La globalización, los mercados modernos, las nuevas filosofías de gestión de organizaciones y los avances en las tecnologías de la información y la comunicación (TIC), alientan a las organizaciones a establecer colaboraciones interorganizacionales o redes de colaboración. Las colaboraciones interorganizacionales consisten en relaciones entre organizaciones por un periodo determinado para alcanzar objetivos comunes.

Una colaboración interorganizacional implica una integración orientada a procesos entre organizaciones heterogéneas y autónomas, que debe ser alcanzada tanto a nivel organizacional como tecnológico.

A nivel organizacional, las partes se centran en el diseño de procesos (de negocio) colaborativos para acordar el comportamiento de la colaboración. En este nivel, la integración y colaboración se alcanza a través de la definición y ejecución de los procesos colaborativos. Un *proceso colaborativo* define una vista global del comportamiento de las interacciones entre las organizaciones para alcanzar las metas de negocio comunes. Para mantener la autonomía de las organizaciones se requiere de una gestión descentralizada de los procesos colaborativos. Ésta se puede alcanzar a través de la ejecución distribuida y sincronizada de los procesos (de negocio) de integración de las organizaciones involucradas. Un *proceso de integración* especifica el comportamiento público y privado que soporta el rol que una organización cumple en un proceso colaborativo, contiene la lógica pública y privada requerida para procesar o generar el intercambio de información entre las organizaciones.

A nivel tecnológico, la solución se centra en la funcionalidad y la interoperabilidad de los sistemas de información de las organizaciones para ejecutar los procesos colaborativos. Esto implica desarrollar e implementar sistemas de información autónomos que sean interoperables y que lleven a cabo la ejecución de los procesos de integración para dar soporte a la gestión descentralizada de los procesos colaborativos.

El enfoque tradicional de gestionar colaboraciones interorganizacionales basadas en acuerdos estáticos que predefinen los procesos colaborativos a ejecutar conlleva costos y tiempos de desarrollo elevados para la implementación y/o adaptación de sistemas de información interorganizacionales ante cambios de requerimientos de negocio en los procesos colaborativos.

Para responder a las nuevas oportunidades de negocio o adaptarse a los frecuentes cambios del entorno, las organizaciones necesitan entablar *colaboraciones interorganizacionales dinámicas*, lo cual implica mediante el uso de TIC dos o más organizaciones pueden acordar llevar a cabo en forma electrónica una colaboración interorganizacional definiendo los procesos colaborativos a ser ejecutados. Ante una colaboración nueva o una ya establecida, las organizaciones acuerdan ejecutar nuevos procesos o nuevas versiones

de los procesos existentes. Lo anterior requiere de nuevas metodologías y métodos de desarrollo de software, arquitecturas de software y de la infraestructura o plataforma que brinde soporte a los diferentes aspectos y requerimientos de las colaboraciones interorganizacionales dinámicas. Por lo tanto, en este trabajo de investigación se propone una solución tecnológica basada en agentes de software para dar soporte a la gestión de procesos colaborativos e interorganizacionales dinámicos. Para ello se propone una plataforma de agentes de software que posibilite que una organización pueda acordar en forma electrónica con otras organizaciones la implementación y ejecución de procesos colaborativos. La plataforma está integrada por agentes estáticos, cuyo comportamiento está predefinido, y agentes dinámicos, cuyo comportamiento es generado en tiempo de ejecución del sistema. Los agentes estáticos posibilitan a las organizaciones entablar y gestionar colaboraciones dinámicas en forma electrónica. Los agentes dinámicos dan soporte a la gestión de procesos colaborativos que se acuerdan y definen ejecutarla durante el transcurso de la colaboración. Los agentes dinámicos son generados en el tiempo que se ejecuta el acuerdo de colaboración a partir de los modelos conceptuales realizados para el proceso.

Los agentes que conforman la plataforma son especificados siguiendo el modelo de agente *Belief-Desire-Intention* (BDI). Los agentes dinámicos se caracterizan por su enfoque orientado a procesos, en donde su comportamiento es gobernado por un modelo de proceso ejecutable mediante un motor de procesos. Además, la plataforma propuesta contiene los mecanismos requeridos para iniciar e incorporar a los agentes en los procesos dinámicos en tiempo de ejecución del sistema. De esta manera, la ejecución de un proceso colaborativo es realizada en forma descentralizada.

Con el propósito de guiar el proceso de desarrollo para generar soluciones tecnológicas (esto es, los agentes orientados a procesos dinámicos) a partir de modelos conceptuales de procesos colaborativos, se propone una metodología que sigue un enfoque *top-down* basada en los principios del desarrollo dirigido por modelos. Esta metodología permite generar para cada organización involucrada en un proceso colaborativo un modelo de proceso ejecutable que implementa un proceso de integración y el código del agente orientado a procesos dinámicos. Éstos en conjunto representan los artefactos de implementación requeridos para ejecutar un proceso colaborativo.

Para satisfacer los requerimientos de las colaboraciones interorganizacionales dinámicas, la metodología se implementa y automatiza mediante un agente estático provisto en la plataforma, el cual genera en forma automática los artefactos de implementación, posteriormente al acuerdo de colaboración en forma electrónica. De esta manera, en tiempo de ejecución de la plataforma se posibilita generar en forma automática la solución tecnológica basada en agentes de software que permiten ejecutar procesos colaborativos.

Por lo tanto, mediante la plataforma y la metodología de desarrollo propuestos se posibilita a las organizaciones gestionar colaboraciones interorganizacionales dinámicas, adaptarse automáticamente a cambios en los requerimientos de las organizaciones

o al ambiente en el que se desenvuelven, reusar componentes y proveer funcionalidades para ejecutar nuevos procesos colaborativos acordados electrónicamente o nuevas versiones de procesos colaborativos.

Edgar Tello Leal, Ana Bertha Ríos Alvarado, Iván López Arévalo, Omar Chiotti, Pablo David Villarreal

# 1. Introducción

En este capítulo se describe el contexto en que se enmarca el presente libro (sección 1.1), detallando los conceptos del dominio de colaboraciones interorganizacionales dinámicas y ejecución de procesos de negocio colaborativos. También se exponen los alcances del trabajo de investigación definiendo los problemas a resolver, se presentan las hipótesis de trabajo y se describen los objetivos de la presente investigación (sección 1.2). Además, se muestran las principales contribuciones alcanzadas en el libro (sección 1.3). Finalmente, se presenta la organización de los capítulos contenidos en el libro (sección 1.4).

## 1.1. El contexto: gestión de colaboraciones interorganizacionales dinámicas

Como resultado de la globalización, las nuevas oportunidades de mercado y los avances en las tecnologías de la información y la comunicación (TIC), las organizaciones apuntan a establecer relaciones estrechas de integración, cooperación y colaboraciones entre ellas, dando lugar a las colaboraciones interorganizacionales o redes de colaboración (20). Las colaboraciones interorganizacionales consisten en organizaciones que se relacionan por un periodo de tiempo determinado para alcanzar objetivos comunes.

La colaboración entre organizaciones ha demostrado traer beneficios para sus participantes, tales como disminución de costos, mejoras de rendimiento, manejo eficiente de información en tiempo real, etc. Una colaboración interorganizacional implica una integración orientada a procesos entre organizaciones heterogéneas y autónomas, que debe ser alcanzada tanto a nivel organizacional (también llamado nivel de negocio) como tecnológico (90, 91).

A nivel organizacional, la solución consiste en aplicar un modelo de gestión que defina las reglas generales que gobiernan la colaboración entre organizaciones. En este nivel, la integración y colaboración entre organizaciones se alcanza a través de la definición y ejecución de los procesos de negocio colaborativos. Un *proceso de negocio colaborativo* (ahora referido como *proceso colaborativo*), también llamado *coreografía de proceso* (91), define el comportamiento de las interacciones entre las organizaciones y sus roles desde un punto de vista global, esto es, cómo coordinan sus acciones e intercambian información con el propósito de tomar decisiones en forma conjunta para alcanzar una meta de negocio común (90, 72).

Una gestión descentralizada de los procesos colaborativos es requerida para mantener la autonomía de las organizaciones (90), de tal forma que cada organización gobierne y gestione en forma independiente sus procesos privados y la información que produce para dar soporte a los procesos colaborativos. La ejecución descentralizada

de un proceso colaborativo se puede alcanzar a través de la ejecución de los procesos de integración de las organizaciones involucradas (49). Un *proceso (de negocio) de integración*, también llamado *proceso privado* (57), *proceso de orquestación* (91) o *proceso ejecutable* (72), define tanto el comportamiento público como el comportamiento privado que compone el rol que una organización cumple en un proceso colaborativo; contiene la lógica pública y privada requerida para procesar o generar el intercambio de información entre las organizaciones (49). Por lo tanto, un proceso colaborativo define lo que las organizaciones acuerdan realizar y establece un contrato del comportamiento de la colaboración, mientras que un proceso de integración define qué debe realizar una organización para cumplir con lo definido en el proceso colaborativo. Un proceso colaborativo es un proceso abstracto, es decir, no es ejecutable directamente sino a través de la ejecución distribuida y descentralizada de los correspondientes procesos de integración de las organizaciones.

A nivel tecnológico, la solución se centra en la funcionalidad y la interoperabilidad de los sistemas de información interorganizacionales para automatizar y ejecutar los procesos colaborativos. Esto implica a las organizaciones desarrollar e implementar s sistemas de información autónomos que sean interoperables y lleven a cabo la ejecución de los procesos de integración, posibilitando el intercambio de información para dar soporte a la gestión descentralizada de los procesos colaborativos (90). Esta interoperabilidad de sistemas puede ser alcanzada a través de la definición de modelos o especificaciones ejecutables tanto de las interfaces de los sistemas como de los procesos de integración de las organizaciones, basadas en estándares de facto o definidos *ad hoc*, que luego puedan ser interpretadas por *sistemas de información orientados a procesos* (SIOPs, o *process-aware information systems*) (23). Un SIOP ejecuta procesos en los que están involucradas personas, aplicaciones y fuentes de información, sobre la base de modelos o especificaciones ejecutables de procesos que son interpretadas por un motor de procesos.

Existen diferentes tipos de colaboraciones interorganizacionales, tales como empresa virtual (*virtual enterprise*), organización virtual (*virtual organization*), integración de cadenas de suministro, empresa extendida (*extended enterprise*) (19), entre otros. El enfoque tradicional en el armado de estas colaboraciones interorganizacionales ha sido realizar negociaciones y acuerdos cara a cara entre las partes para definir de antemano los procesos colaborativos a ejecutar. Estos acuerdos en general tienen una duración de mediano o largo plazo.

No obstante, para responder a las nuevas oportunidades de negocio o adaptarse a los frecuentes cambios que surgen en los ambientes en que están insertadas las organizaciones, se requiere entablar *colaboraciones interorganizacionales dinámicas*, con el propósito de facilitar acuerdos de corto plazo. Esto ha dado lugar a nuevos tipos de colaboraciones, tales como cadenas de suministros dinámicas y organizaciones virtuales dinámicas (19).

Una colaboración interorganizacional dinámica implica que a través del uso de TIC (80, 53, 30, 54) una organización puede entablar una relación de colaboración y acor-

dar en forma electrónica con otras organizaciones la ejecución de procesos colaborativos. Ante una nueva colaboración o una colaboración ya establecida, las organizaciones acuerdan ejecutar nuevos procesos o nuevas versiones de los procesos existentes para adaptarlos a nuevos requerimientos de negocio. Para ello, el acuerdo de cuáles procesos colaborativos ejecutar se realiza en forma electrónica a través de un proceso de negociación en donde modelos predefinidos son intercambiados y acordados entre las organizaciones. Estos modelos pueden ser obtenidos de repositorios públicos o privados o bien definidos *ad hoc* en forma conjunta por las organizaciones. Los procesos a ejecutar se seleccionan sobre la marcha de la colaboración. Además, las colaboraciones interorganizacionales dinámicas se caracterizan por realizar acuerdos entre organizaciones que se establecen, modifican y cancelan en periodos relativamente cortos (19, 53).

Los sistemas de información interorganizacionales posibilitan gobernar las transacciones y comunicaciones entre organizaciones, facilitando la implementación y realización de las colaboraciones interorganizacionales. La eficiencia y eficacia en la gestión entre organizaciones dependen de la capacidad de los sistemas de información interorganizacionales para soportar la constante evolución o cambio de los procesos de negocio involucrados. Los cambios a los procesos o nuevos procesos requieren que se modifiquen los sistemas que los soportan, esto es, que se realice un nuevo desarrollo para adaptar el sistema a los cambios de los procesos. Este nuevo desarrollo implica esfuerzos importantes de recursos humanos, tiempos y costos en que deben incurrir las organizaciones, para el rediseño, recodificación, recopilación y redespliegue de los sistemas interorganizacionales.

Para el desarrollo de estos sistemas se requiere de nuevas metodologías de desarrollo de software, arquitecturas de software y de la infraestructura adecuada que posibilite que se adapten rápidamente y den soporte a los cambios en los procesos de negocio que se definen en un nivel organizacional y que se implementan en un nivel tecnológico.

A continuación se describen las principales características de la gestión de colaboraciones interorganizacionales dinámicas a las que deberían dar soporte los sistemas de información interorganizacionales:

- **Gestión descentralizada de procesos colaborativos.** En la ejecución de los procesos colaborativos, cada organización participante gobierna el rol que desempeña en el proceso colaborativo (90). Como se mencionó anteriormente cada organización debe gestionar la ejecución de los procesos de integración que soportan el rol de la organización en los procesos colaborativos en los que está involucrada, a través de sus sistemas de información interorganizacionales.

- **Capacidad de adaptación a cambios.** Los procesos colaborativos son conducidos por la necesidad de la agilidad, adaptabilidad y flexibilidad del negocio o el entorno interorganizacional. Nuevos procesos requieren ser gestionados en nuevas o existentes relaciones de colaboración interorganizacionales, así como procesos

existentes requieren modificaciones durante el transcurso de una colaboración. Los cambios que se realizan en un proceso colaborativo requieren que se modifiquen los sistemas de información interorganizacionales que lo soportan, para adaptar el sistema a los cambios del proceso (100).

- **Soporte para negociaciones.** Para acordar en forma electrónica los modelos de procesos colaborativos a ser ejecutados se requiere de mecanismos de negociación electrónica que los soporten, así como la habilidad de los sistemas interorganizacionales para posibilitar la ejecución de los procesos colaborativos acordados.

- **Interoperabilidad.** Los sistemas de información interorganizacionales están conformados por componentes distribuidos, autónomos y heterogéneos que se despliegan en cada organización para gestionar en forma conjunta los procesos colaborativos con otras organizaciones (90). La integración de los mismos requiere que se logre la interoperabilidad en diferentes capas a través del uso de lenguajes comunes y/o estándares, como son: la capa de red con los protocolos de comunicación a usar (por ejemplo: HTTP o SOAP), la capa de información con los documentos a intercambiar basados en una sintaxis y semántica común, la capa de procesos de negocio con las especificaciones de las interfaces de los sistemas y de los procesos de integración que gobiernan el orden en que los sistemas se invocan e intercambian información.

- **Interacciones punto-a-punto** *(peer-to-peer).* Los sistemas de información interorganizacionales están basados en relaciones simétricas entre nodos (organizaciones) que requieren y proporcionan datos o servicios en un contexto distribuido, heterogéneo y descentralizado (2). Esto requiere que los sistemas de información interorganizacionales se comuniquen en forma directa sin la utilización de un sistema intermediario, mediante la especificación de mecanismos de interacción que habiliten el intercambio y uso de información entre nodos heterogéneos.

- **Autonomía de las organizaciones.** En las colaboraciones interorganizacionales dinámicas, las organizaciones se comportan como entidades autónomas, las cuales tienen la capacidad de decidir las condiciones de las interacciones, es decir, cuándo, cómo y con quién se integran y colaboran al mismo tiempo que protegen sus actividades y decisiones internas (12). Cada organización tiene plena autonomía para diseñar, implementar, ejecutar y supervisar sus procesos internos.

- **Alineación entre la solución interorganizacional o de negocio y la solución tecnológica.** El grado de alineación existente entre un proceso de negocio y los sistemas que lo soportan puede afectar fuertemente al rendimiento de la ejecución de los procesos de negocio (6). La alineación se puede describir

en dos niveles de abstracción (estratégico y funcional), que abarcan diferentes aspectos, tales como metas, entidades, estrategias, procesos, sistemas de información y datos (7). En nuestro caso, el interés radica en la relación existente entre la solución interorganizacional (procesos de negocio) y la solución tecnológica (sistemas de información) en un nivel funcional. La solución interorganizacional contiene los modelos conceptuales de procesos colaborativos y de integración, y la solución tecnológica contiene los modelos y código de implementación (de procesos y sistemas) en una tecnología específica. Por lo tanto, cualquier modificación realizada en la lógica o actividades de los procesos colaborativos o de integración debe ser plasmada en las funcionalidades de los sistemas de información interorganizacional, así como los cambios en estos sistemas pueden afectar los requerimientos definidos en la lógica y actividades de los procesos colaborativos, y por consiguiente, provocar una desalineación entre las soluciones interorganizacional y tecnológica.

## 1.2. Alcance de la investigación

La tecnología de agentes de software se ha aplicado a la integración de sistemas heterogéneos distribuidos, proporcionando un enfoque de integración funcional y promoviendo la inteligencia de negocios y la colaboración entre organizaciones. Esta tecnología permite una colaboración automática y dinámica, esencial en los sistemas de información interorganizacionales con transacciones complejas y en ambientes distribuidos (75), facilitando la implementación de arquitecturas distribuidas con alta capacidad de comunicación y negociación entre los componentes que la conforman (52).

Los agentes de software se caracterizan por la autonomía, interacción, reactividad, proactividad y movilidad, así como por sus características inherentes de comunicación, coordinación, cooperación y descentralización (39, 95, 11, 93), las cuales son funcionalidades deseables en la implementación de colaboraciones interorganizacionales dinámicas y en la ejecución de procesos de negocio colaborativos (90, 81, 84). Además, los sistemas basados en agentes de software son capaces de realizar acciones flexibles y autónomas en entornos dinámicos, impredecibles y abiertos. Por lo tanto, la tecnología de agentes de software tiene un alto potencial para soportar el modelado centrado en procesos de negocio de las organizaciones (55, 84).

Adicionalmente, la tecnología multiagente permite la construcción de sistemas que contienen múltiples componentes con intereses propios, a través de la definición de metas y el uso extensivo de la negociación entre componentes, lo cual permite a los agentes alcanzar resultados mutuamente aceptables. Esta combinación de funcionalidades hace que los sistemas multiagentes sean capaces de representar los objetivos o metas de cada organización a través de agentes individuales (74). Esto es muy conveniente para soportar la creación de organizaciones virtuales, coordinar el trabajo de los participan-

tes en una cadena de suministro, controlar procesos de *scheduling*, establecer acuerdos entre organizaciones que participan en colaboraciones interorganizacionales dinámicas y gestionar procesos de negocio colaborativos (53, 83, 84). Por lo cual, el uso de esta tecnología puede ser considerado como apropiado para el dominio de problemas en los que se enfoca esta obra.

El desarrollo dirigido por modelos (*Model-Driven Development*, MDD, por sus siglas en inglés) es un enfoque para la construcción de software que permite un desarrollo eficiente mediante el modelado en diferentes niveles de abstracción (45, 28). Los tiempos de desarrollo son mejorados a través de la automatización, donde el código ejecutable es generado a partir de modelos formales usando una o más etapas de transformación (77, 69). La transformación de modelos es la operación central en el desarrollo dirigido por modelos, ya que especifica la forma de producir una serie de modelos basados en un conjunto de modelos de origen. El uso de lenguajes formales de modelado y transformaciones automáticas permite mejorar la calidad de software, facilita el mantenimiento de los sistemas, previene la redundancia, y eleva el nivel de reutilización de modelos, especificaciones y componentes del software (16, 69).

El lenguaje UP-ColBPIP permite modelar procesos colaborativos mediante protocolos de interacción, posibilitando la definición del comportamiento y flujo de control de los procesos colaborativos (90, 88). Este lenguaje cumple con los requerimientos del dominio de colaboraciones interorganizacionales basadas en la integración de los procesos, tales como las interacciones *peer-to-peer*, la preservación de la autonomía de las organizaciones y el soporte a negociaciones complejas (87). Mediante el lenguaje BPMN (57) se pueden definir modelos conceptuales de procesos de integración (48).

Como resultado la definición de métodos de transformación basados en el desarrollo dirigido por modelos permite proveer un soporte automatizado para el diseño de modelos BPMN de procesos de integración, los cuales son requeridos por las organizaciones para implementar procesos colaborativos definidos con el lenguaje UP-ColBPIP (48). A través de métodos MDD es posible garantizar que los modelos de procesos de integración de las organizaciones sean interoperables y consistentes con los procesos colaborativos (48). Por *consistencia* se entiende que debe existir coherencia entre el comportamiento definido en un proceso de integración y su correspondiente proceso colaborativo (49). *Interoperabilidad* (también llamado compatibilidad, (91) se refiere a la capacidad de los procesos de integración de interactuar a través de un intercambio de mensajes sincronizados, de acuerdo al comportamiento definido en un proceso colaborativo.

Además, los modelos conceptuales de procesos pueden ser usados para derivar y diseñar sistemas de agentes en un nivel abstracto y transferir estos conceptos a implementaciones de aplicaciones basadas en agentes de software, utilizando un enfoque de desarrollo dirigido por modelos (32). Mediante métodos de transformación es posible generar automáticamente una implementación de un agente de software, como así también sus instancias, en una plataforma de agentes de software específica, en donde

los agentes de software generados son interoperables (43). La interoperabilidad a nivel tecnológico se refiere a la capacidad de los agentes de software para interactuar e intercambiar mensajes con otros agentes de forma sincronizada para la ejecución descentralizada de los procesos colaborativos.

### 1.2.1. Problemas a resolver

Como se mencionó antes, el enfoque tradicional para gestionar colaboraciones inteorganizacionales basadas en acuerdos estáticos, en las cuales se deciden los procesos colaborativos a ejecutar en forma manual o cara a cara entre las partes, conlleva costos y tiempos de desarrollo elevados para la implementación y/o adaptación de los sistemas de información interorganizacionales ante cambios de requerimientos en los procesos colaborativos. La disminución de los costos de transacción y la eficiencia y eficacia en la gestión de colaboraciones interorganizacionales depende de la capacidad de los sistemas de información interorganizacionales para soportar la constante evolución o cambio de los procesos. Los cambios en los procesos o nuevos procesos requieren que se modifiquen los sistemas que los soportan realizando un nuevo desarrollo para adaptar el sistema a los cambios.

La gestión de colaboraciones interorganizacionales dinámicas necesita de otro enfoque, que consista en nuevas plataformas de software y sistemas de información interorganizacionales flexibles y adaptables en tiempo de ejecución, que posibiliten a las organizaciones establecer acuerdos dinámicos en forma electrónica en colaboraciones interorganizacionales para ejecutar procesos colaborativos nuevos o rediseñados. Es así que se requiere de nuevas metodologías de desarrollo de software, arquitecturas de software y de la infraestructura que dé soporte a los diferentes aspectos y requerimientos de las colaboraciones interorganizacionales dinámicas.

Los sistemas de gestión de procesos de negocio actuales carecen de dichas funcionalidades, ya que las soluciones alcanzadas están basadas en coreografías y/o orquestaciones de servicios Web (75, 46, 36, 88) que no dan soporte a la gestión descentralizada y distribuida de procesos colaborativos. Por otra parte, si bien se encuentran propuestas para la gestión distribuida de procesos y *workflows* (17, 102, 44), éstas no toman en cuenta los requerimientos de las colaboraciones interorganizacionales dinámicas, tales como la negociación en forma electrónica de los procesos colaborativos y la posterior adaptación, en tiempo de ejecución, de los sistemas para ejecutar los correspondientes procesos de integración. Esta adaptación requerida puede ser alcanzada a través de una plataforma que permita, en tiempo de ejecución, la generación automática de modelos de procesos de integración ejecutables y de sistemas de información orientados a procesos que los interpreten.

## 1.3. Principales contribuciones

En la presente obra se muestra una plataforma flexible, adaptable y configurable basada en agentes de software para la gestión de colaboraciones interorganizacionales dinámicas y de los procesos colaborativos.

Se proponen agentes estáticos que representan a las organizaciones, cuyo comportamiento es definido en tiempo de diseño y están incorporados en la plataforma, los cuales dan soporte a procesos de negociación para entablar colaboraciones. A través de éstos las organizaciones negocian en forma electrónica los procesos colaborativos a ejecutar, a partir de modelos predefinidos de dichos procesos que son intercambiados entre las organizaciones participantes. Además se proveen agentes específicos para recuperar modelos de procesos desde repositorios y gestionarlos en la plataforma.

Se proponen agentes de software dinámicos orientados a procesos, los cuales tienen incluido un motor de procesos para ejecutar procesos de integración sobre la base de modelos de procesos BPMN ejecutables. El comportamiento, planes y acciones de estos agentes es gobernado por un modelo de procesos, siguiendo los conceptos de los sistemas de información orientados a procesos. Estos agentes dinámicos no están predefinidos en la plataforma sino que son generados en tiempo de ejecución, tanto su implementación como sus instancias.

Se contribuye con una metodología para el desarrollo de soluciones tecnológicas basadas en agentes de software para colaboraciones interorganizacionales dinámicas, la cual expresa claramente los pasos a seguir para generar un modelo de proceso ejecutable y el código de un agente de software dinámico. Dichos modelos constituyen la solución tecnológica generada mediante métodos basados en los conceptos del desarrollo dirigido por modelos. A través de los métodos MDD definidos en esta metodología se garantiza la alineación entre la solución interorganizacional y la solución tecnológica.

Se proporciona un método basado en MDD para generar un modelo de proceso de integración ejecutable, que es interpretado por los agentes dinámicos orientados a procesos a partir de un modelo de procesos de integración definido con el lenguaje BPMN.

Se proporciona también un conjunto de métodos basados en MDD para generar el código XML final de un agente de software dinámico que puede ser ejecutado e implementado en la plataforma propuesta. Un primer método genera un modelo del agente de software basado en el enfoque BDI (*Belief-Desire-Intention*), derivado de un modelo de un proceso de integración. Un segundo método genera un modelo específico de implementación que contiene el código del agente de software formado por metas, planes y eventos. Estos métodos MDD habilitan al comportamiento del agente de software dinámico para que sea generado a partir de los modelos de procesos de integración de acuerdo a los modelos de procesos colaborativos que son acordados por las partes.

Se proponen también agentes que generan la solución tecnológica en la plataforma para dar soporte a los procesos colaborativos que son acordados. Estos agentes implementan y automatizan la metodología mencionada anteriormente. Esto posibilita gene-

rar en la plataforma los artefactos de implementación y las instancias de los agentes de software dinámicos para dar soporte a la ejecución de nuevos procesos colaborativos o nuevas versiones de dichos procesos.

Finalmente se presentan casos de estudio mediante los cuales se realiza la validación de las hipótesis de trabajo. Se muestra un caso de estudio de colaboración interorganizacional dinámica para la integración de servicios de salud entre hospitales clínicos (*e-Healthcare*). Además, se detalla un caso de estudio de colaboración interorganizacional dinámica en el dominio de la industria de las telecomunicaciones (*Business-to-Business*). En ambos casos se implementó la plataforma basada en agentes de software para establecier colaboraciones interorganizacionales en forma dinámica, generando en tiempo de ejecución la solución tecnológica e instanciando a los agentes de software dinámicos que permiten la ejecución de los procesos colaborativos acordados por las organizaciones participantes.

El aporte global tiene un alcance amplio y significativo, debido al enfoque propuesto para la gestión de colaboraciones interorganizacionales dinámicas, y la ejecución de procesos de negocio colaborativos puede ser implementada en diferentes dominios, por ejemplo, entre industrias (*Business-to-Business*), entre organizaciones gubernamentales y privadas (*Goverment-to-Business*), entre organizaciones gubernamentales (*Goverment-to-Goverment*) o entre instituciones de atención a la salud (*e-Healthcare*).

## 1.4. Organización del libro

El capítulo 2 introduce el marco teórico en el que se encuadra el trabajo de investigación, presentando una revisión del estado de la gestión de colaboraciones interorganizacionales, métodos de desarrollo de software dirigidos por modelos y el paradigma de agentes de software. Asimismo, los principales trabajos relacionados son presentados.

El capítulo 3 describe una propuesta metodológica para el desarrollo e implementación de soluciones tecnológicas basadas en agentes de software que mediante el desarrollo de software dirigidos por modelos permiten generar modelos de procesos ejecutables y el código final de agentes de software que habilitan la implementación y ejecución de procesos de negocio colaborativos.

El capítulo 4 presenta una plataforma basada en agentes de software que posibilita a las organizaciones establecer colaboraciones interorganizacionales dinámicas y gestionar la ejecución de procesos colaborativos. También se detallan los mecanismos de la plataforma tecnológica que permiten la generación de los modelos de procesos de negocio ejecutables y la construcción de los agentes de software donde se implementan.

El capítulo 5 describe los métodos basados en el desarrollo dirigido por modelos utilizados para generar los modelos de procesos ejecutables y el código de implementación de los agentes, detallando las transformaciones de modelos de tipo modelo-a-modelo y modelo-a-código propuestos.

El capítulo 6 presenta dos casos de estudio donde se valida la plataforma de agentes, el enfoque metodológico, los métodos y herramientas propuestos que permiten la gestión de colaboración interorganizacional dinámicas y de procesos colaborativos. El primer escenario describe la integración de servicios electrónicos de salud (*e-Healthcare*) y el segundo detalla un caso de estudio en la industria de telecomunicaciones (*Business-to-Business*).

El capítulo 7 presenta las conclusiones del trabajo, una síntesis de las principales contribuciones e identifica los trabajos futuros relacionados con el enfoque de investigación presentado.

# Parte I
# Fundamentos

# 2. Marco teórico

El presente capítulo describe el estado en el que se sustenta la investigación, expone conceptos y detalles de lenguajes, paradigmas, metodologías y plataformas de desarrollo de software referidos en el capítulo 1. Se muestran conceptos relacionados con la gestión de colaboraciones interorganizacionales y se describen los lenguajes de modelado de procesos de negocio que se utilizan en el presente libro (sección 2.1). Se describen los principios del desarrollo dirigido por modelos, el funcionamiento de las transformaciones de modelos, lenguajes y herramientas que lo soportan (sección 2.2). Asimismo se hace una descripción del paradigma de los agentes de software, metodologías y plataformas para su implementación; se detalla la metodología Tropos orientada al desarrollo de agentes de software y se presenta la plataforma de agentes Jadex (sección 2.3). Por último se analizan y discuten los trabajos de investigación relacionados con el enfoque del presente trabajo de investigación (sección 2.4).

## 2.1. Gestión de colaboraciones interorganizacionales

Los enfoques interorganizacionales de integración orientados a procesos tienen como propósito capturar el flujo de información entre las organizaciones involucradas en acuerdos de colaboración (10, 89, 100, 50, 92). Mediante los modelos de procesos de negocio se captura la información de negocio que se requiere en cada paso de un proceso colaborativo, evitando información redundante y reduciendo el riesgo de diferencias semánticas (22). Estos modelos permiten a las organizaciones alinear sus procesos internos con los procesos colaborativos.

### 2.1.1. Sistemas de gestión de procesos de negocio

La gestión de procesos de negocio *(Business Process Management,* BPM) consiste en llevar a cabo un ciclo de mejora continua en los procesos de las organizaciones, el cual implica realizar varias etapas por cada proceso: análisis/evaluación, diseño, implementación y ejecución. La BPM también puede ser vista como la aplicación de un conjunto de tecnologías, métodos, herramientas de software y estándares que permiten analizar, diseñar, implementar y gestionar los procesos de las organizaciones en forma explícita, con el objetivo de automatizarlos integrando a las personas y aplicaciones de software. La BPM no sólo es relevante para integrar a las personas y aplicaciones de una organización, sino también para integrar, gestionar y ejecutar satisfactoriamente los procesos colaborativos entre organizaciones.

La implementación se puede realizar usando un *sistema de gestión de procesos de negocio* GNP (*Business Process Management Systems*, BPMS), el cual es un tipo de sistema de información orientado a procesos (23, 2, 24). El BPMS es un sistema de software genérico que coordina la ejecución de procesos de negocio con base en representaciones explícitas (modelos) de los mismos (91). Estos sistemas contienen motores de procesos que controlan directamente los procesos automatizados o *workflows* mediante modelos de procesos o descripciones formales de procesos (4).

### 2.1.2. Modelado de procesos de negocio

El diseño y modelado de procesos de negocio es la primera y más importante etapa del ciclo de la GNP (BPM) (1). A través del modelado se intenta separar la lógica del proceso de la lógica de las aplicaciones, de tal manera que el proceso de negocio pueda ser automatizado mediante un BPMS. Dentro de los desafíos que el modelado de procesos de negocio implica, se pueden mencionar: la expresividad del lenguaje de modelado y la complejidad de la verificación de modelos. Algunos lenguajes ofrecen una sintaxis apropiada para expresar las actividades de negocio más relevantes y sus relaciones en el modelo de proceso, mientras que otros lenguajes ofrecen constructores que facilitan una eficiente verificación del modelo de proceso en tiempo de diseño (52).

### 2.1.3. Lenguajes para el modelado de procesos colaborativos

El análisis y diseño de los procesos colaborativos requieren de lenguajes de modelado que cumplan con las características de las colaboraciones interorganizacionales, tales como: preservar la autonomía de las organizaciones, descripción global de las interacciones entre las partes, representación de negociaciones complejas, gestión descentralizada, vistas de integración y aspectos complementarios de los procesos colaborativos. Existen varios lenguajes de modelado de procesos de negocio: *Business Process Management Notation* (BPMN) (57), *Event-Driven Process Chains* (EPCs) (Scheer, 2000), Diagramas de Actividad de UML (63). No obstante, aunque éstos permiten definir todos los aspectos de los procesos de negocio privados de las organizaciones, no satisfacen los requerimientos de las colaboraciones interorganizacionales (87). Un lenguaje que satisface estos requerimientos es el UML *Profile for Collaborative Business Processes based on Interaction Protocols* (UP-ColBPIP) (90, 89, 87).

A continuación se describen los lenguajes de modelado utilizados en el enfoque propuesto que en conjunto cumplen con los requerimientos de la integración de los procesos en entornos interorganizacionales.

### 2.1.3.1. UP-ColBPIP

El lenguaje UP-ColBPIP extiende la semántica de UML2 para modelar procesos colaborativos con independencia de la tecnología (90). Fue definido como un perfil UML proporcionando notaciones gráficas fáciles de comprender por los analistas de negocio y diseñadores de sistemas (87). Este lenguaje utiliza un enfoque de desarrollo *top-down* y ofrece elementos conceptuales para el modelado de cinco vistas:

- **Definición de la vista de colaboración interorganizacional.** 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 integración y describe en forma jerárquica las metas de negocio comunes que han acordado.

- **Definición de la vista de procesos colaborativos.** En esta vista se representan los procesos de colaboración 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.

- **Definición de la 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 interorganizacional son representados en la vista del protocolo de interacción mediante el uso de actos de comunicación (*speech acts*). Cada mensaje de negocio tiene un acto de comunicación asociado que representa la intención del emisor con respecto al documento de negocio.

- **Definición de la 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 les hace referencia en varias de las vistas del lenguaje.

- **Definición de la vista de interfaz de negocio.** Es una vista estática de la colaboración que describe las interfaces de negocio en donde 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.

En este trabajo de investigación, el lenguaje UP-ColBPIP es utilizado para expresar los modelos de procesos colaborativos en términos de protocolos de interacción.

### 2.1.3.2. BPMN

El lenguaje BPMN permite definir diagramas de proceso de negocio que describen el flujo de un proceso de negocio utilizando técnicas de diagramas de flujo (4, 94). La versión 2.0 de BPMN está basada en una definición semiformal mediante un metamodelo (57). El metamodelo contiene constructores adicionales al lenguaje, los cuales pueden ser utilizados por los motores de proceso para capturar la información necesaria para ejecutar el proceso; también puede servir de base para el desarrollo de un formato de intercambio de modelos basados en BPMN (4). El lenguaje BPMN provee de un mapeo al lenguaje de ejecución de procesos WS-BPEL (98). Este último es un lenguaje para la implementación de procesos de negocio basados en servicios Web.

BPMN 2.0 presenta tres tipos básicos de submodelos: procesos privados y públicos, coreografías y colaboraciones, los cuales se describen a continuación:

- Los **procesos privados** representan los procesos internos de una organización. Son usualmente llamados *workflows*, procesos BPM (57) o procesos de orquestación de servicios en el área de servicios Web. Se clasifican en dos tipos: ejecutables y no ejecutables. Un proceso ejecutable se modela con el propósito de ser ejecutado de acuerdo con la semántica definida en el BPEL. Un proceso no ejecutable se modela con el propósito de documentar a nivel conceptual el comportamiento del proceso.

- Los **procesos públicos** representan las interacciones entre un proceso de negocio privado con otro proceso o participante. En la versión 1.2 del BPMN fue denominado *proceso abstracto* (56). En este proceso sólo se representan las actividades que se utilizan para comunicarse con el otro participante. Las actividades internas que corresponden al proceso privado no se muestran en el proceso público.

- Una **coreografía** define el comportamiento esperado de una interacción interorganizacional. Es básicamente el contrato de un procedimiento entre los participantes (organizaciones). En un diagrama de coreografía, las actividades representan las interacciones que tienen lugar a través del intercambio de mensajes.

- Una **colaboración** muestra las interacciones entre dos o más entidades de negocio. Usualmente contiene dos o más *pools* que representan a los participantes. El intercambio de mensajes se muestra mediante un flujo de mensajes que conecta dos *pools* u objetos. La colaboración puede mostrar dos o más procesos públicos que se comunican entre sí.

En la presente investigación, el lenguaje BPMN es utilizado para expresar los modelos de procesos de integración, tanto conceptuales como ejecutables, como se detalla en los siguientes capítulos.

## 2.2. Desarrollo dirigido por modelos

El desarrollo dirigido por modelos (*Model Driven Development*, MDD) es un enfoque para el desarrollo de software, el cual se basa en el uso de modelos como los principales artefactos en el desarrollo junto con el uso de transformaciones automáticas de modelos para generar la implementación o código del software (77, 69). Siguiendo este enfoque, modelos de alto nivel de abstracción son transformados en modelos con detalle de bajo nivel (28, 69). Las relaciones entre estos modelos proporcionan una red de dependencias que registran el proceso por el cual es creada una solución y ayudan a entender las implicaciones de los cambios en cualquier punto del proceso de desarrollo (16). Una de las propuestas de OMG más conocida y utilizada en el ámbito de MDD es la arquitectura dirigida por modelos (*Model Driven Architecture*, MDA) desarrollada por OMG (59), la cual se describe en las siguientes subsecciones.

La transformación de modelos puede ser vertical, la cual refina modelos abstractos en modelos más concretos, u horizontal, la cual describe mapeos entre modelos del mismo nivel de abstracción. Para simplificar la tarea de codificar transformaciones de modelos se han desarrollado lenguajes de alto nivel que proporcionan una sintaxis estándar y la semántica de ejecución basadas en las especificaciones de MOF (61).

### 2.2.1. Arquitectura dirigida por modelos

Definiciones iniciales de OMG (59) describen a MDA como un enfoque de construcción de un conjunto de artefactos de software que permiten la generación de especificaciones o un código mediante la aplicación de transformaciones de modelos. Un desarrollo MDA se centra, primero, en la funcionalidad y comportamiento de la aplicación, dejando a un lado la plataforma tecnológica en la que será implementada, separando las funciones de negocio de los detalles de implementación. MDA está basado en un grupo de estándares emergentes que especifican cómo definir un conjunto de modelos, notaciones y reglas de transformación. Ofrece una infraestructura abierta, neutral e independiente de enfoques de propietarios, garantizando la interoperabilidad de los sistemas a través de los estándares de modelado establecidos por OMG: *Meta-Object Facility* (MOF) (61), *Unified Modeling Language* (UML) (63) y el *Common Warehouse Meta-model* (CWM) (58).

Usando estos estándares de modelado se pueden desarrollar soluciones de negocio basadas en descripciones independientes de la plataforma que pueden ser transformados en una plataforma abierta o propietaria basada en CORBA, J2EE, NET, XMI/XML, plataformas basadas en servicios Web u otras. Los lenguajes de modelado y los modelos para una MDA deben ser construidos utilizando UML, MOF y perfiles de UML. La definición actual de MDA incluye el desarrollo y el uso arbitrario de lenguajes específicos de dominio (*Domain Specific Languages*, DSL) que no están basados en UML.

Los lenguajes de modelado deben ser descritos en términos de MOF, que es el lenguaje estándar de OMG para definir metamodelos de lenguajes. Esto garantiza la interoperabilidad de los modelos y metamodelos intercambiados entre herramientas de MDA y la capacidad de realizar transformaciones automáticas entre modelos. En MDA se especifican tres modelos por defecto para el desarrollo de un sistema (28, 45, 69):

- **Modelo independiente de la computación** (*Computation Independent Model*, CIM). Un 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. Utiliza un vocabulario que es familiar para los especialistas del dominio y es expresado en un lenguaje específico del negocio o dominio.

- **Modelo independiente de la plataforma** (*Platform Independent Model*, PIM). Un PIM es una vista del sistema independiente de la plataforma. Es un modelo con alto nivel de abstracción independiente de cualquier tecnología o lenguaje de implementación que exhibe un grado suficiente de independencia de la plataforma a fin de permitir su mapeo en una o más plataformas. Esto se logra mediante la definición de un conjunto de servicios con abstracción de los detalles técnicos.

- **Modelo específico de la plataforma** (*Platform Specific Model*, PSM). Un PSM presenta una vista del sistema desde la perspectiva de la plataforma tecnológica específica. Es un modelo de solución asociado a una plataforma que incluye los detalles del PIM que describen cómo se puede implementar el CIM, y los detalles que describen cómo realizar la implementación en dicha plataforma.

En resumen, para la generación del código de una aplicación o sistema se diseña un modelo PIM que puede ser implementado en diferentes plataformas tecnológicas mediante un modelo PSM. Un modelo PSM puede ser transformado en código de una plataforma tecnológica seleccionada. Esto permite disponer de sistemas más adaptables a los cambios tecnológicos y realizar el diseño y modelado de una aplicación sin considerar los detalles de la tecnología de implementación.

### 2.2.2. MOF

MOF (61) es utilizado para la definición de lenguajes de modelado y para la construcción de herramientas que implementan lenguajes de modelado (45). MOF es el núcleo que permite el desarrollo y aplicación del enfoque MDA ya que proporciona los conceptos y herramientas para razonar acerca de los lenguajes de modelado. Mediante lenguajes de modelado basado en MOF se pueden definir transformaciones de modelos. Estas transformaciones son definidas en términos de los metamodelos de los lenguajes

de modelado involucrados y pueden ser aplicadas a cualquier modelo definido con estos lenguajes (45). Los lenguajes de modelado basados en MOF definen una forma estándar para generar el formato de intercambio de los modelos de esos lenguajes en un tipo de formato basado en XML denominado XMI (XML *Metadata Interchange*).

La arquitectura de MOF está compuesta por cuatro capas orientadas a estandarizar conceptos relacionados con el modelado, que van desde los más abstractos hasta los más concretos (28, 45, 69):

- **Nivel M3: Metametamodelo.** Es el nivel más abstracto que permite definir metamodelos (MOF se encuentra en este nivel). MOF es un lenguaje para describir lenguajes de modelado. En este nivel se definen los conceptos necesarios para razonar acerca de los conceptos del nivel M2. Los elementos en el nivel M3 clasifican a los elementos del nivel M2.

- **Nivel M2: Metamodelo.** Los elementos del nivel M2 son metamodelos de lenguajes de modelado definidos mediante constructores de MOF (por ejemplo, el metamodelo de UML describe al lenguaje UML). Los constructores de M2 son instancias de los constructores de M3.

- **Nivel M1: Modelo del sistema.** Representa el modelo de un sistema de software, sus elementos son modelos de los datos. Los conceptos del nivel M1 son categorías o clasificaciones de las instancias del nivel M0. Por ejemplo, si en el nivel M2 reside el metamodelo de UML, en el nivel M1 tendríamos un modelo UML.

- **Nivel M0: Instancias.** En el nivel M0 se encuentran las instancias "reales" del sistema, los objetos y datos de la aplicación. Cada elemento en el nivel M0 es una instancia de un elemento en el nivel M1.

### 2.2.3. Transformaciones de modelos

Una transformación de modelos implica el uso de uno o más modelos como entrada al proceso de transformación, cuya salida puede ser uno o más modelos o diferentes tipos de código ejecutable. Los principales tipos de transformaciones son: *refactorización* (*refactoring*), *modelo-a-modelo* y *modelo-a-texto*. Esta última también llamada *modelo-a-código* (16). La transformación *refactorización* reorganiza un modelo basándose en criterios bien definidos y la salida es una versión revisada del modelo original. Las transformaciones *modelo-a-modelo* convierten un modelo en otro o en conjunto de modelos diferentes al de entrada. Una transformación *modelo-a-código* convierte un modelo a un fragmento de código (16).

En el enfoque MDA existen cuatro categorías de técnicas para aplicar transformaciones de modelos. En la transformación *manual* el desarrollador examina el modelo

de entrada y para crear un modelo de salida. La transformación basada en un *perfil* es una extensión de la semántica de UML; mediante la aplicación de un perfil en el que se definen las reglas de transformación, un modelo definido en UML es transformado en otro modelo. La transformación basada en *patrones* es una disposición particular de los elementos de un modelo; los patrones pueden ser aplicados a un modelo de entrada y como resultado se crean los elementos del modelo de salida. La transformación *automática* se basa en reglas de transformación. Estas reglas pueden estar predefinidas en la herramienta de transformación que se utiliza o pueden ser definidas para un dominio específico. La transformación automática requiere que el modelo de entrada esté sintáctica y semánticamente bien definido.

#### 2.2.3.1. Lenguajes y herramientas para transformaciones de modelos

En los últimos años han sido propuestos varios lenguajes y herramientas de tipo *open-source* o comerciales que brindan soporte a la transformación de modelos. La OMG ofrece varios lenguajes para transformaciones *modelo-a-modelo*, como son: el lenguaje QVT *Operational* (62) de naturaleza imperativa, los lenguajes QVT *Core* y QVT *Relational* (62) de tipo declarativos, y el lenguaje ATL (5) que se considera de tipo híbrido. Para transformaciones *modelo-a-código* los principales lenguajes son: MOF *Model to Text Transformation Language* (MOFM2T) (60), *Java Emitter Templates* (JET) (40), Acceleo (3) y Xpand (99).

En el presente trabajo de investigación se utilizó el lenguaje ATL tanto para transformaciones *modelo-a-modelo* como para transformaciones *modelo-a-código*, el cual se describe en la subsección siguiente.

#### 2.2.3.2. ATL

ATL (5) es un lenguaje de transformación de modelos que proporciona un conjunto de herramientas de desarrollo estándar, basadas en el ambiente de desarrollo Eclipse, que tienen como objetivo facilitar las transformaciones *modelo-a-modelo*. Las transformaciones de modelos pueden ser especificadas por medio de módulos ATL. El lenguaje permite definir consultas ATL para calcular valores primitivos o simples, tales como un *string*, entero o booleano, desde un modelo de origen. También ofrece la posibilidad de desarrollar librerías ATL independientes que pueden ser importadas desde cualquier unidad ATL.

Un módulo de transformación ATL contiene un número de variables del modelo de entrada y del modelo de salida. Este módulo contiene las siguientes secciones: encabezado, importación, *helpers* y reglas.

La sección de *encabezado* (*header*) define el nombre del módulo de transformación y de las variables correspondientes al modelo de origen y modelo destino.

La sección de *importación* permite importar librerías ATL existentes. Una librería ATL permite hacer un conjunto de ATL *helpers* que pueden ser llamados desde diferentes unidades ATL. Estas librerías pueden ser utilizadas por los módulos y las consultas ATL.

La sección *helpers* permite encapsular expresiones complejas de OCL que son aplicadas a un número de parámetros. Mediante los *helpers* es posible elaborar un código que se puede traer desde diferentes puntos de una transformación ATL. Los atributos de un *helper* son usados para almacenar el valor de una expresión OCL.

La sección de *reglas* define la forma en que los modelos destino son generados desde los modelos de origen. En ATL existen tres diferentes tipos de reglas que rigen a los dos modos de programación: las reglas *matched* que ordenan la programación declarativa y las reglas *called* y *lazy* que se utilizan para la programación imperativa.

## 2.3. Paradigma de desarrollo de software orientado a agentes

### 2.3.1. Tecnología de agentes de software

La tecnología de agentes de software, desarrollada para hacer frente a entornos complejos y dinámicos, ha registrado un fuerte interés tanto en la academia como en la industria. Un *agente de software* es una entidad de software que funciona continua y autónomamente en un entorno particular, el cual a menudo es habitado por otros agentes y procesos (76).

Un sistema multiagente (*MultiAgent System*, MAS) está compuesto por agentes autónomos, competitivos y cooperativos que interactúan intercambiando mensajes con el fin de alcanzar metas individuales o metas comunes (34, 96, 74). Los principales lenguajes de comunicación entre agentes son *Foundation for Intelligent Physical Agents* (FIPA) (27) y *Knowledge Query and Manipulation Language* (KQML) (26). Estos lenguajes se basan en la teoría de los actos de comunicación (*speech acts*) (73) para facilitar interacciones de alto nivel entre los agentes, tales como las negociaciones y la coordinación.

#### 2.3.1.1. Arquitectura de agente BDI

La arquitectura de agente *BeliefDesire-Intention* (BDI) está basada en la teoría del razonamiento práctico propuesta en (13). BDI es un enfoque para modelar los estados internos de un agente en función de los elementos de creencias (*Beliefs*), deseos (*Desires*) e intenciones (*Intentions*). Las *creencias* representan el conocimiento que tiene un agente acerca del mundo o su entorno. Los *deseos* son un conjunto de objetivos que el agente desea alcanzar. Las *intenciones* son las actividades que el agente se ha comprometido a ejecutar para satisfacer un deseo, típicamente en forma de *metas* intermedias asociadas con los *planes* que el agente ha seleccionado ejecutar.

Las *metas* son los deseos que el agente intenta cumplir, con la creencia de que son alcanzables. Los *planes* contienen los medios para alcanzar ciertos estados futuros de ese mundo, esto es, las posibles acciones a ejecutar con el fin de alcanzar una meta (71). Entonces, las intenciones son un conjunto de planes que el sistema intenta ejecutar para alcanzar una meta.

## 2.3.2. Metodologías para el desarrollo de sistemas basados en agentes

En el desarrollo de sistemas de software basados en agentes se debe tener en cuenta la flexibilidad de los agentes, la autonomía, el comportamiento orientado a metas, la complejidad de las dependencias y las interacciones entre los agentes. Todos estos requerimientos deben considerarse en el diseño de una plataforma de agentes utilizando una metodología apropiada. Dentro de las principales metodologías para el desarrollo de sistemas de agentes de software se pueden mencionar: GAIA (95), MAS-CommonKADS (38), MaSE (21), MESSAGE (18), AUML (8), Prometheus (64) y Tropos (15).

Para el desarrollo de la plataforma de agentes de software propuesta en este libro se utilizó la metodología Tropos, misma que es recomendada para el desarrollo de agentes con un enfoque BDI e implementaciones basadas en la plataforma Jadex (67, 65).

### 2.3.2.1. Metodología Tropos

La metodología Tropos (15) se basa en la idea de construir un modelo del sistema y de su entorno, el cual será refinado y extendido progresivamente, proporcionando una interfaz común a las diferentes actividades del proceso de desarrollo de los agentes, así como una base para la documentación y la evolución del sistema, mediante un enfoque de desarrollo *top-down*. Tropos está basado en el *framework* de modelado organizacional *i** (101), cuyos conceptos principales son *actor*, *meta* y *actor-dependencia* y los utiliza como base para generar los modelos. El concepto meta representa los intereses estratégicos de un actor. Las metas pueden ser del tipo meta dura (*hardgoal*) o meta suave (*softgoal*). Las metas suaves son típicamente usadas para modelar requerimientos no funcionales. Los planes representan una manera de realizar una acción, en un nivel abstracto. La ejecución de un plan puede ser una manera de lograr una meta dura o satisfacer una meta suave.

El análisis de requerimientos en Tropos se divide en dos fases: requerimientos tempranos y requerimientos tardíos. Ambos comparten el mismo enfoque conceptual y metodológico. Durante la primera fase se identifican los actores o *stakeholders* del dominio, los cuales son modelados como actores sociales. Entre los actores se definen dependencias que permiten que las metas sean alcanzadas, los planes realizados y los recursos suministrados. Una dependencia es una relación intencional y estratégica entre dos actores. En el análisis de requerimientos tardíos el modelo conceptual es extendido

incluyendo a un nuevo actor, el cual representará al sistema y las dependencias con otros actores del entorno. En estas dependencias se definen todos los requerimientos funcionales y no funcionales del sistema.

Las fases del diseño arquitectónico y del diseño detallado se centran en la especificación del sistema, de acuerdo con los requerimientos derivados de las fases anteriores. El diseño arquitectónico ofrece un mapeo de los actores del sistema a un conjunto de agentes de software, detallando los planes y recursos necesarios para alcanzar una meta. La fase de diseño detallado tiene por objetivo especificar las capacidades de los agentes y sus interacciones. Por lo general, en esta fase la plataforma de aplicación ya ha sido seleccionada, y esto puede ser tomado en cuenta con el fin de realizar un diseño detallado que se correlaciona directamente con el código. La fase de implementación consiste en la generación de un bosquejo del código de los agentes de software, realizando un mapeo del modelo del diseño detallado para los conceptos de una plataforma específica de agentes.

En la metodología Tropos se utilizan los conceptos de creencia (*belief*) y capacidad (*capability*). Las creencias representan el conocimiento del mundo que posee un actor y la capacidad representa la habilidad de un actor para definir, elegir y ejecutar un plan para el cumplimiento de una meta, de acuerdo con ciertas condiciones en el mundo y en presencia de un evento específico.

Los principales diagramas a definir son el de actores y el de metas, los cuales se van refinando en cada una de las fases de la metodología. El diagrama de actores consiste en identificar y analizar los actores del entorno, del sistema y los agentes. El diagrama de metas proporciona una vista completa del dominio de aplicación, con el cual se pueden determinar estrategias para cumplir las metas relacionadas con un actor. En la fase de diseño detallado se generan los diagramas de capacidades y el diagrama de planes que están basados en el diagrama de actividades de AUML (8). El diagrama de interacción del agente utiliza como base el diagrama de secuencia de AUML.

### 2.3.3. Plataformas para el desarrollo de agentes de software

Las plataformas de agentes de software están principalmente basadas en Java y tratan de abstraerse de paradigmas orientados a objetos, introduciendo extensiones que abarcan a los agentes. Las plataformas de agentes son un *middleware* que ofrece servicios básicos para la realización de un MAS, tales como protocolos de comunicación y servicios especiales. Entre las plataformas de agentes de software más conocidas están Jack (37), FIPA-OS (70) y JADE (11). La plataforma Jadex (67) es una base para el diseño, desarrollo e implementación de agentes de software. La plataforma Jadex desarrolla los agentes orientados a metas, siguiendo la arquitectura de agentes BDI (*Beliefs*, *Desires*, *Intentions*) (13) y es totalmente compatible con el estándar FIPA (27).

### 2.3.3.1. Plataforma de agentes Jadex

El *framework* y la plataforma de agentes Jadex (67) es un entorno de desarrollo con el propósito general de crear aplicaciones basadas en MAS, permitiendo la construcción de agentes con comportamientos reactivos (basados en eventos) y deliberativos (basados en metas). El *framework* Jadex está construido sobre la plataforma JADE y permite implementar agentes JADE de tipo racional con comportamiento orientado a metas (opuesto a los agentes orientados a tareas) mediante la implementación de una arquitectura BDI (71).

Los agentes en la plataforma Jadex son programados mediante la definición de un archivo XML, llamado *archivo de definición de agente* (*Agent Definition File*, ADF), en combinación con las distintas clases del lenguaje Java. En el ADF, el agente se especifica mediante la enunciación de creencias (*beliefs*), metas (*goals*), planes (*plans*) y algunas otras propiedades. En Jadex, el comportamiento específico de un agente es determinado por las creencias, metas y planes. Para implementar los planes utilizados por los agentes se debe definir una clase de lenguaje Java, el cual debe extenderse más allá de las clases que proporciona la plataforma. Los principales elementos en un agente BDI basado en la plataforma Jadex se describen a continuación:

- **Creencias** (*Beliefs*). Define el contenido de una base de datos orientada a objetos. Se pueden almacenar creencias individuales y conjuntos de creencias que consisten en objetos Java. Las creencias pueden ser consultadas mediante el uso de simples consultas OQL (select-sql), que pueden ser predefinidas en la sección de expresiones del ADF. Se puede acceder a las creencias en cualquier momento del planteamiento de las metas y los planes del agente.

- **Metas** (*Goals*). Es un concepto central que representa un cierto estado que el agente está intentando alcanzar. En un diseño orientado a metas como Jadex, éstas explícitamente representan los estados a alcanzar y las razones por las que las acciones (planes) se ejecutan. En Jadex, las metas se representan como objetos explícitos contenidos en una base de metas, que son accesibles para el componente de razonamiento y para los planes con la posibilidad de conocer o modificar las metas actuales del agente. La plataforma Jadex soporta diferentes tipos de metas, las cuales presentan diferentes comportamientos con respecto a su procesamiento (14). Una meta de tipo *perform* está directamente relacionada con la ejecución de acciones. La meta se considera alcanzada cuando algunas de sus acciones han sido ejecutadas, sin considerar el resultado de éstas. Una meta *achieve* define un resultado deseado sin especificar cómo se debe alcanzar. En este caso, los agentes pueden probar planes alternativos para lograr una meta de este tipo. En una meta *query* el resultado no está definido como un estado del mundo, sino como una información que el agente quiere conocer. En las metas *maintain*, un agente comprueba el estado deseado y continuamente ejecuta planes para restablecer el estado del mundo que se quiere mantener.

- **Planes** (*Plans*). Representan los elementos del comportamiento de un agente y contienen las acciones que se pueden ejecutar para alcanzar una meta. En la sección de planes del ADF, una definición del plan contiene las referencias a las metas y los eventos (mensajes) que pueden desencadenar la ejecución del plan, las precondiciones para la ejecución del plan y el contexto de ejecución. Los agentes pueden realizar un seguimiento de las acciones y submetas ejecutadas por cada plan para determinar y gestionar los fallos del plan.

- **Capacidades** (*Capabilities*). Representan un mecanismo para agrupar los elementos de un agente, tales como creencias, metas, planes y eventos. De esta manera, los elementos que están estrechamente relacionados se pueden almacenar en un módulo reutilizable.

## 2.4. Trabajos relacionados

El desarrollo de soluciones tecnológicas para la ejecución de procesos colaborativos mediante plataformas basadas en agentes de software, utilizando en algunos casos los principios de MDD y MDA, se han estudiado previamente en algunos trabajos. Las propuestas más relevantes que tienen relación con el enfoque del presente trabajo de investigación se discuten a continuación.

Para dar soporte al modelado y especificación de procesos colaborativos, en (89) se ha propuesto un método sustentado en los principios de MDA, en el cual se define un marco conceptual para la construcción específica de interfaces de sistemas y procesos de negocio para ejecutar procesos colaborativos. En dicho método se propone la utilización del lenguaje UP-ColBPIP (87) para el modelado de los procesos colaborativos, el cual cumple con los requerimientos del dominio de colaboraciones interorganizacionales y con la representación de la vista global de las interacciones entre las organizaciones. Este método explota dos premisas de la MDA: los modelos son los principales productos del desarrollo de software, en lugar del código o programas; el código debe ser generado automáticamente a partir de sus correspondientes modelos. Por lo tanto, los modelos no sólo son utilizados para documentar el sistema, sino también para construir y generar el producto final. En (48) se propone un método basado en MDA para el diseño de modelos de procesos de integración en colaboraciones interorganizacionales, el cual utiliza modelos de procesos colaborativos formados con el lenguaje UP-ColBPIP para generar modelos de procesos de integración definidos con el lenguaje BPMN. En (89) y (90) se presentan métodos basados en MDA que permiten generar soluciones tecnológicas basadas en el estándar BPEL de servicios Web y en el estándar ebXML de transacciones de negocio. No obstante, estos trabajos se enfocan sólo en el aspecto de modelado y generación de algunos artefactos de implementación, pero no proveen una plataforma con los mecanismos requeridos para la gestión de colaboraciones interorganizacionales.

En (9) se presenta una arquitectura basada en MDA para generar transformaciones en un proceso colaborativo en una plataforma basada en una arquitectura orientada a servicios. La transformación se genera de un nivel CIM a nivel PIM. También muestra la forma en que se puede cambiar automáticamente un modelo CIM que contiene la información necesaria a una vista de proceso en un nivel PIM, así como los vínculos entre los procesos públicos y la vista de procesos. Se identifican los constructores, basados en un metamodelo de la arquitectura orientada a servicios, necesarios para el modelado que permite describir una arquitectura independiente de la plataforma que posteriormente puede ser mapeada en diferentes tecnologías como BPEL o MAS.

En (41) se presenta una plataforma de agentes de software para la gestión de procesos interorganizacionales utilizando PetriNets (29). Los procesos son modelados en una vista global, la cual tiene la función de describir procesos abstractos. Estos procesos son mapeados mediante reglas a PetriNets. La plataforma está compuesta por los agentes locales de cada organización involucrada y un agente global, el cual tiene la función de mediador entre los agentes locales y ejecutar el proceso interorganizacional basado en PetriNets. Los agentes locales ejecutan los procesos abstractos de acuerdo con la solicitud del agente global. Este enfoque requiere de un agente con una función de mediador que centraliza las interacciones entre los agentes responsables de ejecutar los procesos, limitando la autonomía de las organizaciones en ambientes de colaboraciones interorganizacionales.

En (43) se describe un enfoque para el diseño y ejecución de procesos interorganizacionales basados en una arquitectura de agentes de software, utilizando los principios de MDA. Los procesos privados son modelados en el nivel de negocio, en donde las actividades internas no son expuestas a otros participantes. En el nivel técnico se introduce un nivel de abstracción adicional entre los procesos privados y los procesos interorganizacionales, llamado vista de procesos. Esta vista de procesos representa una interface para la interacción con otras organizaciones, describiendo las interacciones entre uno o más procesos privados desde la perspectiva de una organización, en comparación con los procesos interorganizacionales que describen las interacciones desde una perspectiva neutral, capturando todas las interacciones permitidas entre las organizaciones de un proceso. En el nivel de ejecución los procesos interorganizacionales se extienden con información de la plataforma específica. El modelo del agente se construye a partir de un modelo de procesos generado en el nivel técnico que a su vez está basado en la arquitectura orientada a servicios, el cual contiene sólo la información necesaria para la interacción con una o más partes de la colaboración. En esta propuesta el modelo del agente generado no tiene una orientación hacia procesos y los planes del agente son especificados en una estructura secuencial y estática.

En (102) se propone un método basado en la MDA para la generación de modelos de agentes de software con capacidad de ejecutar procesos colaborativos. Se generan las especificaciones de un modelo de agente ejecutable en una plataforma específica, que utiliza capacidades (*capabilities*) definidas como módulos que encapsulan una funciona-

lidad. Éstas son definidas mediante el lenguaje *Web Services Description Language* (WSDL) para especificar invocaciones a sistemas internos de la organización. Estas capacidades son definidas en los planes del agente. El modelo del agente sólo contiene las interacciones de mensajes generadas a partir del proceso colaborativo y los planes del agente son especificados con invocaciones a otros servicios, delegando la ejecución de las tareas. Además, la etapa de la generación del código del agente requiere de la intervención del diseñador de sistemas para asignar las capacidades requeridas por el agente e invocar, de ese modo, un servicio específico.

En (47) se presenta una metodología que permite modelar procesos de negocio y generar las especificaciones para su ejecución en una plataforma de agentes. La metodología utiliza un enfoque basado en MDD que permite realizar las transformaciones necesarias mediante el *framework* JIAC V (35), el cual proporciona herramientas para transformaciones de modelos, editores de modelos y plataforma de ejecución de agentes. El desarrollo de procesos inicia con el análisis mediante casos de uso. Luego, para cada caso de uso se crea un diagrama de proceso, estos diagramas están basados en BPMN. A partir de los diagramas de procesos y los diagramas de casos de uso, se derivan el rol de cada participante, el comportamiento y las capacidades. En la etapa de diseño se puede realizar un refinamiento de los modelos mediante los editores de la herramienta. Finalmente, los modelos de organización (roles y agentes) de cada agente y el comportamiento de los agentes (planes, reglas y servicios) son integrados, lo cual permite generar el código del agente.

En (32) se describe una propuesta para el diseño de protocolos de interacción basado en un lenguaje de modelado independiente de la plataforma para el dominio de MAS, llamada DSML4MAS (33). La propuesta se basa en los principios de desarrollo dirigido por modelos para la generación de código. En primer lugar, en una vista de interacción se define el protocolo de interacción. El protocolo se utiliza para generar el comportamiento del agente. La vista de interacción permite definir y modelar las interacciones entre los actores involucrados. Luego, el diseñador del sistema puede refinar la descripción del comportamiento mediante una vista de comportamiento, agregando instancias del proceso privado e información adicional. La especificación del comportamiento se genera utilizando el metamodelo DSML4MAS. Finalmente, el comportamiento se transforma a código, siguiendo la estructura de un agente de la plataforma Jack. En esta etapa, el código puede ser modificado manualmente, agregando lo necesario, para que el agente pueda ser ejecutado.

Las propuestas antes mencionadas ofrecen ventajas en el desarrollo de soluciones tecnológicas. Sin embargo, tienen como característica un enfoque tradicional en el diseño de MAS, en donde el comportamiento del agente que representa una organización en la ejecución de un proceso colaborativo es definido en tiempo del diseño y no puede ser creado o modificado en tiempo de la ejecución de la solución tecnológica. Esto implica que los procesos colaborativos y de integración a los que dan soporte los agentes son acordados y definidos también en tiempo de diseño a través de acuerdos estáticos

de colaboración. Por lo tanto, los agentes no pueden adaptarse en tiempo de ejecución de cambios o ejecutar nuevos procesos. Lo anterior representa la principal desventaja que limita la implementación de dichas propuestas en ambientes de colaboración interorganizacionales dinámicos. En estos ambientes se requiere una plataforma que permita adicionar nuevos comportamientos a los agentes o crear nuevos agentes para que puedan gestionar procesos colaborativos nuevos o rediseñados que las organizaciones acuerdan en forma dinámica, y comunicarse con otros agentes que representan a las organizaciones involucradas. Esto requiere que la generación de estos agentes se realice en tiempo de ejecución a través de la plataforma que los contiene. La propuesta presentada en este libro apunta a dar solución a la necesidad de atender el aspecto dinámico de la gestión de colaboraciones interorganizacionales y la ejecución descentralizada de los procesos colaborativos, ofreciendo agentes orientados a procesos.

# Parte II
# Metodología e implementación

# 3. Metodología de desarrollo de soluciones tecnológicas basadas en agentes de software

En este capítulo se presenta una metodología para el desarrollo de una solución tecnológica basada en agentes de software que permite la gestión de procesos colaborativos en ambientes de colaboración interorganizacionales dinámicos. La metodología sigue un enfoque *top-down* basado en los principios del desarrollo dirigido por modelos (MDD). El propósito es guiar el proceso de desarrollo para generar la solución tecnológica a partir de modelos conceptuales de procesos colaborativos. En primer lugar se presenta el marco conceptual en el que se basa la metodología propuesta (sección 3.1). Luego se describen las fases y actividades de la metodología, así como lenguajes, métodos y herramientas (sección 3.2). Finalmente se discute la aplicación y adaptación de esta metodología al contexto de las colaboraciones interorganizacionales dinámicas (sección 3.3).

## 3.1. Marco conceptual para el desarrollo de colaboraciones interorganizacionales

La metodología propuesta se basa en un marco conceptual que proporciona los diferentes dominios o niveles de abstracción en los que se debe enfocar el desarrollo de colaboraciones interorganizacionales (49). Este marco conceptual ayuda a los analistas de negocio, a los involucrados en la operación y gestión de los procesos colaborativos (*stakeholders*) y a los diseñadores y desarrolladores de sistemas, a analizar y diseñar la solución desde diferentes niveles de abstracción a través de la separación de intereses (*concerns*) y la identificación de los productos o artefactos requeridos en el desarrollo de colaboraciones interorganizacionales. La figura 3.1 muestra que el marco conceptual está conformado por: los requerimientos organizacionales o de negocio (dominio del problema), la solución interorganizacional (dominio de la solución) y la solución tecnológica (dominio de la implementación).

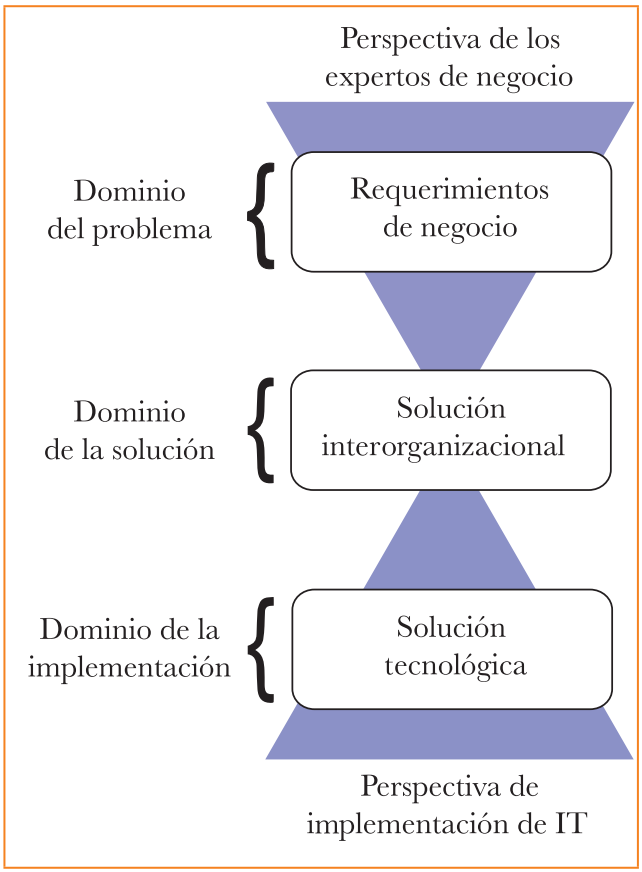



Figura 3.1. *Marco conceptual para el desarrollo de colaboraciones interorganizacionales.*

El *dominio del problema* se refiere a la identificación de los requerimientos de las colaboraciones interorganizacionales desde un punto de vista organizacional o de negocio. Estos requerimientos se explicitan identificando e indicando: las metas en común que deben ser alcanzadas, las organizaciones involucradas en la colaboración, los roles y los procesos colaborativos que las organizaciones deben ejecutar en conjunto para alcanzar las metas establecidas y la información a intercambiar en dichos procesos.

El *dominio de la solución* consiste en la definición de cómo las organizaciones gestionarán la colaboración interorganizacional para satisfacer las metas (requerimientos) establecidas. La solución interorganizacional (también llamada solución de negocio) define el comportamiento explícito de los procesos colaborativos, describiendo la vista global de las interacciones entre las organizaciones involucradas. La solución

interorganizacional también define los procesos de negocio privados que cada organización debe realizar para soportar la ejecución de los procesos colaborativos. La solución de este dominio debe ser independiente de la plataforma, expresada a través de modelos de procesos de alto nivel de abstracción que permitan capturar los conceptos clave del dominio de la solución. De esta manera, la solución interorganizacional puede ser reutilizada para proporcinar posteriormente implementaciones en diferentes tecnologías.

El *dominio de la implementación* se refiere al desarrollo de la solución tecnológica. Ésta se conforma de modelos o especificaciones ejecutables de los procesos de integración y de los sistemas de información interorganizacionales que los implementan. Los modelos o especificaciones ejecutables son expresados usando conceptos específicos de una plataforma. En este nivel se apunta a la integración e interoperabilidad de sistemas de información interorganizacionales heterogéneos, de manera que puedan interactuar y soportar la ejecución de los procesos colaborativos.

## 3.2. Fases de la metodología para el desarrollo de soluciones tecnológicas basadas en agentes

La metodología que guía el proceso de desarrollo de soluciones tecnológicas basadas en agentes de software para la implementación de colaboraciones interorganizacionales se presentó en (82) aplicada en un caso de estudio específico. Ésta se basa en el marco conceptual descrito anteriormente para separar el proceso de desarrollo en tres fases diferentes. La metodología apunta al uso de modelos conceptuales en diferentes puntos de vista, niveles de abstracción y de granularidad. Los artefactos de salida de las fases de esta metodología se representan por medio de modelos (de procesos y/o sistemas), generados mediante la aplicación de los principios de la MDD. El resultado final son los artefactos de implementación, esto es, los modelos de procesos de integración ejecutable y el código de los agentes que representan los sistemas interorganizacionales de cada participante de la colaboración.

En la figura 3.2 se presentan las fases que componen esta metodología: definición del acuerdo de colaboración, diseño de la solución interorganizacional y generación de la solución tecnológica. En las siguientes subsecciones se describe en detalle cada una de estas fases.

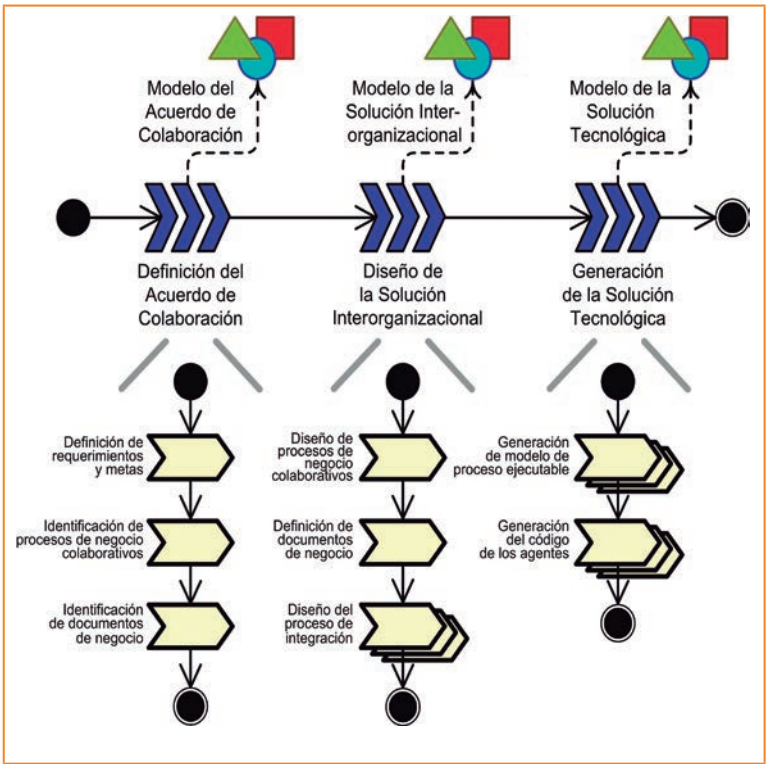



Figura 3.2. *Fases de la metodología para el desarrollo de soluciones tecnológicas basadas en agentes para colaboraciones interorganizacionales.*

## 3.2.1. Definición del acuerdo de colaboración

En esta fase, los requerimientos de la colaboración son definidos en forma conjunta por las organizaciones involucradas a través de un modelo del acuerdo, el cual define las metas comunes y la identificación tanto de los procesos colaborativos como de los documentos de negocio. El modelo del acuerdo de colaboración se expresa mediante el lenguaje UP-ColBPIP (87, 89). La definición de los requerimientos y del acuerdo de colaboración se representan mediante la vista de colaboración interorganizacional del lenguaje UP-ColBPIP, haciendo uso de diagramas de colaboración y diagramas de clases UML2. En un diagrama de colaboración se expresa una colaboración con las organizaciones involucradas, el rol que desempeñan estas organizaciones en la colaboración y sus vínculos de comunicación. La figura 3.3 muestra un ejemplo de la definición de un acuerdo de colaboración basado en el modelo de negocio con referencia CPFR (*Collaborative Planning, Forecasting and Replenishment*) (86) entre dos organizaciones en una cadena de abastecimiento. El diagrama de colaboración describe a los participantes, es decir, la organización A que desempeña el rol de cliente

y la organización B que desempeña el rol de proveedor; ambos se relacionan a través de un enlace que une la colaboración y sus interfaces correspondientes.

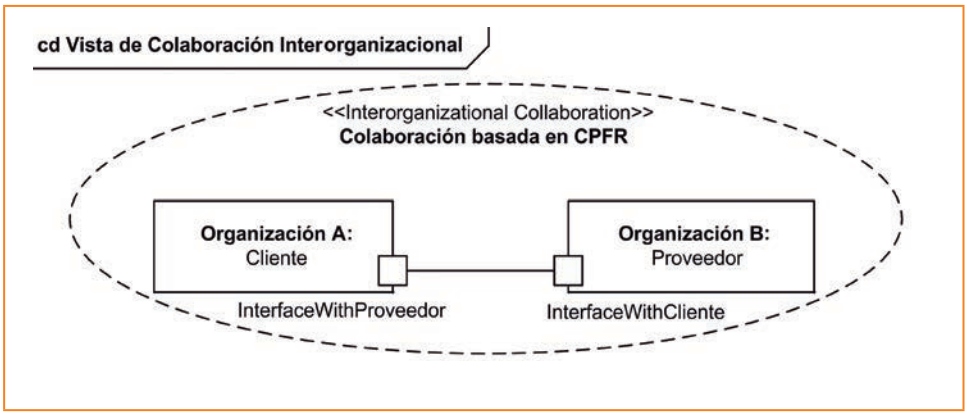



Figura 3.3. *Vista de la colaboración interorganizacional.*

Además, las metas de negocio o de la colaboración que tienen en común las organizaciones, las cuales se deberán cumplir durante el proceso, son capturadas en esta vista a través de diagramas de clases. La definición de las metas permite a las organizaciones evaluar el rendimiento o desempeño del acuerdo de la colaboración. La figura 3.4 presenta un ejemplo de la definición de las metas comunes entre las organizaciones participantes en el acuerdo de colaboración mediante un diagrama de clases.

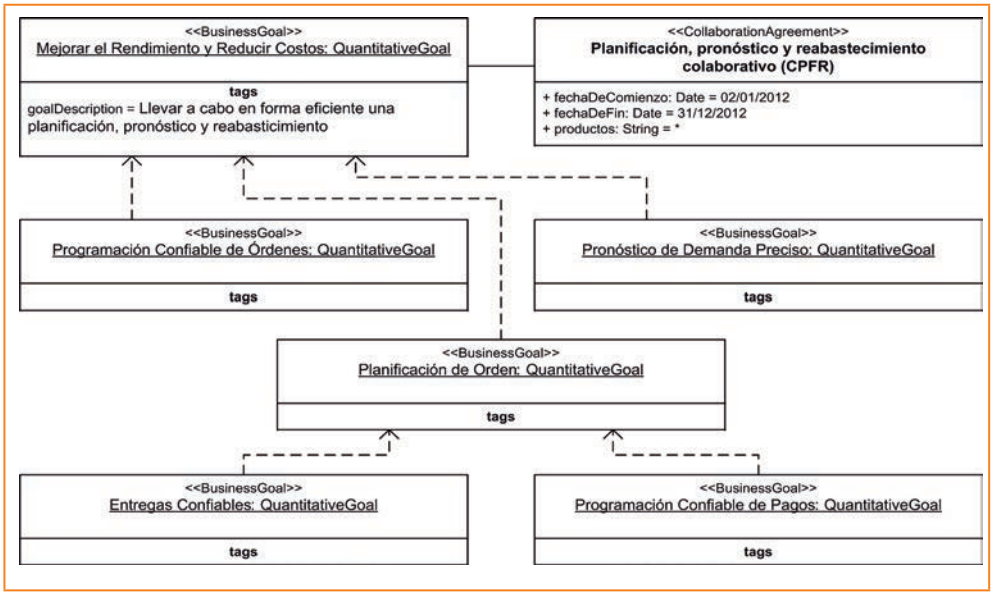



Figura 3.4. *Diagrama de clases del acuerdo de colaboración y las metas de negocio.*

Las metas son presentadas en un orden jerárquico, en este caso, la meta madre, "Mejorar el rendimiento y reducir costos", tiene tres submetas: "Pronóstico de demanda preciso", "Programación de orden confiable" y "Planificación de orden". Esta última tiene a su vez dos submetas.

Luego, se identifican los procesos colaborativos necesarios para cumplir con las metas comunes acordadas previamente. Además, se identifican los documentos de negocio que son requeridos en los procesos colaborativos y que contienen la información a intercambiar. Los procesos colaborativos y los documentos de negocio se muestran utilizando la *vista de procesos de negocio colaborativos* del lenguaje UP-ColBPIP. Los procesos, junto con las metas asociadas a los mismos y los documentos de negocio, se expresan en diagramas de casos de uso.

La figura 3.5 muestra un ejemplo de la definición de procesos colaborativos y su relación con las metas de negocio acordadas entre las organizaciones mediante la *vista de procesos de negocio colaborativos*. En la misma se ha definido el proceso colaborativo "Colaboración basada en CPFR" entre la organización A y la organización B, mediante dicho proceso se puede cumplir la meta de negocio común "Mejorar el rendimiento y reducir costos". Este proceso colaborativo está conformado por tres subprocesos: "Pronóstico de demanda colaborativo", "Planificación de orden" y "Ejecución de orden". El proceso colaborativo "Ejecución de orden" a su vez está compuesto de los subprocesos: "Gestión de entregas" y "Gestión de pagos". Cada uno de los procesos colaborativos definidos en la *vista de procesos de negocio colaborativos* está relacionado con una meta de la colaboración y es mediante dicho proceso que se puede alcanzarla.

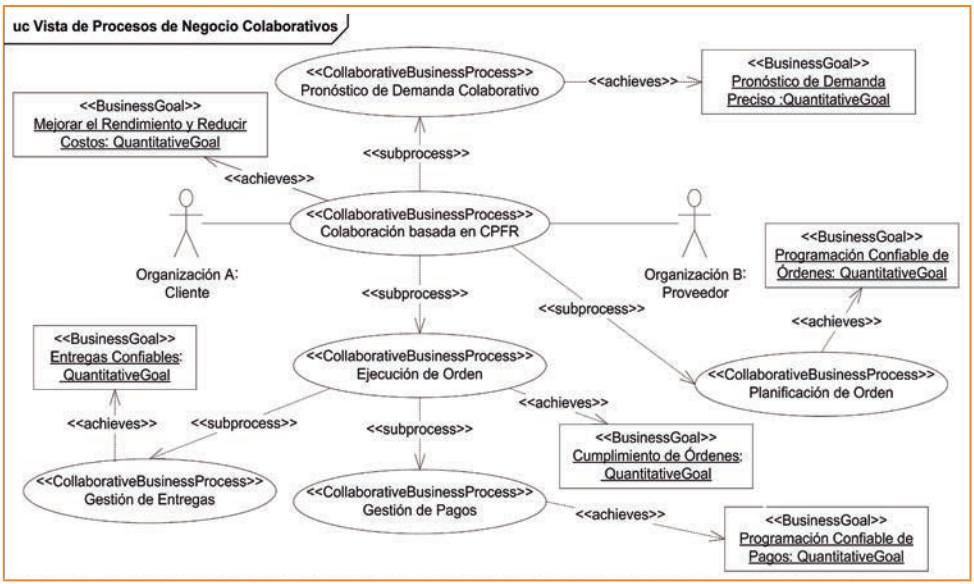



Figura 3.5. *Vista de procesos de negocio colaborativos.*

## 3.2.2. Diseño de la solución interorganizacional

Esta fase implica tres actividades: diseño del comportamiento de los procesos colaborativos requeridos para cumplir con las metas de la colaboración, definición de los documentos de negocio que se intercambiarán durante la ejecución de los procesos colaborativos y diseño de los procesos de integración de cada organización, conforme al rol que la organización desempeña en la colaboración interorganizacional.

### 3.2.2.1. Diseño de procesos de negocio colaborativos

En la actividad de *diseño de procesos de negocio colaborativos* se utiliza el lenguaje UP-ColB-PIP para definir el comportamiento de los procesos colaborativos a través de la *vista de protocolos de interacción*. La representación de un proceso colaborativo mediante un protocolo de interacción permite describir las interacciones entre las organizaciones desde un punto de vista global, 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 y la semántica del mensaje es descrita por un acto de comunicación (*speech act*), el cual permite explicitar la intención que el emisor tiene en relación con el documento que se envía. Decisiones y compromisos entre las partes pueden ser establecidos y conocidos a partir de estos actos de comunicación. Esto posibilita la definición de negociaciones complejas y evita la ambigüedad en la semántica y mejora el entendimiento de las interacciones entre organizaciones en los procesos colaborativos. Los protocolos de interacción son expresados a través de diagramas de secuencia de UML2. Un ejemplo de un protocolo de interacción es la figura 3.6. Muestra el diagrama de secuencia del protocolo que representa el proceso colaborativo "Pronóstico de demanda colaborativo".

Este protocolo soporta una negociación entre una organización que cumple el rol de *cliente* y otra organización con el rol de *proveedor* para acordar un pronóstico de demanda. El protocolo inicia con el rol de *cliente*, que solicita al proveedor la realización de un pronóstico de demanda. El documento de negocio contenido en el mensaje de solicitud transmite los datos necesarios para el pronóstico (por ejemplo: productos, horizontes de tiempos del pronóstico, etc.). El rol de *proveedor* procesa la solicitud y puede responder aceptando o rechazándola, esto es indicado en el protocolo mediante un segmento de control de flujo XOR, que representa dos caminos mutuamente excluyentes.

En caso de que el rol proveedor acepte la propuesta, se compromete a realizar el pronóstico de demanda requerido, de lo contrario el proveedor rechazará la propuesta y el proceso finaliza con una terminación de error. Cuando el proveedor acepta la solicitud, el cliente, en forma paralela, mediante dos mensajes informa: el pronóstico de ventas de sus puntos de venta (POS) y los eventos y políticas de ventas planificadas, esto se indica

a través del segmento de control de flujo AND que expresa dos caminos en paralelo. Por último, con esta información, el proveedor genera un pronóstico de demanda y lo envía al cliente, con lo cual el proceso termina.

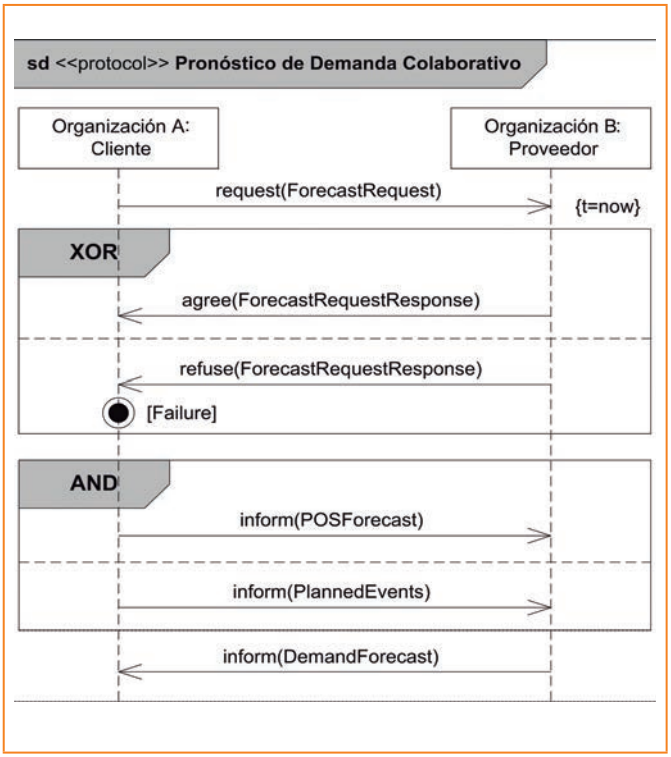



Figura 3.6. *Vista de protocolos de interacción.*

### 3.2.2.2. Definición de documentos de negocio

La actividad de *definición de documentos de negocio* se realiza mediante la *vista de documentos de negocio* del lenguaje UP-ColBPIP. La definición de documentos de negocio se centra en la representación de la información que será intercambiada en los procesos colaborativos. Todos los tipos de documentos de negocio son representados mediante diagramas de clases. Los documentos de negocio son identificados previamente en la *vista de procesos de negocio colaborativos* y referenciados en la *vista de protocolos de interacción*. El lenguaje UP-ColBPIP no proporciona conceptos para definir la estructura semántica de los documentos de negocio. En su lugar se debería seleccionar y utilizar un lenguaje

apropiado para ello. No obstante, la representación semántica de los documentos de negocio está fuera del alcance de este libro.

#### 3.2.2.3. Diseño del proceso de integración

La actividad *diseño del proceso de integración* de las organizaciones consiste en llevar a cabo la definición y derivación de un modelo de proceso de integración a partir de un modelo de proceso colaborativo. Esto implica que esta actividad se realiza varias veces en paralelo por las diferentes organizaciones (figura 3.2). Por cada proceso colaborativo, cada una de las organizaciones involucradas debe realizar esta actividad para generar su correspondiente modelo de proceso de integración.

El *diseño de un proceso de integración* implica la definición, desde el punto de vista de una organización, tanto de las actividades públicas que soportan el envío y recepción de mensajes (documentos de negocio) con otras organizaciones, como de las actividades privadas que procesan los documentos de negocio recibidos y los generados que serán intercambiados a través de los mensajes. Para representar los modelos de procesos de integración se utiliza el lenguaje BPMN.

Estos modelos se crean mediante la aplicación de un método MDD propuesto en (48), el cual permite generar un modelo BPMN del proceso de integración de una organización a partir de un modelo de proceso colaborativo definido con el lenguaje UP-ColBPIP. Esto es, un modelo de protocolo de interacción, que representa un proceso colaborativo, se utiliza como modelo de entrada para generar un diagrama de proceso BPMN que representa un proceso de integración de una organización involucrada en el proceso colaborativo.

Las reglas de transformación definidas en el método (48) agregan las actividades públicas del proceso de integración a partir de los mensajes contenidos en el protocolo de interacción. Las actividades privadas del proceso, que soportan el intercambio de mensajes, son generadas utilizando reglas de transformación basadas en la teoría de patrones de actividades (85).

La figura 3.7 muestra un ejemplo del proceso de integración, el cual representa a la organización B en el proceso "Pronóstico de demanda colaborativo". En este diagrama por cada flujo de mensaje en el protocolo de interacción se genera una tarea de tipo *send* o *receive*. Además, se agregan tareas de tipo *service* (tareas automáticas ejecutadas por una aplicación) o *user* (tareas de *workflow* realizadas por una persona a través de una aplicación) que permiten soportar el intercambio de información durante el proceso. Por ejemplo: posterior a la tarea de tipo *receive*, "request ForecastRequest", se agrega una tarea de tipo *user*, "evaluate ForecastRequest", mediante la cual se evaluará el documento de negocio recibido en la tarea previa y permitirá determinar el posible camino a seguir en el proceso de integración.

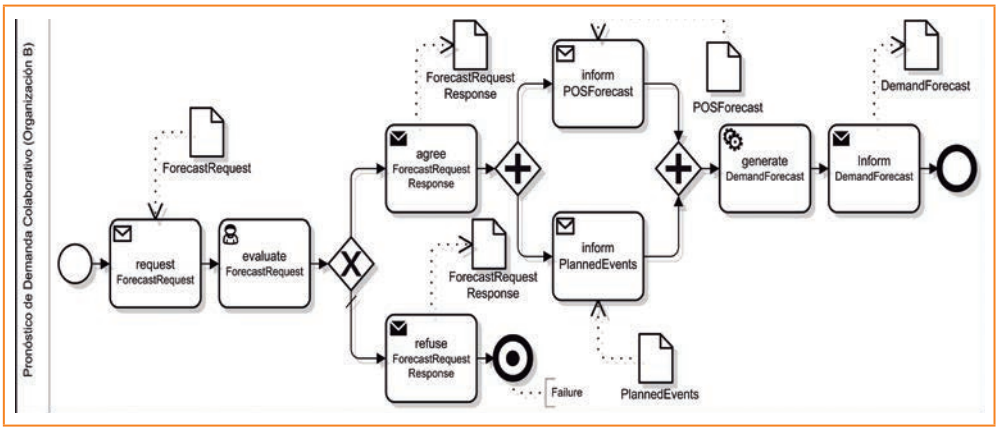



Figura 3.7. *Proceso de integración definido con el lenguaje BPMN.*

### 3.2.3. Generación de la solución tecnológica

La fase *generación de la solución tecnológica* está conformada por las actividades: *generación de modelo de proceso ejecutable* y *generación del código de los agentes*, como se muestra en la figura 3.2. La solución tecnológica apunta a posibilitar la ejecución descentralizada de los procesos colaborativos mediante la implementación, en cada organización, de procesos de integración que son ejecutados a través de agentes Estos agentes representan los sistemas de información interorganizacionales. Un modelo de proceso ejecutable se refiere a la implementación de un proceso de integración. Un agente orientado a procesos está compuesto de un motor enfocado en los procesos, a través del cual se interpreta un modelo de proceso ejecutable para realizar la ejecución de un proceso de integración del rol de la organización que representa. Los modelos de esta fase son definidos usando conceptos de una plataforma de implementación específica.

Esta fase y sus actividades son realizadas en paralelo por cada organización involucrada en los procesos colaborativos, las cuales tienen que estar de acuerdo en la tecnología o plataforma que se utilizará para la implementación. El resultado de esta fase son los artefactos de implementación: los modelos de procesos ejecutables y el código que representa la estructura y comportamiento de los agentes orientados a procesos. Mediante estos artefactos los agentes pueden ser instalados en la plataforma de agentes de software propuesta en el capítulo 4, y llevar a cabo la ejecución de los procesos colaborativos.

### 3.2.3.1. Generación de modelo de proceso ejecutable

La actividad *generación de modelo de proceso ejecutable* consiste en la definición de un modelo de proceso basado en el lenguaje BPMN, que conteniene los parámetros de configuración y la semántica necesaria para su ejecución. Para cada proceso colaborativo, esta actividad debe ser realizada en paralelo por cada organización participante. Por cada organización, se toma como entrada un modelo conceptual (definido en un nivel independiente de la plataforma) de un proceso de integración, el cual representa el comportamiento del rol que la organización desempeña en un proceso colaborativo, y se obtiene como salida un modelo de proceso ejecutable, definido en un nivel específico de la plataforma. En esta investigación se propone que ambos modelos sean especificados utilizando el lenguaje BPMN.

La *generación de modelo de proceso ejecutable* implica la definición del proceso basado en una plataforma específica de implementación, en donde el modelo contiene anotaciones y detalles de implementación en cada uno de sus elementos, los cuales son requeridos para habilitar su ejecución. De esta manera, el modelo generado almacena la información necesaria para ser ejecutado mediante un motor de procesos.

### 3.2.3.2. Generación del código de los agentes

La *generación del código de los agentes* consiste en llevar a cabo la definición y generación de los artefactos de implementación finales de los agentes de software a partir de un modelo conceptual de proceso de integración definido con el lenguaje BPMN. Esto también se realiza en forma paralela por las diferentes organizaciones. El código generado representa la estructura y el comportamiento de un agente orientado a procesos que permite la ejecución de un modelo de proceso. El rol que desempeña un participante (organización) en una colaboración se utiliza para identificar el rol del agente. Para cada participante, un rol es definido con su respectivo comportamiento, el cual se deriva del modelo de proceso de integración correspondiente al participante. Este comportamiento es implementado por un modelo de proceso y se ejecuta a través de un plan definido en el agente, el cual desempeña el rol que le corresponde. Dicho plan consiste en ejecutar el modelo de proceso mediante el motor de procesos enfocado en el agente. De esta manera se expresa la relación entre el rol y su comportamiento en el agente. Los mecanismos de interacción son agregados en el código del agente a partir de eventos de mensajes definidos en el modelo de proceso de integración.

## 3.3. Adaptación de la metodología para colaboraciones interorganizacionales dinámicas

La metodología propuesta previamente se ajusta adecuadamente a colaboraciones interorganizacionales con un enfoque tradicional o estático, que se caracterizan porque las negociaciones y acuerdos de colaboración se realizan cara a cara entre las organizaciones participantes para definir los procesos colaborativos a ejecutar. Es por ello que la primera fase implica la definición del acuerdo de colaboración, previo a la definición de los procesos colaborativos.

No obstante, para proveer una metodología de desarrollo de soluciones tecnológicas basadas en agentes de software para colaboraciones interorganizacionales dinámicas, se propone una adaptación de la metodología presentada previamente. La figura 3.8 es comparativa, y muestra las fases de la metodología presentada para colaboraciones interorganizacionales estáticas, y cuáles fases son usadas para la metodología de colaboraciones interorganizacionales dinámicas.

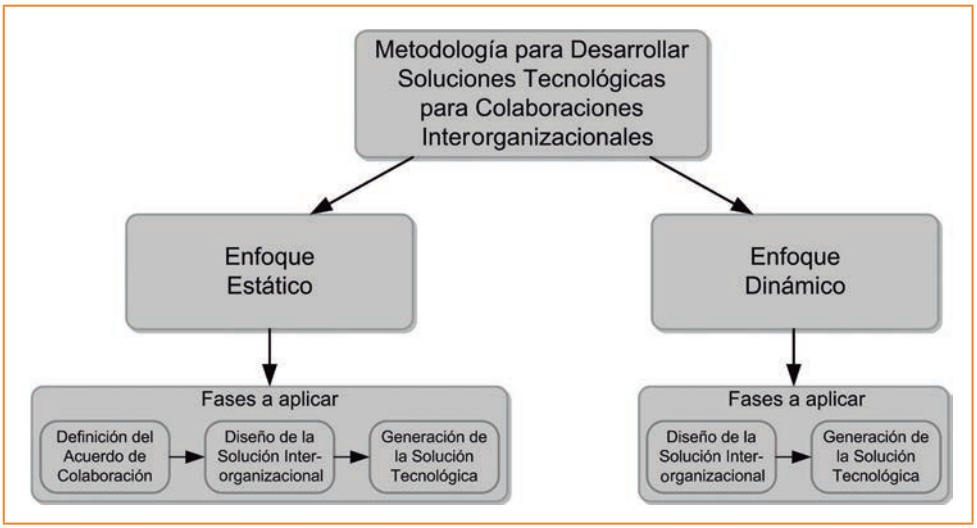



Figura 3.8. *Aplicación de la metodología en ambientes de colaboración.*

Los ambientes de colaboraciones interorganizacionales dinámicas no se caracterizan por un acuerdo predefinido, sino que la colaboración se establece a través de una negociación en forma electrónica entre las partes para definir los procesos colaborativos a ejecutar. La negociación se basa en el intercambio de modelos predefinidos de dichos procesos, que las organizaciones proponen, analizan y/o modifican, hasta llegar a aceptarlas. Los modelos de procesos colaborativos que se intercambian pueden haber sido definidos

*ad hoc,* es decir, en forma conjunta por las partes o en forma individual por una de las partes, o bien recuperados por una de las partes de repositorios públicos o privados.

Por lo tanto, la metodología para colaboraciones dinámicas parte de la fase de *diseño de la solución interorganizacional*, y ahí mismo se aplica sólo la actividad de *diseño del proceso de integración*, que se identifica por tener como entrada un modelo de proceso colaborativo.

A partir de la fase *generación de la solución tecnológica* se aplican sus dos actividades: generación del modelo de proceso ejecutable y generación del código de los agentes, los cuales usan como entrada al modelo de la solución interorganizacional representado por el modelo conceptual del proceso de integración generado a su vez en la fase anterior de la metodología, tal como se muestra en la figura 3.9.

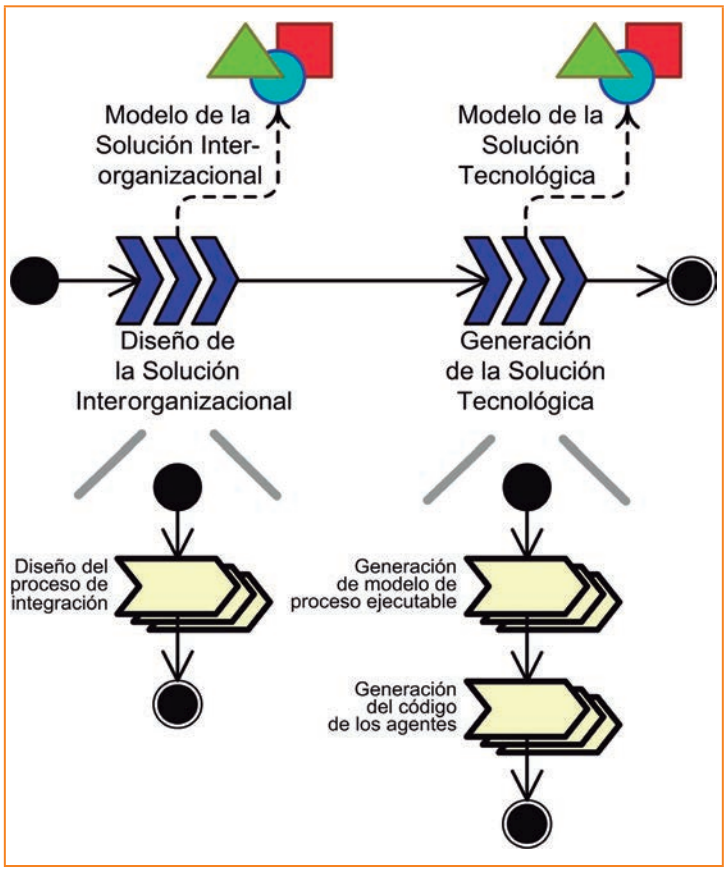



Figura 3.9. *Fases de la metodología aplicada en colaboraciones interorganizacionales dinámicas.*

En las colaboraciones interorganizacionales dinámicas, la solución tecnológica se tiene que generar en el tiempo de la ejecución de la plataforma para dar soporte a la

implementación de nuevos procesos colaborativos o a la adaptación de procesos colaborativos, con el propósito de soportar cambios en los requerimientos de negocio.

Por tal motivo, se requiere automatizar las actividades de las fases de la metodología para colaboraciones interorganizacionales dinámicas. Para ello, los métodos MDD que soportan las actividades de estas fases deben ser implementados por motores de transformación de modelos. Para dar soporte a esto, en el capítulo 4 se propone elaborar agentes de software para conformar una plataforma que tiene la responsabilidad de ejecutar las transformaciones de modelos requeridas por la metodología.

### 3.3.1. Métodos de desarrollo dirigido por modelos para colaboraciones interorganizacionales dinámicas

A continuación se describen los métodos MDD propuestos en la presente investigación para dar soporte a las actividades de las fases de la metodología para colaboraciones interorganizacionales dinámicas.

La figura 3.10 muestra los métodos MDD que se proponen para generar soluciones tecnológicas en ambientes de colaboración interorganizacionales dinámicos. Ahí se detallan los niveles de abstracción de modelos de acuerdo a MDA, que van desde el PIM hasta el código. También se detallan los artefactos (modelos y código) que se generan en cada fase. Para cada uno de ellos se indica la transformación que se realiza, el modelo y el nivel desde donde se parte y el modelo o código que resulta y el nuevo nivel al que corresponde. Mediante la transformación T1 (figura 3.10) se genera un modelo conceptual de proceso de integración de una organización a partir de un modelo de proceso colaborativo; es una transformación horizontal PIM-a-PIM, que utiliza el método propuesto en (48), el cual se describió en la subsección 3.2.2.3.

La transformación modelo-a-modelo T2 apunta a generar un modelo BPMN ejecutable, el cual representa la implementación de un proceso de integración, usando como entrada un modelo BPMN. Por un lado, el modelo BPMN del proceso de integración conforma un modelo conceptual en un nivel independiente de la plataforma PIM con alto nivel de abstracción. Por otro lado, el modelo BPMN de proceso ejecutable se define en un nivel PSM utilizando conceptos específicos de la plataforma de ejecución del proceso que se seleccione. El modelo generado es replicado a nivel de código porque es usado también como artefacto de implementación para realizar la ejecución del mismo, tal como se muestra en la figura 3.10.

El código de un agente orientado a procesos es generado mediante dos métodos dirigidos por modelos. El primer método aplica la transformación modelo-a-modelo T3 (figura 3.10), generando un modelo de salida basado en la plataforma de implementación de agentes que se seleccione, utilizando un modelo de proceso de integración como entrada. El modelo generado es definido en un nivel específico de la plataforma PSM utilizando conceptos de la plataforma de implementación de agentes. Las reglas

de transformación definidas en el método agregan un agente a partir del participante contenido en el modelo de proceso de integración. El plan que implementará el agente se deriva del nombre del proceso que acordaron las organizaciones.

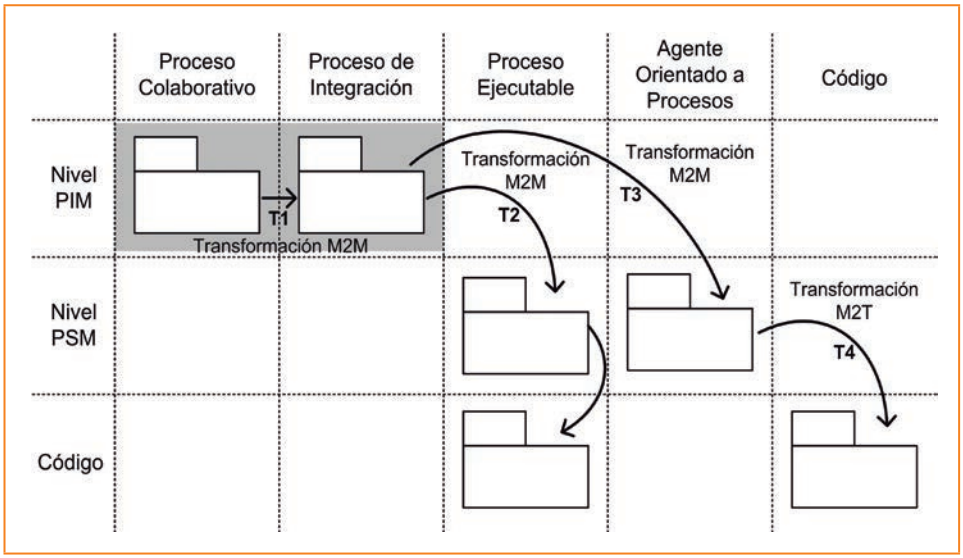



Figura 3.10. *Métodos dirigidos por modelos para colaboraciones interorganizacionales dinámicas.*

Los eventos de mensaje del agente son generados a partir de las tareas de enviar o recibir contenidas en el proceso de integración. Los eventos de mensaje representan la lógica de interacción del agente y habilitan los mecanismos de comunicación con los agentes que desempeñan otros roles en el proceso colaborativo. La meta definida entre los participantes es utilizada para generar la meta que debe alcanzar el agente mediante la ejecución de sus planes y eventos de mensaje.

El segundo método se lleva a cabo mediante la transformación directa modelo-a-código T4 (figura 3.10), que consiste en la generación de un documento con el código fuente que representa la estructura y comportamiento de un agente orientado a procesos en el formato de implementación de la plataforma de agentes de software seleccionada. Este método utiliza como entrada al modelo del agente, generado en la transformación previa, y se obtiene un documento ejecutable con el código que posibilita la implementación del agente.

En resumen, los métodos anteriores posibilitan generar soluciones tecnológicas basadas en agentes de software a partir de modelos de procesos colaborativos intercambiados entre las organizaciones. La figura 3.11 presenta, desde el punto de vista de los modelos y de la colaboración interorganizacional, el proceso a seguir para generar una solución tecnológica.

En primer lugar, a partir de un modelo de proceso colaborativo descrito como un protocolo de interacción, cada organización se centra en generar el modelo de proceso de integración del rol que esta organización desempeña en el modelo del proceso colaborativo.

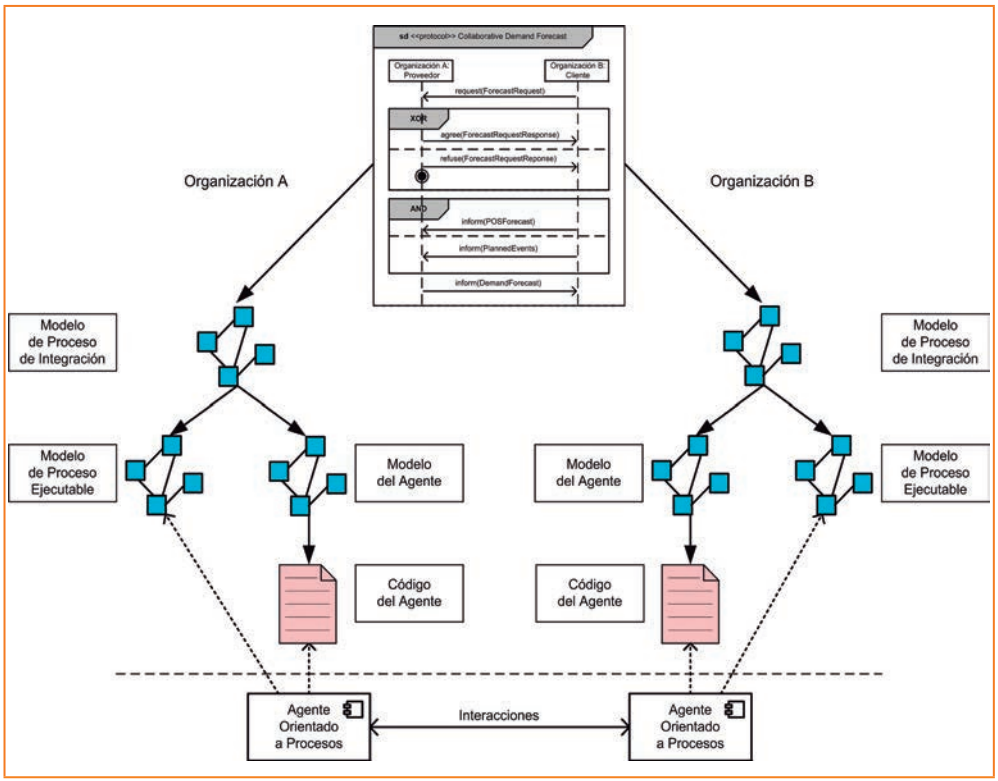



Figura 3.11. *Generación automática de un agente orientado a procesos mediante métodos de desarrollo dirigido por modelos.*

Luego, en forma paralela, un modelo de proceso ejecutable y un modelo de agente son construidos a partir del modelo de proceso de integración. Este procedimiento es llevado a cabo en forma independiente por cada organización participante en la colaboración. A continuación, el modelo del agente es transformado en un documento que contiene el código de un agente, el cual representa el artefacto final del mismo.

Por lo tanto, la metodología propuesta, a través de los métodos MDD definidos, utiliza modelos conceptuales de procesos de integración, generados a partir de procesos colaborativos para construir la estructura de agentes orientados a procesos que ejecutan los modelos de procesos, tal como se muestra en la figura 3.11. Esto permite lograr una alineación entre las soluciones tanto a nivel interorganizacional como a nivel tecnológico, ya que el comportamiento de los artefactos finales de desarrollo (la solución tecno-

lógica) se ajustan al comportamiento definido en los modelos conceptuales de procesos (solución interorganizacional). De esta manera se garantiza que la solución tecnológica ofrezca un soporte completo a la ejecución de los procesos colaborativos. Los métodos MDD de las transformaciones de modelos T2, T3 y T4 (figura 3.10) propuestos se describen en detalle en el capítulo 5.

# 4. Plataforma de gestión de colaboraciones interorganizacionales dinámicas

En este capítulo se describe la plataforma basada en agentes de software propuesta para la gestión de colaboraciones interorganizacionales dinámicas. En primer lugar se presentan los requerimientos funcionales identificados para la plataforma (sección 4.1). Se describe la arquitectura de la plataforma junto con los agentes considerados como necesarios para cumplir con los requerimientos, detallando las dependencias de metas, planes y recursos existentes entre dichos agentes (sección 4.2). Aquí también se expone la arquitectura interna de los agentes. Se presentan las interacciones definidas entre los agentes que les permiten alcanzar las metas asignadas (sección 4.3). Finalmente, se presenta la implementación de la plataforma de acuerdo con la arquitectura de agentes (sección 4.4).

## 4.1. Requerimientos funcionales de la plataforma

Con el propósito de dar soporte a la gestión de colaboraciones interorganizacionales dinámicas, se propone una plataforma basada en agentes de software que habilita a las organizaciones a negociar acuerdos de colaboración en forma electrónica, definiendo los procesos colaborativos a ejecutar y posibilitando su ejecución. En la tabla 4.1 se detallan las principales funcionalidades identificadas para la gestión de colaboraciones interorganizacionales dinámicas, a las que da soporte la plataforma de agentes.

El primer requerimiento es que la plataforma debe permitir a las organizaciones definir y alcanzar acuerdos de colaboración en forma electrónica a través de procesos de negociación, en donde las partes establecen los procesos colaborativos a ejecutar. Para conocer y analizar los procesos colaborativos a acordar, la plataforma debe permitir a las organizaciones intercambiar modelos conceptuales de dichos procesos. Además, se requieren mecanismos de negociación para lograr que las partes arriben o no a un acuerdo en forma correcta.

El segundo requerimiento es dar soporte a la ejecución de los procesos colaborativos acordados. Como se mencionó en los capítulos 1 y 3, la gestión descentralizada de procesos colaborativos puede ser alcanzada a través de la ejecución distribuida de los procesos de integración de las partes de la colaboración. Es por ello que la plataforma requiere dar soporte a la interpretación y ejecución de modelos de procesos de integración. Aplicando la teoría de sistemas de información orientados a procesos, es posible proveer a la plataforma de agentes orientados a procesos, basados en un motor enfocado en procesos, que les permita interpretar modelos de procesos y llevar a cabo la ejecución de instancias de los mismos.

| | Requerimientos |
|---|---|
| 1 | Ejecutar procesos de negociación en forma electrónica que permitan establecer acuerdos de colaboraciones interorganizacionales dinámicas.<br>1.1. Permitir el intercambio de modelos de procesos colaborativos entre las organizaciones.<br>1.2. Permitir que las partes arriben a un acuerdo acerca de los procesos colaborativos a ejecutar. |
| 2 | Ejecutar en forma descentralizada procesos colaborativos a través de la ejecución distribuida de los procesos de integración de las organizaciones.<br>2.1. Interpretar modelos de procesos de integración ejecutables.<br>2.2. Gestionar instancias (ejecución) de modelos de procesos de integración mediante un motor de procesos. |
| 3 | Gestionar para cada organización un repositorio local de modelos de procesos colaborativos intercambiados, es decir, aquellos que ha acordado ejecutar con otras organizaciones. |
| 4 | Generar, en forma automática en la plataforma, la implementación que permite ejecutar un proceso colaborativo, el cual fue acordado en una colaboración interorganizacional dinámica. A partir de modelos conceptuales de procesos colaborativos se requiere:<br>4.1. Generar modelos de procesos de integración ejecutables para cada organización involucrada en un proceso colaborativo.<br>4.2. Generar el código de implementación de agentes orientados a procesos, que representan el sistema de las organizaciones, y que ejecutarán los modelos de procesos de integración ejecutables. |
| 5 | Mediar el intercambio de información requerido en la ejecución de los procesos colaborativos con los sistemas internos de la organización.<br>5.1. Gestionar la generación y procesamiento de los documentos de negocio, que representan la información a intercambiar en los procesos colaborativos, con los sistemas internos de la organización. |

Tabla 4.1. *Requerimientos para la plataforma de gestión de colaboraciones interorganizacionales.*

El tercer requerimiento es dar soporte a cada organización, a la gestión (almacenamiento, actualización y recuperación) de los modelos de procesos colaborativos que han hecho acuerdos, a través de un repositorio de modelos local de la organización. Además, se requiere dar soporte a la gestión de los modelos de procesos (de integración) ejecutables y al código de los agentes orientados a procesos que los ejecutan. La gestión de modelos de procesos colaborativos por parte de las organizaciones tiene como propósito de reutilizar dichos modelos para establecer nuevas colaboraciones interorganizacionales dinámicas.

El cuarto requerimiento se refiere a la funcionalidad necesaria para generar los artefactos de implementación que permiten ejecutar procesos colaborativos. Dos artefactos de implementación se requieren en cada organización que participa en un proceso co-

laborativo. Un artefacto es el modelo de proceso ejecutable, que implemente el proceso de integración de la organización, y que pueda ser interpretado y automatizado por un motor de procesos. Otro artefacto es la implementación del agente, esto es, el código del agente orientado a procesos, con un motor de procesos enfocado que interprete el modelo del proceso para ejecutarlo e interactuar con los agentes de las organizaciones involucradas en el proceso colaborativo. Para guiar la generación de estos artefactos en cada una de las organizaciones a partir de un modelo conceptual de procesos colaborativos, en este libro se propusieron la metodología y los métodos MDD definidos en el capítulo 3, secciones 3.2 y 3.3. Por lo tanto, este requerimiento es para que la plataforma provea el soporte a la automatización de estos métodos, para realizar las transformaciones de modelos que generan los artefactos de implementación. Esto también es un aspecto importante para poder establecer colaboraciones dinámicas, ya que cuando las organizaciones acuerdan los procesos colaborativos en tiempo de ejecución tienen que generar implementaciones y desplegar los sistemas.

Finalmente, un aspecto importante en la ejecución de procesos de integración es la implementación de las tareas internas que permiten crear los documentos de negocio que se envían y los que se reciben. Estas tareas pueden ser ejecutadas por personas, quienes requieren de un soporte para crear o modificar documentos de negocio, o bien por los sistemas internos de las organizaciones. Esto implica la necesidad de integrar personas y los agentes orientados a procesos con los sistemas. Esto requiere un componente que permita ejecutar tareas realizadas por personas para manipular documentos de negocio. Además requiere de un componente que realice funciones de mediador entre los sistemas internos y los componentes de la plataforma, de manera que se puedan interpretar los requerimientos de información por el agente que ejecuta un proceso de integración. El quinto requerimiento definido se refiere a estos aspectos de manejo de los documentos de negocio para intercambiar en la ejecución de procesos colaborativos.

## 4.2. Arquitectura de la plataforma basada en agentes de software

Para cumplir con los requerimientos antes mencionados, se propone crear una plataforma para la gestión de colaboraciones interorganizacionales dinámicas (A4IOC, *Agents for InterOrganizational Collaborations*). La arquitectura de la misma se definió aplicando la metodología de análisis y diseño de sistemas multiagentes TROPOS (15), la cual sugiere que el sistema se estructure con los agentes denominados subagentes (15, 65). La arquitectura global de la plataforma está representada por sistemas multiagentes que son desplegados en cada una de las organizaciones. Los agentes que integran la plataforma están basados en el modelo BDI (*Belief-Desire-Intention*). La arquitectura de estos sistemas multiagentes se construye a través de la delegación de las metas de la plataforma para los agentes. Esta arquitectura se compone de cinco agentes, cuyas responsabilidades se describen a continuación.

- *Agente AdministradorDeColaboraciones* (AC). Este agente desempeña un rol central en la plataforma ya que varias funcionalidades están a su cargo. Representa a una organización y es responsable de establecer comunicaciones con agentes AC de otras organizaciones, con el objetivo de establecer un acuerdo de colaboración que permita la ejecución de procesos colaborativos. El proceso de negociación entre agentes AC para establecer un acuerdo de colaboración es alcanzado mediante la ejecución de protocolos de interacción predefinidos. A través de éstos, el agente AC. gestiona el intercambio de modelos de procesos colaborativos con otros agentes AC. Además, es responsable de crear para cada organización instancias de un *agente AdministradorDeProceso* (AP). Las responsabilidades delegadas al agente AC están centradas en satisfacer el primer requerimiento descrito en la tabla 4.1.

- *Agente AdministradorDeProceso* (AP). Es responsable de desempeñar el rol que una organización cumple en la ejecución de un proceso colaborativo. Para desempeñar este rol interpreta un modelo de proceso de integración ejecutable mediante un motor de procesos centrado en el agente. Un agente AP se inicia por un agente AC al establecer un acuerdo de colaboración. Durante el tiempo en que el acuerdo de colaboración está vigente, una instancia de un agente AP de cada organización está disponible para ejecutar una nueva instancia del proceso colaborativo acordado. Mediante la ejecución coordinada de instancias de agentes AP se satisface el segundo requerimiento detallado en la tabla 4.1. El agente AP, al tener un motor enfocado en procesos, es un agente que cumple con las características de los sistemas de información orientados a procesos.

- *Agente AdministradorDeModelos* (AM). Es responsable de gestionar, almacenar en repositorios locales de esa organización, y recuperar los modelos de procesos de negocio de las colaboraciones. Estos modelos se refieren a modelos de procesos colaborativos, procesos de integración, procesos (de integración) ejecutables. Además es responsable de almacenar y recuperar el código de implementación de los agentes AP. El agente AM permite satisfacer el tercer requerimiento de la tabla 4.1.

- *Agente GeneradorDeImplementaciones* (GI). Es responsable de realizar las transformaciones de modelos de procesos y generar los artefactos de implementación, esto es, los modelos de procesos ejecutables y el código de los agentes AP que los interpretan. Para ello, este agente automatiza los métodos MDD para colaboraciones interorganizacionales dinámicas (capítulo 3). Las funciones que desempeña el agente GI están enfocadas en satisfacer el cuarto requerimiento presentado en la tabla 4.1.

- *Agente IntegradorDeSistemas* (IS). Es responsable de la integración de los sistemas internos de la organización y la plataforma, lo cual permite recuperar la información

generada por los sistemas internos de la organización y/o actualizar las bases de datos del sistema interno con la información intercambiada en la ejecución de los procesos colaborativos. Esta información es representada mediante documentos de negocio. Las responsabilidades delegadas al agente IS posibilitan cumplir con el quinto requerimiento detallado en la tabla 4.1.

Los agentes definidos en la plataforma se clasifican en dos tipos: estáticos y dinámicos. Los *agentes estáticos* se refieren a agentes cuyo comportamiento es definido en tiempo de diseño e implementación. Estos agentes son provistos durante la instalación de la plataforma en cada organización. Las funciones del comportamiento de estos agentes se mantienen sin cambios en tiempo de ejecución de la plataforma, y sólo sus creencias son modificadas durante la ejecución del agente. La plataforma provee agentes estáticos para dar soporte a todos los requerimientos definidos previamente, excepto al segundo requerimiento. El uso de agentes estáticos es un enfoque tradicional utilizado en el desarrollo de sistemas multiagentes. Los agentes estáticos definidos en la plataforma son: agente *AdministradorDeColaboraciones* (AC), agente *AdministradorDeModelos* (AM), agente *GeneradorDeImplementaciones* (GI) y agente *IntegradorDeSistemas* (IS). En el sistema multiagente de una organización puede existir una instancia de un agente AC por cada organización con la cual establece una colaboración. Además, existe una única instancia de los restantes agentes estáticos en dicho sistema.

Los *agentes dinámicos* se refieren a un nuevo tipo de agente propuesto en este libro, cuyas funciones, comportamiento, implementación e instancias se generan en tiempo de ejecución de la plataforma. Las funciones de estos agentes no se conocen de antemano en tiempo de diseño. La implementación de estos agentes puede ser generada en tiempo de ejecución a través de la automatización de métodos MDD que soporten transformaciones de modelos que generan los artefactos y el código de implementación de los agentes. En la plataforma propuesta, el tipo de agente *AdministradorDeProceso* (AP) es un agente dinámico, el cual a su vez es orientado a procesos, ya que tiene un motor enfocado en procesos. Estos agentes dinámicos dan soporte a la ejecución de los procesos colaborativos a través de la ejecución de los procesos de integración. La lógica y comportamiento de estos agentes se define en un modelo de proceso ejecutable, en el que se basan para gobernar y realizar sus actividades y cumplir con la meta de ejecución de un proceso de integración.

Los procesos colaborativos no están definidos de antemano, por tal motivo, no es posible tener predefinidos en la plataforma los agentes para dichos procesos. Por lo tanto, el modelo de proceso ejecutable y el agente dinámico que lo ejecuta se generan en tiempo de ejecución, a través de la automatización de los métodos MDD de la metodología propuesta en el capítulo 3, sección 3.3. Existe una instancia de un agente AP en el sistema multiagente de una organización cada vez que se requiere realizar la ejecución del proceso colaborativo al que da soporte el agente.

Por otra parte, los agentes que conforman la plataforma se pueden clasificar en agentes interorganizacionales (*front-end*) y agentes intraorganizacionales (*back-end*). Los primeros representan a las organizaciones y tienen las responsabilidades de llevar a cabo la colaboración interorganizacional entre las mismas (figura 4.1). Permiten establecer de acuerdos de colaboración y la ejecución de procesos colaborativos. Los tipos de agentes interorganizacionales de la plataforma son: el agente *AdministradorDeColaboraciones* (AC) y el agente *AdministradorDeProceso* (AP).

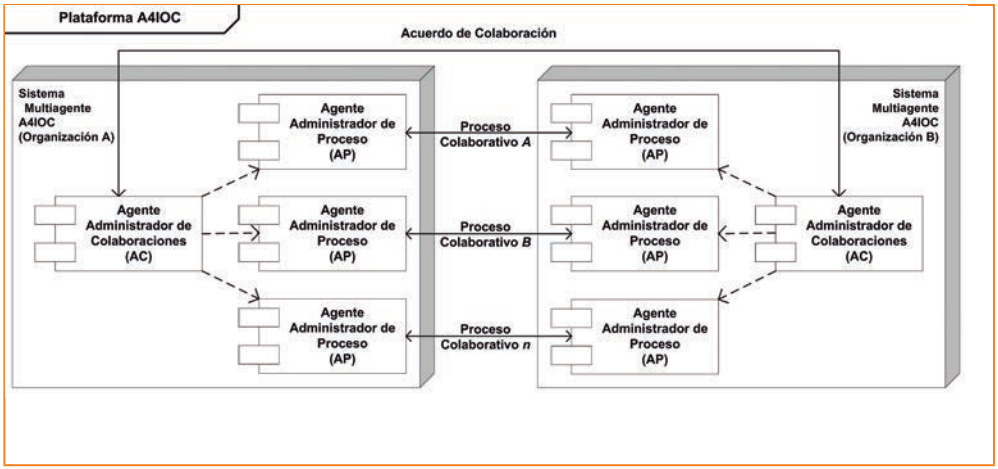



Figura 4.1. *Arquitectura global de la plataforma A4IOC compuesta de sistemas multiagentes.*

La figura 4.1 muestra una representación de los agentes interorganizacionales que componen la plataforma. En este caso, la plataforma consiste en dos sistemas multiagentes, cada uno desplegado en una de las organizaciones, los cuales contienen los agentes interorganizacionales de las organizaciones. Los agentes AC de las organizaciones llevan adelante un proceso de negociación, en donde interactúan para establecer el acuerdo de colaboración acerca de los procesos colaborativos a ejecutar. En esta negociación, por cada proceso colaborativo acordado entre las partes, los agentes AC se encargan de generar e instanciar los agentes AP que ejecutarán el rol de la organización en los procesos colaborativos acordados. Luego, los agentes AP de las organizaciones, que implementan un proceso colaborativo, interactúan de acuerdo a lo establecido en sus modelos de procesos ejecutables, realizando de esta manera la ejecución descentralizada y distribuida del proceso colaborativo.

Los agentes intraorganizacionales se comunican sólo con los agentes de la organización a la que pertenecen. Son responsables de realizar tareas que los agentes interorganizacionales les delegan, tales como la gestión de los documentos de negocio, la invocación a sistemas internos, la gestión de los modelos de procesos y la automatización de los métodos MDD de transformaciones de modelos. En la tabla 4.2 se presenta un resumen de la clasificación de los agentes definidos en la plataforma.

| Agente | Estático | Dinámico | *Front-end* | *Back-end* |
|---|---|---|---|---|
| Administrador de colaboraciones | • | | • | |
| Administrador de proceso | | • | • | |
| Administrador de modelos | • | | | • |
| Generador de implementaciones | • | | | • |
| Integrador de sistemas | • | | | • |

Tabla 4.2. *Clasificación de los agentes de software que integran la plataforma A4IOC.*

### 4.2.1. Arquitectura del sistema multiagente de la plataforma

Siguiendo la metodología Tropos, los requerimientos funcionales definidos para la plataforma son representados mediante metas duras (*hard goals*) (15, 65), las cuales son delegadas a los agentes definidos anteriormente. Esto es representado en el diagrama de actores que muestra la figura 4.2. Un *diagrama de actores* describe las relaciones de dependencia entre el conjunto de actores (agentes) del sistema. La relación de dependencia en este diagrama se detalla mediante dependencias de metas o recursos entre los agentes del sistema. Este diagrama está basado en la fase *diseño arquitectónico* del enfoque metodológico Tropos (15).

Como se mencionó previamente, la plataforma está compuesta de sistemas multiagentes, cada uno de ellos representan a una organización. El actor A4IOC que representa al sistema multiagente de una organización, mostrado en la parte superior de la figura 4.2, delega las metas que representan los requerimientos (definidos en la sección 4.1) a los diferentes agentes que componen el sistema.

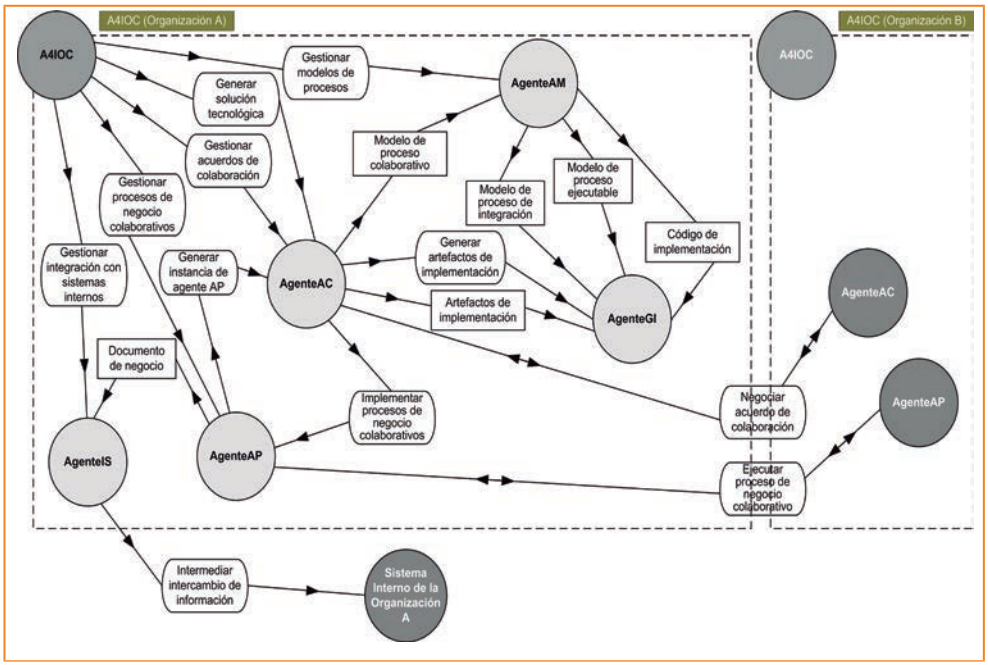



Figura 4.2. *Diagrama de actores de la plataforma A4IOC.*

El actor A4IOC delega al agente *AdministradorDeColaboraciones* (AC) la meta de *gestionar acuerdos de colaboración,* el cual habilita a este agente a establecer acuerdos de colaboración con otro agente AC, que representa otra organización participante en la colaboración. Entonces, un agente AC depende de otro agente AC para alcanzar esta meta, por medio de la submeta *negociar acuerdo de colaboración*, la cual es representada por una relación de dependencia entre los agentes AC en ambos sentidos. El agente AC depende del agente AM para obtener los modelos de procesos colaborativos y depende del agente GI para obtener los artefactos de implementación que permiten la ejecución de los procesos colaborativos acordados. Esto es indicado mediante los elementos de recurso *modelo de proceso colaborativo* y *artefactos de implementación*. El agente AM tiene delegada la meta *gestionar los modelos de procesos*.

La meta *gestionar procesos de negocio colaborativos* es delegada a un agente *AdministradorDeProceso* (AP). Para cumplir con esta meta, previamente el agente AC debe crear una instancia de un agente AP, para lo cual el agente AC delega la responsabilidad de implementar procesos colaborativos al agente AP mediante la submeta *implementar procesos de negocio colaborativos*, la cual contribuye a satisfacer la meta principal *gestionar procesos de negocio colaborativos* del agente AP. El agente AP tiene una relación de dependencia con los agentes AP de otros sistemas que representan a las organizaciones participantes en el proceso colaborativo que ejecutan mediante la submeta *ejecutar proceso de negocio colaborativo*.

Ésta es una submeta de la meta *gestionar procesos de negocio colaborativos*. La relación entre estos agentes se presenta en ambos sentidos de la interacción y en conjunto alcanzan la meta, impactando a las organizaciones involucradas en el proceso colaborativo que gestionan.

También un agente AP tiene una dependencia con el agente IS para obtener o entregar los documentos de negocio desde o hacia los sistemas internos de la organización. Por lo tanto, las interacciones entre los agentes AP y la ejecución coordinada de las tareas mencionadas de cada agente AP permiten satisfacer el requerimiento de ejecutar en forma descentralizada los procesos colaborativos en la plataforma.

Por otro lado, para alcanzar la meta *generar solución tecnológica*, el agente AC delega la submeta *generar artefactos de implementación* al agente GI, para lo cual el agente GI implementa un conjunto de planes para la ejecución en forma secuencial de diferentes transformaciones de modelos. Estas transformaciones generan un modelo de proceso ejecutable y el código del agente AP (artefactos de implementación) para implementar el rol de la organización en el proceso colaborativo acordado en la colaboración. Con lo cual se da soporte al cuarto requerimiento de la plataforma, presentado en la tabla 4.1.

La plataforma depende también de la capacidad de interactuar con los sistemas internos de las organizaciones participantes en la colaboración. Por tal motivo, la meta *gestionar integración con sistemas internos* es delegada al agente IS. Para cumplir con esta meta, el agente IS establece una relación de dependencia con el sistema interno de la organización a través de la meta *intermediar intercambio de información*. Esta meta tiene como propósito implementar los mecanismos que posibiliten recuperar, actualizar o almacenar la información intercambiada al momento de la ejecución de los procesos colaborativos. Las responsabilidades delegadas al agente IS posibilitan cumplir con el quinto requerimiento funcional (tabla 4.1) planteado para la plataforma.

### 4.2.2. Arquitectura de los agentes de la plataforma

El diagrama de actores mostrado en la figura 4.2 es utilizado como base para generar, mediante un refinamiento, un *diagrama de metas*, el cual permite entender mejor cómo serán alcanzadas las metas a través de planes, detallando las relaciones de dependencia entre los agentes de la plataforma. Un *diagrama de metas* representa la perspectiva interna de los agentes, describiendo cómo pueden dividirse las metas de un agente en submetas e identificando los planes y recursos necesarios para satisfacer la meta, así como también explicitando las relaciones de dependencia identificadas entres los agentes del sistema. La figura 4.3 presenta el diagrama de metas del sistema A4IOC, mostrando la división de las metas de los agentes en submetas y/o planes. Un plan representa la forma de llevar a cabo acciones mediante las cuales se puede satisfacer una meta. Además de los planes, en ese diagrama se incluyen los recursos, los cuales para la plataforma

representan los modelos de procesos, el código de los agentes AP o los documentos de negocio que se intercambian entre los agentes.

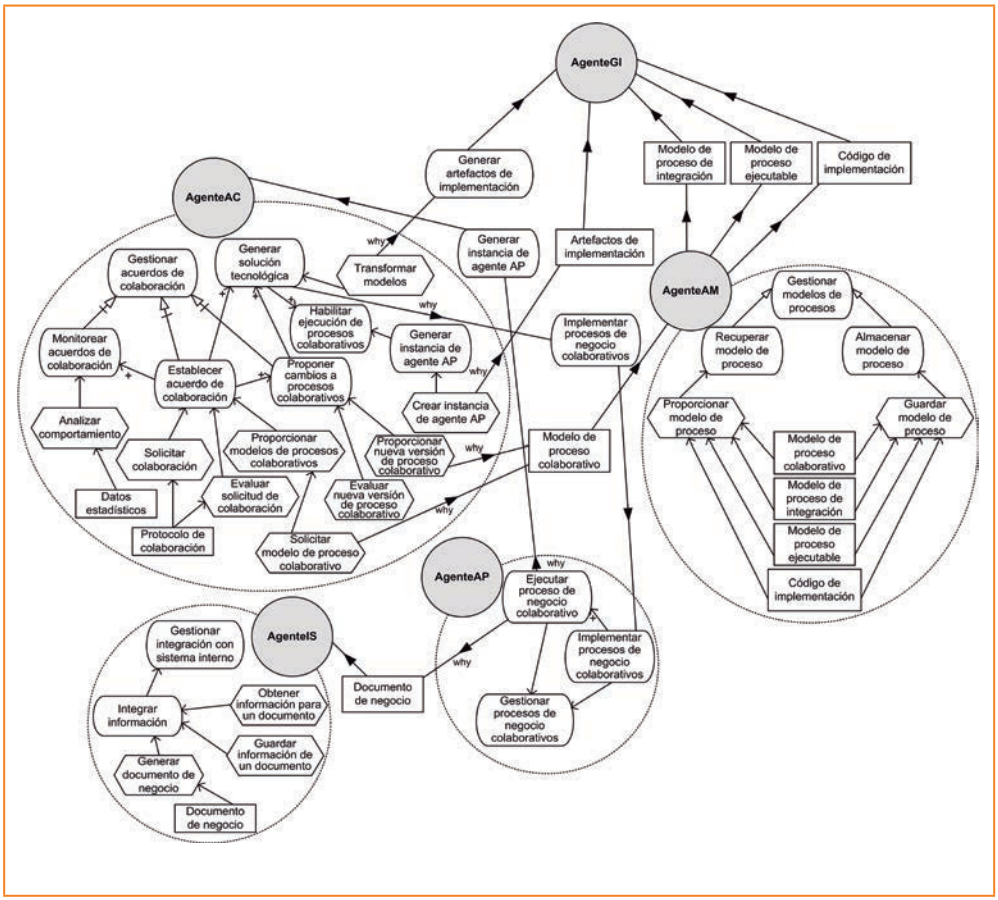



Figura 4.3. *Diagrama de actores de la plataforma A4IOC.*

En el agente AC, la meta *gestionar acuerdos de colaboración* es refinada en las submetas *establecer acuerdo de colaboración*, *monitorear acuerdos de colaboración* y *proponer cambios a procesos colaborativos*, mediante las cuales se intenta satisfacer esta meta. La submeta *establecer acuerdo de colaboración* tiene el objetivo de implementar la comunicación con otros agentes AC y permitir una negociación entre estos agentes. Esta submeta puede ser alcanzada mediante la ejecución de los planes *solicitar colaboración*, *evaluar solicitud de colaboración* y *proporcionar modelos de procesos colaborativos*. Estos planes permiten al agente interactuar, negociar e intercambiar un documento electrónico que describe la intención del acuerdo de colaboración y los procesos colaborativos que se proponen implementar entre las organizaciones. Además, con el fin de establecer una colaboración, el

agente AC necesita acceso al recurso *protocolo de colaboración*, el cual es indicado como un medio para la ejecución de los planes *solicitar colaboración* y *evaluar solicitud de colaboración*. El protocolo de colaboración define la secuencia de interacciones del proceso de negociación entre los agentes AC para establecer un acuerdo de colaboración entre dos organizaciones.

El plan *proporcionar modelos de procesos colaborativos* requiere del recurso *modelo de proceso colaborativo* que es administrado por el agente AM. Para lo cual mediante el subplan *solicitar modelo de proceso colaborativo* se lleva a cabo la recuperación del modelo, existiendo una relación de dependencia entre el agente AC y el agente AM por medio de este recurso. Por lo tanto, la submeta *establecer acuerdo de colaboración* contribuye positivamente a que se alcancen la submeta *monitorear acuerdos de colaboración* y la meta *generar solución tecnológica*, lo cual es indicado mediante una relación de contribución positiva, como se muestra en la figura 4.3.

La submeta *proponer cambios a procesos colaborativos* puede ser alcanzada mediante la ejecución de los planes *proporcionar nueva versión de proceso colaborativo* y *evaluar nueva versión de proceso colaborativo*. El primer plan permite al agente promover una nueva versión de un modelo de proceso colaborativo, con base en cambios o nuevos requerimientos que se han presentado en la organización. Este plan requiere del recurso *modelo de proceso colaborativo* administrado por el agente AM. El segundo plan habilita al agente para analizar los cambios propuestos en un proceso colaborativo y lo que implican en las actividades internas y la información de la organización.

En la meta *generar solución tecnológica* se identifica al plan *transformar modelos* como un medio para satisfacer esta meta. Este plan permite iniciar el proceso de transformación de modelos, delegando la responsabilidad del proceso de generación de los artefactos de implementación (modelo de proceso ejecutable y código del agente AP) al agente GI, mediante la meta *generar artefactos de implementación*, con la cual se establece una relación de dependencia entre el agente AC y el agente GI. Esta meta contribuye positivamente a satisfacer la meta *habilitar ejecución de procesos colaborativos*, llevada a cabo por el agente AC (figura 4.3).

La responsabilidad de la meta *habilitar ejecución de procesos colaborativos* está a cargo del agente AC. Esta meta puede ser alcanzada a través de la submeta *generar instancia de agente AP* y la ejecución del plan *crear instancia de agente AP*, mediante el cual se instancia un agente AP por cada proceso colaborativo acordado. El plan *crear instancia de agente* requiere del recurso *artefactos de implementación*, que es generado por el agente GI. Por tal motivo, se establece una relación de dependencia entre los agentes AC y GI por medio de este recurso, como se muestra en la figura 4.3. El agente AC tiene una relación de dependencia con el agente AP, por medio de la meta *implementar procesos de negocio colaborativos*, la cual es una submeta de la meta *generar solución tecnológica*. Esta meta contribuye a la ejecución del proceso colaborativo mediante una instancia del agente AP.

En el agente AM se delega la meta *gestionar modelos de procesos* y consiste en recuperar o almacenar los modelos de procesos colaborativos que son intercambiados por un

agente AC cuando establece una colaboración con otro agente AC. La meta *gestionar modelos de procesos* está compuesta por las submetas *recuperar modelo de proceso* y *almacenar modelo de proceso*. Estas dos submetas pueden ser consideradas como caminos alternativos para el cumplimiento de la meta *gestionar modelos de procesos*. El agente AM puede cumplir estas submetas por medio de la ejecución de los planes *proporcionar modelo de proceso* y *guardar modelo de proceso*. Estos planes deben tener la capacidad de almacenar o recuperar modelos de procesos colaborativos, modelos de procesos de integración, modelos de procesos ejecutables y código de agentes, los cuales son representados como elementos de recurso en el diagrama mostrado en la figura 4.3. Además, el agente AM mantiene tres relaciones de dependencia de recurso (*modelo de proceso de integración*, *modelo de proceso ejecutable* y *código de implementación*) con el agente GI, debido a que este agente es responsable de generar estos recursos.

La arquitectura del agente AP contiene la meta *gestionar procesos de negocio colaborativos*, la cual es subdividida en la meta *ejecutar proceso de negocio colaborativo* mediante la cual se representa una relación de dependencia, en ambos sentidos, con el agente AP de otra organización participante en la colaboración. También se define la meta *implementar procesos de negocio colaborativos* que contribuye positivamente a alcanzar la meta *ejecutar proceso de negocio colaborativo*. En la arquitectura del agente AP no se incluye mayor detalle debido a que los planes del agente se desconocen al momento del diseño y son construidos en tiempo de ejecución del sistema. Los planes de dichos agentes corresponderán a la ejecución, a través de un motor de procesos, de modelos de procesos ejecutables requeridos para ejecutar los procesos colaborativos.

La *meta gestionar integración con sistemas internos* delegada al agente IS está compuesta por la submeta *integrar información*. Esta submeta requiere de un sistema interno de la organización, estableciendo una relación de dependencia con este sistema mediante la submeta *intermediar intercambio de información*. La submeta *integrar información* puede ser alcanzada a través de los planes *generar documento de negocio*, *guardar información de un documento* y *obtener información para un documento*. El plan *generar documento de negocio* requiere del recurso *documento de negocio* para representar el resultado del plan ejecutado. Los documentos de negocio son el medio por el cual se intercambia la información en los procesos colaborativos. Estos documentos de negocio son generados por el agente IS con base en la información proporcionada por el sistema interno de la organización. Por tal motivo, en los planes se integran mecanismos que permiten convertir los documentos de negocio al formato de intercambio solicitado por el sistema interno (plan *guardar información de un documento*), los mecanismos que habiliten la obtención de la información requerida en los documentos a enviar (plan *obtener información para un documento*) y la generación de estos documentos en el formato de intercambio de la plataforma (plan *generar documento de negocio*).

## 4.3. Interacciones entre los agentes de software que componen la plataforma

En esta sección se describen las interacciones definidas para los agentes estáticos que conforman la plataforma. Las interacciones entre los agentes son modeladas como protocolos de interacción utilizando el lenguaje UP-ColBPIP (88). Mediante diagramas de interacción se describe el comportamiento de las interacciones entre los agentes, definiendo una secuencia permitida de mensajes entre los agentes participantes. Esto permite construir parte de la estructura de cada agente y da lugar a las especificaciones que describen cómo cada agente se comporta con el fin de ejecutar un plan o satisfacer una meta.

### 4.3.1. Interacciones de agentes para establecer una colaboración interorganizacional dinámica

El comportamiento predefinido de las interacciones entre los agentes involucrados para satisfacer la meta *establecer acuerdo de colaboración* se detalla en el protocolo de interacción *Establecer Acuerdo de Colaboración* presentado en la figura 4.4. Dicho protocolo inicia cuando el agente AC de la Organización-A, desempeñando el rol de *iniciador*, envía un mensaje con un *acto de comunicación* de tipo *request* al agente AC de la Organización-B, desempeñando el rol de *receptor*. Este mensaje contiene un documento que describe los términos de un acuerdo de colaboración (por ejemplo: objetivo de la colaboración, alcances del acuerdo, duración de la colaboración, descripción y responsabilidades de cada rol, roles asignados a cada organización) y los procesos colaborativos que se requieren ejecutar para alcanzar una meta de negocio común. En este documento se detallan los nombres con una descripción informal de los procesos colaborativos que se solicita ejecutar.

El agente *receptor* procesa el mensaje, realizando una evaluación de la solicitud mediante una actividad interna del agente. El agente *receptor* puede responder mediante un mensaje *refuse*, con lo cual deniega establecer la colaboración y el protocolo finaliza. De lo contrario, puede responder con un mensaje *agree*, aceptando las condiciones contenidas en el documento propuesto. Cuando el agente *iniciador* procesa el mensaje *agree* recibido, inmediatamente realiza un protocolo de interacción con el agente AM de su organización, para obtener los modelos a enviar de los procesos colaborativos que aceptó ejecutar el receptor (esto se describe en la sección 4.3.2).

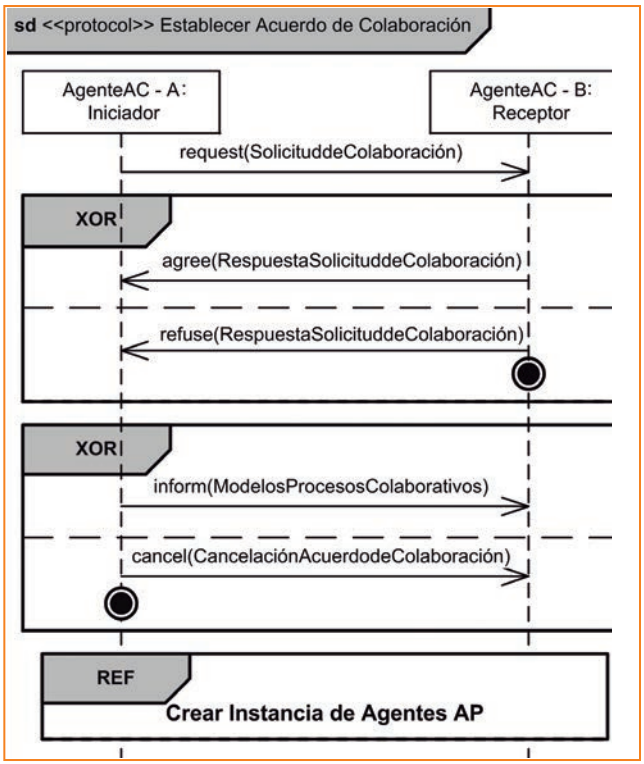



Figura 4.4. *Protocolo de interacción: Establecer acuerdo de colaboración.*

Si dicho protocolo finaliza correctamente, es decir, si el agente AM pudo recuperar y enviar al agente AC los modelos de procesos colaborativos solicitados, el agente AC *iniciador* de inmediato retransmite estos modelos al agente AC *receptor* mediante un mensaje de tipo *inform*. De lo contrario, si no se pudieron recuperar todos los modelos de procesos colaborativos solicitados, el agente AC *iniciador* envía un mensaje *cancel* indicando la cancelación del acuerdo de colaboración y la finalización de las interacciones entre los agentes AC.

Finalmente, los agentes AC de las organizaciones ejecutarán el subprotocolo *Crear Instancia de Agentes AP,* a través del cual se realizará la generación de la solución tecnológica y la creación de las instancias de los agentes AP de cada una de las partes. Dichos agentes darán soporte a la ejecución de los procesos colaborativos acordados (este subprotocolo se describe en la sección 4.3.4).

Las interacciones definidas en el protocolo de interacción *Establecer Acuerdo de Colaboración* conducen a la ejecución de los planes *solicitar colaboración*, *evaluar solicitud de colaboración*, *solicitar modelo de proceso colaborativo* y *proporcionar modelos de procesos colaborativos* (mediante la ejecución del protocolo *Obtener Modelos de Procesos Colaborativos*), así como las acciones que se deben realizar y cómo responder a eventos que se presentan en el

contexto de la ejecución. Mediante la ejecución de estos planes por los agentes AC es posible satisfacer la meta *establecer acuerdo de colaboración*.

El protocolo de interacción detallado en esta sección permite cumplir con los requerimientos de las colaboraciones interorganizacionales dinámicas de establecer la colaboración acordando los procesos colaborativos a ejecutar e intercambiando los modelos de los mismos. Sin embargo, en las colaboraciones interorganizacionales dinámicas también se requiere que todas las partes involucradas puedan proponer modificaciones tanto cuando se establece la colaboración como en los procesos colaborativos acordados.

Por tal motivo, en las siguientes subsecciones se presenta, por un lado, una extensión del protocolo de interacción *Establecer Acuerdo de Colaboración*, que permite proponer cambios al documento de la solicitud de colaboración. Por otro lado, se presenta un protocolo de interacción, que complementa al protocolo presentado en esta sección, el cual habilita a las organizaciones a proponer cambios en los modelos de procesos colaborativos que se ha acordado ejecutar, es decir, negociar nuevas versiones de modelos de dichos procesos o proponer nuevos modelos de procesos colaborativos.

#### 4.3.1.1. Extensión del protocolo *Establecer Acuerdo de Colaboración*

En la figura 4.5 se presenta una extensión de la primera etapa de negociación del protocolo mostrado en la figura 4.4. Este protocolo permite a las organizaciones involucradas en una colaboración, representadas por los agentes AC, realizar cambios en la solicitud de colaboración.

En el protocolo de interacción *Establecer Acuerdo de Colaboración-Extendido*, el agente AC *receptor* tiene un camino adicional para responder a la solicitud de colaboración recibida del agente AC *iniciador* (figura 4.5), mediante el cual tiene la posibilidad de proponer un cambio en el documento de solicitud a través de un mensaje *propose*. Este cambio consiste en agregar procesos colaborativos a ejecutar o eliminar algún proceso con el cual no está de acuerdo en ejecutar la organización B.

Cuando el agente AC *iniciador* recibe una propuesta de cambio en la solicitud de colaboración, realiza una evaluación a los cambios propuestos y puede responder mediante un mensaje *accept-proposal* o *reject-proposal*. En caso de que el agente AC *iniciador* determine no aceptar, enviará un mensaje *reject-proposal*, con lo cual finaliza el proceso de negociación y la solicitud de colaboración es cancelada. En caso contrario, el agente AC *iniciador* responde con un mensaje *accept-proposal*, confirmando que acepta los cambios propuestos en la solicitud de colaboración. Luego, la ejecución del protocolo continúa con la solicitud de los modelos de procesos de colaboración acordados por parte del agente AC *iniciador* al agente AM de su organización. A continuación, el protocolo sigue su ejecución tal como se describió en la subsección 4.3.1 y como se muestra en la figura 4.4.

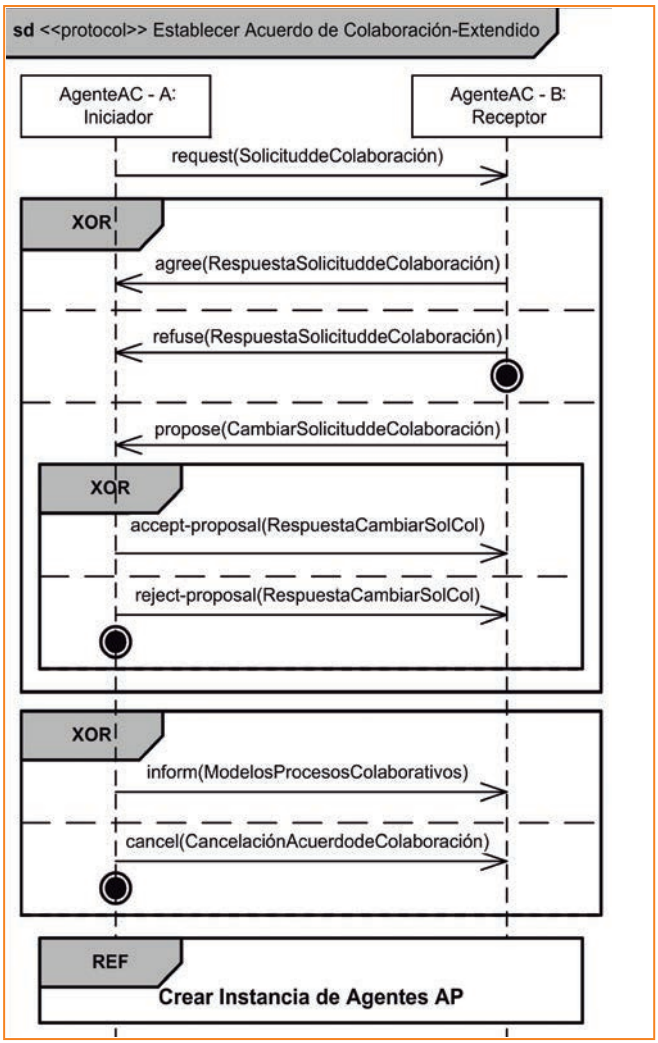



Figura 4.5. *Protocolo de interacción: Establecer acuerdo de colaboración con negociación extendida.*

### 4.3.1.2. Interacciones de agentes para proponer cambios a procesos colaborativos acordados

El comportamiento detallado en el protocolo de interacción *Proponer Cambios en Proceso Colaborativo*, presentado en la figura 4.6, puede ser requerido cuando las organizaciones desean realizar cambios a los procesos colaborativos ya acordados en una colaboración interorganizacional dinámica establecida previamente. Los cambios a los procesos se reflejan a través de nuevas versiones de modelos de dichos procesos. Este protocolo

puede ser ejecutado entre dos agentes AC siempre que hayan ejecutado previamente el protocolo que les permite establecer el acuerdo de colaboración mostrado en la figura 4.4. Las interacciones definidas en este protocolo habilitan la ejecución de los planes *proporcionar nueva versión de proceso colaborativo* y *evaluar nueva versión de proceso colaborativo*, mediante los cuales posibilitan satisfacer la meta *proponer cambios a procesos colaborativos*.

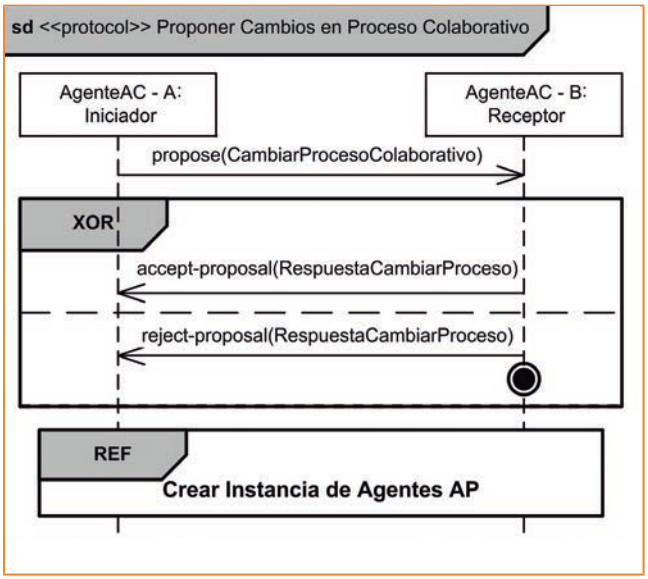



Figura 4.6. *Protocolo de interacción: Proponer Cambios en Proceso Colaborativo.*

El protocolo habilita a una organización, a través de su agente AC, que desempeñará el rol de *iniciador*, a proponer una nueva versión de los modelos de procesos colaborativos acordados al agente AC de otra organización, que desempeñará el rol de *receptor*. En este caso, el agente AC *iniciador* del protocolo puede enviar un mensaje *propose* con una propuesta de cambios en el proceso colaborativo a ejecutar.

Cuando el agente AC *receptor* recibe el mensaje, debe realizar una evaluación a los cambios propuestos, para determinar las implicaciones que éstos tienen en las operaciones de la organización. Posterior a esto, el agente AC *receptor* puede responder con un mensaje *reject-proposal*, con lo cual informa a su contraparte que no está en condiciones de aceptar dichos cambios en los modelos. Ello implica la finalización del proceso de negociación. En caso contrario, el agente AC *receptor* puede responder con un mensaje *accept-proposal*, con lo que notifica que acepta los cambios propuestos en los modelos y que esta versión de los modelos se utilizará en la ejecución del proceso colaborativo.

Luego, cuando los cambios propuestos son aceptados por al agente AC *receptor*, los agentes AC de cada organización ejecutarán el subprotocolo *Crear Instancia de Agentes AP*

para generar la solución tecnológica y la creación de las instancias de los agentes AP de cada una de las organizaciones, correspondientes a la nueva versión de los modelos de procesos colaborativos.

### 4.3.2. Interacciones de agentes para obtener modelos de procesos colaborativos

Las interacciones definidas en el protocolo *Obtener Modelos de Procesos Colaborativos,* que se presenta en la figura 4.7, posibilitan a un agente AC obtener los modelos de procesos colaborativos requeridos en un acuerdo de colaboración a través del agente AM de su organización. Cuando un agente AC que está negociando un acuerdo de colaboración recibe una notificación de aceptación de la solicitud mediante un mensaje *agree*, procede a enviar un mensaje *request* al agente AM requiriendo los modelos de procesos colaborativos que se acordaron ejecutar en el acuerdo de colaboración.

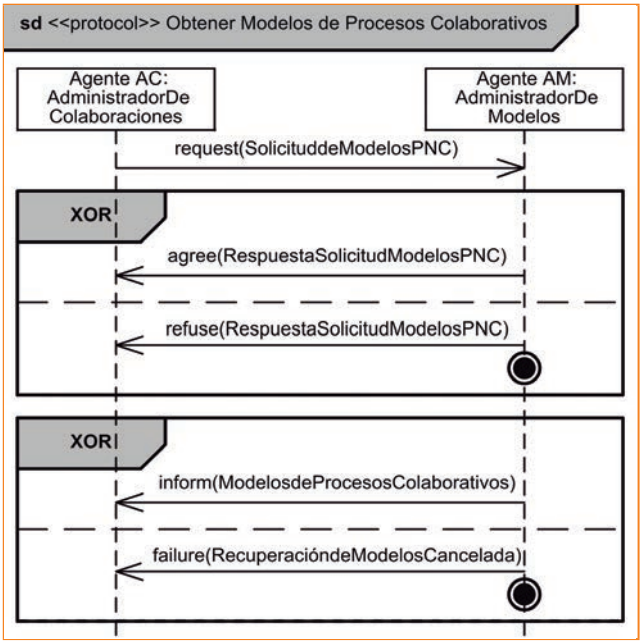



Figura 4.7. *Protocolo de interacción: Obtener Modelos de Procesos Colaborativos.*

El agente AM procesa el mensaje de solicitud y realiza una consulta en su repositorio local de modelos para verificar que tiene almacenados los modelos solicitados, esto se lleva a cabo mediante una actividad interna. En caso de que el agente AM no encuentre los modelos en el repositorio, enviará un mensaje de tipo *refuse* y finaliza la interacción

entre el agente AC y agente AM. En caso contrario, cuando el agente AM encuentra los modelos en el repositorio, responde con un mensaje *agree*, confirmando la aceptación de la solicitud.

Posterior a este mensaje, el agente AM lleva a cabo la recuperación de los modelos de procesos colaborativos, los cuales son enviados en un mensaje *inform* que incluye en su contenido un documento con los modelos de procesos colaborativos solicitados. En caso de presentar alguna falla en la ejecución del procedimiento de recuperación de los modelos, el agente AM enviará un mensaje *failure* al agente AC, notificando dicha falla y finalizando la interacción. El comportamiento de las interacciones del protocolo *Obtener Modelos de Procesos Colaborativos* permite al agente AM alcanzar la meta *gestionar modelos de procesos*.

### 4.3.3. Interacciones de agentes para generar la solución tecnológica

Luego de establecer un acuerdo de colaboración interorganizacional mediante la plataforma de agentes y que se han intercambiado electrónicamente los modelos de procesos colaborativos acordados, se procede a generar la solución tecnológica mediante la ejecución de transformaciones de modelos que permiten derivar los artefactos de implementación. Este proceso es realizado por cada una de las organizaciones involucradas en el acuerdo de colaboración.

El comportamiento de las interacciones requeridas por los agentes para alcanzar la meta *generar solución tecnológica* se presenta en la figura 4.8. En este protocolo, el agente AC solicita al agente GI realizar las acciones necesarias para transformar los modelos de procesos colaborativos que acordó con otra organización. Esto es representado con un mensaje *request* que contiene estos modelos. El agente GI puede responder con un mensaje *agree* o con un mensaje *refuse*, rechazando la solicitud y finalizando la interacción. Cuando el agente GI responde con un mensaje *agree*, aceptando la solicitud de transformación de modelos, procede a ejecutar las transformaciones de modelos por cada modelo de proceso colaborativo recibido.

Al finalizar el proceso de transformaciones, el agente GI notifica al agente AC, mediante un mensaje de tipo *inform*, que las transformaciones fueron realizadas y generadas sus salidas en forma correcta. También puede notificar con un mensaje *failure* cuando se presente alguna falla en el proceso de transformación. Cuando el proceso de transformación finaliza correctamente en el agente GI, éste notifica al agente AM mediante un mensaje *inform* los modelos generados. Entonces, el agente AM actualiza su repositorio de modelos, incorporando los modelos de procesos de integración, procesos ejecutables y código de los agentes generados.

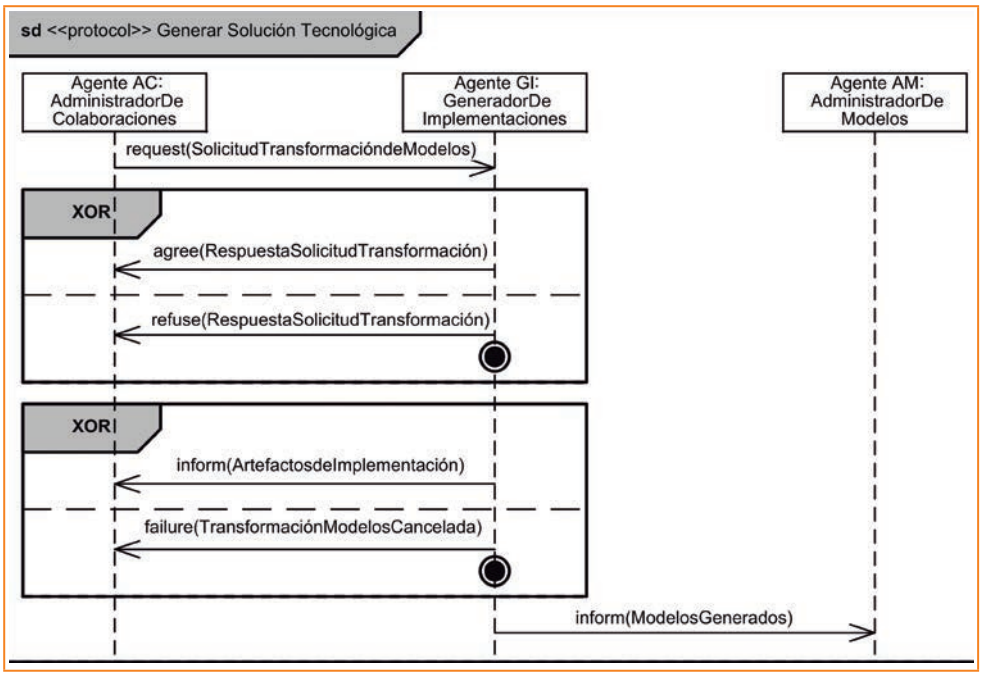



Figura 4.8. *Protocolo de interacción: Generar Solución Tecnológica.*

### 4.3.4. Interacciones de agentes para instanciar agentes *AdministradorDeProceso*

El protocolo de la figura 4.9 permite que los agentes AC cumplan la meta *generar instancia de agente AP*, que consiste en crear un agente AP, incorporarlo a la plataforma y vincularlo con el agente AP de la otra organización participante en el proceso colaborativo. Un agente AP es creado e inicializado en forma dinámica por un agente AC, basado en los artefactos de implementación que genera un agente GI de acuerdo con el protocolo expresado en la sección anterior.

Luego de intercambiados los modelos de procesos colaborativos entre los agentes AC (protocolo *Establecer Acuerdo de Colaboración*), el agente AC *receptor* procederá internamente a ejecutar las interacciones con un agente GI para generar la solución tecnológica y obtener los artefactos de implementación de cada uno de los procesos colaborativos acordados. Cuando el agente AC *receptor* recibe los artefactos de implementación desde un agente GI, envía un mensaje *inform* para notificar a su contraparte (el agente AC *iniciador*) que la transformación finalizó en forma correcta o envía un mensaje *failure de* que la transformación no finalizó correctamente y la colaboración es cancelada (figura 4.9). Cuando el agente AC *receptor* envía un mensaje *inform*, a continuación estará en un estatus de espera de un mensaje que le autorice iniciar el procedimiento de instanciación del agente AP.

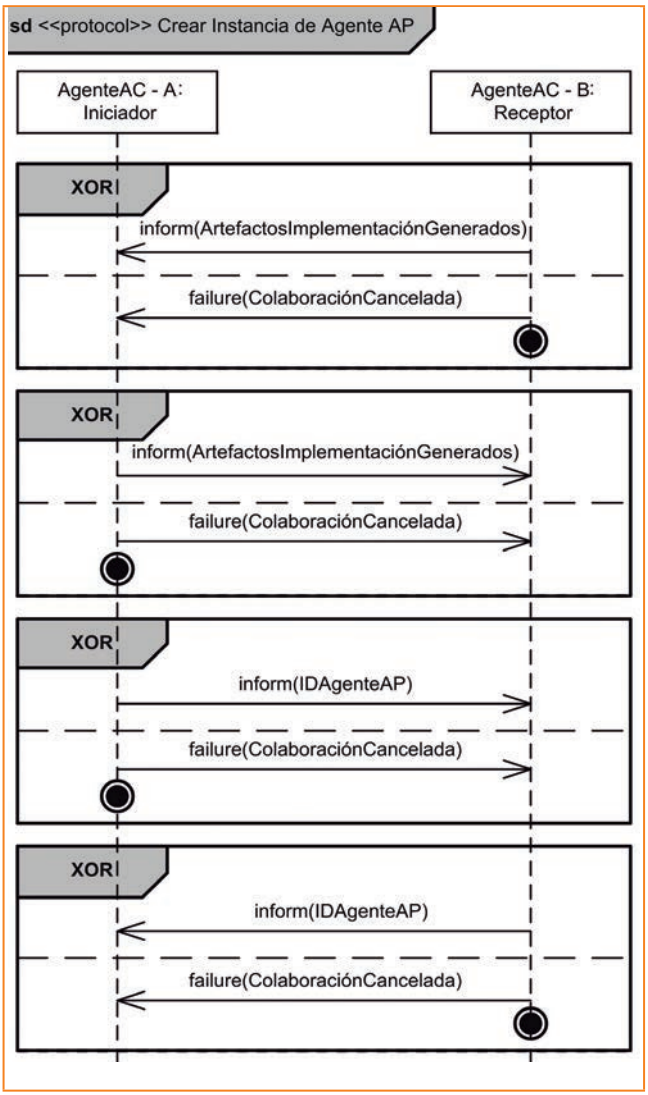



Figura 4.9. *Protocolo de interacción: Crear Instancia de Agente AP.*

De la misma manera, cuando el agente AC *iniciador* recibe un mensaje *inform*, internamente realiza el protocolo de interacción con un agente GI, a través del cual el agente GI realizará un proceso de transformaciones de modelos y le enviará al agente AC el resultado de las transformaciones. En forma similar al agente AC *receptor*, el agente AC *iniciador* notificará el resultado de las transformaciones mediante mensajes de tipo *inform* o *failure*, según sea el caso.

A continuación, tomando los artefactos de implementación generados para un modelo de proceso colaborativo, el agente AC *iniciador* internamente crea una instancia de un agente AP por cada proceso colaborativo acordado. Luego ejecuta un procedimiento que permite capturar los datos de identificación de los agentes AP instanciados. Mediante los datos de identificación capturados se permite establecer la comunicación entre los agentes AP. Los datos de identificación de los agentes AP son agregados al contenido de un mensaje *inform* que es enviado al agente AC *receptor*. En caso de que se presente alguna falla al momento de crear alguno de los agentes AP, el agente AC *iniciador* envía un mensaje de tipo *failure*, notificando la cancelación de la colaboración.

El agente AC *receptor* ejecuta un procedimiento similar de creación de agentes AP, capturando luego los datos de identificación de estos agentes instanciados, los cuales son enviados mediante un mensaje tipo *inform* al agente AC *iniciador*. En caso de presentarse alguna falla en la creación de las instancias de los agentes AP, el agente AC *receptor* genera y envía un mensaje *failure* notificando la falla al agente AC *iniciador* y el proceso de colaboración se cancela.

De esta manera, luego de crearse una instancia de los agentes AP de cada una de las partes, queda habilitada la ejecución de los procesos colaborativos acordados.

## 4.4. Implementación de la plataforma de agentes de software

La implementación de la plataforma A4IOC se llevó a cabo con el uso del *framework* y plataforma de agentes Jadex (67, 68), la cual permite construir aplicaciones con componentes heterogéneos, tales como agentes y BPMN *workflows* (automatización y ejecución de modelos de procesos basados en BPMN). Estos componentes se pueden utilizar en una misma aplicación sin problemas de interoperabilidad. Los agentes desarrollados con Jadex siguen el modelo de agente BDI (*Belief-Desire-Intention*). Una aplicación desarrollada con Jadex representa, en tiempo de ejecución, una entidad que actúa principalmente como contenedor de los agentes y de otros componentes (66). Por lo tanto, el sistema multiagente A4IOC es implementado como una aplicación Jadex, conteniendo los cinco tipos de agentes de software BDI definidos en la plataforma, así como también los modelos de procesos ejecutable BPMN requeridos para la ejecución de los procesos colaborativos.

Todos los agentes propuestos se especifican como agentes BDI de Jadex (agentes Jadex-BDI), de tal forma que se reflejan en la implementación de los mismos las creencias, metas, planes y recursos definidos en la arquitectura de la plataforma A4IOC. A su vez, la especificación de los agentes AC, AM, IS y GI (agentes estáticos) sigue un enfoque de agentes BDI orientados a procesos, ya que el comportamiento de estos agentes se implementó y definió en modelos de procesos (BPMN *workflows*). Estos modelos de procesos son ejecutados como parte de los planes de los agentes.

También el agente AP se implementó como un agente BDI que sigue este enfoque de agente orientado a procesos, utilizando un modelo de procesos ejecutable, tal como se definió en la arquitectura de la plataforma propuesta. Sin embargo, la diferencia con los

otros agentes (AC, AM, GI, IS) es que un agente AP es dinámico, ya que es construido en tiempo de ejecución del sistema por un agente GI, basado en la metodología presentada en el capítulo 3 (sección 3.3) y en los métodos MDD detallados en el capítulo 5.

En las subsecciones siguientes de este capítulo se le llamará modelo Jadex-Processes a un modelo de proceso ejecutable definido con BPMN, utilizado tanto por los agentes estáticos como por un agente AP. En ambos casos, estos modelos son ejecutados por el componente BPMN *Kernel*, el cual es un motor de procesos BPMN provisto por la plataforma Jadex.

La figura 4.10 muestra los componentes que integran un sistema multiagente A4IOC, que es definido como una aplicación Jadex llamada A4IOC. Estos componentes son agentes BDI, modelos Jadex-Processes y clases Java. En la aplicación A4IOC se especifican todos los agentes y sus roles, lo cual permite, al ejecutar la aplicación, inicializar los agentes estáticos de la plataforma.

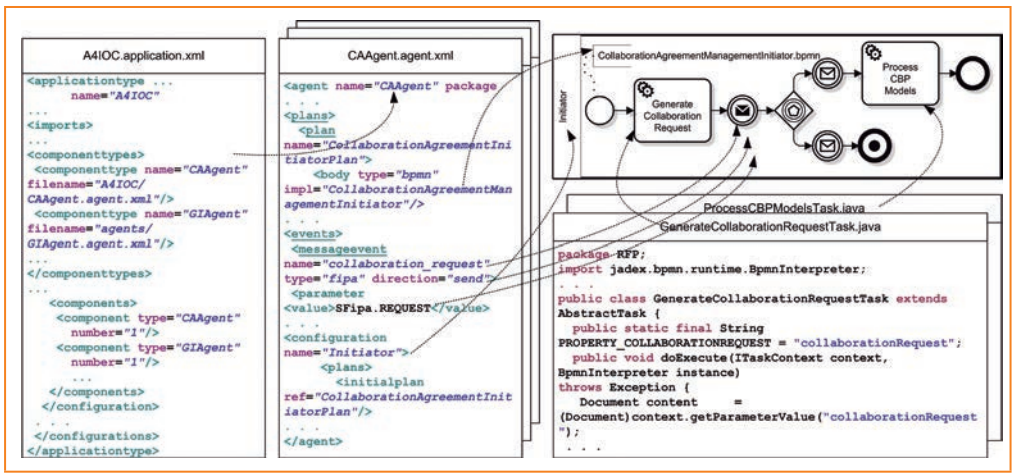



Figura 4.10. *Componentes que integran la plataforma A4IOC.*

La especificación de los agentes se lleva a cabo en un archivo de definición de agente (*Agent Definition File*, ADF), usando el lenguaje XML y siguiendo la estructura del metamodelo Jadex definido como XML Schema, lo cual permite la programación de agentes en forma declarativa. Un ADF representa la estructura de un agente BDI conformado por creencias, metas, planes, eventos y los valores iniciales de configuración, los cuales habilitan su ejecución en la plataforma Jadex. Por lo tanto, el comportamiento de estos agentes es determinado por las creencias, metas y planes especificados en el ADF.

El archivo ADF está conformado por los siguientes elementos: `agent`, `beliefs`, `goals`, `plans`, `events` y `configurations`. Cada agente es identificado por un nombre y la declaración de un paquete de implementación, así como la descripción de la funcionalidad del agente en la sección `agent`.

El elemento `beliefs` (creencias) permite definir el conocimiento del agente representado mediante un enfoque orientado a objetos permitiendo almacenar hechos (`facts`) en forma de objetos Java. Este conocimiento del agente puede ser declarado en forma estática o dinámica. El conocimiento manejado mediante `beliefs` en los agentes que conforman la plataforma A4IOC es dinámico, debido a que su valor es constantemente percibido por el contexto del agente, esto es, agregando hechos en las creencias, así como también consultan y/o modifican las creencias en tiempo de ejecución del sistema.

En Jadex se pueden utilizar dos tipos de creencias: `beliefs` que permiten al agente almacenar un hecho y `beliefset` que habilitan al agente almacenar un conjunto de hechos en la `<beliefbase>`. El manejo de la `<beliefbase>` es similar al realizado en una base de datos relacional, en donde las creencias son identificadas mediante un nombre, y los hechos `(facts)` son el valor contenido en esa creencia. Por lo tanto, en la `<beliefbase>` se almacena el conocimiento del agente de su contexto y del mismo agente.

En el elemento `goals` se definen los deseos del agente. En la definición de las metas se sigue la idea general de que las metas son los deseos concretos de un agente. Las metas son representadas como objetos explícitos contenidos en una `goalbase`, que es accesible para el componente de razonamiento, así como para los planes que necesitan conocer o desean cambiar las metas actuales del agente. Las metas son representadas separadas de los planes, lo que permite al sistema definir metas que no estén relacionadas con un plan.

En el elemento `plans` se definen un plan o conjunto de planes que se requieren para satisfacer la meta de un agente. Cada plan definido requiere la especificación de parámetros que describan las circunstancias sobre las cuales el plan es aplicado, tales como eventos internos, mensajes, metas o estado de una creencia, que permitan desencadenar la ejecución del plan. En el elemento `events` se definen los eventos de mensaje (de los protocolos de interacción) que permiten la comunicación e interacción con otros agentes de la plataforma. Los eventos de mensaje están relacionados con el plan definido en el agente.

En la sección `configurations` se especifican los parámetros iniciales que un agente ejecutará al tener un estado activo en la plataforma, esto es, un estado mental inicial del agente que puede ser declarado por un plan y/o una meta, el cual es instanciado cuando el agente es creado. Adicionalmente, la sección `imports` es agregada para definir detalles de implementación.

En la figura 4.10 se observa la relación entre la especificación de la aplicación A4IOC y el agente AC definido en el archivo ADF mostrado, mediante el nombre del agente. Los agentes de la plataforma propuesta se implementan con un enfoque de agentes orientados a procesos, en donde el comportamiento del agente es guidado y controlado por un modelo Jadex-Processes. El modelo Jadex-Processes es implementando en este tipo de agentes orientados a procesos como su plan principal, en donde los elementos de tipo tarea contenidos en el modelo Jadex-Processes representan los subplanes que el agente ejecutará. Estos subplanes, en conjunto con las metas y eventos intermedios de mensaje, representan el comportamiento de un agente.

Los elementos de evento intermedio de mensaje (que representan el envío o recepción de un mensaje) contenidos en la especificación de un modelo Jadex-Processes mantienen una relación con los mensajes definidos en la sección `events` del archivo ADF del agente, debido a que cuando se ejecuta un evento intermedio de mensaje en el modelo Jadex-Processes se invoca al agente para que ejecute los mecanismos de interacción, lo que habilita al agente a establecer la comunicación con otros agentes de la plataforma.

Los elementos de tipo tarea contenidos en el modelo Jadex-Processes son implementados mediante clases Java, que realizan acciones específicas como generar un documento, invocar un componente de la plataforma o guardar un archivo, similar al enfoque tradicional de agentes que implementan sus planes mediante programación orientada a objetos.

### 4.4.1. Infraestructura de gestión de la plataforma implementada

La plataforma Jadex es soportada por una *infraestructura de gestión*, la cual representa un contenedor de componentes Jadex (agentes BDI, modelos Jadex-Processes, y/o microagentes de Jadex) y es responsable de la operación coordinada de cada uno de estos componentes. La *infraestructura de gestión* está integrada por una *infraestructura de servicios* y por la *arquitectura interna* de cada componente Jadex. Por lo tanto, cada plataforma que se desarrolle sobre la plataforma Jadex estará soportada sobre la *infraestructura de gestión*. En la figura 4.11 se muestra cómo está integrada la infraestructura de gestión del sistema multiagente A4IOC propuesto. En esa figura se detallan los subcomponentes que conforman la arquitectura interna de un agente AP, el resto de agentes de la plataforma A4IOC tiene una arquitectura interna similar.

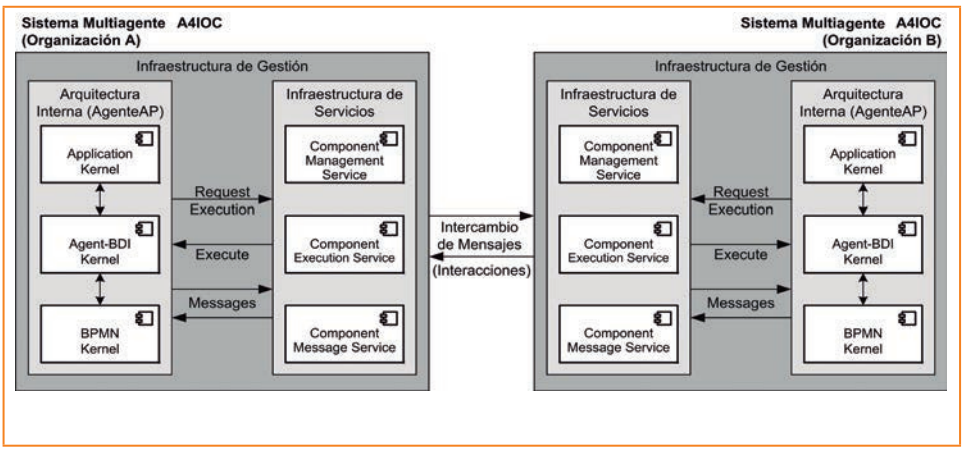



Figura 4.11. *Infraestructura de gestión del Sistema Multiagente A4IOC.*

La *infraestructura de servicios* es utilizada por todos los componentes de la plataforma permitiendo la comunicación tanto interna como externa a los agentes de la plataforma. Las interacciones entre los agentes que conforman la plataforma son realizadas mediante el envío y recepción de mensajes a través del componente *message service*. Este componente es activado cuando el componente *execution service* ejecuta un evento de enviar o recibir un mensaje. Este componente *execution service* recibe las solicitudes de ejecución de los subcomponentes de la arquitectura interna de cada agente. El componente *management service* habilita la creación de nuevas instancias de agentes y de modelos Jadex-Processes. Este componente es utilizado en la plataforma propuesta para instanciar e incorporar los agentes AP y el motor de procesos BPMN en tiempo de ejecución del sistema.

El comportamiento de un componente está determinado por su *arquitectura interna*, en donde su estructura puede incluir un conjunto de subcomponentes. En la figura 4.11 se detalla, como ejemplo, la arquitectura interna del agente AP, que está conformada por los subcomponentes *agent-BDI kernel*, *BPMN-kernel* y *application kernel*. Cuando se crea una instancia del agente AP en la plataforma se hace uso del componente *agent-BDI kernel*, permitiendo incorporar y generar un estado (activo, en ejecución, etc.) para el agente en la plataforma. También en el agente AP se ha definido el uso de un motor de proceso (componente BPMN *kernel*) que permite la ejecución de modelos Jadex-Processes. Además, en el agente AP se definió el uso del componente *application kernel*, el cual habilita la utilización de módulos que contienen ciertas capacidades (*capabilities*). Las *capabilities* es un mecanismo provisto por la plataforma Jadex, que permite agrupar elementos de agentes BDI (tales como creencias, metas, planes y eventos) en un módulo reusable, encapsulando ciertas funcionalidades que pueden ser importadas en los componentes de la plataforma.

Por lo tanto, este agente AP tiene la capacidad de iniciar otros componentes de la plataforma en tiempo de ejecución del sistema mediante la infraestructura de servicios. Cuando el agente AP es inicializado en la plataforma, instancia un motor de procesos que posibilita la ejecución del modelo de proceso ejecutable definido en su sección *plans*, utilizando el componente BPMN *kernel*.

### 4.4.2. Implementación del agente AC

El comportamiento implementado en el agente AC se lleva a cabo mediante modelos Jadex-Processes. En estos modelos se especifican las tareas y eventos intermedios de mensaje que debe realizar el agente para dar soporte a los protocolos de interacción, definidos en la sección 4.3 de este capítulo, en los que está involucrado. Estos modelos de procesos integran las tareas y eventos que debe realizar el agente en forma coordinada para llevar adelante estos protocolos.

Los eventos intermedios de mensaje definidos en un modelo Jadex-Processes representan las comunicaciones e interacciones de los protocolos de interacción que realiza un agente AC con otro agente AC, y entre el agente AC con sus agentes AM, GI y AP que integran el sistema muliagente. Estas interacciones son realizadas a través del intercambio de mensajes definidos con el lenguaje FIPA ACL (27). La información intercambiada mediante estos eventos intermedios de mensaje es gestionada por actividades de tipo tarea que permiten al agente generar o almacenar la información que es intercambiada en los mensajes, y/o invocar sistemas internos u otros agentes de la plataforma.

En el agente AC se ha definido el comportamiento de los roles *iniciador* y *receptor* del protocolo de interacción presentando en la sección 4.3.1, que permiten al agente AC, por un lado, iniciar una solicitud de acuerdo de colaboración y, por otro lado, recibir una solicitud para establecer un acuerdo de colaboración. En la sección `configurations` del archivo ADF del agente AC se especifican estos roles que puede ejecutar el agente y los planes relacionados con cada rol definido, tal como se muestra en la figura 4.12. El comportamiento de los roles que cumple un agente AC se implementa en modelos Jadex-Processes, uno para cada rol, de acuerdo con los dos planes definidos en el agente AC.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
<H3>Collaboration Administrator Agent - CAAgent </H3>
This agent implements an Initiator or Participant role in a Dynamic Inter-Organizational
Collaboration.
-->
<agent xmlns="http://jadex.sourceforge.net/jadex"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                           http://jadex.sourceforge.net/jadex-bdi-2.0.xsd"
       name="CAAgent" package="A4IOC"
       ...
       <configurations>
              <configuration name="Initiator">
                     <plans>
                            <initialplan ref="CollaborationAgreementInitiatorPlan"/>
                     </plans>
              </configuration>
              <configuration name="Participant">
                     <plans>
                            <initialplan ref="CollaborationAgreementParticipantPlan"/>
                     </plans>
              </configuration>
       </configurations>
</agent>
```

Figura 4.12. *Archivo ADF del tipo de agente AC con la definición de los roles en la sección* `configurations`.

A continuación se describe el modelo Jadex-Processes con el que un agente AC implementa el comportamiento del rol *iniciador*. Este modelo se muestra en la figura 4.13.

Cuando se inicia la ejecución del agente AC con el rol *iniciador*, se ejecuta este modelo Jadex-Processes definido en la sección `plans` del agente. Al entrar en operación este plan se ejecuta el elemento de tipo *tarea* `Generar solicitud de Colaboración`,

que tiene la función de generar un documento XML, el cual contiene una solicitud de colaboración con el nombre y la descripción de la colaboración, así como también el nombre, identificador y descripción de los procesos colaborativos que solicita ejecutar a la otra organización. Este documento es generado mediante la clase Java `generate-collaborationrequesttask`, la cual permite generar el documento mediante un formato predefinido. En el parámetro de entrada de la tarea se especifican los términos del acuerdo de colaboración y los procesos colaborativos que se solicita ejecutar. En el parámetro de salida se define el nombre del documento (`cagreement`) que se genera mediante esta tarea. Además, en la tarea se define un parámetro que permite almacenar el documento generado en el contenedor de creencias `<beliefbase>` del agente AC, con el propósito de consultar y/o modificar la información contenida en las creencias, en el transcurso de la ejecución del comportamiento del agente. En los agentes AC este conocimiento (creencias) es representado por documentos de negocio, modelos de procesos colaborativos, nombres e identificadores de agentes AP.

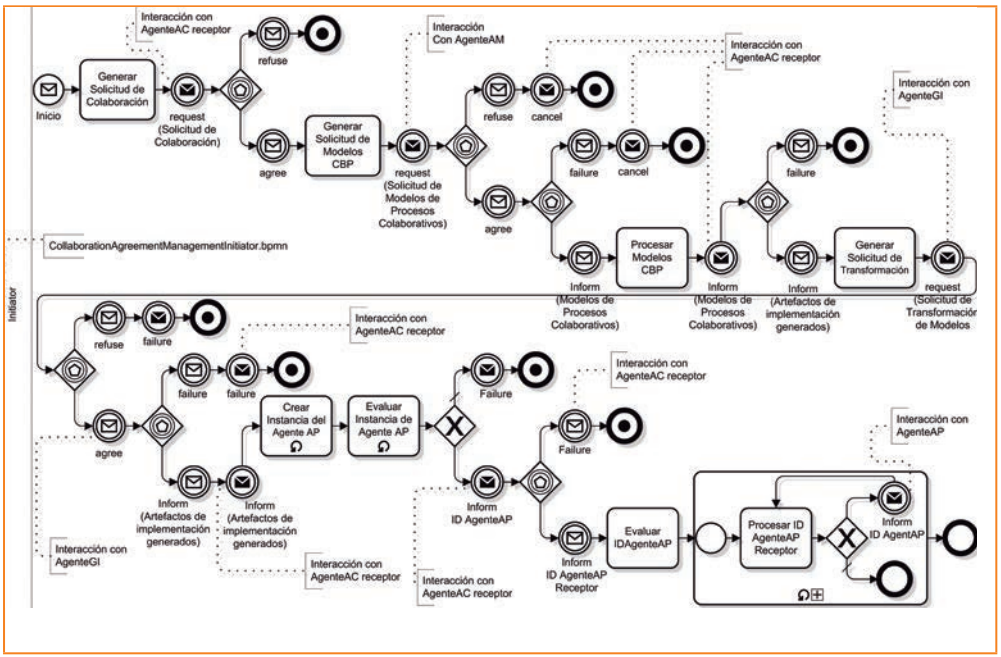



Figura 4.13. *Modelo Jadex-Processes definido para el rol de iniciador de un agente AC.*

A continuación, el agente AC envía el documento generado mediante un evento intermedio de mensaje con un acto de comunicación *request*, solicitando establecer un acuerdo de colaboración, detallado en el documento contenido en el mensaje. En la plataforma Jadex un acto de comunicación (*speech act*) es llamado *performative*. El agente AC puede recibir como respuesta a la solicitud, un mensaje *agree* o un mensaje *refuse*.

En caso de recibir un mensaje *refuse* del agente AC *receptor* rechazando la solicitud, provocará que la negociación sea cancelada y se da por terminado el proceso y la interacción con el agente AC *receptor*.

Cuando el agente AC recibe una notificación de aceptación mediante un mensaje *agree*, procede a ejecutar la tarea generar solicitud de modelos CBP. Esta tarea consiste en generar un documento para solicitar los modelos de procesos colaborativos a ejecutar, detallando el nombre y el identificador de cada proceso colaborativo acordado. Estos datos son recuperados del documento que fue previamente almacenado en el contenedor de creencias del agente AC y que contiene la descripción de los procesos colaborativos que las organizaciones acordaron ejecutar. El documento generado es enviado mediante un evento intermedio de mensaje con un acto de comunicación *request* al agente AM, requiriendo los modelos de procesos colaborativos.

Si el agente AC recibe un mensaje *refuse* desde el agente AM rechazando la solicitud, entonces finaliza la comunicación con el otro agente AC enviando un mensaje *cancel*. Cuando el agente AC recibe un mensaje *agree* desde el agente AM, luego recibe un mensaje *inform* desde el mismo agente AM con los modelos de procesos colaborativos solicitados, el contenido del mensaje *inform* es transferido como entrada a la tarea `procesar modelos CBP`, lo cual se realiza mediante la definición de un mapeo (*mapping*) en el conector de secuencia que une al evento intermedio de mensaje con la tarea. En este conector se define la variable `CBPModels`, la cual almacenará el contenido del mensaje y la instrucción `$event.content`, que permite recuperar el contenido del mensaje, tal como se muestra en la figura 4.14, que presenta la tabla de parámetros del conector de secuencia.

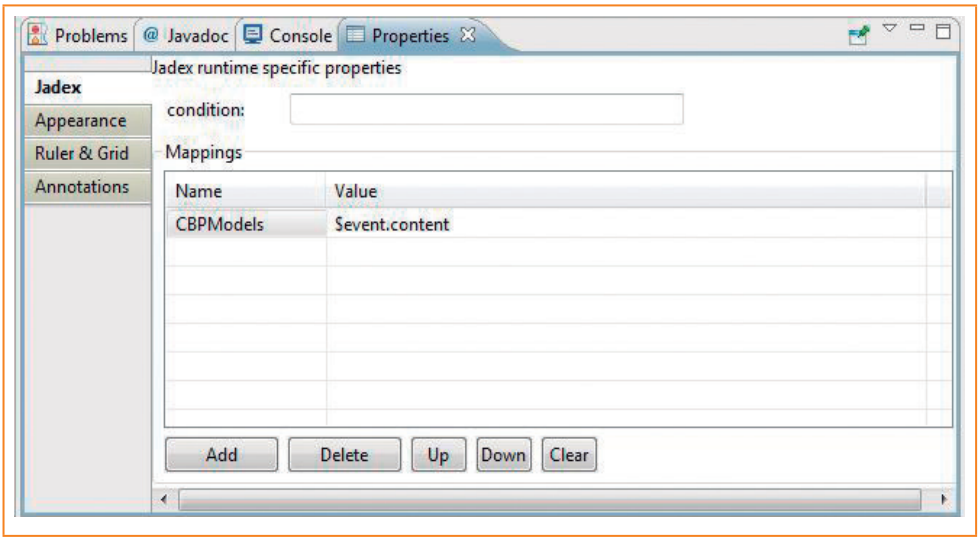



Figura 4.14. *Recuperación del documento contenido en un mensaje recibido mediante un* mapping.

En la tarea `procesar modelos CBP` se define como parámetro de entrada el documento almacenado en la variable `CBPModels`. Mediante este parámetro se actualizan las creencias del agente utilizando el método `BeliefSet()`. También se define un parámetro de salida con la variable `CBPInform`, que contiene los modelos de procesos colaborativos que serán transmitidos al agente AC *receptor*. En resumen, los modelos de procesos colaborativos contenidos en la variable `CBPModels` son utilizados para actualizar las creencias del agente y transferidos a la variable `CBPInform` para ser transmitidos al agente AC *receptor* mediante el siguiente evento intermedio de mensaje definido en el modelo Jadex-Processes.

Entonces, los modelos de procesos colaborativos contenidos en el documento CBPInform son enviados mediante un mensaje *inform* al agente AC *receptor*. Del lado del agente AC *receptor*, al recibir el mensaje con los modelos, solicita a su agente GI la transformación de los modelos para generar las soluciones tecnológicas. Cuando el proceso de transformación finaliza, el agente AC *receptor* notifica al agente AC *iniciador,* mediante un mensaje *failure,* que ocurrió alguna falla en la generación de la solución, finalizando la comunicación y la colaboración entre los agentes AC. Si el proceso de transformación finaliza correctamente, el agente AC *receptor* notifica, mediante un mensaje *inform* al agente AC *iniciador*, que la solución tecnológica se generó correctamente.

En el momento en que el agente AC *iniciador* recibe el mensaje que notifica la generación de la solución tecnológica exitosa, ejecuta la tarea `generar solicitud de transformación`, que consiste en generar un documento que contiene la solicitud de transformación y los modelos de procesos colaborativos a transformar. Esta tarea es implementada mediante la clase Java `generatetransformation request`, la cual implementa un método para recuperar cada modelo de proceso colaborativo almacenado en las creencias del agente zzzz<`Model()CBPModel =(Model())getBelief-base(). getBeliefSet("CBPModels").getFacts()`).

El documento generado es enviado mediante un evento intermedio de mensaje *request* al agente GI, solicitando ejecución de las transformaciones de modelos. El agente GI puede responder mediante un mensaje *agree* o *refuse*, aceptando o rechazando la solicitud, en el último caso la comunicación es finalizada entre los agentes AC y GI. El agente GI realiza el proceso de transformación como se detalla en la siguiente subsección.

El agente AC puede recibir un mensaje *refuse* o un mensaje *agree* desde el agente GI. En el primer caso, el agente AC notifica al agente AC *receptor* mediante un evento intermedio de mensaje *failure* que no se pudieron obtener los artefactos de implementación, con lo cual se cancela comunicación entre los agentes AC. En el segundo caso, el agente AC recibe un mensaje *inform* con los artefactos de implementación, el cual incluye los nombres de los agentes AP generados. La plataforma Jadex tiene como condición que el nombre del archivo ADF que contiene el código del agente debe ser el mismo que el nombre del agente que se especifica en el código. Mediante estos nombres el agente AC instanciará a los agentes AP. Luego, el agente AC *iniciador* notifica mediante un evento intermedio de mensaje *inform* al agente AC *receptor* que los artefactos de implementación se generaron correctamente.

A continuación, el agente AC ejecuta la tarea `crear instancia de agente AP`, cuya función es crear un agente AP, asignarle el rol e incorporarlo a la plataforma de agentes. Esta tarea es implementada mediante la clase Java `instantiateagenteAP`. En esta clase se invoca al servicio `IComponentManagementService` provisto por la plataforma Jadex, el cual representa un servicio de páginas blancas que se puede utilizar para todas las operaciones del ciclo de vida de un componente, por ejemplo: crear o destruir una instancia de un agente. Este servicio mantiene un registro de todas las instancias de los componentes creados y gestiona las entradas que contienen el identificador del agente y la lista de direcciones de transporte del componente. Entonces, utilizando el método `CMScreateComponent` proporcionado por este servicio, es creada una instancia del agente utilizando el nombre del agente que fue generado en la transformación. Mediante el método `IComponentIdentifier` se le crea un identificador al agente, con lo cual se habilita la incorporación del mismo a la plataforma y permite asignarle un rol dentro de la plataforma, así como también monitorear el desempeño del agente AP a través de las herramientas proporcionadas por la plataforma Jadex. En este caso, a la instancia del agente se le asigna el rol con el tipo de agente *AdministradorDeProceso* (especificado como AgenteAP) que se predefinió en la aplicación A4IOC y puede tener *n* instancias del mismo tipo.

Posteriormente, en el comportamiento del agente AC se ejecuta la tarea evaluar instancia del agente AP, que es implementada a través de la clase Java `evaluate-instantiateagent`, y que tiene la función de confirmar la existencia del agente en la plataforma, el estado del agente y recuperar los datos de identificación del agente. Esta clase utiliza también el servicio `IComponentManagementService` y mediante el método `ComponentDescription` recupera los datos del agente y su estado en la plataforma utilizando el nombre del agente para su búsqueda. La descripción del agente recuperada mediante este método contiene el nombre del agente, identificador, estado, dirección IP y puerto. Los estados que puede tener un agente en la plataforma Jadex son: activo (*active*), suspendido (*suspended*), en terminación (*terminating*), terminado (*terminated*), listo (*ready*) y en ejecución (*running*).

Estos datos del agente son almacenados en un documento, definido como parámetro de salida de la tarea `evaluar instancia del agente AP`, con el nombre de `IDA-gent` y es agregado en el contenedor de creencias del agente AC. Finalmente, en esta tarea se evalúa el estado del agente mediante los datos recuperados del agente. En caso de que el agente AP se encuentre en estado *terminating* o *terminated*, se envía un mensaje *failure* al agente AC *receptor*, con lo cual se notifica que ocurrió algún error en la creación de la instancia del agente AP y el proceso de establecer la colaboración finaliza. En caso contrario, que el agente AP se encuentre en estado *active* o *suspended*, se envía un mensaje *inform* que contiene el documento IDAgent con los datos de identificación del agente AP al agente AC *receptor*. Estos datos de identificación del agente AP son intercambios entre los agentes AC debido a que se requiere configurar el nombre y la dirección IP del agente AP con el que se establecerá la comunicación para ejecutar el proceso co-

laborativo. Las tareas `crear instancia de agente AP` y `evaluar instancia del agente AP` son realizadas en forma iterativa por cada agente AP que el agente AC debe instanciar.

A continuación, el agente AC estará en espera de recibir un mensaje *failure* o *inform* del agente AC *receptor*, con el cual notifique el resultado del proceso de creación de instancias de sus agentes AP. Cuando el agente AC recibe un *inform* con los datos de identificación de los agentes AP de la otra organización, procede a recuperar el contenido del mensaje mediante un *mapping* definido en el conector de secuencia que relaciona el evento intermedio de mensaje con la tarea `evaluar ID agenteAP`. La especificación en el conector es la variable `IDAgentReceptor`, en la cual se almacenará el documento recibido, y la sentencia `$event.content`, para recuperar el contenido del evento intermedio de mensaje, similar a lo desplegado en la figura 4.14.

A continuación, la tarea `evaluar ID agenteAP` es ejecutada. En esta tarea se ha definido un parámetro de entrada para el documento `IDAgentReceptor`, lo que permite evaluar el documento recibido, esto es, identificar y contar las descripciones de agentes generados por el agente AC *receptor* (un agente por cada proceso colaborativo que fue acordado ejecutar). El valor del total de descripciones de agentes es almacenado en la variable temporal `TotalID` y es agregada con el mismo nombre en la `<beliefbase>` como una creencia del agente AC. También el documento con las descripciones de los agentes `IDAgentReceptor` es agregado a las creencias del agente AC.

Entonces se ejecuta una tarea subproceso tipo bucle (iterativo) que contiene la tarea `procesar ID agenteAP receptor` y un evento intermedio de mensaje de tipo de envío. En la tarea de tipo sub-proceso se define un parámetro de entrada con el nombre de `TotalIDAgent` de tipo entero (`Int`), en el cual es asignado el valor contenido en la creencia `TotalID`. Este valor representa las repeticiones que serán ejecutadas del subproceso iterativo.

La tarea `procesar ID agenteAP receptor` tiene la función de procesar las descripciones de los agentes contenidas en la creencia `IDAgentReceptor`. En esta tarea se ha definido un parámetro de entrada con el nombre `agent` de tipo `Int`, que tiene la función de contador, el cual es inicializado con un valor 1 y que será incrementado por cada mensaje que sea enviado al agente AP correspondiente. También se ha definido un parámetro de salida con el nombre de `IDAgent`, el cual permitirá almacenar el contenido de la descripción e identificación de cada agente. Para lo cual se recupera de la `<beliefbase>` la creencia `IDAgentReceptor` y se almacena el contenido de la descripción del agente en el parámetro de salida `IDAgent`. La posición o número de identificación del agente a recuperar es determinado por el valor del parámetro `agent`.

El contenido de `IDAgent` representa el documento que será enviado al agente AP. Entonces, por cada documento generado es enviado un mensaje *inform* al agente AP correspondiente, incrementando el valor del parámetro `agent`. La condición que permite incrementar el valor `agent` está definida en el conector de secuencia que relaciona el evento intermedio de mensaje y la tarea `procesar ID agenteAP` receptor. Cuando

el valor del parámetro `agent` es mayor que el valor contenido en `TotalIDAgent`, la ejecución del subproceso finaliza. Esta condición se ha definido en la compuerta exclusiva basada en datos, posterior a la tarea procesar `ID agenteAP receptor`.

Posteriormente, el evento de fin es ejecutado en el modelo Jadex-Processes, quedando en estado activo las instancias de los agentes AP generados, en espera del mensaje que inicialice la ejecución de los mismos y empiecen la interacción con sus agentes AP contraparte.

### 4.4.3. Implementación del agente GI

El agente GI implementa y automatiza la metodología y los métodos MDD propuestos en el capítulo 3 (subsección 3.3), posibilitando la generación de la solución tecnológica en forma automática. Para lo cual los métodos MDD fueron implementados mediante el lenguaje de transformación de modelos ATL (5, 42) (como se detalla en el capítulo 5), permitiendo la configuración de un motor de transformaciones de modelos ATL para cada transformación. Estos motores son ejecutados mediante los subplanes definidos en el agente GI, habilitando la generación de la solución tecnológica en tiempo de ejecución del sistema.

A continuación se describe el comportamiento definido para el agente GI especificado en un modelo Jadex-Processes (figura 4.15). Dicho modelo es implementado como el plan del agente. Los elementos de tipo tarea contenidos en el modelo Jadex-Processes representan los subplanes que ejecuta el agente GI mediante este modelo.

La ejecución del agente GI es activada cuando recibe un mensaje *request* con una solicitud para generar la solución tecnológica (figura 4.15). El contenido del mensaje recibido es recuperado mediante la sentencia $event.content, transfiriendo su valor a una variable temporal llamada CBPModels. El contenido de esta variable es evaluado mediante la tarea evaluar solicitud de transformación que es implementada a través de la clase Java evaluatecontent. La evaluación consiste en verificar que la solicitud contenga al menos un modelo de proceso colaborativo, así como contar el número de modelos contenidos en el documento de solicitud.

En caso de contener uno o más modelos, la tarea agrega en la `<beliefbase>` del agente GI una nueva creencia con el nombre CBPModels y el valor (hechos) de esta creencia son los modelos recuperados del contenido del mensaje recibido. Entonces, el agente GI procede a enviar un mensaje *agree*, confirmando la aceptación de la solicitud. En caso contrario, cuando el documento de la solicitud no contiene ningún modelo, se envía un mensaje *refuse*, rechazando la solicitud y cancelando la comunicación entre los agentes AC y GI. Además, el número total de modelos contados en el documento CBPModels es almacenado en la variable Total. Esta variable es agregada a la `<beliefbase>` del agente con el mismo nombre, con el propósito de detectar el fin del ciclo de transformaciones cuando se alcance el número total de modelos contenidos en la solicitud recibida.

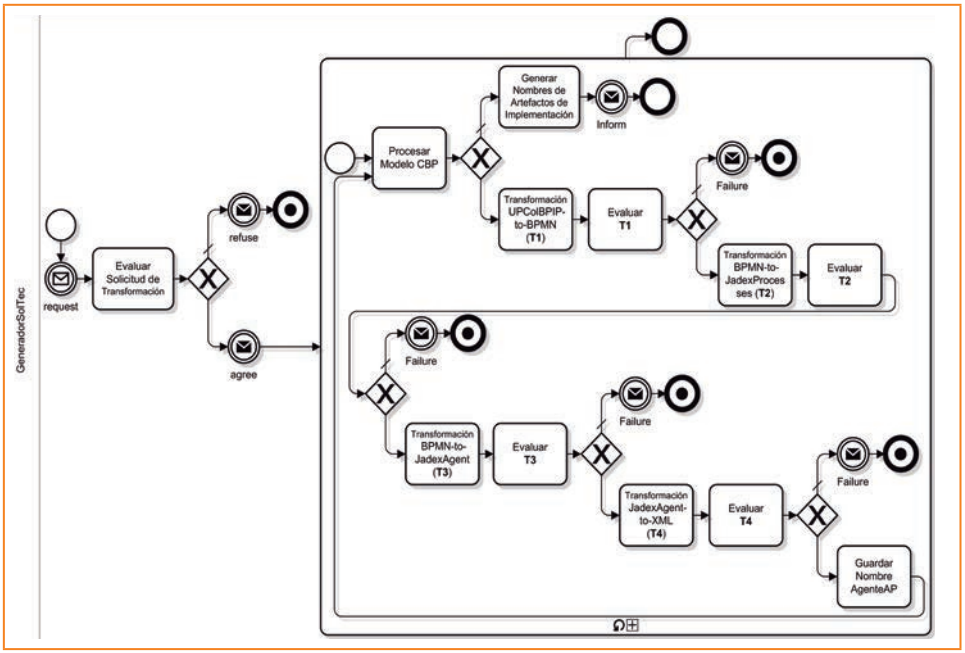



Figura 4.15. *Modelo Jadex-Processes implementado en el plan del agente GI.*

A continuación es implementado un subproceso mediante el cual se inicia la transformación de modelos. Este subproceso se realiza en forma iterativa por cada modelo de proceso colaborativo recibido por el agente. En la configuración de ejecución del subproceso se define un parámetro con el nombre de `TotalModels`, con un tipo de dato `Int` y asignándole el valor contenido en la creencia `Total`, previamente recuperado de la `<beliefbase>` del agente GI, dando inicio a la ejecución de una instancia del subproceso por cada modelo de proceso acordado.

Dentro de este subproceso, el primer elemento ejecutado es la tarea `procesar modelo CBP` que es implementada mediante la clase Java `processCBPModels`. En esta tarea se ha configurado el parámetro `Transform` de tipo `Int` (entero), que es utilizado como un contador que permite conocer el número de modelos de proceso colaborativo que se han transformado. Este parámetro es inicializado con valor `1`, en donde este valor representa el número del modelo de proceso colaborativo que se debe recuperar y transformar. También en esta clase se define un parámetro de salida llamado `Model`, el cual es utilizado para almacenar el modelo a transformar. Entonces, se recupera la creencia `CBPModels` de la `<beliefbase>` del agente GI, copiando el modelo del proceso colaborativo a transformar, dependiendo del valor del parámetro `Transform`, en el parámetro `Model`. Este documento generado mediante el parámetro de salida de la tarea procesar modelos CBP representa el modelo de entrada de la primera transformación a ejecutar.

El proceso de transformación se realiza en forma secuencial, por lo cual la primera transformación es ejecutada mediante la tarea `Transformación UP-ColBPIP-to- BPMN (T1)`, que tiene la función de generar un modelo BPMN del proceso de integración de la organización. Esta tarea es implementada mediante la clase Java `upcolbpip2bmpnPlan`, la cual permite copiar el modelo contenido en el documento `Model` al directorio de trabajo, permitiendo al usuario (si así fue configurado en la plataforma) confirmar la ruta de almacenamiento del modelo de entrada y modelo de salida mediante una interfaz. Posteriormente, la clase invoca a un plug-in ATL (`upcolbpip2bpmn`) que permite lanzar el motor ATL que ejecuta la transformación `upcolbpip2bpmn.atl`. En el plug-in se configuró que se capturen todos los mensajes y excepciones generados por el motor de transformaciones de modelos ATL. Cuando finaliza la ejecución del motor ATL, son recuperadas las excepciones y son almacenadas en un archivo.

A continuación se ejecuta la tarea `evaluar T1,` que lleva a cabo un análisis del archivo de excepciones generadas por la primera transformación, para detectar algún error en la transformación. En caso de encontrar un error, se envía un mensaje *failure* al agente AC, notificando la falla en el proceso de transformación. En caso contrario, se ejecuta la tarea `Transformación BPMN-to-JadexProcesses (T2)`, que tiene la función de generar un modelo de proceso ejecutable como modelo Jadex-Processes. La tarea es implementada mediante la clase Java `bmpn2jadexPlan`, que invoca al plug-in ATL `bmpn2jadex` para lanzar el motor ATL que ejecuta la segunda transformación, capturando las excepciones del motor de transformación.

Luego se ejecuta la tarea `evaluar T2,` que permite verificar que el modelo de proceso ejecutable se generó correctamente. Además, esta tarea permite, cuando el proceso de transformación finaliza correctamente, copiar el modelo de salida de la transformación al directorio de ejecución de la plataforma de agentes. Esta tarea tiene predefinida la ubicación que deben tener los artefactos de implementación generados por los motores de transformación. Cuando se detecta alguna falla en la transformación, se cancela la ejecución de las transformaciones, generando y enviando un mensaje *failure* al agente AC, notificando la falla en el proceso de transformación.

El proceso de transformación continúa con la ejecución de las tareas `Transformación BPMN-to-JadexBDI (T3)` y `Transformación JadexBDI-to- -XML (T4)` para generar el modelo del agente y código de implementación del agente, respectivamente, en forma similar a lo descrito en las transformaciones `T1` y `T2`. Adicionalmente, en la tarea evaluar `T4` se lleva a cabo un proceso que permite copiar el documento generado por la transformación `T4` al directorio de implementación y ejecución de la plataforma de agentes, mediante una ruta predefinida.

Cuando la transformación `T4` finaliza correctamente, se ejecuta la tarea guardar nombre Agente AP, que tiene la función de almacenar el nombre del agente en una creencia llamada `AgentName` en la `<beliefbase>`. El nombre del agente es recuperado del documento generado, el cual representa el código de implementación de un

agente AP. El código de implementación contiene un elemento `agent` con un atributo `name`, cuyo valor representa el nombre del agente.

Al finalizar esta tarea es incrementado el valor del contador de la variable `Transform`, este parámetro es definido en el conector de secuencia que une la tarea `guardar nombre Agente AP` con la tarea `procesar modelo CBP`. Entonces, una nueva iteración (instancia) del subproceso se inicia, con la ejecución de la tarea `procesar modelo CBP`, generando el documento con el siguiente modelo de proceso colaborativo a transformar.

En la compuerta exclusiva basada en datos definida posteriormente a la tarea `procesar modelo CBP`, se ha especificado la condición que permite determinar cuándo se han transformado todos los modelos almacenados en la creencia `CBPModels`. Esto se realiza mediante comparación del valor contenido en las variables `TotalModels` y `Transform`. Cuando el valor de `Transform` es mayor que el valor de `TotalModels`, significa que ya no existen más modelos a transformar. Con lo cual se habilita la ejecución de la tarea `Generar Nombres de Artefactos de Implementación` que consiste en recuperar la creencia `AgentName` contenida en la `<beliefbase>` del agente GI. Mediante el valor de esta creencia se genera un documento con los nombres de todos los agentes AP generados, el cual es enviado al agente AC mediante un mensaje *inform*. Con la ejecución del evento anterior finaliza la comunicación entre el agente GI y el agente AC. En caso contrario, que el valor de `Transform` sea menor al valor contenido en `TotalModels`, se ejecutará un nuevo ciclo del proceso de transformación.

### 4.4.4. Implementación del agente AP

Los agentes AP forman parte de la solución tecnológica que se genera al establecer una colaboración interorganizacional dinámica entre los agentes AC. Un agente AP se construye en forma automática y en tiempo de ejecución de la plataforma mediante los métodos MDD implementados y ejecutados por el agente GI, como se explicó en la subsección anterior. Por lo tanto, el plan del agente AP que posibilita que el agente ejecute el rol de la organización en un proceso colaborativo también se genera en tiempo de ejecución.

Sin embargo, para la implementación de un agente AP se usa un archivo ADF de plantilla que tiene predefinido un plan y un conjunto de clases Java. El plan permite actualizar los datos del agente receptor, habilitando la comunicación entre los agentes AP *iniciador* y *receptor*. En el caso de las clases Java, éstas son utilizadas para ejecutar los diferentes tipos de tareas que pueden ser definidas en un modelo de proceso ejecutable.

El plan predefinido en el agente AP está conformado por un evento intermedio de mensaje y por la tarea `actualizar ID agente receptor`, como se muestra en la figura 4.16.

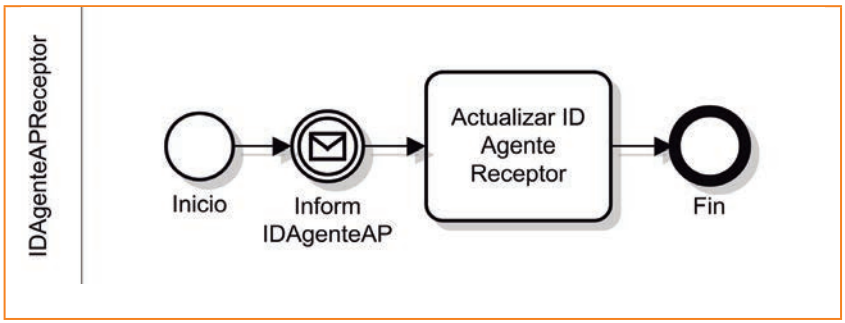



Figura 4.16. *Plan predefinido para establecer la comunicación entre agentes AP.*

Cuando el agente AC crea una instancia del agente AP, éste es inicializado en la plataforma y se ejecutará primeramente este plan. El agente AP estará en espera de recibir un mensaje del agente AC. Al recibir el mensaje *inform* que contiene el documento `IDAgent`, conoce los datos de identificación del agente AP *receptor*. Este documento es mapeado como entrada en los parámetros de ejecución de la tarea `actualizar ID agente receptor`, la cual interpreta el contenido del documento y mediante el método `IComponentIdentifier[]` establece los datos de identificación del agente AP *receptor* en los parámetros de comunicación del agente AP *iniciador*.

Adicionalmente, en la plataforma se predefinieron las siguientes clases Java: `UserInteractionTask`, `WriteContextTask` y `GenerateDocTask`, que posibilitan la implementación de los elementos de tipo tarea contenidos en el modelo del proceso ejecutable. Estas clases son asignadas a los elementos de tipo tarea mediante un método MDD que se detalla en el capítulo 5.

Las tareas de tipo usuario (*UserTask*) etiquetadas como *evaluate* son implementadas mediante la clase `UserInteractionTask`. En esta clase se definen tanto los parámetros de entrada como los de salida. La entrada es un documento de negocio que debe ser evaluado por un usuario (persona) que realiza la tarea, para lo cual en la clase se ha definido una variable temporal para el documento de entrada, el tipo de dato de la variable y el valor de la variable que corresponde al contenido del documento de negocio. Mediante esta clase el contenido del documento de negocio es desplegado en pantalla para que el usuario pueda analizarlo y lo evalúe.

La forma en que el documento de negocio es establecido como entrada de la clase `UserInteractionTask` se describe en la subsección 5.1.3.7 del capítulo 5. El parámetro de salida de la clase permite almacenar el resultado de la evaluación al documento llevada a cabo por el usuario, para lo cual se definieron el nombre de una variable temporal y el tipo de dato de la variable. En este caso se especifica la variable temporal `option` y el tipo de dato `Int`, ya que esta tarea almacena el resultado de la evaluación como un valor entero. Esto es realizado mediante un panel en el que el usuario esta-

blece el resultado de la evaluación mediante sección de botones, los cuales representan para la clase un valor numérico de tipo entero.

Las tareas de tipo servicio (*ServiceTask*) representan tareas automáticas. Las tareas de este tipo etiquetadas como store son implementadas mediante la clase `WriteContextTask`. Esta clase permite almacenar un documento de negocio en un sistema interno de una organización que participa en una colaboración interorganizacional, a través de la invocación o delegación de esta tarea a un agente IS. En la clase `WriteContextTask` sólo se definen los parámetros de entrada, éstos son: el nombre de una variable temporal para almacenar el documento de negocio, el tipo de dato y el valor de la variable. El documento de negocio es definido en la entrada de tarea utilizando el método descrito en el capítulo 5 (subsección 5.1.3.7). Además, en la clase se especifica un conjunto de acciones que permiten establecer la comunicación e interactuar con el agente IS, con el propósito de transmitir el documento de negocio al agente IS mediante un mensaje interno de tipo *inform*.

En el agente IS se ha configurado un comportamiento que permite establecer la comunicación con un agente AP y procesar el contenido de los mensajes recibidos del agente AP. Este procesamiento consiste en transformar el documento de negocio contenido en mensaje al formato requerido por el sistema interno de la organización. Al finalizar este proceso el agente IS envía un mensaje interno *inform* notificando al agente AP que el documento de negocio fue almacenado correctamente. Al recibir el mensaje, la clase `WriteContextTask` finaliza su ejecución, liberando el proceso para continuar la ejecución del siguiente elemento contenido en el proceso ejecutable.

Este comportamiento, definido mediante la clase `WriteContextTask`, se implementa por medio de mensajes internos entre los agentes, debido a que el modelo de proceso ejecutable (derivado del modelo de proceso de integración) únicamente contiene mecanismos de interacción mediante intercambio de mensajes con su contraparte (otro agente AP). Por lo cual, al requerir interactuar con otros agentes de la plataforma o sistemas internos en una tarea del proceso, es necesario predefinir este comportamiento a través de una clase Java en donde se pueden implementar estas interacciones. Por lo tanto, el agente IS realiza una función de mediador entre un agente AP y un sistema interno de la organización.

Las tareas de tipo servicio (*ServiceTask*) etiquetadas como `generate` son implementadas mediante la clase `GenerateDocTask`. En esta clase se definen, en forma similar a las clases anteriores, parámetros de entrada y salida, pero la salida generada por esta tarea es un documento de negocio que será transmitido por un elemento de evento intermedio de mensaje, posterior a esta tarea. Además, se agrega un parámetro de entrada/salida (`inout`) que permite integrar los datos de identificación (nombre e identificador del agente) del agente que ejecuta el modelo de proceso ejecutable mediante una acción que implementa el método `IComponentIdentifier[]` provisto por la plataforma Jadex. Para generar el documento de negocio, la clase `GenerateDocTask` invoca a un agente IS mediante un comportamiento que permite establecer la comunicación con este agente.

El agente IS recibe la solicitud mediante un mensaje *inform*, conteniendo los datos y tipo de documento que se requiere generar. Este agente interactúa con el sistema interno para recopilar la información necesaria y poder construir un documento de negocio en XML basado en plantillas predefinidas XML del documento de negocio. El documento generado es transmitido a la clase `GenerateDocTask`, para que lleve a cabo la integración de los datos de identificación, con lo cual el documento de negocio estará disponible para su utilización en el proceso de ejecutable. Normalmente será transmitido por el siguiente elemento de evento intermedio de mensaje contenido en el modelo de proceso ejecutable.

# 5. Métodos de desarrollo dirigido por modelos para agentes orientados a procesos

En este capítulo se presentan los métodos de transformación de modelos, propuestos en este libro, basados en los conceptos del desarrollo dirigido por modelos (MDD), que dan soporte a la metodología que permite la generación de soluciones tecnológicas para colaboraciones interorganizacionales dinámicas, definida en el capítulo 1, sección 3. Estos métodos de transformación de modelos permiten realizar el desarrollo dirigido por modelos de los agentes *AdministradorDeProceso*, para generar la implementación en tiempo de ejecución de estos agentes en la plataforma de agentes propuesta en el capítulo 2. Estos métodos son automatizados por el agente GI de dicha plataforma. Los artefactos (modelos y código) generados por estos métodos están basados en los conceptos y componentes de la plataforma de implementación Jadex, sobre la que se implementan los agentes propuestos en el capítulo 2.

En primer lugar, se presenta el método de transformación para generar un modelo de proceso ejecutable, que implementa un proceso de integración, desde un modelo conceptual del proceso de integración. El modelo generado es especificado para su implementación mediante el componente Jadex-Processes (sección 5.1). El método de transformación para generar un modelo de un agente Jadex-BDI, que representa el modelo de un agente *AdministradorDeProceso*, a partir de un modelo conceptual de un proceso de integración, es luego definido (sección 5.2). El modelo generado está basado en el componente agente BDI de Jadex. Este modelo generado es utilizado por el tercer método de transformación que genera el código de un agente *AdministradorDeProceso* que representa la estructura y el comportamiento de un agente BDI para la plataforma Jadex (sección 5.3). Estos métodos con sus reglas de transformación se definieron usando el lenguaje y las herramientas de transformación de modelos ATL (véase la sección 2.2.3.2 para obtener una descripción de ATL).

## 5.1. Método de transformación para generar modelos de proceso ejecutable

Este método de transformación de modelo-a-modelo tiene por objetivo generar un modelo de proceso ejecutable que implementa un proceso de integración, en el formato del modelo Jadex-Processes, que luego puede ser interpretado y ejecutado por el componente BPMN-Kernel de la plataforma Jadex.

El método utiliza como entrada un modelo conceptual de un proceso de integración (modelo origen), el cual está especificado mediante el lenguaje estándar BPMN y definido en un nivel de modelo independiente de la plataforma (PIM). En la definición del patrón

de entrada del método de transformación se utiliza el metamodelo BPMN2 (metamodelo origen) que está basado en la especificación 2.0 de BPMN e implementado con el *Eclipse Modeling Framework* (EMF) (78). EMF es un componente *open-source* dentro del proyecto *Eclipse Modeling Project* de Eclipse (25), que permite implementar metamodelos y generar el código Java que posibilita manipular modelos basados en dichos metamodelos.

El método de transformación genera como salida un modelo de proceso ejecutable (modelo destino) como modelo Jadex-Processes. Éste es un modelo específico de la plataforma (PSM). El modelo es derivado mediante los patrones de destino especificados en el método, los cuales están basados en el metamodelo Jadex-Processes (metamodelo destino). En el modelo de proceso ejecutable se definen anotaciones semánticas para todos los elementos del modelo origen. El metamodelo Jadex-Processes está implementado con EMF y es una extensión del proyecto *STP BPMN Modeler* de Eclipse. Éste combina las especificaciones de elementos basados en STP-BPMN con elementos del proyecto Jadex que permiten la definición de parámetros de ejecución de un modelo de proceso BPMN para la plataforma Jadex.

Como resultado, el modelo destino es un modelo de proceso ejecutable que implementa un proceso de integración, anotado con expresiones y parámetros basados en los conceptos de Jadex-Processes e interpretables por el componente BPMN-*Kernel* de la plataforma Jadex, el cual es un motor de procesos que ejecuta modelos BPMN.

En el método de transformación se definen reglas que permiten generar el modelo destino, dentro de las cuales se pueden destacar las reglas relacionadas con los elementos tarea (*task*) en el modelo origen. Por ejemplo, los elementos *task* que están definidos como tareas abstractas en el modelo conceptual de origen son trasformados en actividades de tipo *task* concretas con sus implementaciones en el modelo destino, lo cual permite generar un modelo de proceso ejecutable completo dentro de la plataforma de implementación.

En las siguientes subsecciones primero se describe el metamodelo de BPMN (metamodelo origen), luego el metamodelo de Jadex-Processes (metamodelo destino) y finalmente las reglas del método de transformación de modelos, cuyos patrones son definidos usando los elementos de los metamodelos de origen y destino.

### 5.1.1. Metamodelo del lenguaje BPMN

El metamodelo BPMN2 especifica los conceptos requeridos para definir un modelo conceptual de procesos de negocio con base en la versión 2.0 del lenguaje BPMN. En la propuesta de este libro, todo modelo conceptual de un proceso de integración (modelo origen) se define de acuerdo con el metamodelo BPMN2.

El elemento `DocumentRoot` contiene a la mayoría de las entidades del metamodelo. En el extracto del metamodelo BPMN2 presentado en la figura 5.1 se muestra que las entidades `UserTask`, `ServiceTask`, `SendTask` y `ReceiveTask` componen el elemento `DocumentRoot`. Todas estas entidades heredan del elemento Task, que a su

vez hereda de las entidades `Activity` e `InteractionNode`. En BPMN2, una tarea es una especialización de actividad y puede ser de un tipo específico: *userTask* es una tarea de *workflow* realizada por una persona a través de una aplicación; *serviceTask* es una tarea automática ejecutada por una aplicación; *sendTask* es una tarea que representa el envío de un mensaje a un participante externo; *receiveTask* es una tarea que representa la recepción de un mensaje enviado por un participante externo.

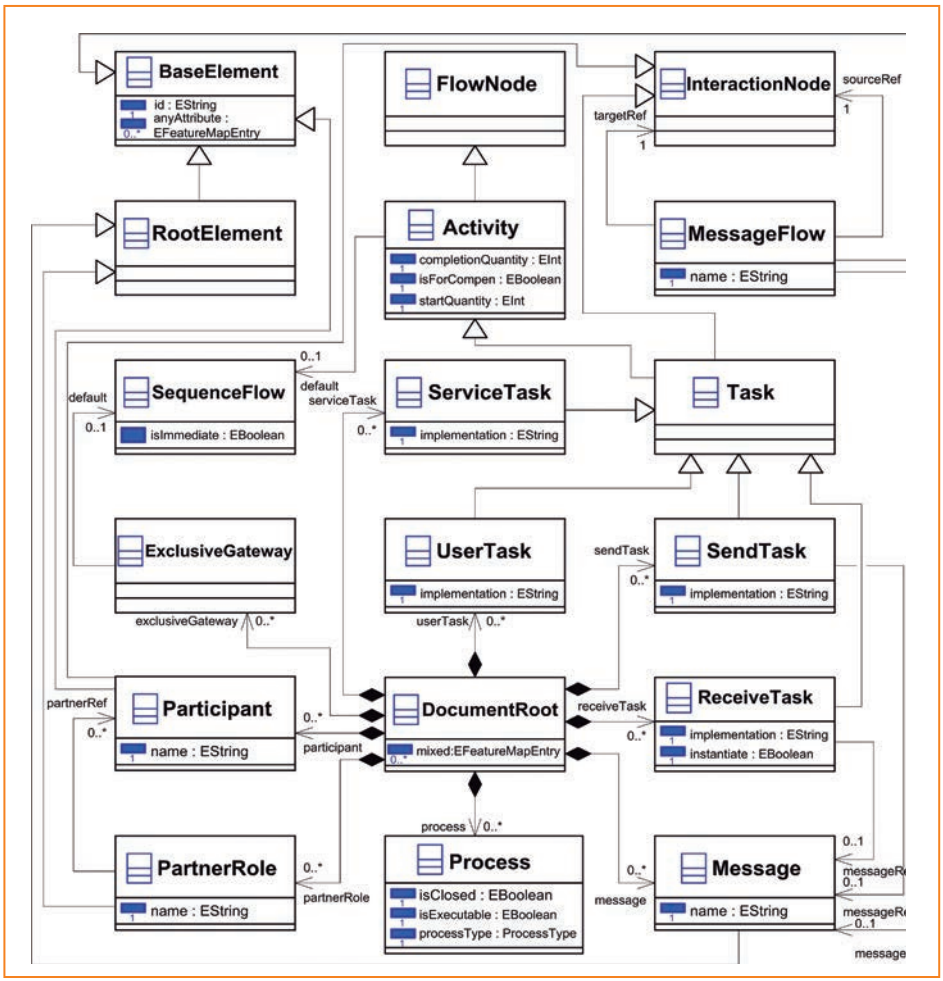



Figura 5.1. *Extracto del metamodelo BPMN2.*

Los elementos `Message`, `Process`, `PartnerRole`, `Participant` y `Exclusive-Gateway` también son contenidos por la entidad `DocumentRoot`. La entidad

Message-Flow define al atributo name de una referencia de mensaje entre actividades del diagrama y hereda del elemento BaseElement, al igual que la entidad Participant. El elemento BaseElement provee el atributo id que habilita la generación de un identificador para cada instancia de los elementos generados conforme al metamodelo. Cada entidad MessageFlow tiene un elemento InteractionNode como sourceRef, en que se define el elemento SendTask y ReceiveTask relacionado con el mensaje, y un elemento InteractionNode como targetRef, en que se especifica el participante emisor o receptor del mensaje, lo que permite representar el intercambio de mensajes entre los participantes. Las entidades SendTask y ReceiveTask pueden contener cero o un elemento Message que define el atributo name mediante la asociación messageRef.

Los elementos Message y PartnerRole heredan de la entidad RootElement. A su vez, RootElement hereda de la entidad BaseElement. La entidad PartnerRole define el atributo name, mediante el cual se le asigna el nombre al rol que una organización desempeña en un proceso de negocio. Una entidad PartnerRole puede tener varios elementos Participant, que define el atributo name, a través del cual se especifica el nombre de una organización o participante en un proceso. El elemento Participant hereda de la entidad InteractionNode y de la entidad BaseElement, la cual provee el atributo id para cada participante.

Los elementos Activity y ExclusiveGateway pueden contener cero o un elemento SequenceFlow mediante la asociación default. El elemento SequenceFlow expresa las secuencias de actividades, eventos o gateways de los caminos de un proceso. La entidad Activity hereda de la entidad abstracta FlowNode, que a su vez hereda del elemento FlowElement, que define el atributo name para todas sus instancias.

Un Gateway permite expresar la división y la unión o sincronización de los caminos de un proceso. El elemento ExclusiveGateway hereda de la entidad abstracta Gateway, que define al atributo gatewayDirection. A su vez la entidad Gateway hereda del elemento FlowNode. Los diferentes tipos del elemento Gateway son: ExclusiveGateway representa una compuerta exclusiva divergente o de decisión basada en datos, se utiliza para crear caminos alternativos dentro del flujo de un proceso, así como también puede ser utilizada como una compuerta convergente, es decir, para unir caminos alternativos dentro de un proceso. InclusiveGateway se utiliza para crear caminos alternativos pero también caminos paralelos dentro del flujo del proceso. ParallelGateway es usado para sincronizar (combinar) o crear caminos paralelos. ComplexGateway representa una forma de modelar un comportamiento de sincronización complejo. EventBasedGateway representa un punto de ramificación en el proceso, donde los caminos alternativos que sigue la compuerta se basan en eventos que se producen u ocurren, en lugar de expresiones de evaluación de datos utilizadas por los tipos de compuertas.

### 5.1.2. Metamodelo de Jadex-Processes

El metamodelo Jadex-Processes especifica los elementos necesarios para definir un modelo de proceso de negocio basado en la notación BPMN con la capacidad de ejecutarse en la plataforma Jadex a través de su componente Jadex-Processes. En la figura 5.2 se muestra un extracto del metamodelo de Jadex-Processes.

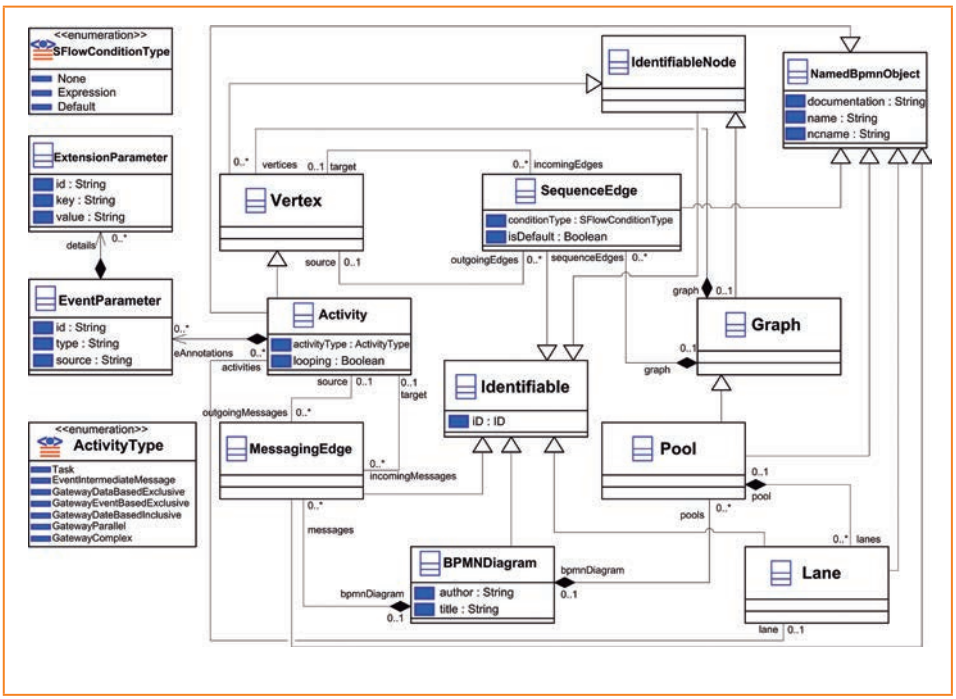



Figura 5.2. *Extracto del metamodelo Jadex-Processes.*

El elemento BPMNDiagram representa un modelo Jadex-Processes. Este elemento está compuesto por elementos MessagingEdge, y elementos Pool. Cada elemento BPMNDiagram se define con la información específica del autor y el título del diagrama del proceso de negocio. Los elementos BPMNDiagram, MessagingEdge, Lane, SequenceEdge e IdentifiableNode heredan de la clase Identifiable, la cual provee el atributo iD, que permite definir un identificador a las instancias de los elementos del diagrama.

Un `Pool` representa un participante de un proceso colaborativo y tiene la función de contener todos los elementos del proceso de dicho participante. Cada entidad `MessagingEdge` tiene un elemento `Activity` como origen y un elemento `Acti-`

vity como destino. El elemento Activity representa una tarea del proceso. A través de los MessagingEdge se define el flujo de mensajes del proceso. Cada entidad Activity puede tener varios elementos MessagingEdge con mensajes entrantes y salientes. La entidad Activity define los atributos looping y activityType. Este último utiliza la entidad ActivityType de tipo <<enumeration>> para especificar el valor del atributo. El atributo activityType indica el tipo de tarea de BPMN, así como también si la misma representa un evento intermedio de mensaje de envío o recepción.

La entidad Activity contiene al elemento EventParameter, que define a los atributos id, type y source, mediante los cuales se definen los datos del encabezado de la anotación de una entidad Activity. El elemento ExtensionParameter define los atributos key y value, a través de estos atributos se definen los detalles de los parámetros de ejecución de un elemento Activity; este elemento es contenido por la entidad EventParameter. En conjunto, las definiciones de los elementos EventParameter y ExtensionParameter habilitan la anotación de los parámetros de ejecución en los elementos de un modelo de proceso de negocio posibilitando su ejecución.

Los elementos SequenceEdge, Pool, Lane, MessagingEdge y Activity heredan de la entidad NameBpmnObject que define los atributos documentation, name y ncname. La entidad Pool hereda de la entidad Graph y puede contener varios elementos Lane. A su vez, esta entidad puede contener diversos elementos Activity a través de la asociación activities. Una Lane representa un recurso (persona o aplicación) que es resposable de realizar las actividades contenidas en la misma.

Las entidades Graph y Vertex permiten expresar el grafo de actividades de un modelo de procesos. Estas entidades heredan del elemento IdentifiableNode. Graph puede contener varios elementos de Vertex y SequenceEdge. La entidad SequenceEdge define los atributos conditionType e isDefault. Cada entidad Vertex puede tener varios elementos SequenceEdge como conectores de secuencia entrantes y salientes. A su vez, cada entidad SequenceEdge tiene un elemento Vertex como origen y un elemento Vertex como destino. A través de estas entidades se expresa el flujo de secuencia y de control de las tareas de un proceso.

### 5.1.3. Reglas de transformación para generar el modelo de proceso ejecutable

A continuación se describen las reglas de transformación declarativas que conforman el método que genera un modelo de proceso ejecutable como modelo Jadex-Processes, a partir de un modelo conceptual de proceso de integración definido con BPMN2.

### 5.1.3.1. Regla 1: participant2pool

Esta regla establece que por cada participante en el modelo origen se genera un *pool* en el modelo destino. El *pool* se etiqueta con el nombre del participante.

El patrón de entrada (`from`) y el patrón de salida (`to`) de la regla `participant2pool` se muestran en la figura 5.3. El patrón de entrada permite localizar todos los objetos `Participant` que tengan definida una referencia a un proceso en el modelo origen, generando un objeto `pool` en el modelo destino por cada coincidencia encontrada.

```
rule participant2pool {
        from
                participant: MMbpmn2!Participant (
                        not participant.processRef.oclIsUndefined()
                )
        to
                pool: MMjadex!Pool (
                        iD <- participant.id,
                        name <- participant.name,
                        vertices <- thisModule.getAllFlowNodes(),
                        sequenceEdges <- thisModule.getAllSequenceFlows()
                )
}
```

Figura 5.3. *Especificación de la regla de transformación* participant2pool.

En el patrón de salida, el atributo `iD` es inicializado con el valor del atributo `id` del modelo origen, lo cual permite mantener el identificador único para el participante de la colaboración. También el atributo de `name` del modelo destino es inicializado con el valor del atributo `name` del objeto `Participant` del metamodelo origen, con lo cual se mantiene el nombre del participante de la colaboración en el `pool`.

### 5.1.3.2. Regla 2: sendTask2ThrowEvent

Esta regla establece que por cada tarea de tipo *send* se genera en el modelo destino un evento intermedio de mensaje anotado en su semántica de ejecución como de tipo *throwing*, lo cual permite representar en el evento la acción de enviar el contenido del mensaje. Además, en el evento intermedio de mensaje se definen los parámetros del receptor (el otro participante de la colaboración), el acto de comunicación contenido en el mensaje (basado en especificaciones FIPA), el nombre del evento y el documento de negocio a enviar en el mensaje.

Los patrones de la regla `sendTask2ThrowEvent` se muestran en la figura 5.4. El patrón de entrada obtiene todos los objetos `SendTasK` en el modelo origen. El patrón de destino crea en el modelo destino un objeto `Activity` de tipo `EventInterme-`

diateMessage por cada coincidencia del patrón de entrada en el modelo origen. Los atributos iD y name son inicializados con el valor de los atributos id y name del objeto SendTask del modelo origen, los cuales representan el identificador y nombre del evento de mensaje. El evento intermedio de mensaje es anotado con el tipo *enviar* mediante el elemento EventParameter, en donde el atributo source es inicializado con el valor *string* de isThrowing. Los detalles de la anotación se realizan a través de un elemento ExtensionParameter, inicializando los atributos key y value con los valores isThrowing y true respectivamente.

```
rule sendTask2ThrowEvent {
	from
		task: MMbpmn2!SendTask
	to
		event: MMjadex!Activity (
			iD <- task.id,
			activityType <- #EventIntermediateMessage,
			name <- task.name,
			incomingEdges <- task.incoming,
			outgoingEdges <- task.outgoing,
			eAnnotations <- annotation1,
			eAnnotations <- annotation2
		),
		annotation1: MMjadex!EventParameter (
			id <- task.id + 'annotation1',
			source <- 'isThrowing',
			details <- annotationdetail0
		),
		annotation2: MMjadex!EventParameter (
			id <- task.id + 'annotation2',
			source <- 'jadex_parameters_table',
			details <- parameter0,
			details <- parameter1,
			details <- parameter2,
			details <- parameter3,
			details <- parameter4,
			details <- parameter5,
			details <- parameter6,
			details <- parameter7,
			details <- parameter8,
			details <- parameter9,
			details <- parameter10
		),
		annotationdetail0: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'isThrowing',
			value <- 'true'
		),
		parameter0: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- 'dimension',
			value <- '4:2'
		),
		parameter1: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- 'uniqueColumnIndex',
			value <- '0'
		),
		parameter2: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- 'complexColumns',
			value <- 'false:false'
		),
		parameter3: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '0:0',
			value <- 'type'
		),
		parameter4: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '0:1',
			value <- task.name
		),
		parameter5: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '1:0',
			value <- 'receivers'
		),
		parameter6: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '1:1',
			value <- thisModule.getReceiver().name
		),
		parameter7: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '2:0',
			value <- 'performative'
		),
		parameter8: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '2:1',
			value <- task.properties.first().name
		),
		parameter9: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '3:0',
			value <- 'content'
		),
		parameter10: MMjadex!ExtensionParameter (
			id <- annotation2.id,
			key <- '3:1',
			value <- task.messageRef.name
		)
}
```

Figura 5.4. *Especificación de regla de transformación* sendTask2ThrowEvent.

Los restantes parámetros de ejecución del evento intermedio de mensaje son construidos mediante la definición de una tabla de parámetros Jadex (jadex_parameters_table) utilizando el elemento EventParameter del metamodelo destino. Los detalles

de la tabla de parámetros son construidos con el uso de la relación details existente entre los elementos `EventParameter` y `ExtensionParameter`. Mediante esta relación se define el valor del parámetro y su ubicación en la tabla.

En la figura 5.5 se presenta un ejemplo de una tabla de parámetros de un evento intermedio de mensaje definida mediante el editor gráfico de Jadex-Processes, en donde `type` de la columna `Name` es especificado con el atributo `key` con un valor 0:0 que representa el renglón `0` y columna `0` de la tabla y el atributo `value` es inicializado con el valor `type`; y `send_proposal` estará ubicado en el renglón `0` con columna `1` definido en el atributo `key` con un valor `0:1` y con el atributo `value` inicializado con el valor `send_proposal`. De esta forma, los parámetros de ejecución requeridos son ubicados mediante la definición de una ubicación de valor de un atributo en una tabla de parámetros a través de la combinación de reglones y columnas. Esta especificación de parámetros es llevada a cabo para los elementos de tipo tarea, evento intermedio de mensaje y conector de secuencia.

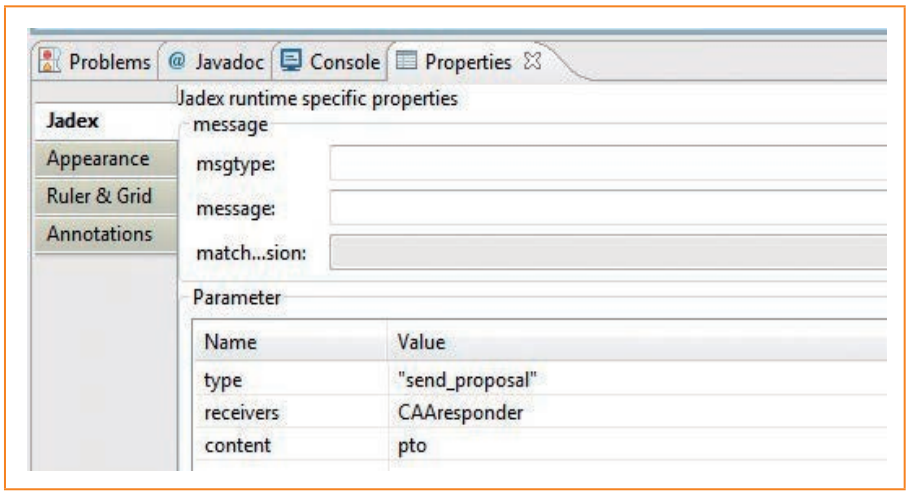



Figura 5.5. *Definición de la tabla de parámetros de un evento de mensaje usando el editor de Jadex-Processes.*

En la variable `parameter4` se asigna el valor `0:1` al atributo `key` y se inicializa el atributo `value` con el nombre de la tarea del modelo origen mediante el atributo `name` de dicha tarea. La variable `parameter6` permite inicializar los atributos `key` y `value` con el valor `1:1` y el nombre del receptor del mensaje utilizando el *helper* `thismodule.getReceiver().name`, el cual localiza el nombre del receptor mediante el uso del elemento `Participant` del metamodelo origen. En la variable `parameter8` se inicializa el atributo `key` con el valor `2:1` y el atributo `value` se inicializa con el valor del nombre del acto de comunicación, el cual es obtenido de las propiedades del objeto `task` del modelo origen.

La variable `parameter10` inicializa el atributo `key` con el valor `3:1` y el atributo value es inicializado con el atributo `task.messageRef.name` que representa el nombre del documento de negocio contenido en la tarea de enviar del modelo origen. Los restantes parámetros representan configuraciones requeridas en el modelo destino que no se derivan de objetos o atributos del modelo origen. El atributo id que representa el identificador de los parámetros es inicializado con el valor de `annotation2.id`, el cual fue previamente construido usando la cadena `task.id + annotation2`, que permite tener un nombre único para estos identificadores.

### 5.1.3.3. Regla 3: receiveTask2CatchEvent

Esta regla establece que por cada tarea de tipo *receive* en el modelo origen, se genera un evento intermedio de mensaje en el modelo destino. Este evento de mensaje del modelo destino es anotado semánticamente definiendo los parámetros del evento, los cuales están compuestos por el acto de comunicación (llamado *performative* en Jadex-Processes) que representa la intención del mensaje y por el nombre del evento intermedio de mensaje.

```
rule receiveTask2CatchEvent {
    from
        task: MMbpmn2!ReceiveTask
    to
        event: MMjadex!Activity (
                id <- task.id,
                activityType <- #EventIntermediateMessage,
                name <- task.name,
                incomingEdges <- task.incoming,
                outgoingEdges <- task.outgoing,
                eAnnotations <-annotation3
        ),
        annotation3: MMjadex!EventParameter (
                id <- task.id + 'annotation3',
                source <- 'jadex_parameters_table',
                details <- parameter1,
                details <- parameter2,
                details <- parameter3,
                details <- parameter4,
                details <- parameter5,
                details <- parameter6,
                details <- parameter7
        ),
        parameter1: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- 'dimension',
                value <- '2:2'
        ),
        parameter2: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- 'uniqueColumnIndex',
                value <- '0'
        ),
        parameter3: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- 'complexColumns',
                value <- 'false:false'
        ),
        parameter4: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- '0:0',
                value <- 'type'
        ),
        parameter5: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- '0:1',
                value <- task.name
        ),
        parameter6: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- '1:0',
                value <- 'performative'
        ),
        parameter7: MMjadex!ExtensionParameter (
                id <- annotation3.id,
                key <- '1:1',
                value <- task.properties.first().name
        )
}
```

Figura 5.6. *Especificación de la regla de transformación* receiveTask2Catch-Event.

El evento intermedio de mensaje es anotado semánticamente mediante la definición de variables. Las variables `parameter1`, `parameter2` y `parameter3` generan la definición de la tabla que contiene la anotación del evento. Las variables `parame-`

ter4 y parameter6 permiten definir el nombre del parámetro, para ello se inicializa el atributo de value con los valores type y performative, respectivamente. Estos valores son ubicados en la tabla mediante el atributo key, el cual es inicializado con los valores 0:0 y 1:0 para cada parámetro. El atributo key en la variable parameter5 es inicializado con el valor 0:1 y el atributo value es inicializado con el valor contenido en el atributo name del objeto ReceiveTask del modelo origen.

La variable parameter7 permite definir el acto de comunicación que se utiliza en el evento de tipo catch, para lo cual el atributo key es inicializado con el valor 1:1, lo que corresponde a la ubicación en la tabla del atributo value. Este atributo es inicializado con el valor del acto de comunicación o *performative* contenido en el atributo task.properties.first().name de la tarea del modelo origen.

El identificador de cada una de las instancias de los parámetros del evento intermedio de mensaje es generado mediante la concatenación del task.id + annotation3. El valor contenido en el atributo id del objeto ReceiveTask corresponde al modelo origen.

### 5.1.3.4. Regla 4: exclusiveGateway2gatewayDataBased

Esta regla establece que por cada compuerta exclusiva en el modelo origen se genera una compuerta exclusiva basada en datos en el modelo destino.

La regla exclusiveGateway2gatewayDataBased se detalla en la figura 5.7. Por cada elemento ExclusiveGateway localizado en el modelo origen se genera un elemento Activity de tipo GatewayDataBasedExclusive en el modelo destino. El identificador de la compuerta es generado en el atributo iD, el cual es inicializado con el valor del atributo id del elemento ExclusiveGateway del modelo origen. Se generan arcos de entrada y salida del objeto ExclusiveGateway del modelo origen.

```
rule exclusiveGateway2gatewayDataBased {
        from
                exclusiveGateway: MMbpmn2!ExclusiveGateway
        to
                gatewayDatabased: MMjadex!Activity (
                        iD <- exclusiveGateway.id,
                        activityType <- #GatewayDataBasedExclusive,
                        incomingEdges <- exclusiveGateway.incoming,
                        outgoingEdges <- exclusiveGateway.outgoing
                )
}
```

Figura 5.7. *Especificación de la regla de transformación* exclusiveGateway2gatewayDataBased.

### 5.1.3.5. Regla 5: parallelGateway2gatewayParallel

Esta regla establece que por cada compuerta paralela en el modelo origen se genera una compuerta paralela en el modelo destino con similares características.

La regla `parallelGateway2gatewayParallel` se detalla en la figura 5.8. La regla genera un objeto `Activity` del tipo `GatewayParallel` en el modelo destino por cada coincidencia encontrada en el modelo origen de un objeto `ParallelGateway`. El identificador de la compuerta paralela (atributo `iD`) es inicializado con el valor del identificador (atributo `id`) del modelo origen. Se generan arcos de entrada y salida de un objeto `GatewayParallel` por cada entrada y salida del correspondiente objeto `ParallelGateway` del modelo origen.

```
rule parallelGateway2gatewayParallel {
        from
                parallelGateway: MMbpmn2!ParallelGateway
        to
                gatewayParallel: MMjadex!Activity (
                        iD <- parallelGateway.id,
                        activityType <- #GatewayParallel,
                        incomingEdges <- parallelGateway.incoming,
                        outgoingEdges <- parallelGateway.outgoing
                )
}
```

Figura 5.8. *Especificación de la regla de transformación* parallelGateway2gatewayParallel.

### 5.1.3.6. Regla 6: EventGateway2gatewayEventBased

Esta regla establece que por cada compuerta basada en eventos en el modelo origen se genera una compuerta exclusiva basada en eventos en el modelo del proceso de negocio ejecutable.

La regla `EventGateway2gatewayEventBased` se detalla en la figura 5.9. La regla define que por cada elemento `EventBasedGateway` localizado en el modelo origen se debe crear un elemento `Activity` del tipo `GatewayEventBasedExclusive` en el modelo destino.

El identificador de cada instancia de la actividad se inicializa en el atributo `iD` del modelo destino con el valor del atributo `id` del elemento `EventBasedGateway` del modelo origen. Se generan arcos de entrada y salida de un `GatewayEventBasedExclusive` por cada entrada y salida del correspondiente objeto `EventBasedGateway` del modelo origen.

```
rule EventGateway2gatewayEventBased {
        from
                eventGateway: MMbpmn2!EventBasedGateway
        to
                gatewayEventBased: MMjadex!Activity (
                        iD <- eventGateway.id,
                        activityType <- #GatewayEventBasedExclusive,
                        incomingEdges <- eventGateway.incoming,
                        outgoingEdges <- eventGateway.outgoing
                )
}
```

Figura 5.9. *Especificación de la regla de transformación* EventGateway2gatewayEventBased.

### 5.1.3.7. Regla 7: sequenceFlow2incomingEdge

Por cada conector de flujo de secuencia en el modelo origen se genera un conector de secuencia en el modelo destino. El conector se anota semánticamente dependiendo del elemento anterior o posterior existente en el modelo destino. Cuando el elemento posterior al conector sea una tarea etiquetada con *evaluate* o *store* se genera un mapeo en el conector de secuencia asignado a una condición que permita recuperar el documento de negocio contenido en el evento intermedio de mensaje previamente recibido. Este documento es almacenado en una variable temporal, la cual es utilizada por la siguiente tarea (*evaluate* o *store*) como entrada en su ejecución. En el caso del conector que precede de una compuerta exclusiva basada en datos se comprueba cuál conector se encuentra asignado como camino por default en el modelo origen y se anota con ese estatus en el modelo destino. Los conectores de secuencia que no se incluyen en los tipos anteriores se deben generar con similares características a las del modelo origen.

La generación del conector de secuencia en el modelo destino se realiza mediante la implementación de tres reglas de transformación que solucionan los posibles tipos de anotación del conector de secuencia en el modelo destino.

La regla `sequenceFlow2incomingEdge` presentada en la figura 5.10 permite generar un objeto `SequenceEdge` en el modelo destino por cada objeto `Sequence-Flow` cuyo destino sea un objeto `ServiceTask` etiquetado como `Store` o un objeto `User-Task` etiquetado como `Evaluate` en el modelo origen y que no se encuentre identificado como conector por `default`. El identificador del conector generado en el modelo destino es inicializado con el valor del atributo `id` del objeto `SequenceFlow` del modelo origen. La anotación del conector de secuencia se realiza mediante una tabla de parámetros que está compuesta por el nombre del documento negocio y por la sintaxis de la instrucción que permite recuperar el contenido de un evento intermedio de mensaje. En la variable `parameter0` se inicializa el atributo `key` con el valor `0:0`

que permite generar la ubicación en la tabla del nombre de una variable local que almacenará el documento de negocio contenido en el mensaje recibido, y el atributo `value` es inicializado con el nombre de la variable local `doc_name_IN`.

```
rule sequenceFlow2incomingEdge {
        from
                sequenceflow: MMbpmn2!SequenceFlow (
                        sequenceflow.targetIsEvaluateOrStore and
                        not sequenceflow.isDefault
                )
        to
                incomingEdge: MMjadex!SequenceEdge (
                        id <- sequenceflow.id,
                        eAnnotations <- annotation0
                ),
                annotation0: MMjadex!EventParameter (
                        id <- sequenceflow.id + 'annotation0',
                        source <- 'jadex_mappings_table',
                        details <- parameter0,
                        details <- parameter1
                ),
                parameter0: MMjadex!ExtensionParameter (
                        id <- annotation0.id,
                        key <- '0:0',
                        value <- 'doc_name_IN'
                ),
                parameter1: MMjadex!ExtensionParameter (
                        id <- annotation0.id,
                        key <- '0:1',
                        value <- '$event.content'
                )
}
```

Figura 5.10. *Especificación de la regla de transformación* sequenceFlow2incomingEdge.

El atributo `key` de la variable `parameter1` es inicializado con el valor `0:1`, el cual indica la ubicación del valor en la tabla, y el atributo `value` es inicializado con valor `$event.content` que corresponde a una instrucción válida en la plataforma de ejecución del modelo destino. El identificador (atributo `id`) de los parámetros en el modelo destino es generado mediante la concatenación del valor del atributo `id` del elemento `sequenceflow` del modelo origen más el *string* `annotation0`. Este identificador es similar para las instancias generadas en el modelo destino mediante las variables `annotation0`, `parameter0` y `parameter1`.

La regla `sequenceFlow2incomingEdge_Default` presentada en la figura 5.11 permite configurar un camino `default` en una compuerta exclusiva basada en datos. La regla encuentra todas las instancias del objeto `SequenceFlow` marcado como `default` cuyo destino no sea un objeto `ServiceTask` o `UserTask` etiquetado con `Evaluate` o `Store`, respectivamente. Por cada coincidencia localizada se genera un objeto

SequenceEdge en el modelo destino. El atributo iD es inicializado con el valor del identificador contenido en el atributo id del elemento SequenceFlow del modelo origen. El atributo conditionType e isDefault del objeto SequenceEdge del modelo destino son inicializados con el valor *string* Default y el valor booleano true, respectivamente. Lo cual permite configurar el conector de secuencia en un estatus de default.

```
rule sequenceFlow2incomingEdge_Default {
        from
                sequenceflow: MMbpmn2!SequenceFlow (
                        not sequenceflow.targetIsEvaluateOrStore and
                        sequenceflow.isDefault
                )
        to
                sequenceEdge: MMjadex!SequenceEdge (
                        iD <- sequenceflow.id,
                        conditionType <- 'Default',
                        isDefault <- true
                )
}
```

Figura 5.11. *Especificación de la regla de transformación* sequenceFlow2incoming-Edge_Default.

La especificación de la regla sequenceFlow2incomingEdge_Others habilita la generación de un elemento SequenceEdge en el modelo destino por cada instancia del elemento SequenceFlow localizada en el modelo origen, la cual no debe estar etiquetada como conector de secuencia de default, y que el conector de secuencia no tenga como destino una tarea ServiceTask etiquetada como Store o una tarea UserTask etiquetada como Evaluate. Por cada coincidencia encontrada en el modelo origen se inicializa el atributo iD con el valor del atributo id del elemento SequenceFlow del modelo origen. La regla sequenceFlow2incomingEdge_Others es presentada en la figura 5.12.

```
le sequenceFlow2incomingEdge_Others {
      from
              sequenceflow: MMbpmn2!SequenceFlow (
                      not sequenceflow.targetIsEvaluateOrStore and
                      not sequenceflow.isDefault
              )
      to
              seqeunceEdge: MMjadex!SequenceEdge (
                      iD <- sequenceflow.id
              )
```

Figura 5.12. *Especificación de la regla de transformación* sequenceFlow2incoming-Edge_Others.

### 5.1.3.8. Regla 8: serviceTask2TaskStore

Esta regla establece que por cada tarea de tipo *service* que tenga la función de almacenar un documento de negocio en el modelo origen se genera una actividad de tipo *task* en el modelo destino. En la actividad se define la clase de implementación que permitirá su ejecución en la plataforma. Además, es anotada con los parámetros de entrada del documento a almacenar. Estos parámetros están compuestos por el tipo, el nombre y el contenido del documento de negocio.

La regla `ServiceTask2TaskStore` se detalla en la figura 5.13. Por cada objeto `ServiceTask` etiquetado como `Store` y encontrado en el modelo origen se genera un objeto `Activity` de tipo `Task` en el modelo destino, inicializando los atributos de `id` y `name` con los valores contenidos en los atributos `iD` y `name` del objeto `ServiceTask` del modelo origen. El objeto `Task` es anotado mediante el atributo `source` del elemento `EventParameter` del metamodelo destino. Este atributo es inicializado con el valor `jadex`, el cual habilita el uso de clases Java predefinidas en la plataforma. El parámetro de la clase que se utilizará en la tarea se define mediante la relación `details` entre los elementos `EventParameter` y `ExtensionParameter`.

Para ello, en la variable `parameter1` se inicializa el atributo `key` con el valor `class`, especificando que en la tarea se utilizará una clase Java, y el atributo `value` es inicializado con el valor proporcionado por el *helper* `task.getClassType`. En este *helper* se ha especificado el nombre de las clases Java predefinidas que permiten implementar los diferentes tipos de objetos `Task` en el modelo destino mediante la plataforma propuesta. En este caso, por el tipo de tarea, el *helper* asignará el valor `WriteContextTask` en el atributo `value` y mediante la ejecución de esta clase Java se almacena un documento de negocio en un repositorio.

La especificación de los parámetros de entrada de la clase Java (`WriteContextTask`) que permiten la implementación de la tarea se realiza mediante la variable `annotation1`, en la que se define una tabla que contiene los parámetros. Las variables `parameter2`, `parameter3` y `parameter4` definen la estructura de la tabla que está formada por un renglón con cuatro columnas. En la variable `parameter5` se inicializa el atributo `key` del elemento `ExtensionParameter` con el valor `0:0` que representa la ubicación del valor del atributo `value` en la tabla, y el atributo `value` se inicializa con el valor `in`, el cual indica que el documento contenido en la tabla es un parámetro de información de entrada de la tarea. Los atributos `key` y `value` de la variable `parameter6` son inicializados con el valor `0:1` y `document`, respectivamente. El valor del atributo `key` representa la celda ubicada entre la segunda columna y el primer renglón de la tabla. El valor del atributo `value` es el nombre con el que se almacenará el documento en el repositorio.

```
rule ServiceTask2TaskStore {
        from
                task: MMbpmn2!ServiceTask (
                        task.name.startsWith('Store')
                )
        to
                taskJadex: MMjadex!Activity (
                        iD <- task.id,
                        activityType <- #Task,
                        name <- task.name,
                        incomingEdges <- task.incoming,
                        outgoingEdges <- task.outgoing,
                        eAnnotations <- annotation0,
                        eAnnotations <- annotation1
                ),
                annotation0: MMjadex!EventParameter (
                        id <- task.id + 'annotation0',
                        source <- 'jadex',
                        details <- parameter1
                ),
                parameter1: MMjadex!ExtensionParameter (
                        id <- annotation0.id,
                        key <- 'class',
                        value <- task.getClassType
                ),
                annotation1: MMjadex!EventParameter (
                        id <- task.id + 'annotation1',
                        source <- 'jadex_parameters_table',
                        details <- parameter2,
                        details <- parameter3,
                        details <- parameter4,
                        details <- parameter5,
                        details <- parameter6,
                        details <- parameter7,
                        details <- parameter8
                ),
                parameter2: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- 'dimension',
                        value <- '1:4'
                ),
                parameter3: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- 'uniqueColumnIndex',
                        value <- '1'
                ),
                parameter4: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- 'complexColumns',
                        value <- 'false:false:false:false'
                ),
                parameter5: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- '0:0',
                        value <- 'in'
                ),
                parameter6: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- '0:1',
                        value <- 'new_doc'
                ),
                parameter7: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- '0:2',
                        value <- 'String'
                ),
                parameter8: MMjadex!ExtensionParameter (
                        id <- annotation1.id,
                        key <- '0:3',
                        value <- 'doc_name_IN'
                )
}
```

Figura 5.13. *Especificación de regla de transformación* serviceTask2TaskStore.

En la variable `parameter7` el atributo `key` es inicializado con el valor `0:2` y el atributo `value` es inicializado con el valor `String`, que representa el tipo de dato del documento que se almacena. Finalmente, en la variable `parameter8` se anotará el documento de negocio que se recibe del conector de secuencia previo a la tarea. El atributo `key` es inicializado con el valor `0:2`, que corresponde a la celda ubicada entre la cuarta columna y el primer renglón de la tabla, y el atributo `value` es inicializado con el valor `doc_name_IN`, que corresponde al nombre de la variable que contiene el documento de negocio, recuperado mediante la regla 7 de esta subsección, que consiste en especificar el conector de secuencia que relaciona un evento intermedio de mensaje y una tarea de tipo *service* etiquetada como `Store`.

### 5.1.3.9. Regla 9: userTask2TaskEvaluate

Esta regla establece que por cada tarea de tipo *user* etiquetada como *evaluate* en el modelo origen se genera una actividad de tipo *task* en el modelo destino. El elemento tarea se debe anotar semánticamente con la clase que permita su implementación en la

plataforma y con los parámetros de entrada y salida requeridos para su ejecución. Los parámetros de entrada están conformados por el nombre, el tipo de dato y el documento de negocio que será evaluado. En los parámetros de salida se deben incluir el nombre de una variable que almacenará el resultado de la evaluación y el tipo de dato de la variable.

La regla `UserTask2TaskEvaluate` se detalla en la figura 5.14. La misma genera un elemento `Activity` de tipo `Task` en el modelo destino por cada coincidencia encontrada del elemento `UserTask` etiquetado como `Evaluate` en el modelo origen. Los atributos `iD` y `name` del elemento `Activity` de tipo Task generado son inicializados con el valor del `id` y `name` del elemento `UserTask` del modelo origen; estos atributos contienen las instancias del identificador y nombre de la tarea.

El proceso de anotación semántica del elemento `Task` en los primeros parámetros es similar al implementado en la regla 8. Por tal motivo, se describen sólo los parámetros adicionales. En la variable `parameter1` se presenta un cambio en el atributo `value`, que es inicializado con el valor proporcionado por el *helper* `task.getClassType2`. Este *helper* proporciona una clase Java predefinida que permite la implementación de la tarea en la plataforma. En este caso, para una instancia del elemento `Task` etiquetado como `Evaluate` se asignará la clase Java `UserInteractionTask`. La implementación de esta clase permite al usuario de la plataforma definir el resultado de la evaluación realizada al documento de negocio.

El atributo `key` de la variable `parameter9` es inicializado con el valor `1:0`, correspondiente a la celda ubicada entre el segundo renglón y la primera columna de la tabla, en la cual se colocará el valor del atributo `value` que es inicializado con el *string* `out`, que indica la dirección del parámetro. En el `parameter10` se asigna al atributo `key` el valor `1:1` y el atributo `value` es inicializado con el valor `option`, que representa el nombre de la variable donde se almacena el resultado de la evaluación. En la variable `parameter11` el atributo `key` se inicializa con el valor que representa la posición entre el primer renglón y la tercera columna, y el atributo `value` es inicializado con el valor `Int`, que representa el tipo de dato de la variable definida en el `parameter10`.

```
rule UserTask2TaskEvaluate {
	from
		task: MMbpmn2!UserTask (
			task.name.startsWith('Evaluate')
		)
	to
		taskJadex: MMjadex!Activity (
			iD <- task.id,
			activityType <- #Task,
			name <- task.name,
			incomingEdges <- task.incoming,
			outgoingEdges <- task.outgoing,
			eAnnotations <- annotation0,
			eAnnotations <- annotation1
		),
		annotation0: MMjadex!EventParameter (
			id <- task.id + 'annotation0',
			source <- 'jadex',
			details <- parameter1
		),
		parameter1: MMjadex!ExtensionParameter (
			id <- annotation0.id,
			key <- 'class',
			value <- task.getClassType2
		),
		annotation1: MMjadex!EventParameter (
			id <- task.id + 'annotation1',
			source <- 'jadex_parameters_table',
			details <- parameter2,
			details <- parameter3,
			details <- parameter4,
			details <- parameter5,
			details <- parameter6,
			details <- parameter7,
			details <- parameter8,
			details <- parameter9,
			details <- parameter10,
			details <- parameter11,
			details <- parameter12
		),
		parameter2: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'dimension',
			value <- '2:4'
		),
		parameter3: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'uniqueColumnIndex',
			value <- '1'
		),
		parameter4: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'complexColumns',
			value <- 'false:false:false:false'
		),
		parameter5: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:0',
			value <- 'in'
		),
		parameter6: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:1',
			value <- 'document'
		),
		parameter7: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:2',
			value <- 'String'
		),
		parameter8: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:3',
			value <- 'doc_name_IN'
		),
		parameter9: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:0',
			value <- 'out'
		),
		parameter10: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:1',
			value <- 'option'
		),
		parameter11: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:2',
			value <- 'Int'
		),
		parameter12: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:3',
			value <- ''
		)
}
```

Figura 5.14. *Especificación de regla de transformación* userTask2TaskEvaluate.

En la variable `parameter12` se almacena el resultado de la evaluación realizada por el usuario de la plataforma, para lo cual el atributo `key` es inicializado con el valor `1:3` y el atributo `value` únicamente se define y su valor será actualizado cuando el modelo destino sea implementado. La ejecución de esta tarea mediante la implementación de la clase Java `UserInteractionTask` despliega un panel en el que el usuario establece el resultado de la evaluación mediante una sección de botones. El valor de la selección del usuario es almacenado en una instancia del atributo value de la variable `parameter12`.

### 5.1.3.10. Regla 10: serviceTask2TaskGenerate

Esta regla establece que por cada tarea de tipo *service* que tenga la función de generar un documento de negocio en el modelo origen se genera una actividad de tipo *task* en el modelo destino, que especifica la clase de implementación que permita su ejecución en la plataforma. La actividad de tipo tarea es anotada semánticamente con el nombre del documento a generar y los datos de identificación del agente responsable de crear el documento de negocio. Además, se define el documento de negocio de entrada de la tarea.

```
rule ServiceTask2TaskGenerate {
	from
		task: MMbpmn2!ServiceTask (
			task.name.startsWith('Generate')
		)
	to
		taskJadex: MMjadex!Activity (
			iD <- task.id,
			activityType <- #Task,
			name <- task.name,
			incomingEdges <- task.incoming,
			outgoingEdges <- task.outgoing,
			eAnnotations <- annotation0,
			eAnnotations <- annotation1
		),
		annotation0: MMjadex!EventParameter (
			id <- task.id + 'annotation0',
			source <- 'jadex',
			details <- parameter1
		),
		parameter1: MMjadex!ExtensionParameter (
			id <- annotation0.id,
			key <- 'class',
			value <- task.getClassType
		),
		annotation1: MMjadex!EventParameter (
			id <- task.id + 'annotation1',
			source <- 'jadex_parameters_table',
			details <- parameter2,
			details <- parameter3,
			details <- parameter4,
			details <- parameter5,
			details <- parameter6,
			details <- parameter7,
			details <- parameter8,
			details <- parameter9,
			details <- parameter10,
			details <- parameter11,
			details <- parameter12,
			details <- parameter13,
			details <- parameter14,
			details <- parameter15,
			details <- parameter16
		),
		parameter2: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'dimension',
			value <- '3:4'
		),
		parameter3: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'uniqueColumnIndex',
			value <- '1'
		),
		parameter4: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- 'complexColumns',
			value <- 'false:false:false:false'
		),
		parameter5: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:0',
			value <- 'in'
		),
		parameter6: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:1',
			value <- 'new_doc'
		),
		parameter7: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:2',
			value <- 'String'
		),
		parameter8: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '0:3',
			value <- 'doc_name'
		),
		parameter9: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:0',
			value <- 'inout'
		),
		parameter10: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:1',
			value <- 'responder'
		),
		parameter11: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:2',
			value <- 'IComponentIdentifier[]'
		),
		parameter12: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '1:3',
			value <- ''
		),
		parameter13: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '2:0',
			value <- 'out'
		),
		parameter14: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '2:1',
			value <- 'document'
		),
		parameter15: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '2:2',
			value <- 'String'
		),
		parameter16: MMjadex!ExtensionParameter (
			id <- annotation1.id,
			key <- '2:3',
			value <- ''
		)
}
```

Figura 5.15. *Especificación de la regla* serviceTask2TaskGenerate.

La regla `ServiceTask2TaskGenerate` se detalla en la figura 5.15. Por cada coincidencia del objeto `ServiceTask` etiquetado como `Generate` en el modelo origen se genera un objeto `Activity` de tipo `Task` en el modelo destino. Los atributos `iD` y `name` del objeto `Activity` son inicializados con el valor contenido en los atributos `id` y `name` del objeto `ServiceTask` del modelo origen. La variable `parameter1` permite anotar la tarea con la clase Java que habilita su implementación. El atributo `key` de esta variable es inicializado con el valor `class`, y el atributo `value` es inicializado con el valor `GenerateDocTask` proporcionado por el *helper* `task.getClassType`, el cual asigna la clase Java dependiendo de la etiqueta identificada en el objeto `ServiceTask` del modelo origen.

Las variables `parameter2`, `parameter3` y `parameter4` permiten definir una tabla de parámetros similar a la mostrada en la figura 5.16. De la variable `parameter5` a la variable `parameter8` se especifica el documento de entrada a la tarea, similar a los parámetros detallados en la regla 8.

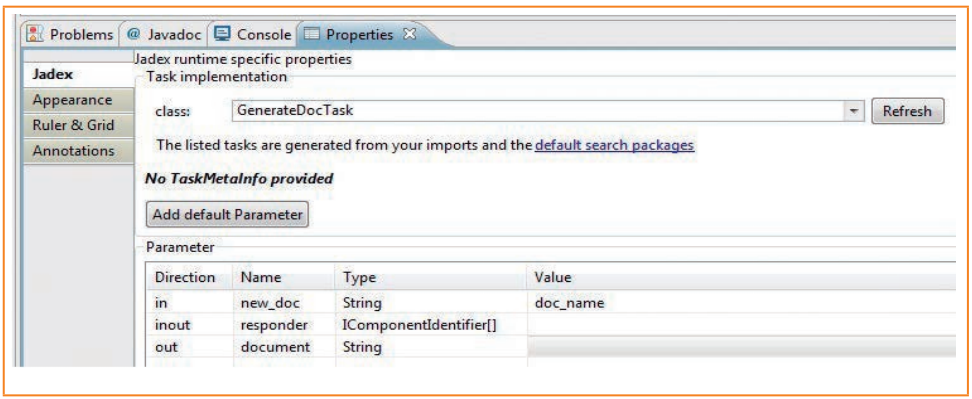



Figura 5.16. *Definición de la tabla de parámetros de una tarea* generate *usando Jadex-Processes.*

En la variable `parameter9` se define la dirección del parámetro mediante el atributo `value` del objeto `ExtensionParameter` que es inicializado con valor de `inout` y es asignado mediante la especificación del atributo `key` en la posición `1:0` que corresponde al segundo renglón con la primera columna de la tabla.

La variable `parameter10` permite generar el nombre del rol que ejecuta la actividad en el proceso de integración, para lo cual el atributo `value` es inicializado con el valor asignado por el *helper* `thisModule.getRoleEntity` y su ubicación es determinada por el valor `1:1` del atributo `key`. En el `parameter11`, el atributo `value` es inicializado con el valor `IComponentIdentifier[]`. El valor asignado invoca a un método válido en la plataforma de ejecución, mediante el cual se agrega el identificador del agente (componente Jadex) que implementa el modelo de proceso ejecutable en el documento generado por esta tarea. En la misma variable, el atributo `key` es inicializado con el valor `1:2`.

En la variable `parameter12` se almacena el valor del identificador del agente en tiempo de ejecución del proceso ejecutable mediante el atributo `value`, por lo cual el atributo sólo es definido y no es inicializado al momento de la transformación del modelo. Este parámetro es ubicado en la posición `1:2` de la tabla mediante el atributo `key`.

Mediante las variables `parameter13`, `parameter14`, `parameter15` y `parameter16` se definen los parámetros que permiten generar el documento de negocio. En la variable `parameter13` se define la dirección del parámetro a través del atributo `value` que es inicializado con el valor `out`. Este valor es colocado en la celda ubicada entre el tercer renglón y la primera columna de la tabla, lo cual es especificado en el atributo `key`, inicializándolo con el valor `2:0`.

En la variable `parameter14` se inicializa el atributo `value` con el nombre de la variable que almacenará el documento, en este caso `document`. Además, el atributo `key` es inicializado con el valor `2:1`, posición en la que se ubicará la variable en la tabla de parámetros.

El atributo `key` de la variable `parameter15` es inicializado con el valor `2:2,` y el atributo `value` se inicializa con el valor `String,` que representa el tipo de dato de la variable previamente definida. En la variable `parameter15` se define el atributo key inicializado con el valor `2:3` y el atributo `value` es solamente definido, debido a que su valor es generado en tiempo de ejecución. El valor del atributo `value` es el documento de negocio generado por el elemento `Activity` de tipo `Task`.

## 5.2. Método de transformación para generar modelos de agentes Jadex-BDI

Este método de transformación de modelo-a-modelo permite generar un modelo de agente Jadex-BDI, que representa un agente *AdministradorDeProceso*, a partir de un modelo conceptual de un proceso de integración. El modelo generado es definido utilizando conceptos de agentes BDI de la plataforma de agentes Jadex, y por lo tanto es un modelo específico de la plataforma (PSM).

Con el fin de soportar métodos de desarrollo dirigido por modelos MDD para generar agentes en Jadex se desarrolló el metamodelo Jadex-BDI, basado en los archivos de la versión 2.0 del Jadex XML Schema (67). En este metamodelo se incluyen los conceptos y recursos requeridos para el desarrollo de agentes BDI para la plataforma Jadex.

El modelo del agente Jadex-BDI es derivado a través de los patrones destino definidos en el método de transformación basados en el metamodelo Jadex-BDI. Mediante el método de transformación se construyen los siguientes elementos en un modelo de agente-BDI: `agent`, `goals`, `plans`, `events` y `configurations`. Éstos conforman la estructura básica de un agente de software en la plataforma Jadex y son derivados de los elementos `Participant`, `TextAnnotation`, `Process`, `SendTask`, `ReceiveTask` y `PartnerRole`, de un modelo conceptual de proceso de integración.

### 5.2.1. Metamodelo de la arquitectura de agentes Jadex-BDI

El metamodelo de Jadex-BDI especifica los conceptos que son necesarios para definir agentes y capacidades (*capabilities*). Los principales conceptos en la arquitectura de agentes Jadex-BDI son `MBDIAgent` y `MCapabilityBase`. Un `MBDIAgent` es un tipo de componente (`ComponentType`) de la plataforma Jadex, que representa un agente Jadex-BDI. Una `MCapability` es un tipo de capacidad (`CapabilityType`) que contiene una colección de entidades (`beliefs`, `goals`, `plans`) habilitando el empaquetamiento de sus funcionalidades dentro de *cluster*, con el propósito del reutilizar estas capacidades dentro de los agentes-BDI de una plataforma, a través de su especificación en el concepto `Imports`. La única diferencia entre agentesBDI y *capabilities* en Jadex es que los agentes poseen su propio *thread* de ejecución. En la figura 5.17 se presenta un extracto del metamodelo de la arquitectura de agentes Jadex-BDI.

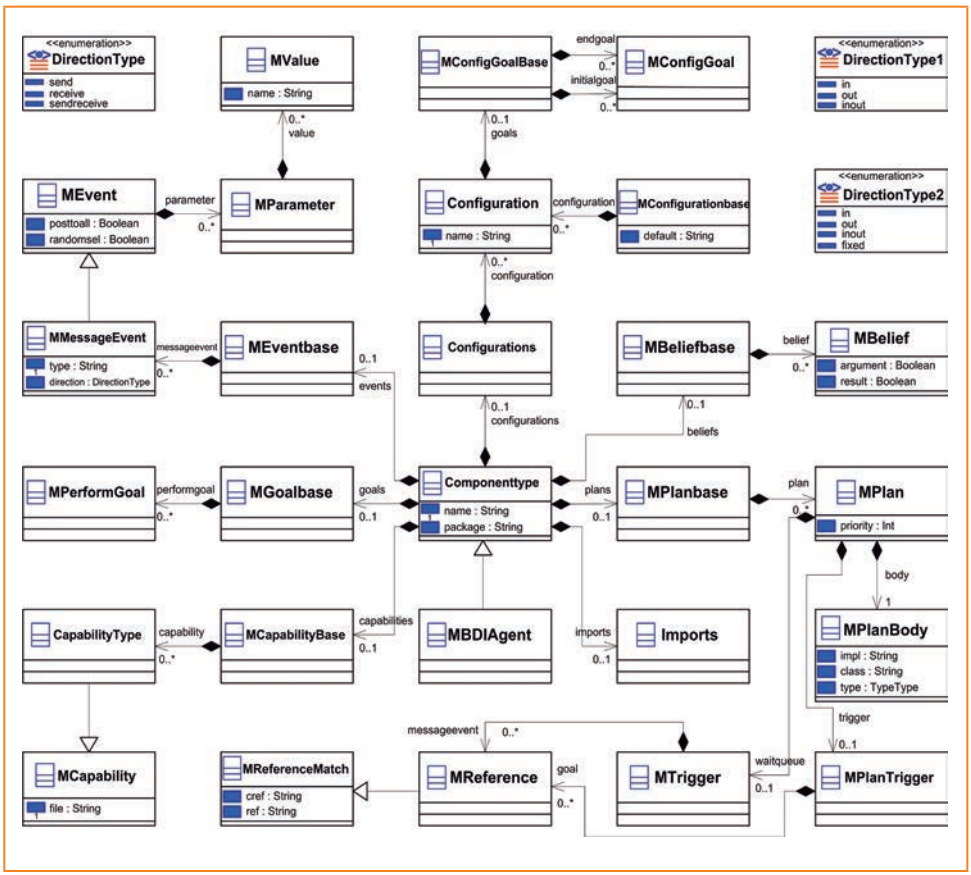



Figura 5.17. *Extracto del metamodelo de agentes Jadex-BDI.*

En el elemento ComponentType se pueden declarar varios eventos mediante los conceptos MInternalEvent y MMessageEvent, los cuales son contenidos por el concepto MEventbase. Los eventos de mensaje (MMessageEvent) representan los mensajes que se intercambian con otros agentes y los eventos internos (MInternalEvent) permiten la comunicación entre las entidades (beliefs, goals, plans) que conforman un agente. Por ejemplo: en la ejecución secuencial de planes, un evento interno activa un segundo plan cuando finaliza un primer plan. Igualmente, las declaraciones de una meta son contenidas por el concepto MGoalbase.

En Jadex existen cuatro tipos de metas: MAchieveGoal, MPerformGoal, MQueryGoal y MMaintainGoal, contenidas por el concepto MGoalbase. Los cuatro conceptos de metas pueden estar compuestos de varios elementos MGoal. Una meta de tipo MAchieveGoal representa el enfoque tradicional, el cual consiste en una definición de un estado del mundo o contexto deseado sin especificar cómo llegar a esa meta. La meta MPerformGoal está directamente relacionada con la ejecución de acciones, y se considera que la meta fue alcanzada cuando algunas acciones han sido ejecutadas, independientemente de los resultados de esas acciones. La meta MQueryGoal es similar a la meta MAchieveGoal, pero el estado deseado no es un estado del mundo (exterior), es un estado interno del agente. En la meta de tipo MMaintainGoal, un agente mantiene un registro de un estado deseado, y continuamente se ejecutarán planes adecuados para restablecer el estado determinado como deseado (67).

Los planes definidos en un agente son declarados mediante el concepto de MPlan, el cual es contenido por el elemento MPlanbase. Un elemento MPlan contiene un concepto MPlanBody que hace referencia a un plan del agente Jadex-BDI, el cual puede ser un plan basado en Java o un plan basado en modelos de procesos ejecutables BPMN (modelo Jadex-Processes). Mediante el concepto MPlanTrigger se hace referencia a los eventos que pueden desencadenar la ejecución de un plan en el agente.

Las creencias (beliefs) de un agente son especificadas a través del elemento MBelief, que es contenido por la entidad MBeliefbase. Las creencias representan el conocimiento de un agente y son consultadas y/o modificadas mediante los planes. Un cambio en una creencia está relacionado directamente con acciones, tales como la generación de un evento o que una meta sea alcanzada. Un elemento ComponentType puede contener varios objetos Configuration, los cuales son contenidos por el elemento Configurations. Mediante la sección de configuration, generada a través de estos elementos, un agente puede ser inicializado con un cierto número de beliefs, goals y/o plans.

## 5.2.2. Reglas de transformación para generar el modelo del agente Jadex-BDI

### 5.2.2.1. Regla 1: participant2agent

La regla establece que por cada organización participante encontrada en el modelo de proceso de integración (modelo origen), con un identificador de proceso definido, se genera un modelo de un agente *AdministradorDeProceso* (modelo destino) basado en los conceptos del agente BDI de Jadex.

La regla `participant2agent` se detalla en la figura 5.18. El patrón de entrada definido habilita localizar los objetos `Participant` en el modelo origen, con la condición de que cada objeto seleccionado tenga una referencia del proceso válida, generando un objeto `Componenttype` en el modelo destino por cada coincidencia encontrada por el patrón de entrada. El atributo `name` del objeto `Componenttype` es inicializado con la concatenación del valor contenido en el atributo `name` del objeto `Participant` más el valor en el atributo `id` del objeto `Process` del modelo origen. El valor del atributo `id` es recuperado mediante el *helper* `this.Module.getProcessID()`. El valor del atributo `name` del objeto `Componenttype` representa el nombre del agente con el cual se instanciará en la plataforma de agentes propuesta. El atributo `package` del objeto `Componenttype` se inicializa con el nombre del paquete en el cual se implementará el agente, en este caso, el paquete utilizado en la plataforma es `agents`.

```
rule participant2agent {
        from
                participant: MMbpmn2!Participant (
                        not participant.processRef.oclIsUndefined()
                )
        to
                agent: MMjadexbdi!Componenttype (
                        name <- participant.name + thisModule.getProcessID(),
                        package <- 'agents'
                )
}
```

Figura 5.18. *Especificación de la regla de transformación* participant2agent.

### 5.2.2.2. Regla 2: businessgoal2goals

Esta regla establece que por cada meta de negocio en el modelo origen se genera una meta de ejecución en el modelo destino.

La regla `businessgoal2goals` se muestra en la figura 5.19. El patrón de entrada de la regla habilita localizar todos los objetos `TextAnnotation` en el modelo origen

que estén etiquetados en el inicio con el texto `Goal:`, generando un objeto `MGoalbase` en el modelo destino por cada coincidencia encontrada por el patrón de entrada. La especificación de la meta del agente se realiza a través de la relación `performgoal` existente entre el elemento `MGoalbase` y el elemento `MPerformGoal` en el metamodelo destino. En la variable `goal` se define el nombre de la meta mediante el atributo name del elemento `MPerformGoal`, que es inicializado con el valor del atributo `text` de la meta de negocio del modelo origen concatenado con el *string* `-Goal`.

```
rule businessgoal2goals{
        from
                bgoals: MMbpmn2!TextAnnotation
                        bgoals.text.startsWith('Goal:')
        to
                goals: MMjadexbdi!MGoalbase(
                        performgoal <- goal
                ),

                goal: MMjadexbdi!MPerformGoal(
                        name <- bgoals.text +'-Goal'
                )
}
```

Figura 5.19. *Especificación de la regla de transformación* businessgoal2goals.

### 5.2.2.3. Regla 3: process2plans

Esta regla establece que por cada proceso en el modelo origen se genera un plan en el modelo destino, definiendo el nombre del proceso de negocio BPMN a ejecutar. Además se anotan los eventos de mensaje que condicionan la ejecución del plan y la meta alcanzada por el agente al finalizar el plan.

La regla `process2plans` se muestra en la figura 5.20. La regla habilita la generación de un elemento `MPlan` en el modelo destino por cada instancia del elemento `Process` localizada en el modelo origen. El nombre del plan del modelo destino (atributo `name` del elemento `MPlan`) se genera mediante la concatenación del valor contenido en el atributo `process.name` del `objeto process` del modelo origen más el *string* `-plan`.

Los datos del proceso integración a ejecutar son copiados al atributo `body`, el cual está compuesto por el tipo de plan a implementar y el nombre del archivo que contiene el plan. En la variable `datosplan` se inicializa el atributo `type` con el valor `bpmn`, que representa el tipo de plan, y en el atributo `impl` es copiado el nombre del plan, obtenido mediante el atributo `name` del proceso del modelo origen. Mediante la relación `waitqueue` entre los elementos `MPlan` y `MTrigger` se definen los eventos de mensaje

que condicionan la ejecución del plan, esto es llevado a cabo en la variable `datosplan2`. Estos eventos de mensaje son construidos mediante la relación `messageevent` existente entre los objetos `MTrigger` y `MReference` del metamodelo destino.

```
rule process2plans{
     from
          process: MMbpmn2!Process
     to
          plan: MMjadexbdi!MPlan(
                name <- process.name + '-plan',
                body <- datosplan,
                waitqueue <- datosplan2,
                trigger <- datosplan3
          ),
          datosplan: MMjadexbdi!MPlanBody(
                type <- 'bpmn',
                impl <- process.name
          ),
          datosplan2: MMjadexbdi!MTrigger(
                messageevent <- Sequence {MMbpmn2!ReceiveTask.allInstances()
                                          -> collect(i | thisModule.process2dataPlan(i))}
          ),
          datosplan3: MMjadexbdi!MPlanTrigger(
                goal <- metaref
          ),
          metaref: MMjadexbdi!MReference(
                ref <- thisModule.getGoals() +'-Goal'
          )
}
```

Figura 5.20. *Especificación de la regla de transformación* process2plans.

El atributo `ref` es inicializado con el valor asignado por la regla de tipo *lazy* `process2dataPlan` presentada en la figura 5.21, que es invocada en el atributo `messageevent` de la variable `datosplan2`. Mediante la invocación a esta regla se recupera el nombre de los eventos de mensaje de tipo `receive` que desencadenan la ejecución de las tareas contenidas en el plan del agente, iterando todas las instancias del elemento `ReceiveTask` contenidas en el modelo origen. Las reglas tipo *lazy* se caracterizan porque sólo son aplicadas en un método de transformación cuando son llamadas por otra regla que utiliza programación imperativa. Una regla *lazy* puede ser llamada múltiples veces (iterar) en una transformación.

Los detalles de la referencia de la meta que el agente alcanzará al finalizar la ejecución del plan son construidos con el uso de la relación `goal` existente entre los elementos `MPlanTrigger` y `MReference`. En la variable `metaref`, el atributo `ref` es inicializado con el valor proporcionado por el *helper* `thisModule.getGoals`, el cual recupera el nombre de la meta.

```
lazy rule process2dataPlan{
        from
                process: MMbpmn2!ReceiveTask
        to
                mensajeref: MMjadexbdi!MReference(
                        ref <- process.name
                )

}
```

Figura 5.21. *Especificación de la regla de transformación de tipo* lazy process2dataPlan.

### 5.2.2.4. Regla 4: sendTask2Eventsend

Esta regla establece que por cada tarea de tipo *send* en el modelo origen se genera un evento de mensaje con dirección de tipo *send* en el modelo destino. Los eventos de mensaje generados se anotan con el nombre, la dirección y el lenguaje de comunicación del agente utilizado en la implementación. En el evento de mensaje se definen los parámetros de ejecución, conformados por el acto de comunicación del mensaje y el lenguaje de definición del contenido del mensaje.

La regla `SendTask2Eventsend` se muestra en la figura 5.22. La regla habilita la generación de un elemento `MMessageEvent` en el modelo destino por cada objeto `SendTask` encontrado en el modelo origen. El atributo `name` es inicializado con el valor del atributo `name` del objeto de tipo `SendTask` del modelo origen. El atributo `type` es inicializado con el valor `fipa` que representa el nombre del lenguaje de comunicación que los agentes utilizarán. El atributo `direction` es inicializado con el valor `#send`, el cual es un tipo de dirección válido definido en el elemento `DirectionType` de tipo `<<enumeration>>` en el metamodelo destino. Los parámetros del evento de mensaje son definidos mediante la relación `parameter` existente entre los elementos `MMessageEvent` y `MParameter` del metamodelo destino.

En la variable `datos1` se especifican las características del acto de comunicación (llamado *performative* en Jadex-BDI) utilizado en el evento de mensaje. El atributo `name` es inicializado con el valor `performative`, lo cual determina que se utilizará un acto de comunicación. En el atributo `class` se define el formato de los datos; en este caso, el tipo de dato que se utiliza para un acto de comunicación es `String`. El atributo `direction` es inicializado con el valor `#fixed`, un tipo de dirección de uso exclusivo en la arquitectura Jadex-BDI (14) para el manejo de eventos de mensaje de tipo *send* o *receive* y que se encuentra definido en el elemento `DirectionType1` de tipo `<<enumeration>>` del metamodelo destino. La especificación del acto de comunicación se realiza mediante la relación `value` entre los objetos de tipo `MParameter` y `MValue` del modelo destino.

Entonces, el atributo `name` de la variable `valor1` es inicializado con la concatenación del *string* `SFipa`. más el valor contenido en el atributo `throw.properties.first().name`, en el cual se almacena el acto de comunicación utilizado en las instancias del objeto `SendTask` del modelo origen.

```
rule SendTask2Eventsend{
        from
                throw: MMbpmn2!SendTask
        to
                events: MMjadexbdi!MMessageEvent(
                        name <- throw.name,
                        type <- 'fipa',
                        direction <- #send,
                        parameter <- datos1,
                        parameter <- datos2
                ),
                datos1: MMjadexbdi!MParameter(
                        name <- 'performative',
                        class <- 'String',
                        direction <- #fixed,
                        value <- valor1
                ),
                valor1: MMjadexbdi!MValue(
                        name <- 'SFipa.'+ throw.properties.first().name
                ),
                datos2: MMjadexbdi!MParameter(
                        name <- 'language',
                        class <- 'String',
                        direction <- #fixed,
                        value <- valor2
                ),
                valor2: MMjadexbdi!MValue(
                name <- 'SFipa.JADEX.XML'
                )
}
```

Figura 5.22. *Especificación de la regla de transformación* sendTask2Eventsend.

En la variable `datos2` se define el lenguaje que se utilizará para el contenido del mensaje que intercambian los agentes en la plataforma. Este requerimiento es obligatorio en los eventos de mensaje de tipo `send`. El atributo `name` es inicializado con el valor `language`, lo cual determina que en un parámetro de nivel inferior se encontrará el tipo de lenguaje a utilizar. Los atributos `class` y `direction` son inicializados con los valores `String` y `#fixed`, respectivamente. En la variable `valor2`, el atributo `name` del objeto de tipo `MValue` es inicializado con valor `Jadex.XML`, con lo cual se determina el lenguaje que se utilizará en el intercambio de información entre los agentes.

### 5.2.2.5. Regla 5: receiveTask2Eventreceive

Esta regla establece que por cada tarea de *receive* en el modelo origen se debe generar un evento de mensaje con dirección de tipo *receive* en el modelo destino. Al evento generado en el modelo destino se le deben definir los parámetros del lenguaje de comunicación del agente y el acto de comunicación que utilizará.

La regla `ReceiveTask2Eventreceive` se muestra en la figura 5.23. En la regla de transformación se define que por cada instancia que coincida con el objeto de tipo `ReceiveTask` del modelo origen se genera un objeto de tipo `MMessageEvent` en el modelo destino.

```
rule ReceiveTask2Eventreceive{
        from
                catch: MMbpmn2!ReceiveTask
        to
                events: MMjadexbdi!MMessageEvent (
                        name <- catch.name,
                        type <- 'fipa',
                        direction <- #receive,
                        parameter <- datos1
                ),
                datos1: MMjadexbdi!MParameter(
                        name <- 'performative',
                        class <- 'String',
                        direction <- #fixed,
                        value <- valor1
                ),
                valor1: MMjadexbdi!MValue(
                        name <- 'SFipa.' + catch.properties.first().name
                )
}
```

Figura 5.23. *Especificación de la regla de transformación* receiveTask2Eventreceive

Los datos de identificación del evento de mensaje son creados mediante los atributos: `name`, inicializado con el valor del atributo `name` de la variable `catch; type`, inicializado con el valor `fipa,` que determina el lenguaje de comunicación entre los agentes, y `direction`, inicializado con el valor `#receive`, lo cual determina que el evento de mensaje debe tener un estado de espera de un mensaje con un acto de comunicación de tipo FIPA. El tipo de dirección del evento es provisto por el elemento `DirectionType` de tipo `<<enumeration>>` del metamodelo destino.

Los parámetros de ejecución del evento de mensaje son especificados mediante la variable `datos1`, en donde el atributo `name` es inicializado con el *string* `performative`, lo cual determina que en un nivel inferior de la estructura del agente se debe encontrar el

valor de un acto de comunicación. Los atributos `class` y `direction` del objeto de tipo `MParameter` son inicializados con el valor `String` y `#fixed`, respectivamente. El valor del acto de comunicación utilizado en el evento de mensaje es inicializado en el atributo `name` mediante la concatenación del *string* `SFipa`. más la instancia contenida en el atributo `catch.properties.first().name` del modelo origen, formándose la estructura válida del acto de comunicación y que puede ser interpretada por la plataforma.

### 5.2.2.6. Regla 6: Role2Configurations

Esta regla establece que por cada rol que una organización participante desempeñe en el proceso de integración se genera un rol para el agente que ejecutará el proceso, así como la meta asignada al rol a ejecutar.

La regla `Role2Configurations` se muestra en la figura 5.24. La misma permite generar un objeto `Configuration` en el modelo destino por cada objeto `PartnerRole` del modelo origen que tenga una relación con un objeto `Participant` y que el participante tenga una referencia al objeto `Process` mediante la relación `processRef` definida en el modelo de entrada. El atributo `name` del objeto `Configuration` es inicializado con el rol encontrado en el atributo `name` del objeto `PartnerRole`.

```
rule Role2Configurations {
        from
                role: MMbpmn2!PartnerRole (
                        not role.participantRef.first().processRef.oclIsUndefined()
                )
        to
                configuration: MMjadexbdi!Configurations(
                        configuration <- config
                ),
                config: MMjadexbdi!Configuration(
                        name <- role.name,
                        goals <- initial
                ),
                initial: MMjadexbdi!MConfigGoalbase(
                        initialgoal <- ref1
                ),
                ref1: MMjadexbdi!MConfigGoal(
                        ref <- thisModule.getInitialGoal()
                )
}
```

Figura 5.24. *Especificación de la regla de transformación* role2Configurations.

La referencia a la meta inicial del rol del agente se define mediante el atributo `ref` del objeto `MConfigGoal`, el cual es inicializado por el valor asignado por el *helper* `thisModule.getInitialGoal`, que habilita la recuperación de la meta principal

relacionada con el rol. Esta referencia a la meta inicial se obtiene mediante la relación `initialgoal` entre los objetos `MConfigGoal` y `MConfigGoalbase`. Para alcanzar el vínculo existente entre el rol y la meta se debe utilizar la relación `goals` entre el objeto `MConfigGoalbase` y el objeto `Configuration` del modelo destino.

## 5.3. Método de transformación para generar el código del agente BDI

Este método de transformación modelo-a-código tiene por objetivo generar el código de un agente Jadex-BDI que implementa un agente *AdministradorDeProceso*, para que pueda ser instanciado en la plataforma de agentes Jadex. El método tiene declarado como modelo de entrada el modelo del agente Jadex-BDI que representa a un agente *AdministradorDeProceso* generado a través del método de transformación detallado en la sección 2.

La definición de un agente Jadex-BDI requiere de dos componentes. Por un lado, un archivo de definición de agente (*Agent Definition File*, ADF), para la especificación de las creencias, metas y planes, así como también los valores de la configuración inicial del agente. En la especificación del archivo ADF es usado el lenguaje XML, siguiendo el metamodelo Jadex-BDI. Por otro lado, la definición del cuerpo de un plan puede llevarse a cabo mediante codificación en lenguaje Java o en modelos Jadex-Processes.

Por lo tanto, la salida de este método de transformación consiste de un documento XML, el cual contiene la estructura del archivo ADF que representa el código de un agente BDI de Jadex. Dicho documento contiene las secciones `agent`, `goals`, `plans`, `events` y `configurations` derivadas del modelo del agente. Mediante estas especificaciones el agente puede ser inicializado en la plataforma Jadex.

En el método de transformación se han predefinido las secciones `beliefs` y `capabilities` que permiten agregar en el agente el uso de conocimiento, capacidades y recursos provistos por Jadex. En la sección `goals` se ha predefinido una meta que permite crear una instancia del agente *AdministradorDeProceso* y una instancia del componente Jadex-Processes embebido en el agente (motor de procesos) en la plataforma cuando se inicializa al agente, así como también otra meta que habilita destruir una instancia del agente y sus componentes al finalizar la ejecución del proceso.

El proceso de generación del código consiste en dos etapas. En la primera etapa se realiza una transformación desde el modelo del agente a un documento definido en XMI, utilizando el metamodelo Jadex-BDI como origen y un metamodelo XML como destino. En la segunda etapa, a partir del documento XMI generado en la etapa anterior, se genera el documento XML, esto es, el modelo XML es transformado en una representación textual XML. Para ello se utiliza un ATL *query* que permite la generación de una salida de texto a partir de un conjunto de modelos de entrada.

### 5.3.1. Metamodelo XML

Se definió un metamodelo XML, que describe los elementos que componen los modelos XML, así como los relacionados que se pueden definir entre los elementos del metamodelo, el cual se muestra en la figura 5.25.

El metamodelo XML tiene un único elemento `Root`, el cual puede contener entidades `Element`, `Attribute` y `Text`. Los elementos `Attribute`, `Text` y `Element` son directamente heredados del elemento abstracto `Node`, mientras que el elemento `Root` es heredado de la entidad `Element`. La entidad `Node` tiene definidos los atributos `name` y `value`. La utilización de estos atributos en las diferentes entidades del metamodelo XML se define de la siguiente forma:

- En la entidad `Element`, "`name`" codifica el nombre de las etiquetas XML del modelo o documento.
- En la entidad `Attribute`, "`name`" codifica el nombre del atributo, mientras que en "`value`" contiene el valor asociado al atributo.
- En la entidad `Text`, "`value`" almacena el contenido textual de la entidad `Text`.

Una entidad `Element` puede contener varios nodos de tipo `Attribute`, `Text` o `Element`. Inversamente, un nodo puede ser contenido por cero o un elemento. De hecho, cada nodo está contenido en un elemento, excepto el elemento `Root` que no tiene nodo padre.

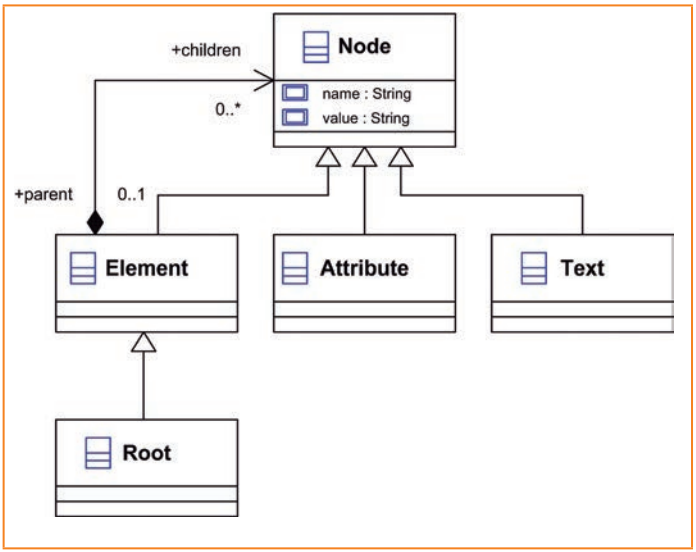



Figura 5.25. *Metamodelo del lenguaje XML.*

## 5.3.2. Reglas de transformación para generar el código del agente BDI

### 5.3.2.1. Regla 1: Root

La regla `Root` establece que por cada componente de tipo agente encontrado en el modelo del agente BDI (modelo origen) se genera un elemento raíz en el código del agente (documento destino). El documento ADF (basado en XML) generado debe contener el nombre del agente y el paquete de implementación.

La regla `Root` genera un elemento raíz en el documento destino, así como un conjunto de atributos, elementos y nodos de texto desde el elemento de entrada `Componenttype` (figura 5.26). El elemento `Root` generado es etiquetado como `agent`, que contiene los atributos XML `package` dentro de la variable `agentPackage` y `name` dentro de la variable `agentName`. El valor del atributo `value` en la variable `agentPackage` es copiado desde el atributo `package` del objeto de entrada `Componenttype`. En la variable `agentName` el valor del atributo `value` del objeto `Attribute` del modelo destino es copiado del objeto de entrada `Componenttype`.

```
rule Root {
        from
                agent : MMjadexbdi!Componenttype
        to
                agentXML : MMxml!Root (
                        name <- 'agent',
                        children <- Sequence {agentPackage, agentName, imports}
                ),
                agentPackage : MMxml!Attribute (
                        name <- 'package',
                        parent <- agentXML,
                        value <- agent.package
                ),
                agentName : MMxml!Attribute (
                        name <- 'name',
                        parent <- agentXML,
                        value <- agent.name
                )
}
```

Figura 5.26. *Especificación de la regla de transformación Root.*

### 5.3.2.2. Regla 2: goal2goalXML

Esta regla establece que por cada definición de meta encontrada en el modelo origen se debe crear un elemento de meta en el documento destino. Cuando se encuentra la

primera meta se debe crear una sección *goals* en el modelo destino y todas las metas encontradas deben quedar contenidas dentro de esta sección.

La regla `goal2goalXML` se muestra en la figura 5.27. En esta regla, cuando se encuentra el primer objeto `MGoalbase` en el modelo origen, se genera un objeto `Element` en el modelo destino asignándole el nombre de `goals`, el cual tiene como padre a la sección `agent` (`root` del documento), a través del atributo `parent` que es inicializado con el valor asignado por el *helper* `thisModule.getAgente()`.

La sección `goals` generada tiene como hijos a los elementos `achievegoalref` y `performgoal`. Mediante la variable `achieveGoalsCreate` se crea una meta predefinida en el documento destino, permitiendo instanciar un agente en la plataforma utilizando una capacidad (`capabability`) que la arquitectura Jadex-BDI provee para la creación de componentes en tiempo de ejecución del sistema. El atributo `name` del objeto `Element` del modelo destino en la variable `achieveGoalsCreate` es inicializado con el valor `achievegoalref`, que representa el nombre de la etiqueta de la entidad `Element` generada. Este elemento tiene como hijos a los atributos `name` y `concrete` creados mediante el atributo `children`.

Mediante la variable `achieveGoalName` el atributo `name` y `value` son inicializados como una entidad de tipo `Attribute` del elemento `achievegoalref`, en donde el atributo `value` es inicializado con el valor `cms_create_component`. La variable `concrete` se inicializa como una entidad del objeto `Element` generando una etiqueta con el nombre de `concrete`, la cual tiene un hijo inicializado con el nombre `ref`. A través de la variable `concreteref` se inicializa el atributo `name` como una entidad del objeto `Attribute` con el valor de `ref` y el atributo `value` es inicializado con el valor `cmscap.cms_create_component`, lo cual permite en tiempo de ejecución de la plataforma instanciar un agente mediante el código generado en la transformación.

En la variable `achieveGoalsDestroy` se crea una meta predefinida en el código del agente que habilita al agente a destruir su instancia creada en la plataforma al finalizar la ejecución del proceso de negocio. El procedimiento que sigue la variable `achieveGoalsDestroy` es similar a la definición de la variable `achieveGoalsCreate`, únicamente cambian los parámetros de implementación en las variables `achieveGoalNameD` y `concreterefD` definidas como entidades del objeto `Attribute`. El atributo `value` en la variable `achieveGoalNameD` es inicializado con el valor `cms_destroy_component`, mientras que en la variable `concreterefD` el atributo es inicializado con el valor `cmscap.cms_destroy_component`.

El elemento `performgoal` creado como hijo de la sección `goals` en el documento destino es implementado mediante la variable `performGoal` del objeto `Element` del modelo `destino`. Mediante esta variable se crea el nombre de la etiqueta `performgoal` y se define la variable `performGoalName` como hijo del elemento `performgoal`. En la variable `performGoalName` se generan los atributos del elemento `performgoal`. En el atributo `value` del objeto `Attribute` del modelo `destino` se copia

el valor del atributo `goals.performgoal.at(1).toString()` del modelo origen que corresponde a la meta inicial que el agente debe alcanzar y la cual se debe satisfacer mediante la ejecución de sus planes y eventos.

```
rule goal2goalXML {
    from
        goals : MMjadexbdi!MGoalbase
    to
        goalsXML : MMxml!Element(
            name <- 'goals',
            parent <- thisModule.resolveTemp(thisModule.getAgente(), 'agentXML'),
            children <- Sequence {achieveGoalsCreate, achieveGoalsDestroy, performGoal}
        ),
        achieveGoalsCreate : MMxml!Element (
            name <- 'achievegoalref',
            parent <- goalsXML,
            children <- Sequence {achieveGoalName, concrete}
        ),
        achieveGoalName : MMxml!Attribute (
            name <- 'name',
            value <- 'cms_create_component',
            parent <- achieveGoalsCreate
        ),
        concrete : MMxml!Element (
            name <- 'concrete',
            parent <- achieveGoalsCreate,
            children <- concreteRef
        ),
        concreteRef : MMxml!Attribute (
            name <- 'ref',
            value <- 'cmscap.cms_create_component',
            parent <- concrete
        ),
        achieveGoalsDestroy : MMxml!Element (
            name <- 'achievegoalref',
            parent <- goalsXML,
            children <- Sequence {achieveGoalNameD, concreteD}
        ),
        achieveGoalNameD : MMxml!Attribute (
            name <- 'name',
            value <- 'cms_destroy_component',
            parent <- achieveGoalsDestroy
        ),
        concreteD : MMxml!Element (
            name <- 'concrete',
            parent <- achieveGoalsDestroy,
            children <- concreteRefD
        ),
        concreteRefD : MMxml!Attribute (
            name <- 'ref',
            value <- 'cmscap.cms_destroy_component',
            parent <- concreteD
        ),
        performGoal : MMxml!Element (
            name <- 'performgoal',
            parent <- goalsXML,
            children <- performGoalName
        ),
        performGoalName : MMxml!Attribute (
            name <- 'name',
            value <- goals.performgoal.at(1).toString(),
            parent <- performGoal
        )
}
```

Figura 5.27. *Especificación de la regla de transformación* goal2goalXML.

### 5.3.2.3. Regla 3: plans2plansXML

Esta regla establece que por cada especificación del plan en el modelo origen se genera un elemento del plan en el modelo destino. Al localizar la primera especificación del plan en el modelo origen se crea una entidad denominada *plans* en el documento destino. Luego, todas las especificaciones encontradas son agregadas como un elemento hijo dentro de la estructura del documento destino. Cada plan es identificado con el nombre, el tipo de plan a implementar y el archivo que contiene la implementación del plan. Los eventos de mensaje que condicionan la ejecución del plan y las metas que estén relacionadas y que se puedan satisfacer mediante la ejecución del plan son especificados.

La regla `plans2plansXML` se muestra en la figura 5.28. Por cada elemento `MPlan` localizado en el modelo origen se debe crear una entidad `Element` en el documento destino. Esta entidad se define mediante la variable `plansXML` que contiene el atributo `name`, que es inicializado con el valor de `plans`, generando una etiqueta con ese nombre. El elemento generado en el documento destino tiene definido como padre al elemento `agent` mediante el atributo `parent` y al elemento `plan` como hijo a través del atributo `children`.

En la variable `plan` que utiliza el objeto `Element` del modelo destino se define el nombre de la etiqueta del elemento y se especifica la secuencia de los datos de implementación y ejecución del plan mediante el atributo `children`. En la variable `planName` se copia el nombre del `plan`, definiéndolo mediante el atributo `value` del objeto `Attribute` del modelo destino, el cual es inicializado con el valor contenido en la variable `plans.name.toString()` del modelo origen. Mediante la variable `plansbody` se define el cuerpo de implementación del plan, generando un elemento con el nombre de `body`, el cual está compuesto por el nombre del modelo Jadex-Processes que representa el modelo de proceso ejecutable. El elemento `body` contiene el tipo de plan y el plan a implementar, lo cual se realiza a través de las variables `bodyImpl` y `bodyType`, configuradas como atributos en la estructura del documento mediante el objeto `Attribute` del modelo destino.

En la variable `bodyImpl` se define la etiqueta `impl` y se asigna el nombre del archivo de proceso ejecutable a implementar (generado a través del método de transformación detallado en la sección 1), contenido en el atributo `plans.body.impl` del modelo origen al atributo `value` del modelo destino. En la variable `bodyType` se especifica el nombre de etiqueta `type`, y el valor del atributo `plans.body.type` del modelo origen se copia al atributo `value` del modelo destino, en este caso, el valor es BPMN que representa el tipo de plan a implementar.

En la variable `planTrigger` se genera la etiqueta `trigger` mediante el atributo `name` del objeto `Element` del modelo destino, y tiene como hijo al elemento `goal` que se especifica mediante la variable `planGoal`. En la variable `goalRef` definida como entidad `Attribute` del elemento `goal` se inicializa el atributo `name` con el valor `ref` y el valor del atributo `value` es copiado desde `plans.trigger.goalfinished.at(1).ref` del modelo origen que contiene la meta que el agente debe satisfacer mediante la ejecución del plan.

La ejecución del plan está condicionada a la interacción mediante eventos de mensaje entre los agentes que implementan el proceso colaborativo. Por ello, en la definición del plan se debe configurar una cola de espera (*waitqueue*) de los eventos de mensaje de los agentes de otra organización que se recibirán en el proceso de integración. La especificación de los eventos de mensaje que forman el `waitqueue` del plan a implementar se realiza mediante una expresión que itera la aplicación de una regla tipo *lazy* (`waitqueue2waitqueueXML`) que define la estructura del elemento `waitqueue` en el documento destino, repitiendo la aplicación por cada instancia del evento de mensaje de tipo `receive` existente en el modelo origen, tal como se muestra en la variable `plan`.

```
lazy rule waitqueue2waitqueueXML {
        from
                plans : MMjadexbdi!MPlan
        to
                waitQueue : MMxml!Element (
                        name <- 'waitqueue',
                        parent <- thisModule.resolveTemp(thisModule.getAgente(), 'plan'),
                        children <- messageEvent
                ),
                messageEvent : MMxml!Element (
                        name <- 'messageevent',
                        parent <- waitQueue,
                        children <- messageRef
                ),
                messageRef : MMxml!Attribute (
                        name <- 'ref',
                        parent <- messageEvent,
                        value <- plans.ref
                )
}
```

Figura 5.29. *Especificación de la regla de transformación de tipo* lazy waitqueue2waitqueueXML.

En la regla `waitqueue2waitqueueXML` mostrada en la figura 5.29 se genera un elemento `waitqueue` que tiene como padre al elemento `plan` y como hijo al elemento `messageevent`, el cual tiene definida la variable `messageRef` como hijo. El atributo `name` del objeto `Attribute` del modelo destino es inicializado con el valor `ref` y el atributo `value` es inicializado con el valor contenido en el atributo `plans.ref` en el modelo origen.

### 5.3.2.4. Regla 4: events2eventsXML

Esta regla establece que por cada evento de mensaje en el modelo origen se genera un evento de mensaje en el modelo destino. Cuando se encuentre la primera instancia de

un evento de mensaje en el modelo origen, se genera una sección *events* en el documento destino. La sección *events* incluirá todos los eventos de mensaje con sus atributos, parámetros y valores de ejecución.

```
rule events2eventsXML {
     from
          msg_events : MMjadexbdi!MMessageEvent
     to
          eventsXML : MMxml!Element (
                name <- 'events',
                parent <- thisModule.resolveTemp(thisModule.getAgente(), 'agentXML'),
                children <- Sequence {message_Event}
          ),
          message_Event : MMxml!Element (
                name <- 'messageevent',
                parent <- eventsXML,
                children <- Sequence {eventName, eventType, eventDirection, eventParameter}
          ),
          eventName : MMxml!Attribute (
                name <- 'name',
                parent <- message_Event,
                value <- msg_events.name
          ),
          eventType : MMxml!Attribute (
                name <- 'type',
                parent <- message_Event,
                value <- msg_events.type
          ),
          eventDirection : MMxml!Attribute (
                name <- 'direction',
                parent <- message_Event,
                value <- msg_events.direction.toString()
          ),
          eventParameter : MMxml!Element (
                name <- 'parameter',
                parent <- message_Event,
                children <- Sequence {paramName, paramClass, paramDirection, paramSpeech}
          ),
          paramName : MMxml!Attribute (
                name <- 'name',
                parent <- eventParameter,
                value <- msg_events.parameter.at(1).name
          ),
          paramClass : MMxml!Attribute (
                name <- 'class',
                parent <- eventParameter,
                value <- msg_events.parameter.at(1).class
          ),
          paramDirection : MMxml!Attribute (
                name <- 'direction',
                parent <- eventParameter,
                value <- msg_events.parameter.at(1).direction.toString()
          ),
          paramSpeech : MMxml!Element (
                name <- 'value',
                parent <- eventParameter,
                children <- paramValue
          ),
          paramValue : MMxml!Text (
                parent <- paramSpeech,
                value <- msg_events.parameter.at(1).value.at(1).toString()
          )
}
```

Figura 5.30. *Especificación de la regla de transformación* events2eventsXML.

La regla events2eventsXML se muestra en la figura 5.30. El patrón de entrada especificado en la regla permite localizar todos los objetos MMessageEvent en el modelo del agente BDI, generando un elemento events del objeto Element del modelo destino que incluye una estructura de todos los eventos de mensaje, así como una colección de atributos, elementos y nodos de texto que conforman la estructura del evento de mensaje.

El elemento messageevent tiene definido como padre al elemento events y como hijos la secuencia de atributos eventName, eventType, eventDirection y al elemento eventParameter.

En la variable eventName se genera el atributo name, asignándole el valor del atributo msg_events.name, que contiene el nombre del evento de mensaje en el modelo origen. En la variable eventType se genera el atributo type copiándole el valor del atributo msg_events.name, que representa el tipo de dato del evento de mensaje contenido en el modelo origen. El atributo direction es generado mediante la variable eventDirection y es inicializado con el valor proporcionado por el atributo msg_events.direction.toString() del patrón de entrada, el cual representa el sentido del evento de mensaje, que puede ser un evento send o receive. El elemento parameter es generado mediante la variable eventParameter y tiene definido al elemento paramSpeech, así como una secuencia de atributos paramName, paramClass, paramDirection como hijos y al elemento messageevent como padre. En la variable paramName se genera el atributo name, al cual se le asigna el valor del atributo msg_events.parameter.at(1).name, que es representado por el valor performative en el modelo origen. Luego, se generan los atributos class y direction copiando el valor contenido en el modelo origen.

Mediante la variable paramSpeech se genera el elemento value a través del atributo name del objeto Element del modelo destino. El elemento value tiene un nodo de texto como hijo, a través del cual se copia el acto de comunicación que utilizará el evento de mensaje. Esto se realiza en la variable paramValue, en donde el atributo value es inicializado con el valor contenido en el modelo origen en el atributo msg_events.parameter.at(1).value.at(1).toString().

### 5.3.2.5. Regla 5: configurations2configurationsXML

Esta regla establece que por cada elemento de configuraciones en el modelo origen se genera una sección de configuraciones en el documento destino. La estructura del agente contenida en el documento destino debe incluir el rol que ejecutará el agente y la meta inicial que tiene asignada el agente.

La regla configurations2configurationsXML se muestra en la figura 5.31. Por cada coincidencia del objeto Configurations encontrada en el modelo origen la regla genera una entidad Element etiquetada con el nombre de configurations

en el documento destino. En la variable `configElement` se genera el nombre del elemento `configuration` mediante el atributo `name` del objeto `Element` del modelo destino. Este elemento generado tiene como padre a la entidad `configurations` y como hijos al atributo contenido en la variable `configName` y al elemento especificado en la variable `configGoal`.

```
rule configurations2configurationsXML {
        from
                configurations : MMjadexbdi!Configurations
        to
                configurationsXML : MMxml!Element(
                        name <- 'configurations',
                        parent <- thisModule.resolveTemp(thisModule.getAgente(), 'agentXML'),
                        children <- Sequence {configElement}
                ),
                configElement : MMxml!Element (
                        name <- 'configuration',
                        parent <- configurationsXML,
                        children <- Sequence {configName, configGoal}
                ),
                configName : MMxml!Attribute (
                        name <- 'name',
                        parent <- configElement,
                        value <- configurations.configuration.at(1).name
                ),
                configGoal : MMxml!Element (
                        name <- 'goals',
                        parent <- configElement,
                        children <- initialGoal
                ),
                initialGoal : MMxml!Element (
                        name <- 'initialgoal',
                        parent <- configGoal,
                        children <- initialGoalRef
                ),
                initialGoalRef : MMxml!Attribute (
                        name <- 'ref',
                        parent <- initialGoal,
                        value <- configurations.configuration.at(1).goals.initialgoal.at(1).ref
                )
}
```

Figura 5.31. *Especificación de la regla de transformación* configurations2configurationsXML.

En la variable `configName` se genera un atributo con el nombre de `name`, al cual se le asigna el valor contenido en el atributo `configuration.at(1).name` del modelo origen mediante el atributo `value` del objeto `Attribute` del modelo destino. Este valor representa el rol que desempeñará el agente en la ejecución del proceso colaborativo.

Mediante la variable `configGoal` se genera el elemento `goals`, el cual tiene al elemento `initialgoal` como hijo, generado mediante la variable `initialGoal`.

Este elemento contiene la meta inicial que el agente debe satisfacer al ser inicializado en la plataforma. La meta inicial es especificada mediante la variable `initialGoalRef` que genera un atributo con el nombre `ref`, al cual se le asigna el valor de la instancia que coincida con el patrón de entrada `configurations.configuration.at(1).goals-.initialgoal.at(1).ref`, el cual crea el valor del atributo en el documento destino.

# Parte III
# Evaluación

# 6. Casos de estudio

En este capítulo se presentan dos casos de estudio utilizados para validar la plataforma tecnológica, los métodos y las herramientas propuestas en el presente libro. El primer caso de estudio describe la integración e implementación de servicios electrónicos de salud (e-healthcare) entre instituciones públicas de servicios de salud mediante una colaboración interorganizacional dinámica (sección 6.1). El segundo caso de estudio detalla una colaboración interorganizacional dinámica del dominio de la industria de las telecomunicaciones (sección 6.2). Finalmente, se analizan los resultados de la implementación de la plataforma A4IOC para estas colaboraciones interorganizacionales dinámicas (sección 6.3).

## 6.1. Colaboración interorganizacional dinámica en dominio de servicios electrónicos de salud

El caso de estudio consiste en un proyecto de integración y coordinación de servicios electrónicos de salud (e-Healthcare) entre el Hospital General y el Hospital de Especialidades ubicados en la zona Norte de México, en una ciudad con una población de aproximadamente 600 000 habitantes. El Hospital General inició sus operaciones hace 25 años. Cada año atiende consultas y realiza tratamientos médicos a aproximadamente 200 000 pacientes y brinda servicios de hospitalización a 22 000 pacientes. El Hospital de Especialidades es un hospital público de reciente creación con cobertura regional, de alta especialidad y equipado con tecnología de punta en el área médica.

Los pacientes hospitalizados en el Hospital General pueden ser derivados al Hospital de Especialidades cuando requieren servicios de atención médica por presentar enfermedades cardiacas y vasculares, neurológicas y/o cáncer. Ambos hospitales han formalizado un acuerdo para el desarrollo de servicios electrónicos integrados y coordinados de atención a la salud. Los objetivos son disminuir los tiempos de gestión de la admisión y transferencia de pacientes, y mejorar la calidad del servicio de referencias médicas (derivación de pacientes) entre los hospitales. Para ello han establecido implementar una colaboración interorganizacional y ejecutar procesos colaborativos que habiliten el intercambio electrónico de documentos estándares que contienen datos clínicos. Esta información posibilita que los especialistas tomen mejores decisiones en relación con la atención médica requerida por los pacientes.

Los datos de los procesos hospitalarios y la información requerida para la integración y coordinación de los servicios de salud fueron recolectados a través de entrevistas a los médicos, enfermeras, administradores y otros *stakeholders* involucrados en la colaboración interorganizacional de cada hospital. Los médicos juegan el rol más importante en el proceso de referencia.

Para lograr el objetivo de mejorar la calidad de atención al paciente, se definió un proceso colaborativo denominado "Gestión de Referencia de Paciente" utilizando el lenguaje UP-ColBPIP. El propósito de este proceso es la gestión de una referencia (también llamada derivación) de un paciente entre los hospitales. El Hospital General desempeña el rol de "Proveedor de Atención Primaria" (*Primary Care Provider*, PCP) y el Hospital de Especialidades desempeña el rol de "Proveedor de Atención Especializada" (*Specialist Care Provider*, SCP).

La referencia o derivación de un paciente por parte del Hospital General implica primero negociar una orden de transferencia de paciente con el Hospital de Especialidades y luego gestionar la correspondiente referencia médica. Por lo tanto, el proceso "Gestión de Referencia de Paciente" se realiza a través de dos subprocesos: "Orden de Transferencia de Paciente" (*Patient Transfer Order*, PTO) y "Gestión de Referencia Médica" (*Medical Referral Management*). El primero de estos procesos colaborativos fue definido para alcanzar la meta de disminuir el tiempo de gestión de admisión de un paciente en el Hospital de Especialidades al momento de realizar una derivación del paciente desde el Hospital General al Hospital de Especialidades. Mediante el protocolo de interacción correspondiente al proceso "Orden de Transferencia del Paciente" (figura 6.1) se define el comportamiento de la negociación entre el Hospital General (PCP) y el Hospital de Especialidades (SCP) para acordar la referencia o derivación de un paciente desde el PCP al SCP.

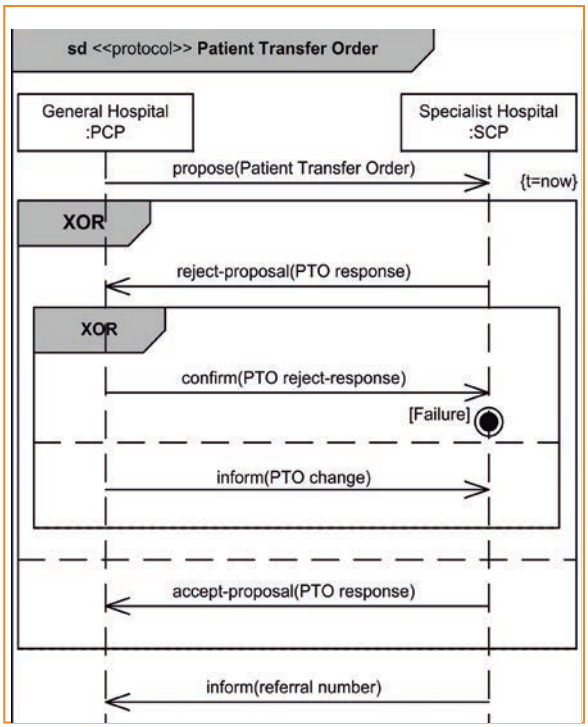



Figura 6.1. *Protocolo de interacción del proceso colaborativo* Orden de Transferencia de Paciente.

El protocolo de interacción correspondiente al segundo proceso colaborativo, "Gestión de Referencia Médica", mostrado en la figura 6.2, se definió con la meta de obtener información clínica confiable sobre el diagnóstico y tratamiento de los pacientes.

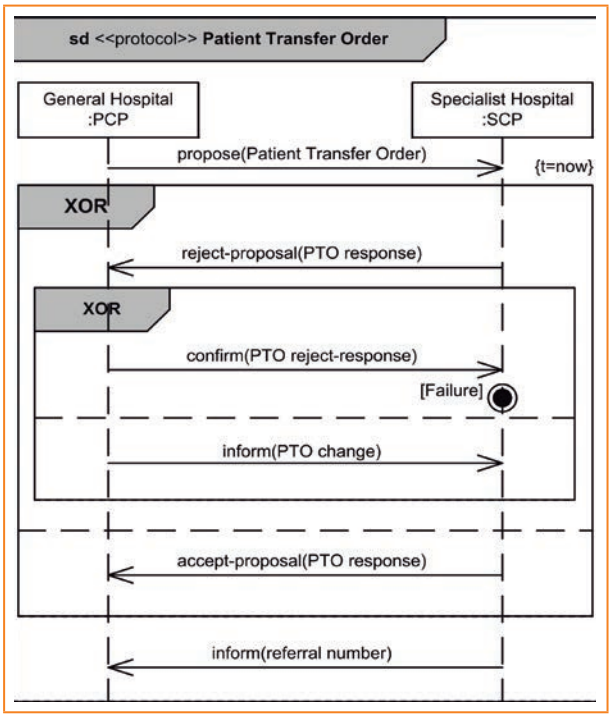



Figura 6.2. *Protocolo de interacción del proceso colaborativo* Gestión de Referencia Médica.

Las metas en común establecidas entre los hospitales involucrados en la colaboración, las cuales permiten evaluar los procesos colaborativos, pueden ser cuantificadas a través de indicadores, tales como el tiempo de gestión de referencias de los pacientes, la cantidad de órdenes de transferencia de pacientes que se completaron con éxito, la cantidad de órdenes de transferencia de pacientes que fueron rechazadas y la confiabilidad de la información clínica del paciente contenida en los documentos que se intercambian en los procesos.

El comportamiento de estos procesos se describe en la sección 6.1.2 en términos de los modelos de procesos de integración de las partes involucradas, los cuales luego son implementados por la plataforma.

### 6.1.1. Implementación de la colaboración interorganizacional dinámica

La implementación de la colaboración interorganizacional dinámica entre el Hospital General (PCP) y el Hospital de Especialidades (SCP) requiere que ambos instalen y desplieguen en servidores separados, conectados vía internet, la plataforma de agentes propuesta. Esto implica que cada hospital instala y despliega en su servidor un sistema multiagente (MAS) con los agentes estáticos definidos en la plataforma propuesta en el capítulo 4.

El comportamiento definido en los agentes de software estáticos de la plataforma posibilita a las organizaciones establecer una colaboración interorganizacional dinámica. En la figura 6.3 se muestran las interacciones ejecutadas entre los agentes de los MAS implementados en la plataforma que representan a cada uno de los hospitales, para establecer una colaboración interorganizacional. Estas interacciones resultan de ejecutar los caminos de los protocolos de interacción entre estos agentes, que permiten realizar el proceso de negociación entre las partes para establecer la colaboración, acordar los procesos colaborativos a ejecutar, generar en forma automática los modelos de procesos ejecutables que implementan los procesos de integración y el código de los agentes *AdministradorDeProceso* que ejecutan los procesos de integración, e inicializar e incorporar dichos agentes a los MAS de las partes para que ejecuten en forma descentralizada y coordinada los procesos colaborativos. En la figura 6.3 los agentes con el sufijo `A4IOC.etello_toshi` representan los agentes instanciados en el MAS del servidor del Hospital General. Los agentes con el sufijo `A4IOC.etello_laptop` son aquellos instanciados en el MAS del servidor del Hospital de Especialidades.

Entre el agente *AdministradorDeColaboraciones* (AC) iniciador que representa al Hospital General o PCP y el agente AC receptor que representa al Hospital de Especialidades o SPC se lleva a cabo una negociación (interacciones 1 y 2).

El AC iniciador solicita al AC receptor iniciar una colaboración y le envía como parte del contenido del mensaje los nombres y una descripción de los procesos colaborativos que le solicita que ejecuten en forma conjunta. Estos procesos son: *Orden de Transferencia de Paciente* y *Gestión de Referencia Médica*. Luego el AC receptor le envía un mensaje *agree* al agente AC iniciador definiendo que acuerda realizar la implementación y ejecución de los procesos colaborativos propuestos.

Entonces, se desencadenan las interacciones entre los agentes del MAS del PCP para obtener los modelos de procesos colaborativos y enviarlos al MAS del SCP. El agente AC del PCP solicita al agente *AdministradorDeModelos* (AM) del PCP que recupere los modelos de los procesos colaborativos acordados (interacción 3). El agente AM responde aceptando la solicitud mediante un mensaje *agree* (interacción 4), inicia la recuperación de los modelos solicitados a través de los mecanismos de búsqueda y recuperación definidos en sus planes y luego envía al agente AC los modelos recuperados mediante un mensaje *inform* (interacción 5).

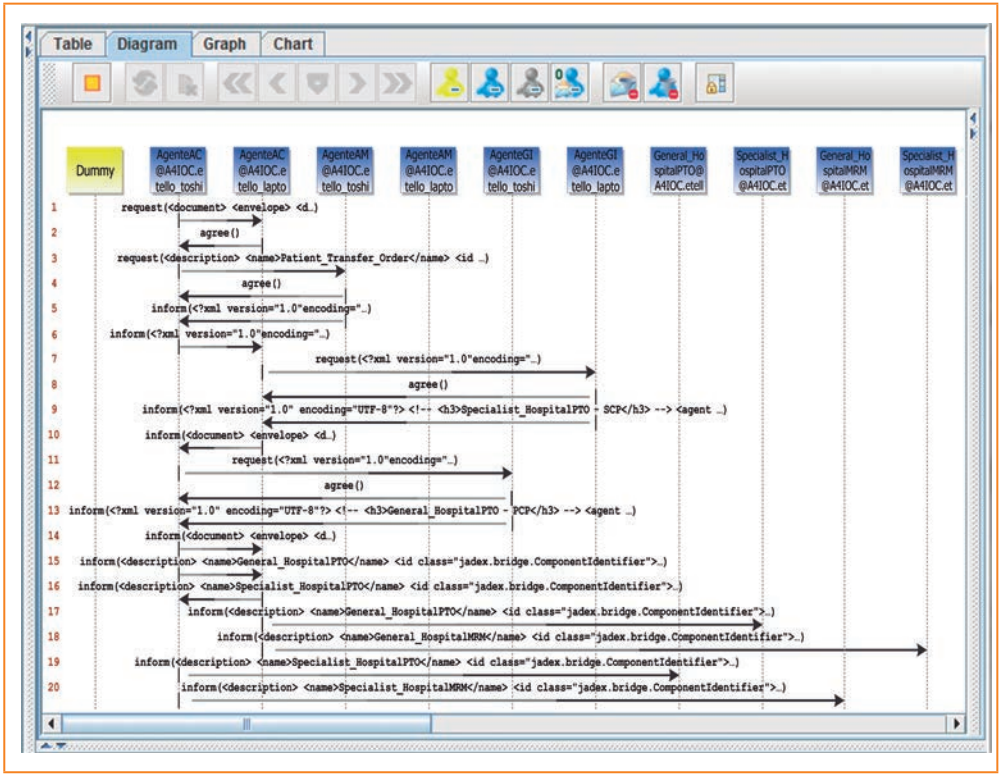



Figura 6.3. *Interacciones entre los agentes para establecer una colaboración interorganizacional dinámica.*

El agente AC del PCP reenvía a su contraparte, el agente AC del SCP, los modelos de procesos colaborativos recuperados desde su repositorio de modelos de procesos a través de un mensaje *inform*.

Del lado del MAS del SCP, su agente AC solicita al agente *GeneradorDeImplementaciones* (GI) del SCP la transformación de los modelos de procesos colaborativos recibidos a través de un mensaje con un acto de comunicación *request* (interacción 7). Aceptada la solicitud, el agente GI inicia la ejecución de su plan, a través del cual ejecuta los métodos de transformaciones de modelos (presentados en el capítulo 5) que generan automáticamente los artefactos de implementación para los procesos colaborativos acordados. Estos artefactos son: los modelos de procesos ejecutables y el código de los agentes *AdministradorDeProceso*. Para ello, el agente GI ejecuta en forma iterativa su plan, por cada modelo de proceso colaborativo que recibió en la solicitud. Dicho plan ejecuta los métodos de transformación de modelos que generan automáticamente: el modelo conceptual del proceso de integración, el cual representa el comportamiento del rol SCP del Hospital de Especialidades en el modelo de proceso colaborativo siendo transformado, el modelo de proceso ejecutable que implementa el proceso de integración

del SCP, y el código (implementación) del agente *AdministradorDeProceso* que ejecutará dicho modelo de proceso ejecutable para realizar el rol SCP del proceso colaborativo. Al finalizar estas transformaciones, el agente GI envía al agente AC un documento con el nombre de los agentes generados (esto se refiere a sus implementaciones) por cada modelo de proceso colaborativo procesado, mediante un mensaje *inform* (interacción 9).

Luego, el agente AC del SCP notifica a su contraparte, el agente AC del PCP, que el proceso de generación de modelos de procesos y código de los agentes *AdministradorDeProceso* finalizó satisfactoriamente y, por lo tanto, tiene habilitada la ejecución de los dos procesos colaborativos acordados.

Por su parte, del lado del MAS del PCP, su agente AC iniciador envía un mensaje a su agente GI con un acto de comunicación *request* conteniendo los modelos de proceso colaborativos a transformar (interacción 11). El agente GI responde con un mensaje *agree*, aceptando la solicitud (interacción 12), y activa el plan que permite ejecutar las transformaciones de modelos. Al finalizar el proceso de transformación, envía al agente AC un mensaje *inform* que contiene un documento con el nombre de los dos agentes *AdministradorDeProceso* generados para los modelos de procesos colaborativos acordados (interacción 13).

El agente AC iniciador procesa el mensaje *inform* notificando a su contraparte, el agente AC receptor del SCP, que la generación de modelos de proceso ejecutable y el código de los agentes *AdministradorDeProceso* terminó correctamente.

El agente AC del PCP inicializa y crea una instancia del agente *AdministradorDeProceso* con el nombre `General_HospitalPTO` y otra instancia del agente *AdministradorDeProceso* con el nombre `General_HospitalMRM`, ambos con el rol PCP en el MAS del Hospital General, correspondientes a los dos procesos colaborativos acordados. Luego consulta y recupera los datos de identificación de cada agente en la plataforma. Estos datos son capturados y guardados en un documento XML, el cual es enviado a su contraparte, el agente AC receptor del SCP, a través de un mensaje *inform* (interacción 15).

Al recibir el mensaje el agente AC *receptor* activa un comportamiento que permite instanciar a los agentes *AdministradorDeProceso* `Specialist_HospitalPTO` y `Specialist_HospitalMRM` con el rol SCP e incorporarlos al MAS del Hospital de Especialidades. Luego recupera los datos de identificación de los dos agentes en la plataforma, generando un documento XML que contiene estos datos y lo envía al agente AC iniciador del PCP mediante un mensaje *inform* (interacción 16).

Por otra parte, el agente AC del SCP recupera del documento recibido de su contraparte (agente AC *iniciador*) los datos de identificación del primer agente y mediante un mensaje *inform* envía un documento al agente *AdministradorDeProceso* `Specialist_HospitalPTO`, el cual contiene los datos de identificación del agente *AdministradorDeProceso* `General_HospitalPTO`, con quien se establecerá comunicación e interactuará para ejecutar el proceso colaborativo (interacción 17). Luego, realiza un proceso similar recuperando los datos de identificación del segundo agente, enviando un men-

saje *inform* con los datos de identificación del agente *AdministradorDeProceso* `General_HospitalMRM` al agente *AdministradorDeProceso* `Specialist_HospitalMRM`, habilitando la comunicación entre estos agentes (interacción 18).

El agente AC del PCP realiza un comportamiento similar, enviando al agente *AdministradorDeProceso* `General_HospitalPTO` los datos de identificación del agente *AdministradorDeProceso* `Specialist_HospitalPTO` (interacción 19) y enviando un mensaje *inform* con los datos de identificación del agente *AdministradorDeProceso* `Specialist_HospitalMRM` al agente *AdministradorDeProceso* `General_HospitalMRM` (interacción 20).

Finalmente, los agentes *AdministradorDeProceso* `General_HospitalPTO`, `Specialist_HospitalPTO`, `General_HospitalMRM` y `Specialist_HospitalMRM` reciben los mensajes *inform*, lo cual habilita la ejecución de su comportamiento para que puedan iniciar sus mecanismos de interacción y dar soporte a la ejecución de sus correspondientes procesos colaborativos a través de la ejecución de sus modelos de procesos ejecutables.

### 6.1.2. Generación de los modelos de procesos de integración

En esta sección se describen los modelos conceptuales de procesos de integración que se derivan a partir de los modelos de procesos colaborativos (protocolos de interacción) acordados por las partes en la colaboración (sección 6.1.1). Se describen, por cada proceso colaborativo, los modelos de procesos de integración que explican el comportamiento del proceso colaborativo, en términos de los envíos y recepciones de mensajes entre las partes a través de estos procesos de integración. De manera que se pueden visualizar no sólo las actividades e interacciones públicas de la colaboración entre las partes, sino también las actividades privadas de las partes, ambas implementadas en la solución tecnológica generada.

Para cada hospital involucrado en un protocolo de interacción de un proceso colaborativo se generó un modelo de proceso de integración que soporta el rol que el mismo desempeña en el proceso colaborativo. Estos modelos de procesos de integración se generaron por los agentes GI de las partes, como respuesta a algunas de las interacciones descritas en la sección previa, a través de la ejecución del método de transformación que implementan (48) (como se mencionó en el capítulo 3).

En primer lugar se describen los modelos de procesos de integración generados a partir del modelo de proceso colaborativo "Orden de Transferencia de Paciente" y luego aquellos generados a partir del modelo de proceso colaborativo "Gestión de Referencia Médica".

### 6.1.2.1. Modelos de procesos de integración de Orden de Transferencia de Paciente

Los modelos conceptuales de los procesos de integración que describen los roles del PCP y el SCP para el proceso colaborativo Orden de Transferencia de Paciente se muestran en la figura 6.4, junto con las interacciones entre los mismos desde el punto de vista de mensajes intercambiados. Este proceso colaborativo se inicia cuando el PCP desea derivar un paciente al SCP. Esto es capturado por el evento de inicio del proceso de Hospital General (figura 6.4.A), el cual representa la recepción de un mensaje *Transferir Paciente* (*Patient Transfer*), en cuyo contenido están los datos del paciente. Con estos datos se ejecuta una tarea de servicio que genera el documento *Orden de Transferencia de Paciente* (*Patient Transfer Order*) para el paciente a derivar, mediante una invocación al sistema interno del Hospital General. Luego, el PCP envía un mensaje (tarea de envío de mensaje *Propose Patient Transfer Order*) proponiendo derivar un paciente al SCP, de acuerdo con el acto de comunicación *propose* definido en el protocolo (figura 6.4.A). Este mensaje contiene el documento *Patient Transfer Order* con los motivos de la referencia médica, el diagnóstico, la gravedad o severidad de la enfermedad y el tratamiento actual del paciente a transferir.

Cuando el rol SCP recibe el mensaje con la propuesta de transferencia del paciente, evalúa la orden recibida (PTO) en la tarea privada *Evaluate Patient Transfer Order*, definida como tarea de usuario (figura 6.4.B). Dependiendo del resultado de la evaluación en esta tarea, responde con un mensaje *accept-proposal* o con un mensaje *reject-proposal*, como lo indican las correspondientes tareas de envío de mensaje definidas. Este comportamiento de caminos alternativos mutuamente excluyentes es indicado mediante una compuerta exclusiva basada en datos que define los caminos alternativos. Según el resultado de evaluación, el proceso continúa por uno de los dos caminos. Si el SCP envía un mensaje *accept-proposal*, genera un documento *Número de Referencia* (*Referral Number*) a través de una tarea de tipo servicio que invoca a un sistema interno.

Luego realiza la tarea que envía un mensaje con un acto de comunicación *inform* conteniendo el documento con el número de referencia (tarea *inform Referral Number*) y su proceso finaliza. El número de referencia representa un folio de autorización y aceptación de la solicitud del servicio de atención médica por parte del SCP. Si el SCP envía un mensaje *reject-proposal*, este mensaje contiene un documento con los motivos del rechazo de la propuesta de transferencia de paciente.

Del lado del proceso del PCP, la recepción de los mensajes *accept-proposal* o *reject-proposal* está representada por la compuerta basada en eventos con los dos caminos alternativos que representan la posible recepción de estos mensajes (figura 6.4.A).

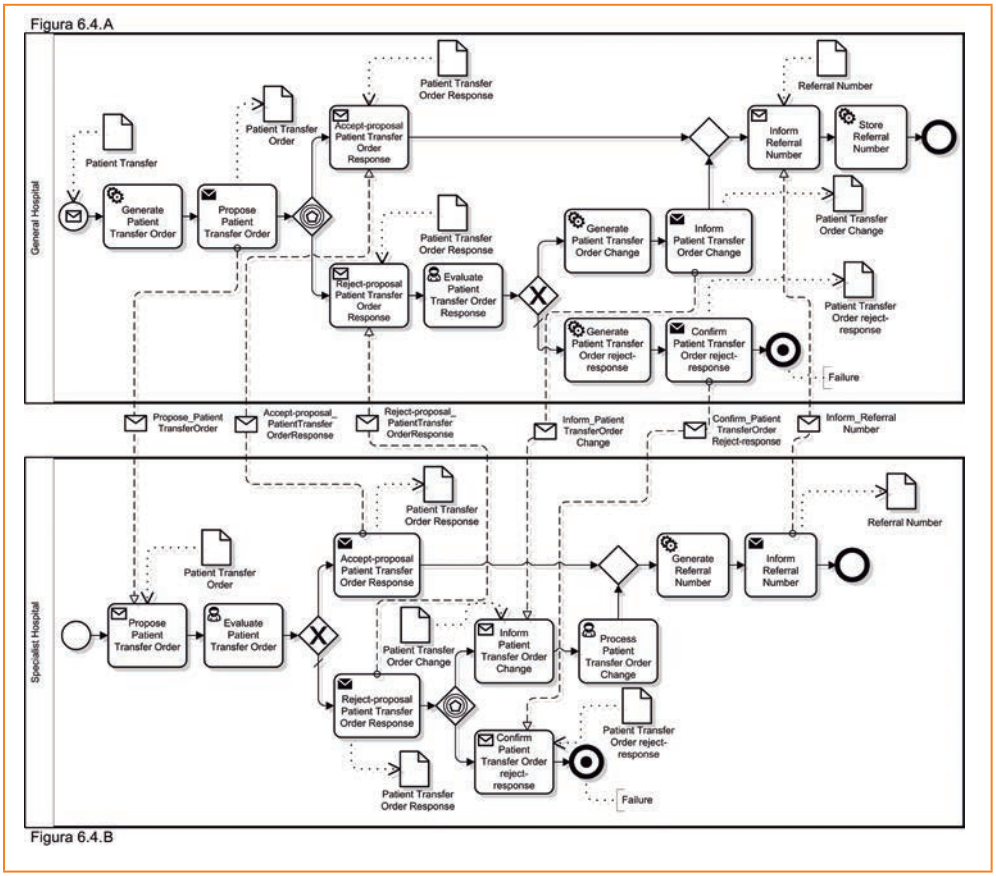



Figura 6.4. *Proceso de integración "Orden de Transferencia de Paciente":* A) *Rol PCP,* B) *Rol SCP.*

Cuando el PCP recibe un mensaje *reject-proposal*, evalúa el documento contenido en el mensaje a través de la tarea de usuario Evaluate Patient Transfer Order Response. En esta tarea, el médico supervisor del área del Hospital General que gestiona la transferencia del paciente evalúa la respuesta de rechazo enviada por el SCP con respecto a la orden de transferencia propuesta, y puede decidir cambiar la orden y establecer que es de una prioridad alta la atención del paciente, o bien acepta las razones del rechazo enviadas por el SCP. Los caminos alternativos que pueden ser realizados de acuerdo con el resultado de evaluación se representan por el elemento de compuerta exclusiva basada en datos con dos caminos alternativos. En caso de que se rechace, la tarea de servicio Generate Patient Transfer Order reject-response genera el documento a enviar al SCP a través de un mensaje confirm (figura 6.4.A).

El mensaje *confirm* indica que los motivos del rechazo son aceptados por el PCP, y las partes deciden en forma conjunta no realizar la derivación o transferencia del paciente.

En este caso, ambos procesos de integración de las partes finalizan, como lo indican los eventos de fin *terminate*, lo cual representa la finalización del proceso colaborativo.

En caso de que el PCP modifique la prioridad de la orden, la tarea de servicio *Generate Patient Transfer Order Change* genera un documento con dichos cambios a la orden inicial (figura 6.4.A). Este documento se envía a través de un mensaje *inform*, el cual indica un cambio en la prioridad de la atención médica del paciente con respecto al documento PTO inicial enviado con la propuesta inicial. En las políticas y reglas de operación del acuerdo de integración de servicios entre hospitales se ha definido una serie de prioridades determinadas por la gravedad del estado del paciente. Las políticas públicas gubernamentales exponen que no se puede negar la atención médica en un hospital público a una persona que presente una emergencia médica, avalada por otro hospital público, por un médico general del mismo hospital o por un médico privado.

Del lado del proceso del SCP (figura 6.4.B) la recepción de los mensajes anteriores *inform* o *confirm* se representa mediante una compuerta exclusiva basada en eventos, posterior a la tarea de tipo enviar *Reject-proposal Patient Transfer Order Response*, y asignando en uno de los conectores de salida de la compuerta exclusiva una tarea que representa la recepción de un mensaje con un acto de comunicación *confirm*, y en el otro conector de salida, una tarea que representa la recepción de un mensaje *inform*. Cuando el SCP recibe un mensaje *inform* conteniendo un documento con una prioridad alta, debe procesar y aceptar en forma automática la propuesta. Para ello genera el documento conteniendo el número de referencia, el cual es enviado con un mensaje *inform* mediante la tarea de envío de mensaje *inform Referral Number*.

Cuando el PCP recibe el mensaje *inform* que contiene el documento con el número de referencia generado por el SCP, almacena el contenido del documento mediante la tarea de tipo de servicio *Store Referral Number* (figura 6.4.A), que invoca a un sistema interno para llevar a cabo la actualización del expediente clínico del paciente.

### 6.1.2.2. Modelos de procesos de integración de Gestión de Referencia Médica

Al finalizar satisfactoriamente la ejecución del proceso colaborativo "Orden de Transferencia de Paciente" se ejecuta el protocolo de interacción del proceso colaborativo "Gestión de Referencia Médica". En este protocolo se define el intercambio de los documentos clínicos requeridos para especificar el tratamiento médico del paciente entre los hospitales. El modelo conceptual del proceso de integración correspondiente al PCP y el modelo conceptual del proceso de integración correspondiente al SCP se muestran en la figura 6.5.

El proceso colaborativo se inicia cuando el PCP internamente recibe un evento de mensaje *AcceptedPacientTransfer*. Esto es capturado por el evento de inicio del proceso, el cual representa la recepción de este mensaje, en cuyo contenido están los datos de la orden de transferencia de paciente generada en el proceso anterior. Con estos datos,

se ejecuta una tarea de servicio que genera el documento clínico *Carta de Referencia de Paciente* (*Patient Referral Letter*) mediante una tarea de tipo de servicio (figura 6.5.A). En dicho documento se detallan los datos acerca del tratamiento médico requerido por el paciente. Este documento es enviado al SCP mediante la tarea de envío de mensaje que representa el envío de un mensaje con un acto de comunicación *inform*.

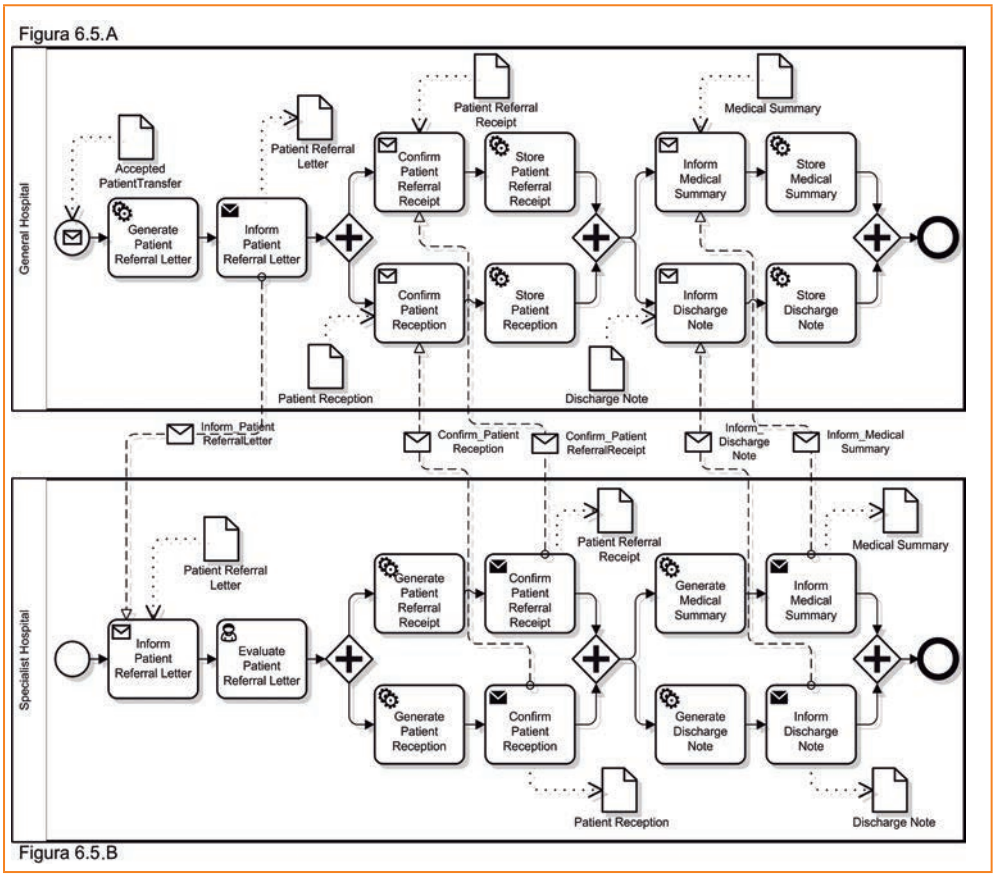



Figura 6.5. *Proceso de integración "Gestión de Referencia Médica":* A) *Rol PCP,* B) *Rol SCP.*

Del lado del proceso del SCP, cuando éste recibe el mensaje, procesa el documento recibido mediante la tarea de usuario *Evaluate Patient Transfer Letter* (figura 6.5.B). Luego, por un lado genera un documento de negocio que contiene un certificado digital emitido por un sistema interno del SCP (tarea de servicio *Generate Patient Referral Receipt*) y envía un acuse de recibo con dicho documento, el cual indica la recepción del documento clínico recibido al inicio. Este documento es enviado al PCP mediante un mensaje con un acto de comunicación *confirm* a través de la tarea de envío *Confirm Patient Referral Receipt*.

Por otro lado, cuando se registra el ingreso (admisión) del paciente en el Hospital de Especialidades, se genera un segundo documento *Patient Reception* con los datos de identificación del paciente que recibirá el tratamiento mediante una tarea de tipo de servicio (figura 6.5.B). Este documento es también enviado al PCP junto con un mensaje *confirm* (tarea de envío *Confirm Patient Reception*). Esto representa el acuse de recibo del paciente por parte del SCP al PCP.

Los caminos que generan estos mensajes y documentos se ejecutan en paralelo, representando de esta manera el primer segmento de control de flujo AND definido en el protocolo de la figura 6.2, que aquí es expresado en ambos procesos de integración a través de compuertas paralelas (*Parallel Gateway* en BPMN) que representan la división y la sincronización de los caminos.

Del lado del PCP, cuando éste recibe los mensajes *confirm*, procede a almacenar la información contenida en los documentos recibidos a través de tareas de servicio que invocan a su sistema interno. La recepción de los mensajes y las tareas de servicio son realizadas también en paralelo, para representar el primer segmento de flujo de control AND del protocolo.

En este punto del proceso del SCP, cuando se enviaron las confirmaciones, comienza el tratamiento médico que es llevado a cabo por los especialistas del SCP. Estas actividades no tienen un tiempo definido y no están expresadas en el proceso; mientras tanto los procesos de las partes se mantienen en ejecución y en espera de ser actualizados. En el momento que finaliza el tratamiento del paciente, el SCP genera un documento clínico *Resumen Clínico/Médico* (*Medical Summary*) mediante el cual se informa al PCP el estado actual de salud del paciente, incluyendo información del diagnóstico, resultados de laboratorio y de estudios médicos realizados, tratamiento realizado, tratamiento futuro y recomendaciones (figura 6.5.B).

Además, se genera un segundo documento clínico identificado como *Alta Médica* (*Discharge Note*), el cual expresa que se autoriza el traslado del paciente y se transfiere la responsabilidad de la atención médica y tratamiento al PCP. Ambos documentos son enviados mediante un mensaje con un acto de comunicación *inform* en forma paralela a través de las tareas de envío de mensaje. Luego de esto, el proceso finaliza del lado del SCP. Posterior a la recepción de estos mensajes por parte del PCP, la ejecución del proceso colaborativo "Gestión de Referencia Médica" finaliza.

### 6.1.3. Generación de los modelos de procesos ejecutables

Los modelos de procesos ejecutables, derivados a partir de los modelos conceptuales de procesos de integración descritos en la sección anterior, y que representan la implementación de estos procesos, se generaron también por los agentes GI de las partes en el establecimiento de la colaboración interorganizacional dinámica (sección 6.1). Esto fue posible debido a que los agentes GI implementan el método de transformación definido en el capítulo 5, sección 5.1. De acuerdo con este método, un modelo de proceso ejecu-

table es derivado a partir de un modelo conceptual de proceso de integración, realizando una anotación semántica que define los parámetros de ejecución en los elementos del modelo, para habilitar su ejecución mediante el motor de procesos embebido en los agentes *AdministradorDeProceso*.

Para ejemplificar una transformación, a continuación se describe la generación del modelo de proceso ejecutable o modelo Jadex-Processes (modelo destino) de la organización General Hospital, derivado del modelo de proceso de integración de "Orden de Transferencia de Paciente" del PCP (modelo de origen). Se muestran los elementos del modelo generado en términos de su representación en un archivo XMI usado para persistir el modelo.

Los elementos de este modelo de proceso ejecutable son contenidos en un `pool`, identificado como `participant1` con el nombre `General Hospital` derivado del participante *General Hospital* del modelo conceptual del proceso de integración (figura 6.6). La actividad *Generate Patient Transfer Order* se derivó de la tarea de servicio con el mismo nombre del modelo origen (figura 6.6).

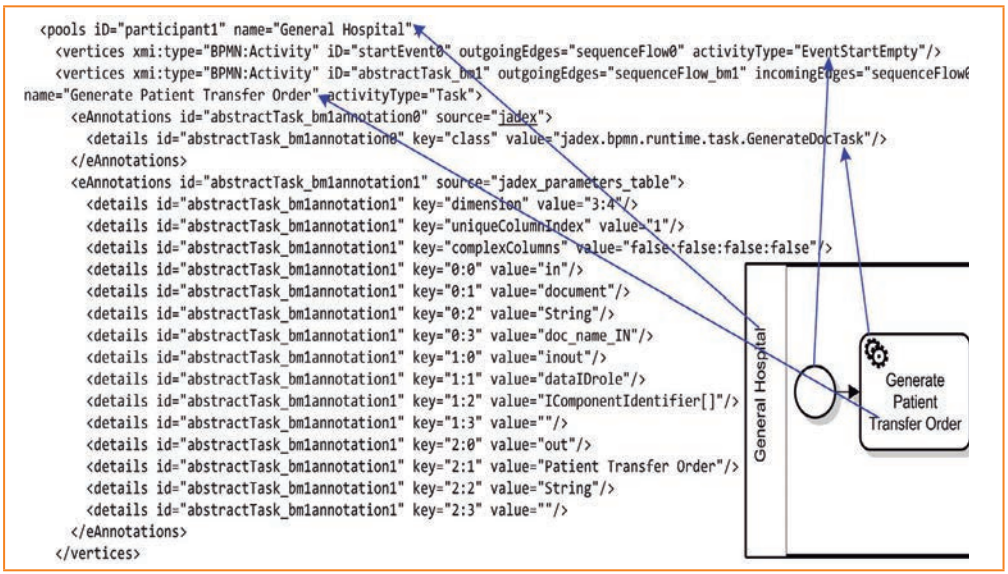



Figura 6.6. *Especificaciones generadas para los elementos* pool, *evento de inicio,* y *tarea* generate.

Las actividades en el modelo destino que representan las tareas de servicio del modelo origen, las cuales generan documentos, son anotadas con la clase Java predefinida `GenerateDocTask`. Los datos de entrada y salida de la actividad que se genera y por ende de la clase que la implementa se especifican mediante la tabla de parámetros `jadex_parameters_table`. El parámetro `document` define el documento de entrada a la tarea y el parámetro `dataIDrole` almacena el identificador del componente que ejecuta el modelo del proceso ejecutable recuperado mediante el método `IComponen-`

tIdentifier(). El identificador generado es integrado al documento, lo cual permite detectar en cualquier momento el rol que generó un documento. En la variable Patient Transfer Order se almacenará el documento generado por la tarea en tiempo de ejecución del sistema.

La tarea de envío de mensaje identificada con el nombre Send Propose Patient Transfer Order en el modelo origen se convierte en una actividad de tipo evento intermedio de mensaje en el modelo destino (figura 6.7). Este evento es anotado como isThrowing, lo cual representa un evento intermedio de envío de mensaje. El evento intermedio de mensaje también es anotado con parámetros que permiten su ejecución. Mediante el valor Patient Transfer Order del parámetro type se identifica el evento internamente en el sistema; con el parámetro receivers se asignan los receptores del mensaje, en este caso el rol receptor del evento es Specialist Hospital. El acto de comunicación Propose enviado por el evento intermedio de mensaje generado es configurado en el parámetro performative, y el documento Patient Transfer Order generado por la actividad anterior al evento intermedio de mensaje es configurado en el parámetro content, que representa el documento de entrada al evento y que será enviado cuando se produzca el mismo.

La tarea Receive Reject-Proposal Patient Transfer Order Response del modelo origen es transformada en una actividad de tipo evento intermedio de mensaje en el modelo destino (figura 6.7). El evento intermedio de mensaje generado es anotado mediante una tabla de parámetros que contienen las variables type y performative. En este caso, la variable type almacena el nombre del evento "Patient Transfer Order Response" y la variable performative contiene el acto de comunicación Reject-Proposal. De idéntica forma a esta actividad, se genera la actividad de evento intermedio de mensaje Accept-Proposal Patient Transfer Order Response.

Los conectores de flujo de secuencia anteriores a una tarea de usuario que realice una actividad de evaluación en el modelo conceptual del proceso de integración son generados con una anotación semántica utilizando una tabla de parámetros para su definición. Esto habilita en tiempo de ejecución de una instancia de un modelo de proceso ejecutable mover el documento de salida de una tarea y ubicar este documento en la entrada de la siguiente tarea (figura 6.8). De esta manera, el conector de secuencia sequenceFlow_bm5 previo a la tarea Evaluate Patient Transfer Order Response es configurado con la variable doc_name_IN, en la cual se almacenará el contenido del documento Patient Transfer Order Response recibido en el evento intermedio de mensaje anterior y recuperado mediante la instrucción $event.content. Este documento será la entrada en la tarea de evaluación siguiente al conector de secuencia.

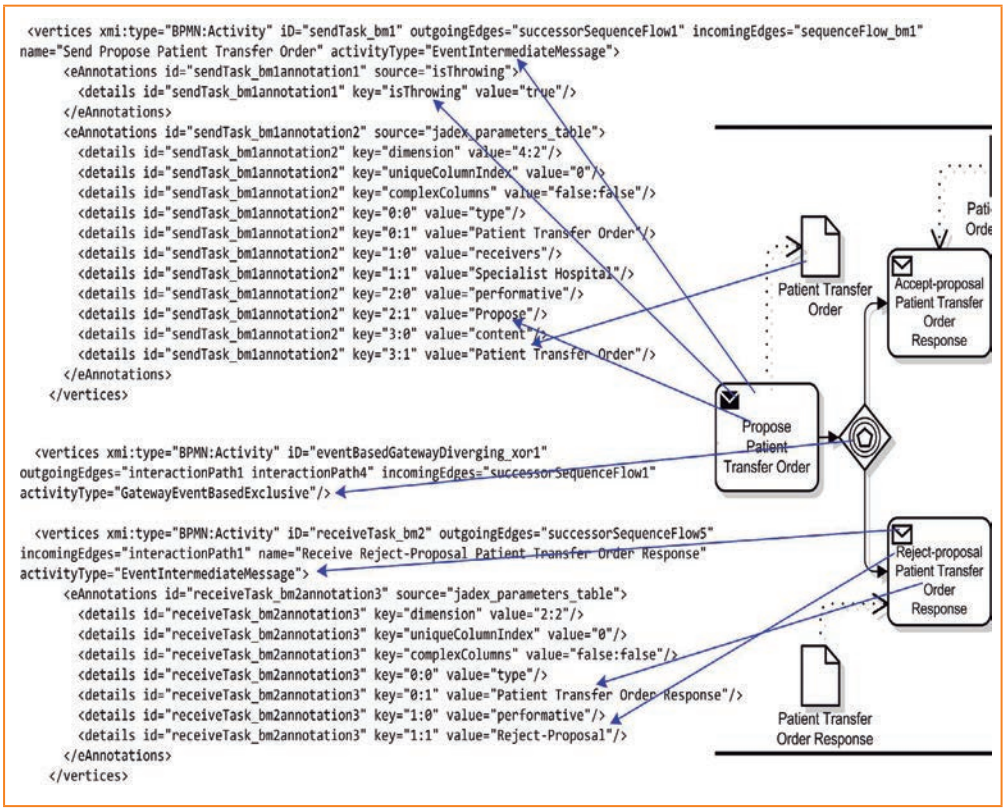
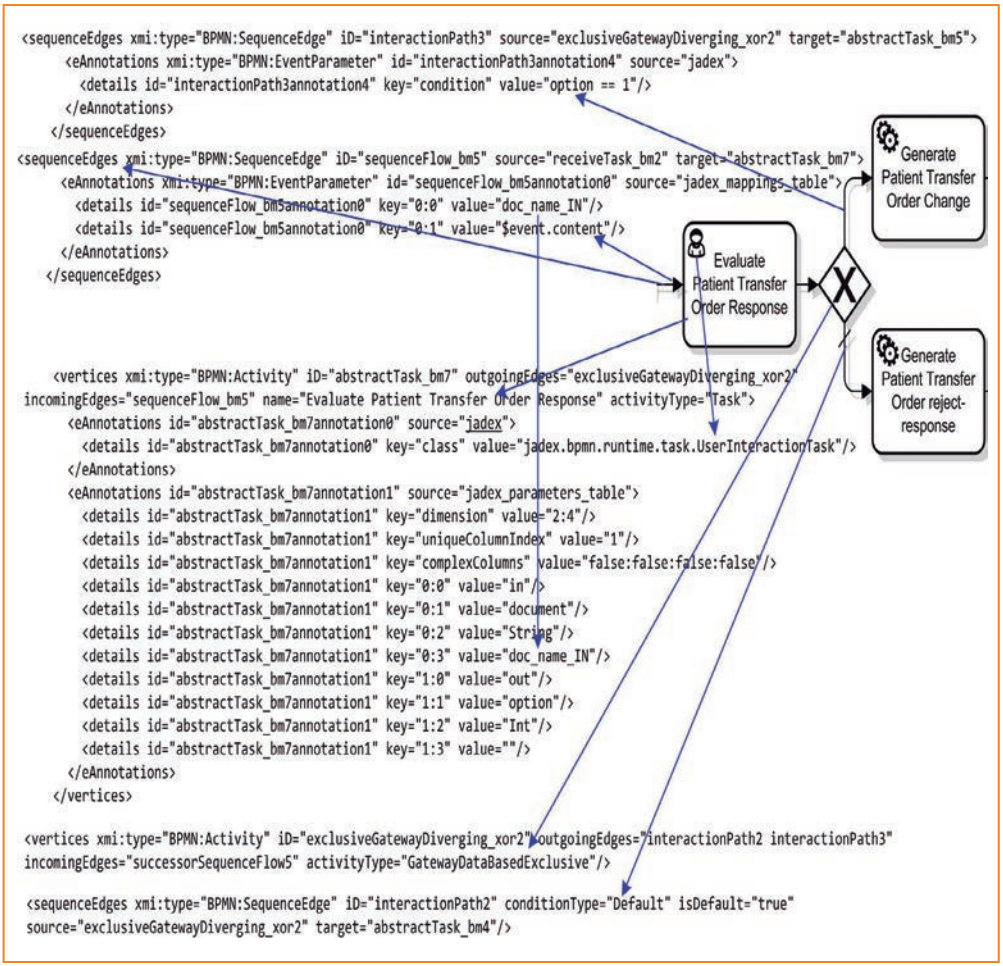



Figura 6.7. *Especificaciones generadas para el evento intermedio de mensaje tipo* enviar *o* recibir.

La tarea de usuario `Evaluate Transfer Order Response` es generada como una actividad de tipo tarea que habilita la intervención de un usuario en la ejecución del proceso colaborativo. Esta tarea es anotada con la clase predefinida `UserInteractionTask`, la cual provee su implementación en la plataforma (figura 6.8). La tarea es generada con una serie de parámetros para la definición del documento de entrada. En este caso, la variable `doc_name_IN`, generada en el elemento conector de secuencia anterior a la tarea, se especifica como el documento de entrada a la tarea. Al momento de la ejecución de una instancia del modelo de proceso ejecutable, esta tarea desplegará el contenido del documento permitiendo su análisis, evaluación y actualización por parte del usuario que realiza la tarea.

El valor contenido en esta variable `option` de salida es utilizado en los conectores de secuencia de salida del `GatewayDataBasedExclusive` (compuerta exclusiva basada en datos) generado, para determinar el camino a seguir en el proceso. Uno de los conectores (`interactionPath2`) es anotado como la salida por `default` de la compuerta exclusiva y al otro conector (`interactionPath3`) se le genera una anotación semántica especificando

una condición que debe cumplirse para seguir ese camino del proceso. En la figura 6.8 se muestra la condición generada `option == 1` mediante el parámetro `condition`. La variable `option` y su valor fueron generados previamente en la tarea de evaluación; de no cumplirse esta condición el proceso seguirá por el conector de secuencia marcado como `default`.




```
<sequenceEdges xmi:type="BPMN:SequenceEdge" iD="interactionPath3" source="exclusiveGatewayDiverging_xor2" target="abstractTask_bm5">
      <eAnnotations xmi:type="BPMN:EventParameter" id="interactionPath3annotation4" source="jadex">
        <details id="interactionPath3annotation4" key="condition" value="option == 1"/>
      </eAnnotations>
    </sequenceEdges>
<sequenceEdges xmi:type="BPMN:SequenceEdge" iD="sequenceFlow_bm5" source="receiveTask_bm2" target="abstractTask_bm7">
      <eAnnotations xmi:type="BPMN:EventParameter" id="sequenceFlow_bm5annotation0" source="jadex_mappings_table">
        <details id="sequenceFlow_bm5annotation0" key="0:0" value="doc_name_IN"/>
        <details id="sequenceFlow_bm5annotation0" key="0:1" value="$event.content"/>
      </eAnnotations>
    </sequenceEdges>

    <vertices xmi:type="BPMN:Activity" iD="abstractTask_bm7" outgoingEdges="exclusiveGatewayDiverging_xor2"
 incomingEdges="sequenceFlow_bm5" name="Evaluate Patient Transfer Order Response" activityType="Task">
      <eAnnotations id="abstractTask_bm7annotation0" source="jadex">
        <details id="abstractTask_bm7annotation0" key="class" value="jadex.bpmn.runtime.task.UserInteractionTask"/>
      </eAnnotations>
      <eAnnotations id="abstractTask_bm7annotation1" source="jadex_parameters_table">
        <details id="abstractTask_bm7annotation1" key="dimension" value="2:4"/>
        <details id="abstractTask_bm7annotation1" key="uniqueColumnIndex" value="1"/>
        <details id="abstractTask_bm7annotation1" key="complexColumns" value="false:false:false:false"/>
        <details id="abstractTask_bm7annotation1" key="0:0" value="in"/>
        <details id="abstractTask_bm7annotation1" key="0:1" value="document"/>
        <details id="abstractTask_bm7annotation1" key="0:2" value="String"/>
        <details id="abstractTask_bm7annotation1" key="0:3" value="doc_name_IN"/>
        <details id="abstractTask_bm7annotation1" key="1:0" value="out"/>
        <details id="abstractTask_bm7annotation1" key="1:1" value="option"/>
        <details id="abstractTask_bm7annotation1" key="1:2" value="Int"/>
        <details id="abstractTask_bm7annotation1" key="1:3" value=""/>
      </eAnnotations>
    </vertices>

<vertices xmi:type="BPMN:Activity" iD="exclusiveGatewayDiverging_xor2" outgoingEdges="interactionPath2 interactionPath3"
incomingEdges="successorSequenceFlow5" activityType="GatewayDataBasedExclusive"/>

<sequenceEdges xmi:type="BPMN:SequenceEdge" iD="interactionPath2" conditionType="Default" isDefault="true"
source="exclusiveGatewayDiverging_xor2" target="abstractTask_bm4"/>
```

Figura 6.8. *Especificaciones para la tarea* evaluate, *compuerta basada en datos y conectores de secuencia.*

La tarea de enviar `Send Confirm Patient Transfer Order reject-response` del modelo origen se convierte en actividad de tipo evento intermedio de mensaje en el modelo destino, el cual es anotado como `isThrowing`, similar a la especificación presentada en la figura 6.7, exceptuando la especificación del parámetro `performative`, el cual es generado con el acto de comunicación `Confirm` (figura 6.9). Asimismo en la figura 6.9 se describe la especificación generada para un evento de fin de tipo `Terminate`.

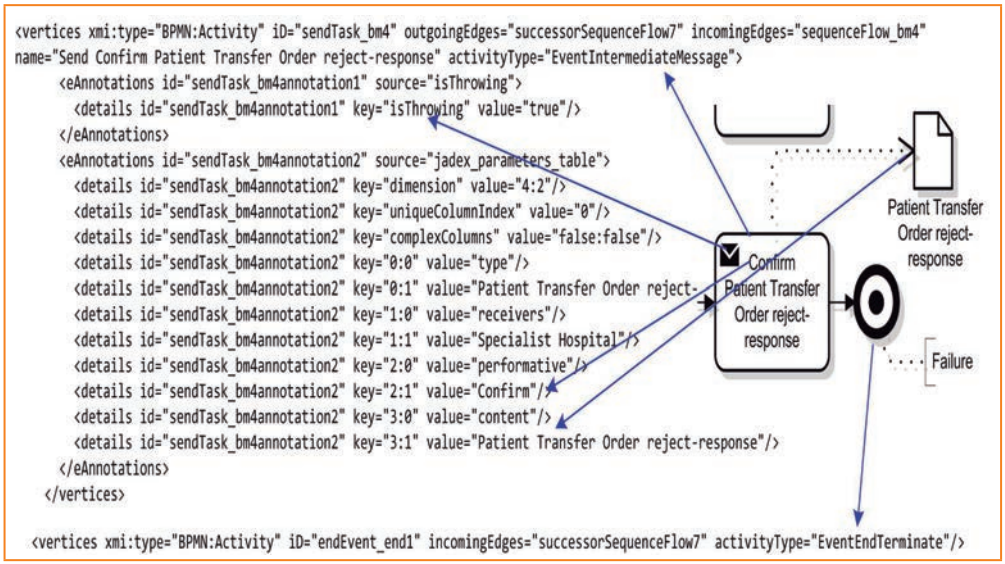



Figura 6.9. *Especificaciones para el evento intermedio de mensaje* enviar *y evento de fin* terminate.

La actividad de tipo evento intermedio de mensaje `Send InformPatientTransfer Order Change`, anotada como `isThrowing`, se generó a partir de la tarea de envío de mensaje con el mismo nombre contenida en el modelo origen (figura 6.10). El parámetro performative es generado con el valor `Inform`. Los demás parámetros son generados en forma similar a la descripción de la actividad de tipo evento intermedio de mensaje (anotada como `isThrowing`) presentada en la figura 6.7.

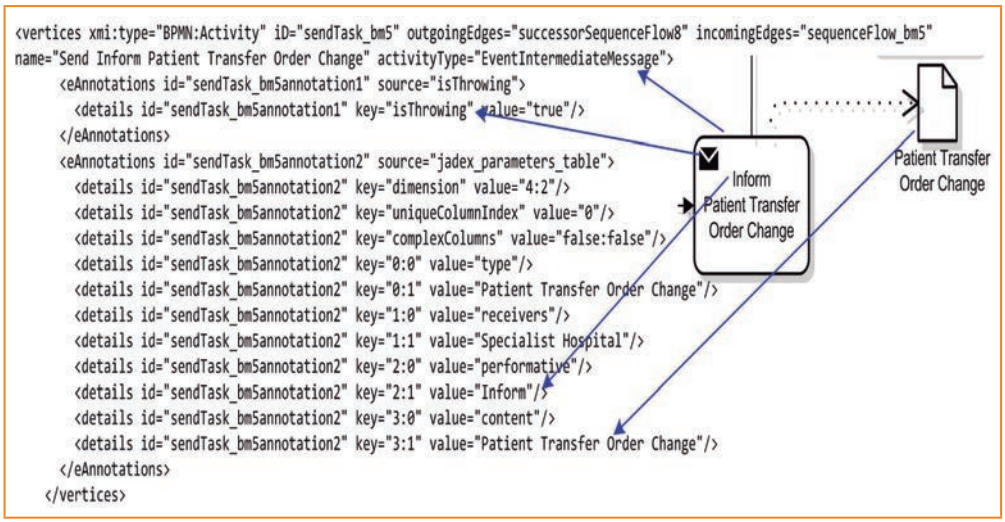
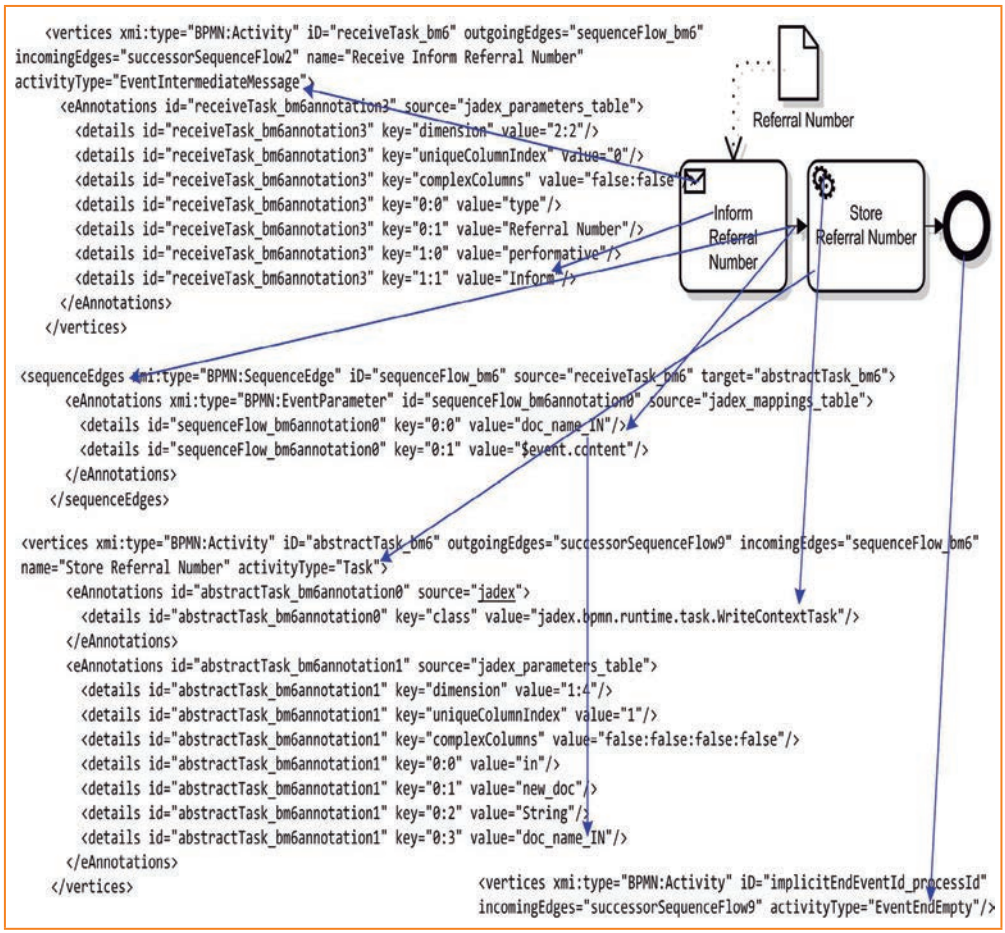



Figura 6.10. *Especificaciones para el evento intermedio de mensaje* enviar.

La tarea de servicio `Store Referral Number` del modelo origen es generada como una actividad de tipo tarea anotada con la clase `WriteContextTask` (figura 6.11). Mediante esta clase se implementa dicha tarea en la plataforma.




```
<vertices xmi:type="BPMN:Activity" iD="receiveTask_bm6" outgoingEdges="sequenceFlow_bm6"
incomingEdges="successorSequenceFlow2" name="Receive Inform Referral Number"
activityType="EventIntermediateMessage">
  <eAnnotations id="receiveTask_bm6annotation3" source="jadex_parameters_table">
    <details id="receiveTask_bm6annotation3" key="dimension" value="2:2"/>
    <details id="receiveTask_bm6annotation3" key="uniqueColumnIndex" value="0"/>
    <details id="receiveTask_bm6annotation3" key="complexColumns" value="false:false"/>
    <details id="receiveTask_bm6annotation3" key="0:0" value="type"/>
    <details id="receiveTask_bm6annotation3" key="0:1" value="Referral Number"/>
    <details id="receiveTask_bm6annotation3" key="1:0" value="performative"/>
    <details id="receiveTask_bm6annotation3" key="1:1" value="Inform"/>
  </eAnnotations>
</vertices>

<sequenceEdges xmi:type="BPMN:SequenceEdge" iD="sequenceFlow_bm6" source="receiveTask_bm6" target="abstractTask_bm6">
  <eAnnotations xmi:type="BPMN:EventParameter" id="sequenceFlow_bm6annotation0" source="jadex_mappings_table">
    <details id="sequenceFlow_bm6annotation0" key="0:0" value="doc_name_IN"/>
    <details id="sequenceFlow_bm6annotation0" key="0:1" value="$event.content"/>
  </eAnnotations>
</sequenceEdges>

<vertices xmi:type="BPMN:Activity" iD="abstractTask_bm6" outgoingEdges="successorSequenceFlow9" incomingEdges="sequenceFlow_bm6"
name="Store Referral Number" activityType="Task">
  <eAnnotations id="abstractTask_bm6annotation0" source="jadex">
    <details id="abstractTask_bm6annotation0" key="class" value="jadex.bpmn.runtime.task.WriteContextTask"/>
  </eAnnotations>
  <eAnnotations id="abstractTask_bm6annotation1" source="jadex_parameters_table">
    <details id="abstractTask_bm6annotation1" key="dimension" value="1:4"/>
    <details id="abstractTask_bm6annotation1" key="uniqueColumnIndex" value="1"/>
    <details id="abstractTask_bm6annotation1" key="complexColumns" value="false:false:false:false"/>
    <details id="abstractTask_bm6annotation1" key="0:0" value="in"/>
    <details id="abstractTask_bm6annotation1" key="0:1" value="new_doc"/>
    <details id="abstractTask_bm6annotation1" key="0:2" value="String"/>
    <details id="abstractTask_bm6annotation1" key="0:3" value="doc_name_IN"/>
  </eAnnotations>
</vertices>

<vertices xmi:type="BPMN:Activity" iD="implicitEndEventId_processId"
incomingEdges="successorSequenceFlow9" activityType="EventEndEmpty"/>
```

Figura 6.11. *Especificaciones para el conector de secuencia, la tarea* store *y el evento de fin.*

También se generan los parámetros de la tarea, que especifican el documento de entrada a la tarea contenido en la variable `doc_name_IN`, el cual será almacenado por la tarea. Este documento es previamente recuperado del contenido del evento intermedio de mensaje recibido, para lo cual se define un mapeo mediante una anotación en el conector de secuencia existente entre el evento intermedio de mensaje `Inform Referral Number` y la tarea `Store Referral Number` (figura 6.11).

El conector de secuencia `sequenceFlow_bm6` es generado con una tabla de mapeo que contiene la definición de una variable local `doc_name_IN` y la instrucción

`$event.content`, que permite en tiempo de ejecución recuperar el contenido del evento intermedio de mensaje y almacenarlo en dicha variable local. Asimismo en la figura 6.11 se describe la especificación generada para un evento de fin de tipo `Empty`.

Para ejemplificar otra transformación, a continuación se describe la generación del modelo de proceso ejecutable (modelo destino) de la organización `Specialist Hospital`, derivado del modelo de proceso de integración de "Gestión de Referencia Médica" del SCP (modelo origen). También se muestra el mapeo de los elementos del modelo destino en términos de su representación en un archivo XMI usado para persistir el modelo. En el modelo destino se genera el identificador `participant2` y el nombre `Specialist Hospital` para el elemento `pool`. El proceso contiene un evento de inicio que es generado como una actividad de tipo `EventStartEmpty` (figura 6.12).

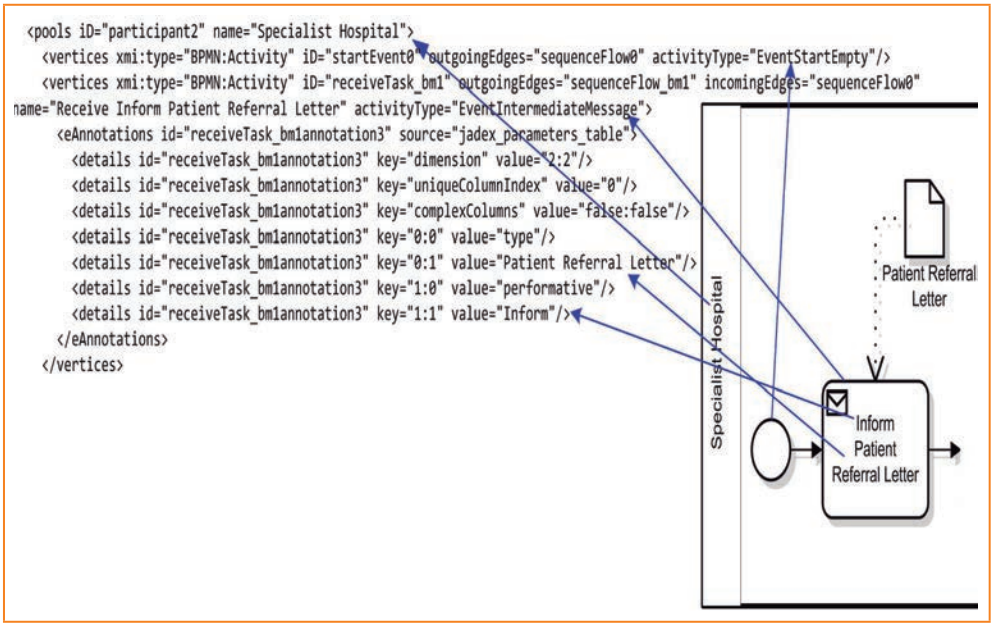



Figura 6.12. *Especificaciones generadas para los eventos de inicio, intermedio de mensaje y* pool.

La tarea de recepción de mensaje `Receive Inform Patient Referral Letter` del modelo origen es generada como una actividad de tipo evento intermedio de mensaje, utilizando el tipo de actividad `EventIntermediateMessage` (figura 6.12). Esta actividad de evento de mensaje es anotada con el nombre `Patient Referral Letter` mediante la variable `type`, con la cual se identificará al evento intermedio de mensaje en tiempo de ejecución y con el acto de comunicación `Inform` mediante el parámetro `performative`.

La actividad de tipo tarea `Evaluate Patient Referral Letter` es derivada de la correspondiente tarea de usuario del modelo origen. La tarea generada es anotada con la clase predefinida `UserInteractionTask`. La tabla de parámetros de dicha tarea

contiene la variable local doc_name_IN mediante la cual se define el documento de entrada a la tarea de evaluación y la variable local option a través de la cual se almacena el resultado de la evaluación realizada por el usuario (figura 6.13).

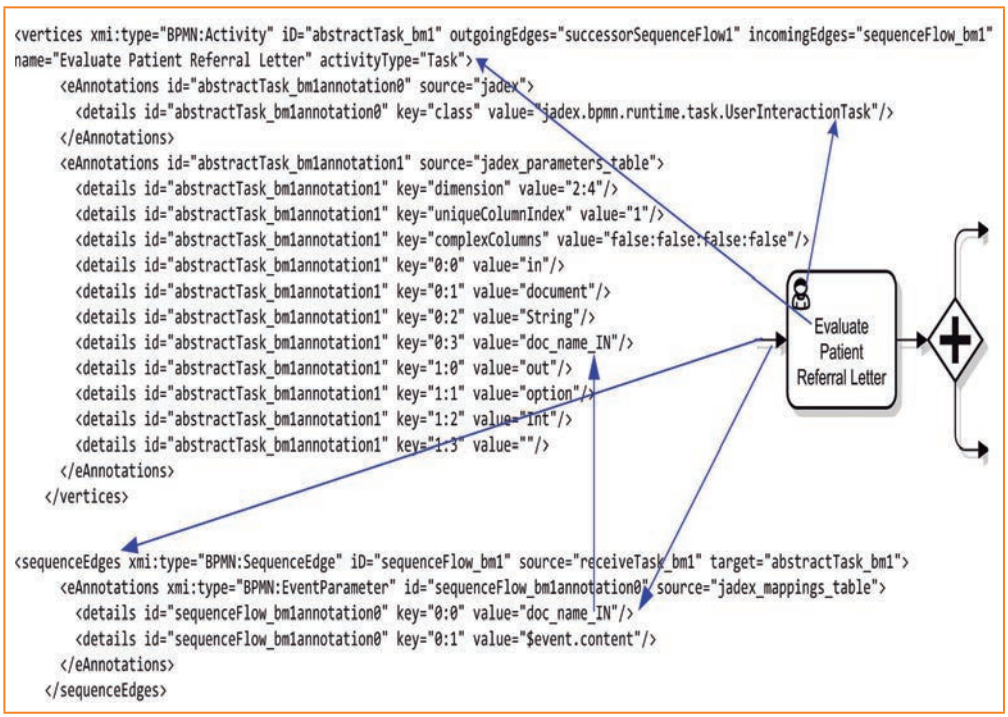



Figura 6.13. *Especificaciones generadas para el conector de secuencia y la tarea* evaluate.

El documento contenido en la variable local doc_name_IN es mapeado (copiado) del documento contenido en el evento intermedio de mensaje Receive Inform Patient Referral Letter (figura 6.13). Este mapeo es realizado mediante la definición y anotación del conector de secuencia sequenceFlow_bm1 que relaciona la actividad del evento intermedio de mensaje con la actividad de tarea Evaluate Patient Referral. En la anotación del conector de secuencia se genera una tabla jadex_mappings_table conteniendo la variable local doc_name_IN, en la que se almacenará el documento y la instrucción $event.content, la cual permite recuperar el *objeto* o *string* contenido en el parámetro content del mensaje recibido.

El siguiente elemento en el modelo origen es una compuerta paralela, la cual es generada en el modelo destino también como una compuerta paralela parallelGateway_and1. Esta compuerta tiene definido un conector de secuencia de entrada y dos conectores de secuencia de salida, permitiendo ejecutar en paralelo las actividades de tarea Generate Patient Referral Receipt y Generate Patient Reception (figura

6.14). Ambas se generan con la clase de implementación `GenerateDocTask` y se anotan mediante la tabla `jadex_parameters_table` con los parámetros de entrada y salida.

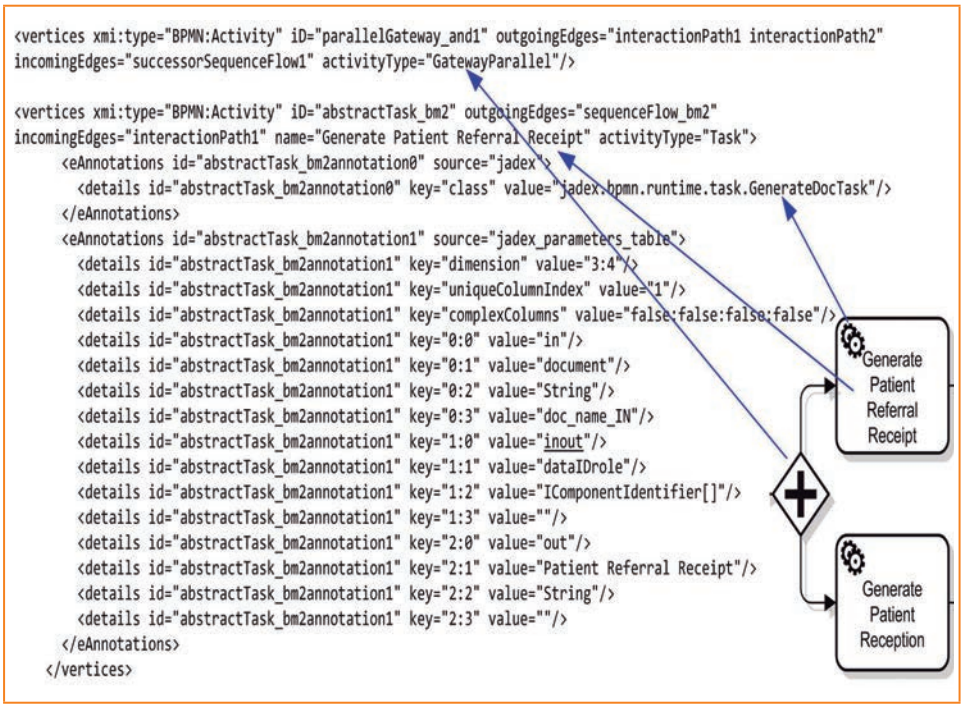



Figura 6.14. *Especificaciones generadas para la compuerta paralela y la tarea* generate.

El documento generado por este tipo de tareas incluye los datos de identificación del componente que ejecuta el proceso; estos datos son almacenados en la variable local `dataIDrole` y son recuperados mediante el método `IComponentIdentifier()` que provee la plataforma de agentes JadexBDI. Estos datos son integrados al documento permitiendo identificar en cualquier momento el rol que generó el mismo. El documento generado (salida de la tarea) se almacena en la variable local `Patient Referral Receipt`, que tiene una dirección de salida `out` (figura 6.14). La tarea de servicio `Generate Patient Reception` es convertida en una actividad de tipo tarea con la clase de implementación y parámetros de ejecución, en forma similar a la especificación de la tarea `Generate Patient Referral Receipt`.

La tarea de envío `Send Confirm Patient Referral Receipt` del modelo origen es generada en el modelo destino como una actividad de evento intermedio de mensaje con la anotación semántica `isThrowing`. Este evento es anotado con los parámetros `type`, `receivers`, `performative` y `content` (figura 6.15). Para este evento intermedio de mensaje, el parámetro `performative` contiene el acto de comunicación `Confirm` y en

el parámetro content se almacena el documento Patient Referral Receipt que es enviado en el mensaje que genera esta actividad de evento intermedio de mensaje. La tarea de envío Send Confirm Patient Reception del modelo origen es generada en el modelo destino como una actividad de evento intermedio de mensaje con anotación semántica y parámetros de ejecución similares a los especificados en la actividad de evento intermedio de mensaje Send Confirm Patient Referral Receipt.

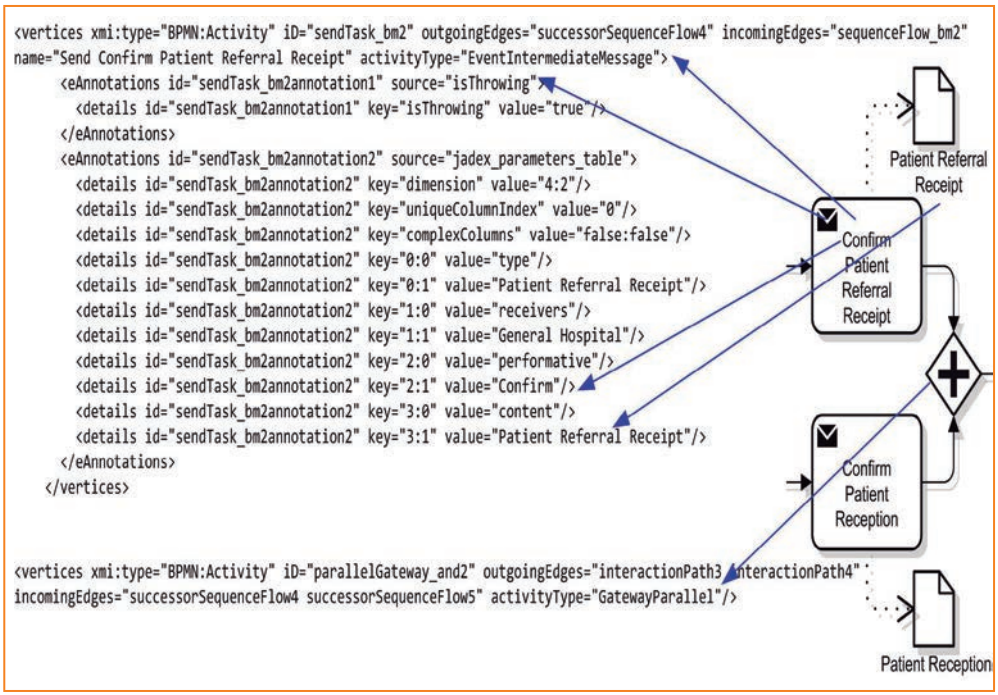



Figura 6.15. *Especificaciones generadas para el evento intermedio de mensaje y la compuerta paralela.*

La compuerta paralela parallelGateway_and2 generada en el modelo destino tiene especificados los conectores de secuencia de entrada successorSequenceFlow4 y successorSequenceFlow5 que definen la sincronización de los caminos de las actividades de eventos intermedios de mensaje Send Confirm Patient Referral Receipt y Send Confirm Patient Reception, los cuales se ejecutan en forma paralela (figura 6.15). En la compuerta paralela parallelGateway_and2 se especifican los conectores de secuencia de salida interactionPath3 y interactionPath4, que habilitan dos caminos para ejecutar en forma paralela las tareas Generate Medical Summary y Generate Discharge Note (figura 6.16).

La tarea de envío Send Inform Medical Summary del modelo origen es derivada en una actividad de evento intermedio de mensaje asignándole el mismo nombre mediante el atributo name (figura 6.16). La anotación semántica y la tabla de paráme-

tros de ejecución son especificadas en forma similar al evento intermedio de mensaje presentado en la figura 6.15. Sólo cambia el valor de los parámetros de acuerdo con el evento generado. Otro caso similar se presenta con la actividad de evento intermedio de mensaje `Send Inform Dicharge Note` generada.

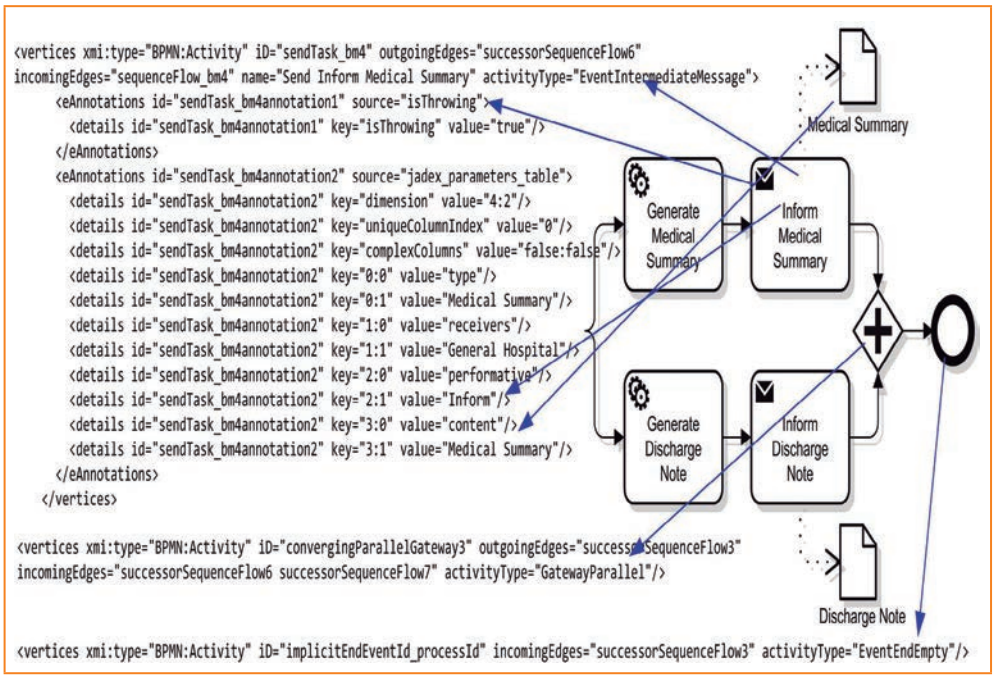



Figura 6.16. *Especificaciones generadas para los eventos de mensaje de tipo* enviar *y evento de fin.*

Posteriormente se genera la compuerta paralela `convergingParallelGateway3` que sincroniza los caminos en paralelo de las dos actividades de eventos intermedios de mensaje generadas previamente (figura 6.16). Esta compuerta paralela tiene un sólo conector de secuencia de salida que la relaciona con el evento de fin `EventEndEmpty`, que finaliza una instancia del modelo de proceso ejecutable que implementa el proceso de integración "Gestión de Referencia Médica".

## 6.1.4. Generación del código de los agentes *AdministradorDeProceso*

El código de los agentes *AdministradorDeProceso* de las partes, definidos como agentes BDI de Jadex son derivados del modelo conceptual del proceso de integración de las partes. A continuación se describe el código del agente *AdministradorDeProceso* generado para el General Hospital con el rol PCP, para el proceso colaborativo "Orden de Transferencia de Paciente".

El agente generado contiene las secciones <agent>, <imports>, <beliefs>, <capabilities>, <goals>, <plans>, <events> y <configurations>, las cuales habilitan crear una instancia, inicializar y ejecutar un agente de este tipo en la plataforma. La sección <agent> contiene el nombre del agente General_HospitalPTO generado a partir del nombre del participante en el modelo del proceso de integración (figura 6.17). En esta sección también se genera el paquete de implementación del agente agents, el cual está predefinido en el método de transformación. En la parte superior del archivo de código ADF generado se crea una notación con el nombre y el rol del agente <!- - <h3>General_HospitalPTO - PCP/<h3> - ->, habilitando desplegar el nombre del agente en algunas secciones de la plataforma.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <h3>General_HospitalPTO - PCP</h3> -->
<agent package="agents" name="General_HospitalPTO" xsi:schemaLocation="http://
jadex.sourceforge.net/jadex http://jadex.sourceforge.net/jadex-bdi-2.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
jadex.sourceforge.net/jadex">
  <imports>
    <import>jadex.*</import>
     . . .
  </imports>
  <beliefs>
    <belief name="agrspace" class="AGRSpace">
      <fact class="IFuture">$scope.getParent().getExtension("myA4IOCagrspace")"</
fact>
    </belief>
  </beliefs>
  <capabilities>
    <capability name="cmscap" file="CMS"/>
  </capabilities>
  <goals>
    <achievegoalref name="cms_create_component">
      <concrete ref="cmscap.cms_create_component"/>
    </achievegoalref>
    <achievegoalref name="cms_destroy_component">
      <concrete ref="cmscap.cms_destroy_component"/>
    </achievegoalref>
    <performgoal name="Decrease the management time on the patient admission-Goal"/>
  </goals>
```

Figura 6.17. *Código del agente generado para las secciones* agent, beliefs, capabilities *y* goals.

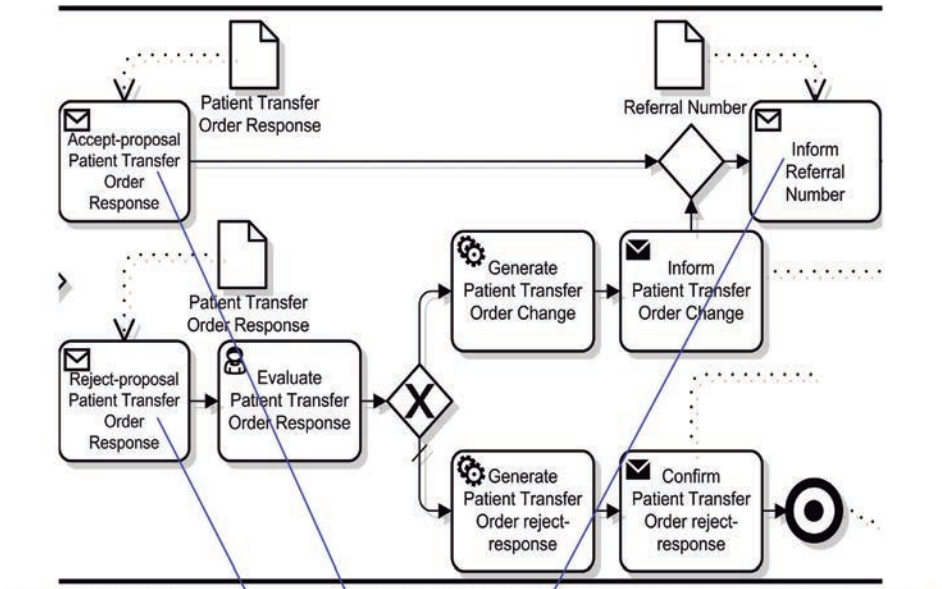
Las secciones <imports>, <beliefs> y <capabilities> se generan por el método de transformación para importar las clases y capacidades requeridas para

la implementación del agente, así como utilizar creencias en forma dinámica y el uso de capacidades provistas por la arquitectura Jadex-BDI, que habilitan crear agentes y componentes en tiempo de ejecución del sistema. El contenido de estas secciones ya está predefinido en los patrones de salida de las reglas del método de transformación.

En la sección `<goals>` se especifica la meta que debe alcanzar el agente mediante la ejecución de planes y eventos que definen su comportamiento. La meta de tipo `performgoal` es derivada de la meta definida en el modelo conceptual del proceso de integración "Orden de Transferencia de Paciente" (figura 6.17). Esta meta está contenida en el modelo de proceso de integración generado, pero la misma no es mostrada en los diagramas BPMN de dichos modelos. Adicionalmente, se generan dos metas de tipo `<achievegoal>` predefinidas en el método de transformación que posibilitan crear componentes, en este caso, inicializar al agente y crear una instancia del modelo de proceso ejecutable embebido en el agente, y destruir los componentes creados al finalizarse la ejecución de dicho proceso.

La sección `<plans>` está conformada por los planes que ejecutará el agente. El plan que implementará el agente *AdministradorDeProceso* es generado con el nombre del proceso colaborativo que acordaron las organizaciones. En este caso, `Patient-TransferOrder-plan` es el nombre del plan a ejecutar, definido como un plan de tipo BPMN (figura 6.18). El nombre del archivo que contiene el plan a ejecutar se define mediante el atributo `impl`. Este archivo `PatientTransferOrder` corresponde al modelo de proceso ejecutable que implementa y puede ejecutar el agente. Cuando el agente inicia su ejecución en la plataforma, ejecuta el plan definido, creando una instancia del modelo de proceso ejecutable e iniciando su ejecución. Con ello, la instancia del proceso mantiene el control del comportamiento del agente, interactuando a través del envío y recepción de mensajes con el otro agente *AdministradorDeProceso*, función que es coordinada por el proceso y ejecutada por el agente.

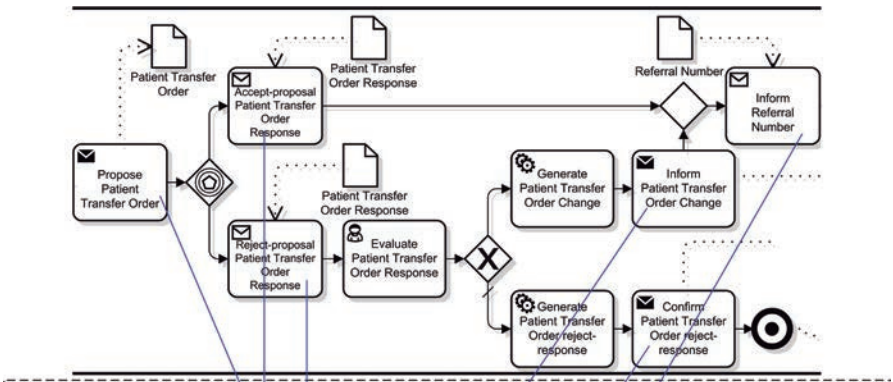
Por lo tanto, la ejecución del plan del agente está condicionada también por la interacción que mantiene con los otros agentes *AdministradorDeProceso* que ejecutan el proceso colaborativo. Esto es definido mediante elementos `<waitqueue>` dentro del plan del agente. Se genera un `<waitqueue>` por cada evento de mensaje que el agente espera recibir en la interacción que realiza con el otro agente. Estos eventos de mensaje son derivados de las tareas de recepción del modelo conceptual de proceso de integración (figura 6.18). De esta manera, el plan `PatientTransferOrder-plan` generado contiene las referencias de los eventos de mensaje `Accept-proposal PatientTransferOrderResponse`, `Reject-proposal Patient Transfer Order Response` e `Inform Referral Number` correspondientes a las tres tareas de recepción de mensajes del modelo conceptual del proceso de integración. En la definición del plan también se hace referencia a la meta `Decrease the management time on the patient admission` que debe ser alcanzada mediante la ejecución del plan.




```
<plans>
  <plan name="PatientTransferOrder-plan">
    <body type="bpmn" impl="PatientTransferOrder"/>
    <waitqueue>
      <messageevent ref="Accept-Proposal Patient Transfer Order response"/>
    </waitqueue>
    <waitqueue>
      <messageevent ref="Reject-Proposal Patient Transfer Order Response"/>
    </waitqueue>
    <waitqueue>
      <messageevent ref="Inform Referral Number"/>
    </waitqueue>
    <trigger>
      <goal ref="Decrease the management time on the patient admission-Goal"/>
    </trigger>
  </plan>
</plans>
```

Figura 6.18. *Código del agente generado para la sección* plans.

La sección `<events>` está definida por los eventos de mensaje generados a partir de las tareas de tipo envío y recepción de mensajes contenidas en el modelo conceptual del proceso de integración. Éstos representan los mensajes que puede enviar o recibir el agente (figura 6.19). Los eventos de mensaje del agente están directamente relacionados con las actividades de tipo evento intermedio de mensaje definidas en el modelo de proceso ejecutable.




```
<events>
  <messageevent name="Propose Patient Transfer Order" type="fipa" direction="send">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.PROPOSE</value>
    </parameter>
    <parameter name="language" class="String" direction="fixed">
      <value>SFipa.JADEX_XML</value>
    </parameter>
  </messageevent>
  <messageevent name="Accept-Proposal Patient Transfer Order response" type="fipa" direction="receive">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.ACCEPT-PROPOSAL</value>
    </parameter>
  </messageevent>
  <messageevent name="Reject-Proposal Patient Transfer Order response" type="fipa" direction="receive">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.REJECT-PROPOSAL</value>
    </parameter>
  </messageevent>
  <messageevent name="Inform Patient Transfer Order change" type="fipa" direction="send">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.INFORM</value>
    </parameter>
    <parameter name="language" class="String" direction="fixed">
      <value>SFipa.JADEX_XML</value>
    </parameter>
  </messageevent>
  <messageevent name="Confirm Patient Transfer Order reject-response" type="fipa" direction="send">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.CONFIRM</value>
    </parameter>
    <parameter name="language" class="String" direction="fixed">
      <value>SFipa.JADEX_XML</value>
    </parameter>
  </messageevent>
  <messageevent name="Inform Referral Number" type="fipa" direction="receive">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.INFORM</value>
    </parameter>
  </messageevent>
</events>
```

Figura 6.19. *Código del agente generado para la sección* events.

Por lo tanto, cuando se ejecuta una instancia del modelo de proceso ejecutable como plan del agente, por cada evento intermedio de mensaje marcado como `isThrowing` que se ejecute se invocará al evento de mensaje que tiene relacionado en la sección `<events>` del agente. El agente ejecutará el evento de mensaje estableciendo la interac-

ción con el otro agente involucrado en el proceso colaborativo. Cuando el agente reciba un mensaje desde el otro agente, ejecutará un proceso inverso, entregando a la instancia en ejecución del modelo de proceso ejecutable el evento de mensaje recibido, el cual lo procesará y continuará la ejecución del mismo por el agente.

Los eventos de mensaje derivados de las tareas de envío son definidos en un elemento `<messageevent>` con la definición de dos elementos `<parameter>` y un elemento `<value>` para cada parámetro. Por ejemplo, el evento de mensaje `Propose Patient Transfer Order` se especifica con una dirección de tipo `send` y en el parámetro `performative` con el valor del acto de comunicación contenido en el mensaje `SFipa.PROPOSE` y un segundo parámetro `language` con un valor `SFipa.JADEX_XML`, el cual es utilizado en los eventos de mensaje de tipo `send` para definir el lenguaje utilizado en el contenido del mensaje (figura 6.19). Los eventos de mensaje derivados de las tareas de recepción también son definidos en un elemento `<messageevent>`, especificados con una dirección de tipo `receive` y sólo tienen un elemento `<parameter>`, en el cual se define el acto de comunicación que deberá contener el evento de mensaje que se reciba.

La sección `<configurations>` permite definir los diferentes roles que el agente puede implementar y la configuración inicial del agente que habilita su ejecución en la plataforma. Para el agente `General_HospitalPTO` en el proceso "Orden de Transferencia de Paciente" se generó en el elemento `<configuration>` una configuración inicial para el agente con el rol PCP derivado del rol que la organización desempeña en el proceso de integración y se definió la meta inicial `Decrease the management time on the patient admission`, derivada de la meta contenida en el modelo origen y configurada en el agente como una meta de tipo `<performgoal>` (figura 6.20).

```
  <configurations>
    <configuration name="PCP">
      <goals>
        <initialgoal ref="Decrease the management time on the patient admission-Goal"/>
      </goals>
    </configuration>
  </configurations>
</agent>
```

Figura 6.20. *Código del agente generado para la sección* configurations.

### 6.1.5. Ejecución de los procesos colaborativos por los agentes *AdministradorDeProceso*

Los agentes *AdministradorDeProceso* derivados de los modelos conceptuales de procesos de integración permiten dar soporte a la ejecución de los procesos colaborativos acordados entre el Hospital General y el Hospital de Especialidades. Éstos son instanciados en los sistemas multiagentes (MAS) de estas organizaciones según la organización a la que representan, como se describe en la sección 6.1.1. Como ejemplo, a continuación se describe la ejecución de una instancia del proceso colaborativo "Orden de Transferencia de Paciente" por medio de los agentes *AdministradorDeProceso* de cada una de las partes.

El comportamiento que siguen los agentes *AdministradorDeProceso* del `General_Hospital` (rol PCP) y del `Specialist_Hospital` (rol SCP) dependen del resultado de las tareas de evaluación (ejecutadas por un usuario) contenidas en sus respectivos modelos de proceso ejecutable, las cuales determinan el camino a seguir en la ejecución de dicho proceso. Por tal motivo, los agentes que implementan y ejecutan el proceso colaborativo pueden presentar diferentes comportamientos e interacciones en tiempo de ejecución para los diferentes escenarios o instancias de ejecución de dicho proceso.

La figura 6.21 muestra el escenario 1 de ejecución de dicho proceso por parte de instancias de estos agentes. Luego de que los agentes *AdministradorDeColaboraciones* (AC) de las partes crean una instancia de los agentes *AdministradorDeProceso* en cada uno de los MAS de las organizaciones, éstos son los agentes `General_HospitalPTO` y `Specialist_HospitalPTO`, ambos reciben un mensaje `inform` de su respectivo agente AC, con los datos de identificación del agente AP con el que establecerán comunicación. Los agentes procesan la información recibida e inician su operación, que consiste en ejecutar el rol definido en su sección `<configuration>` con la implementación del plan definido. A partir de aquí los agentes *AdministradorDeProceso* están habilitados para ejecutar su modelo de proceso ejecutable e interactuar entre ellos para ejecutar una instancia del proceso colaborativo.

En el escenario 1 (figura 6.21), el proceso comienza con el agente `General_HospitalPTO` que genera y envía una propuesta de transferencia de un paciente mediante un mensaje `propose` conteniendo el documento `Patient Transfer Order` (interacción 21). El agente `Specialist_HospitalPTO` evalúa la propuesta y responde con un mensaje `accept-proposal`, aceptando la propuesta de transferencia de paciente (interacción 22). El agente `Specialist_HospitalPTO` procesa los datos del paciente y genera un documento que contiene el número de referencia para el paciente, el cual permitirá identificar los datos clínicos del paciente en el Hospital de Especialidades. Este documento es enviado al agente `General_HospitalPTO` mediante un mensaje `inform` (interacción 23). El agente `General_HospitalPTO` recibe el mensaje con el número de referencia, lo almacena mediante una tarea interna y finaliza la ejecución del proceso de integración.

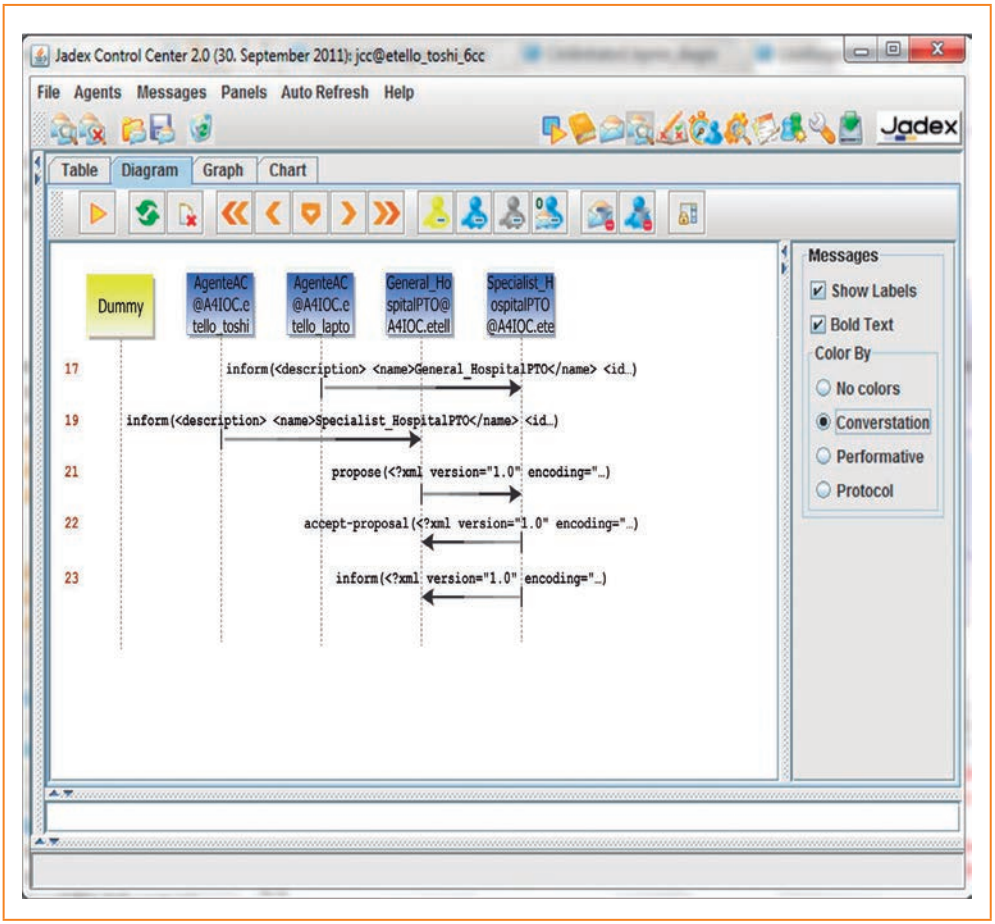



Figura 6.21. *Ejecución del proceso colaborativo "Orden de Transferencia de Paciente", escenario-1.*

La figura 6.22 muestra otro escenario de ejecución del proceso. El agente `General_HospitalPTO` envía una propuesta de transferencia de paciente al agente `Specialist_HospitalPTO` mediante un mensaje `propose` (interacción 21). El agente `Specialist_HospitalPTO` evalúa el documento `Patient Transfer Order` y declina la aceptación del paciente, generando un documento con los motivos del rechazo de la propuesta, el cual es enviado mediante un mensaje `reject-proposal` (interacción 22). El agente `General_HospitalPTO` recibe el mensaje, evalúa el documento contenido y decide generar un documento con una contrapropuesta, modificando la prioridad de atención del paciente, el cual es enviado mediante un mensaje `inform` (interacción 23). El agente `Specialist_HospitalPTO` procesa el documento recibido y genera en forma automática el número de referencia con que será admitido el paciente

en el Hospital de Especialidades. Este número de referencia es enviado mediante documento contenido en un mensaje `inform` (interacción 24). El agente `General_HospitalPTO` recibe el mensaje y lo procesa mediante una tarea interna de almacenar.

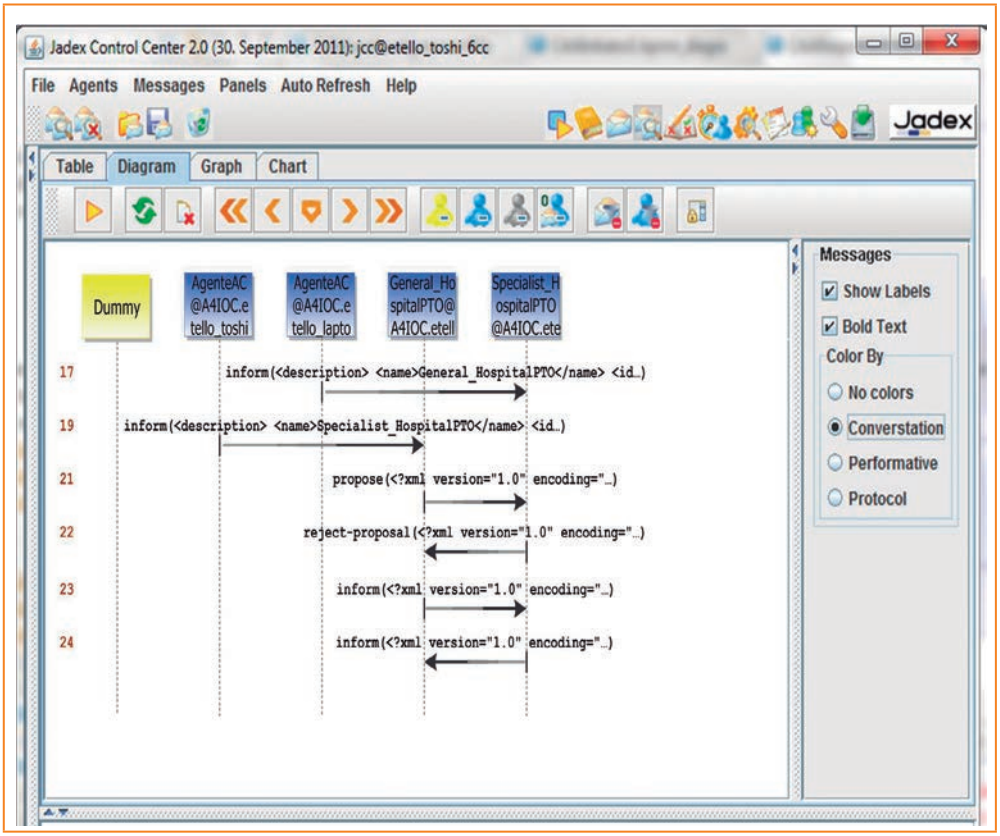



Figura 6.22. *Ejecución del proceso colaborativo "Orden de Transferencia de Paciente", escenario-2.*

Finalmente, otro comportamiento que pueden implementar los agentes `General_HospitalPTO` y `Specialist_HospitalPTO` en la ejecución del proceso colaborativo "Orden de Transferencia de Paciente" corresponde al escenario 3 (figura 6.23). El agente `General_HospitalPTO` envía una propuesta de transferencia del paciente mediante un mensaje `propose` (interacción 21). El agente `Specialist_HospitalPTO` evalúa el documento `Patient Transfer Order` recibido, declinando la propuesta, y genera un documento en el que justifica la decisión tomada, el cual es enviado mediante un mensaje `reject-proposal` (interacción 22). El agente `General_HospitalPTO` recibe el mensaje y ejecuta una tarea interna de evaluación del documento contenido en el mensaje, generando un documento con la aceptación de los

motivos del rechazo presentados en el documento recibido. Este documento es enviado al agente `Specialist_HospitalPTO` mediante un mensaje `confirm` (interacción 23). El agente `Specialist_HospitalPTO` recibe el mensaje, dando por terminada la comunicación e interacciones entre los agentes. En este escenario el paciente no es derivado al Hospital de Especialidades.

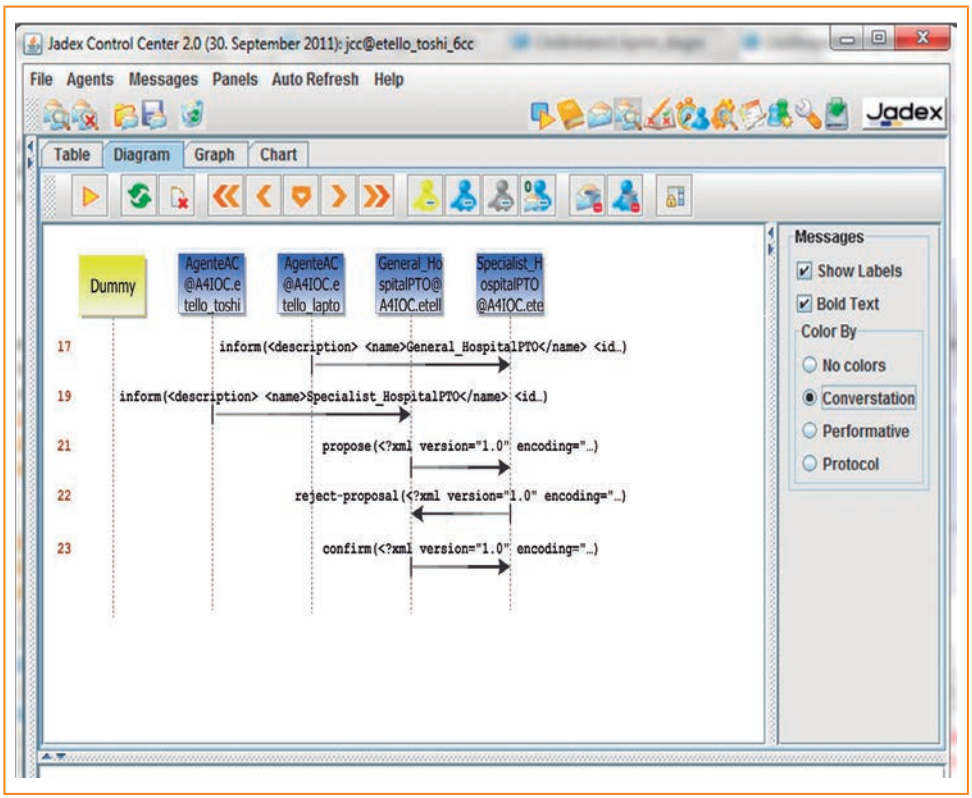



Figura 6.23. *Ejecución del proceso colaborativo "Orden de Transferencia de Paciente", escenario-3.*

### 6.1.6. Negociación de modificaciones en modelos de procesos colaborativos acordados

En esta sección se describe la negociación que se llevó a cabo entre el Hospital General y el Hospital de Especialidades cuando se presentó un nuevo requerimiento en los servicios integrados de atención a la salud, lo cual implicó la modificación de un proceso colaborativo previamente acordado en la colaboración interorganizacional dinámica. En la figura 6.24 se presentan las interacciones ejecutadas entre los agentes de los MAS

de la plataforma que representan a cada uno de los hospitales, para llevar a cabo una propuesta de modificación al modelo de proceso colaborativo "Gestión de Referencia Médica" acordado previamente.

El proceso de negociación inicia cuando el agente AC que representa al Hospital de Especialidades o SCP envía una propuesta de un cambio en dicho modelo de proceso colaborativo (interacción 1). Como parte del contenido del mensaje `propose`, el agente AC del SCP incluye un documento con los detalles de las modificaciones propuestas y el modelo del proceso colaborativo modificado. Luego el agente AC que representa al Hospital General o PCP envía un mensaje `accept-proposal` notificando que acepta la propuesta de cambio (interacción 2).

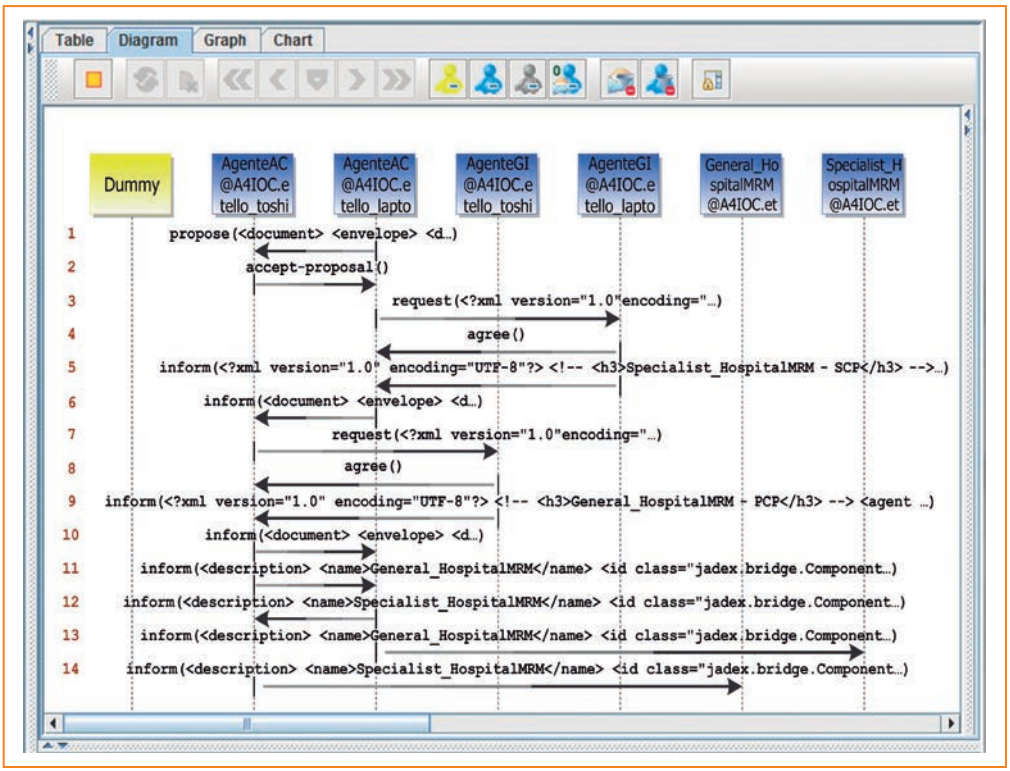



Figura 6.24. *Interacciones entre agentes AC para negociar en un modelo de proceso colaborativo.*

A continuación se desencadena un conjunto de interacciones en el MAS de SCP para generar la solución tecnológica (interacciones 3-5). Comparado con la negociación que se describió en la sección 6.1.1, en este caso no es necesario establecer la comunicación con el agente AM de cada MAS para recuperar el modelo de proceso colaborativo, debido a que los agentes AC de cada organización tienen almacenado

en sus creencias el modelo del proceso colaborativo modificado. Por lo tanto, a partir de la interacción 6 el comportamiento de las interacciones entre los agentes del MAS del PCP y SCP son similares a lo detallado en la figura 6.3 de la sección 6.1.1. La negociación de propuesta de cambios en el modelo del proceso colaborativo "Gestión de Referencia Médica" finaliza con la implementación de los agentes AP que dan soporte a la ejecución de la nueva versión del proceso colaborativo.

En la figura 6.25 se presenta la nueva versión del protocolo de interacción que representa al proceso colaborativo "Gestión de Referencia Médica". Éste muestra cambios a partir del segundo segmento de flujo de control AND, comparado con el protocolo presentado en la figura 6.2. En este segundo segmento se incluye un camino con un mensaje `inform` que contiene el documento *Planificación de Alta de Paciente* (*PacientDischargePlanning*), con el cual el SCP informa al PCP que el tratamiento médico del paciente está por terminar y notifica la programación de los eventos de alta médica del paciente. En esta programación el SCP puede informar que el paciente será dado de alta en el Hospital de Especialidades o que será transferido al Hospital General para continuar con su proceso de recuperación.

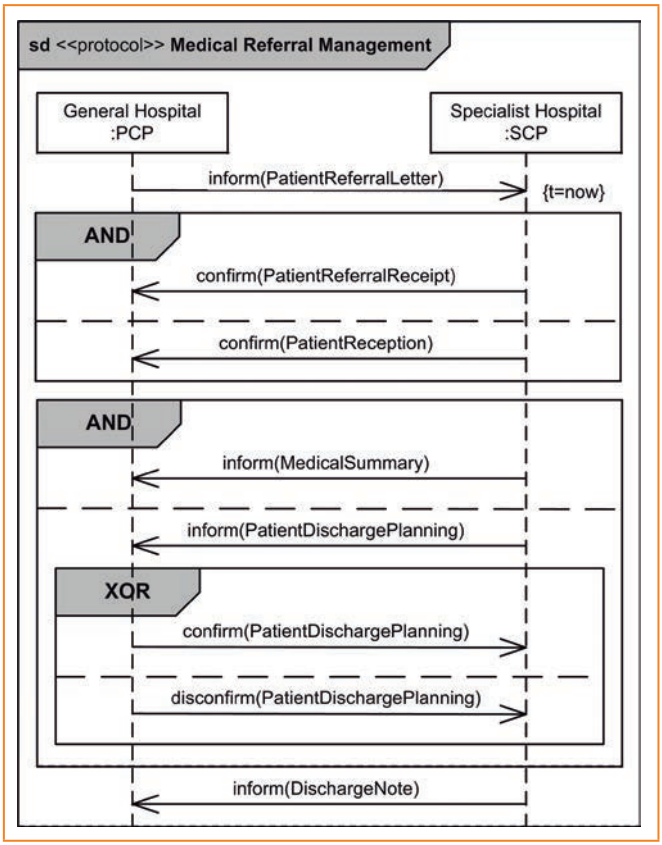



Figura 6.25. *Proceso colaborativo* Gestión de Referencia Médica *modificado.*

Cuando el rol PCP recibe el mensaje con la planificación de alta de paciente, puede responder con un mensaje *confirm* o *disconfirm*. Mediante el primer mensaje el PCP confirma que está de acuerdo con la acción a realizar por el SCP, y con el segundo mensaje el PCP notifica que no acepta el proceso de alta médica del paciente. Del lado del SCP, genera un documento de *alta de paciente* (*DischargeNote*) con base en la respuesta recibida del PCP. Este documento representa la autorización para que el paciente abandone el Hospital de Especialidades, continuando el tratamiento médico y la etapa de recuperación en el Hospital General o en forma independiente.

La ejecución de la nueva versión del proceso colaborativo "Gestión de Referencia Médica" por los agentes `AdministradorDeProceso` `General_HospitalMRM` y `Specialist_HospitalMRM` puede presentar dos diferentes comportamientos, determinado por el segmento de flujo de control XOR. En la figura 6.26 se muestra un posible escenario de la ejecución de este proceso. El proceso inicia con el agente `General_HospitalMRM` que genera y envía un mensaje `inform` notificando la transferencia del paciente al Hospital de Especialidades mediante el documento `PatientReferralLetter` (interacción 15). El agente `Specialist_HospitalMRM` evalúa y procesa el informe, respondiendo con un mensaje `inform`, con un documento `PatientReferralReceipt` que representa un acuse de recibo (interacción 16). Cuando se recibe al paciente en las instalaciones del SCP se genera un segundo mensaje `inform`, notificando la admisión del paciente mediante un documento `PatientReception` (interacción 17).

A continuación, el Hospital de Especialidades (rol SCP) inicia el tratamiento médico requerido para el paciente. Al finalizar este procedimiento, el agente `Specialist_HospitalMRM` genera el documento *Resumen Clínico/Médico* (`Medical Summary`), así como también un documento con la *Planificación de Alta de Paciente* (`Patient-DischargePlanning`), los cuales son enviados en forma paralela mediante mensajes `inform` al agente `General_HospitalMRM` (interacciones 18 y 19).

El agente `General_HospitalMRM` recibe ambos mensajes, almacenando el documento `Medical Summary` mediante una tarea interna. En el caso del documento `PatientDischargePlanning`, éste es evaluado mediante una tarea interna de usuario. Luego el agente `General_HospitalMRM` genera y envía un mensaje `confirm`, notificando que está de acuerdo con la planificación de alta de paciente (interacción 20). El agente `Specialist_HospitalMRM` recibe el mensaje con la confirmación, generando un documento con el *Alta de Paciente* (`DischargeNote`). Este documento es enviado mediante un mensaje `inform` (interacción 21). El agente `General_HospitalMRM` recibe el mensaje con el *Alta de Paciente*, lo almacena mediante una tarea interna y finaliza la ejecución del proceso de integración.

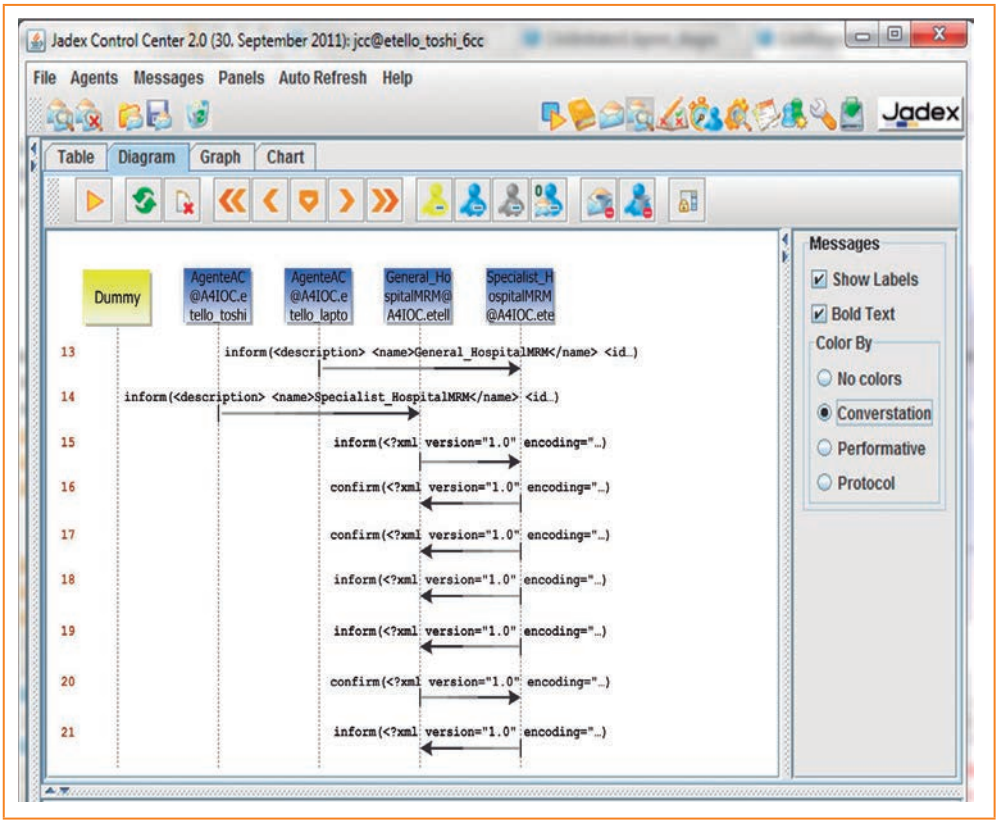



Figura 6.26. *Ejecución del proceso colaborativo* Gestión de Referencia Médica, *escenario-1*.

## 6.2. Colaboración interorganizacional dinámica en dominio de industria de telecomunicaciones

En esta sección se describe la implementación de una colaboración interorganizacional dinámica para un caso de estudio del dominio de la industria de las telecomunicaciones mediante la plataforma presentada en este libro.

El caso de estudio consiste en la instalación e implementación de la plataforma de agentes propuesta en las empresas MotoRepair y MotoParts, para que puedan establecer colaboraciones interorganizacionales dinámicas y ejecutar procesos colaborativos relacionados con la gestión de órdenes de compras y la provisión de productos de MotoParts a MotoRepair.

MotoRepair es el centro de reclamo de garantías de los productos fabricados por MotoBCS, en el cual se realiza la mayor cantidad de reparaciones de los productos MotoBCS a nivel mundial. Entre los productos fabricados por MotoBCS, de los cuales se atienden

solicitudes de garantía en MotoRepair, están los receptores de señal de televisión satelital, receptores digitales y de alta definición (HD) para sistemas de televisión por cable, módem para servicios de internet por cable y módem para servicios de telefonía por cable. MotoParts es una división de MotoBCS dedicada a la fabricación de componentes exclusivos para los productos fabricados por MotoBCS, los cuales son utilizados en los procesos de reparación en la empresa MotoRepair.

MotoRepair atiende solicitudes de garantía de los distribuidores de productos de MotoBCS, así como usuarios finales cuando se ha vencido el periodo de garantía del producto. MotoRepair tiene tres centros autorizados (empresas filiales) que brindan servicios de reparación de los productos de MotoBCS en países ubicados en el sur del continente americano. MotoRepair es el distribuidor oficial de los componentes de MotoParts utilizados en los procesos de reparación. Por ello, los centros autorizados de reparación de productos MotoBCS se convierten en clientes de MotoRepair.

El departamento de adquisiciones de MotoRepair realiza una estimación de los componentes que serán requeridos para reparar los diferentes modelos de productos MotoBCS con base en sus pronósticos de reparación y los pronósticos de reparación de los centros autorizados. El departamento de adquisiciones de componentes genera órdenes de compra globales con el objetivo de obtener mejores precios, con base en el volumen de compra, las cuales negocia con su contraparte MotoParts. A continuación se describe uno de los procesos colaborativos que las organizaciones han acordado ejecutar al establecer un acuerdo de colaboración en forma dinámica a través de la plataforma. Esto se realiza de la misma manera como se detalló en la sección 6.1.1 para el caso descrito allí. La diferencia con el caso anterior es que aquí el modelo de proceso colaborativo fue definido previamente por sólo una de las partes, en este caso MotoRepair. Y a través del establecimiento de una colaboración interorganizacional dinámica, MotoRepair le propone a MotoParts colaborar a través de la ejecución del proceso colaborativo "Gestión de Orden de Compra" (*Purchase Order Management*).

En este proceso colaborativo, cuyo comportamiento se muestra en el protocolo de interacción de la figura 6.27, la empresa MotoRepair desempeña el rol de cliente y MotoParts desempeña el rol de proveedor de componentes. El proceso colaborativo tiene como meta de negocio reducir los tiempos de la gestión de la adquisición de componentes y acelerar el proceso de compra en MotoRepair mediante la automatización de las decisiones de confirmación en forma electrónica por parte del proveedor. El proceso permite a las partes negociar los tiempos de entrega de los componentes y proponer cambios en la orden de compra.

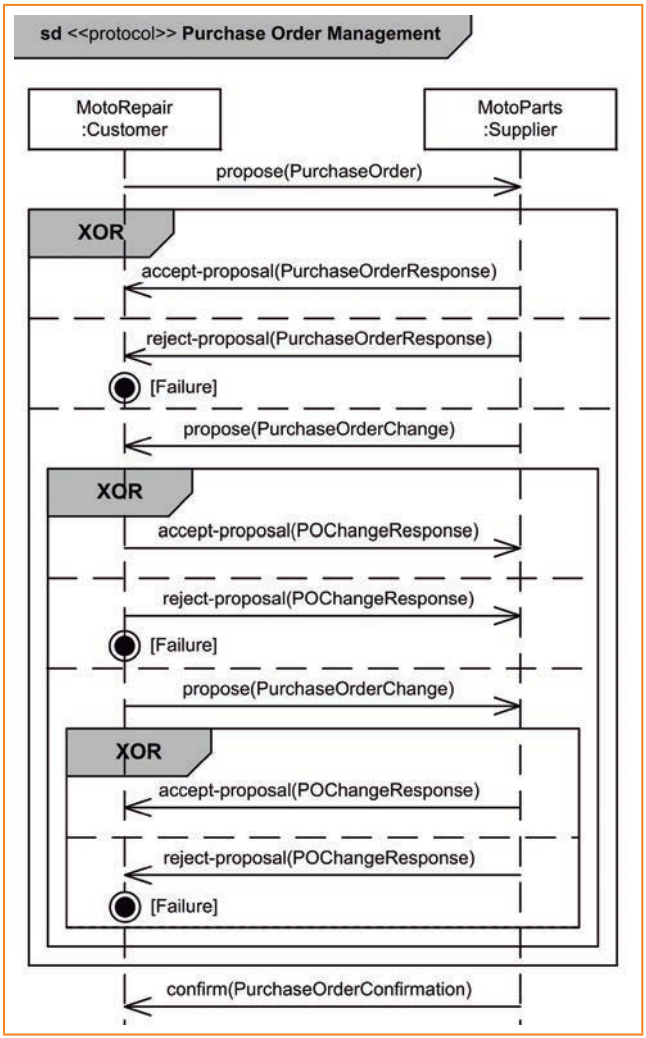



Figura 6.27. *Protocolo de interacción que representa el proceso colaborativo* Gestión de Orden de Compra.

### 6.2.1. Modelos de procesos de integración

En esta subsección se describe el comportamiento del proceso colaborativo "Gestión de Orden de Compra" en términos del comportamiento de los modelos de procesos de integración que se generan para cada una de las partes.

El proceso colaborativo inicia cuando el cliente, MotoRepair, inicia su proceso de integración. Aquí genera y propone una orden de compra a su proveedor MotoParts

(figura 6.28.A). El documento de negocio *PurchaseOrder* contiene los identificadores de partes requeridas, cantidades, fechas de entrega propuestas, total de entregas a realizar en la vigencia de la orden de compra, cantidad de partes por entrega.

Del lado del proceso del proveedor, MotoParts, éste recibe la propuesta del cliente, la evalúa, y puede responder aceptando, rechazando o proponiendo cambios a la orden de compra enviada por el cliente. Esto es representado mediante una compuerta exclusiva basada en datos con tres conectores de flujo de secuencia de salida (figura 6.28.B). En caso de aceptar la propuesta, el proveedor genera un documento de negocio con la confirmación de la orden de compra, el cual representa la aceptación de los términos de la orden de compra. Este documento es enviado mediante un mensaje *confirm* (tarea *Confirm PurchaseOrderConfirmation*).

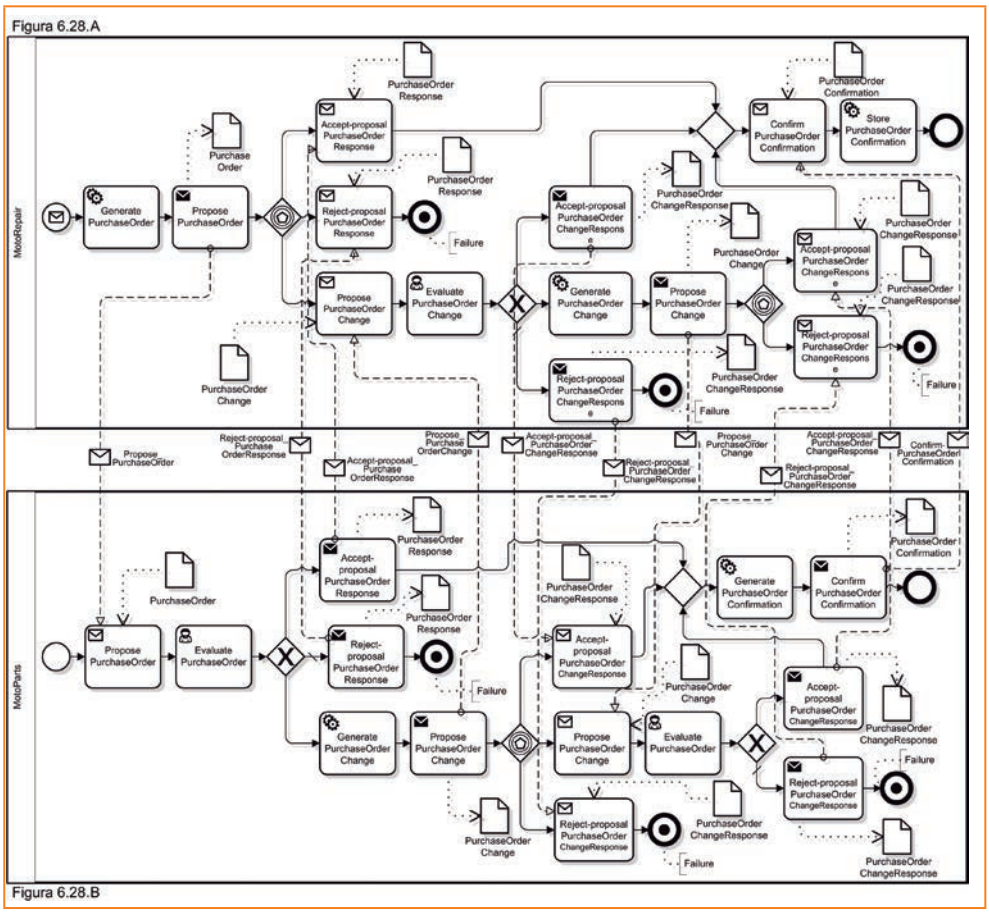



Figura 6.28. *Proceso de integración* Gestión de Orden de Compra: A, *Rol MotoRepair*, B, *Rol MotoParts*.

Cuando MotoRepair recibe la confirmación de aceptación de la orden, procesa y almacena la información contenida en el documento de negocio recibido. Entonces, el proceso "Gestión de Orden de Compra" finaliza y la empresa MotoRepair estará en espera de la recepción de las partes en los tiempos programados. En esta etapa otros departamentos, como Finanzas, son involucrados para iniciar los trámites del pago de la orden de compra en los términos pactados. Esto se realiza a través de otros procesos colaborativos.

Por otro lado, cuando MotoParts rechaza la orden de compra propuesta lo notifica al cliente mediante una tarea que envía un mensaje *reject-proposal* (figura 6.28.B). Esto se puede presentar cuando el proveedor no está en condiciones de cumplir con los términos de la propuesta en ninguno de los componentes requeridos, lo que se notifica en el documento de negocio contenido en el mensaje, terminando el proceso colaborativo a través de la finalización de los procesos de integración de ambas partes. Además, el proveedor puede responder con una contrapropuesta, en la cual se indican las condiciones en que puede cumplir la propuesta original y los cambios que se proponen. Estos cambios pueden estar relacionados con los plazos de entrega, las cantidades o los números de entregas. Para esto genera un documento de negocio (*Purchase Order Change*) y lo envía como propuesta al cliente.

Cuando el cliente recibe la propuesta, evalúa el documento de negocio con los cambios propuestos y puede responder mediante un mensaje *accept-proposal*, lo cual indica que se aceptan los cambios en la orden de compra y el proveedor debe generar el documento con la confirmación de la orden de compra y enviarlo como contenido de un mensaje *confirm* al cliente (figura 6.28.A). En caso que el cliente determine que la propuesta del proveedor no cumple sus requerimientos, puede responder mediante un mensaje *reject-proposal*, con lo cual se da por terminada la negociación y los procesos de integración de las partes finalizan.

El cliente también puede responder con un documento de negocio que contenga cambios en la contrapropuesta del proveedor, generando un documento con la nueva propuesta y enviándolo mediante un mensaje *propose*. Esta opción es utilizada cuando el rol cliente toma la decisión de adquirir sólo los componentes en los que el proveedor cumple con los términos previamente estipulados, y poniendo en espera la compra de los componentes en los que no se cumpla con sus requerimientos. Esta decisión puede estar soportada por la existencia que MotoRepair tiene de estos componentes en el inventario. También, debido a que se puede realizar una reprogramación de la fecha de inicio de la reparación de las unidades y que se tienen unidades disponibles en el almacén con las que se puede sustituir las unidades que requieren reparación.

Cuando MotoParts recibe el mensaje *propose* que contiene el documento de negocio con la nueva orden compra propuesta, evalúa la misma y puede decidir aceptar la propuesta respondiendo con un mensaje *accept-proposal* y generar un documento con la confirmación de la orden de compra, el cual es enviado como contenido de un mensaje *confirm*, con lo cual finaliza el proceso colaborativo satisfactoriamente, o rechazar la propuesta respondiendo con un mensaje *reject-proposal*, con lo cual también finaliza el proceso colaborativo (figura 6.28.B).

### 6.2.2. Ejecución del proceso colaborativo mediante la plataforma

En la ejecución del proceso colaborativo "Gestión de Orden de Compra" por parte de los agentes *AdministradorDeProceso* `MotoRepairPOM` y `MotoPartsPOM` de los respectivos MAS implementados con la plataforma, se pueden presentar seis escenarios con comportamientos diferentes, determinados por las negociaciones que se llevan a cabo en dicho proceso. A continuación se describen tres de estos escenarios.

La ejecución del proceso colaborativo "Gestión de Orden de Compra" involucra un agente *AdministradorDeProceso* `MotoRepairPOM` que desempeña el rol `Customer`, y al agente *AdministradorDeProceso* `MotoPartsPOM` que desempeña el rol `Supplier`.

En el escenario 1 (figura 6.29), el proceso inicia cuando el agente `MotoRepairPOM`, mediante el plan implementado, genera un documento de negocio `PurchaseOrder`, el cual se envía mediante un mensaje `propose` (interacción 19). Mediante este mensaje se plantea a la organización `MotoParts` una orden de compra, la cual contiene datos de los artículos requeridos y las políticas de compra.

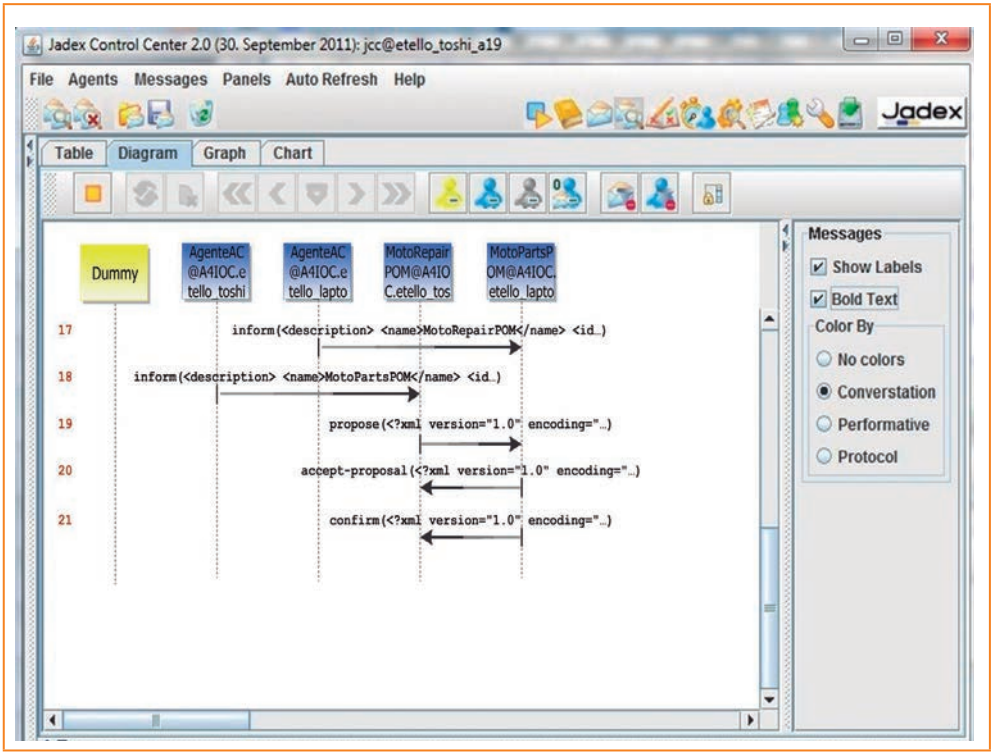



Figura 6.29. *Ejecución del proceso colaborativo* Gestión de Orden de Compra, *escenario-1.*

El agente `MotoPartsPOM` evalúa el documento de negocio `PurchaseOrder`, y responde con un mensaje `accept-proposal` (interacción 20), con lo cual se compromete a proveer los componentes requeridos en la orden. El agente `MotoPartsPOM` genera un documento de negocio `PurchaseOrderConfirmation`, con el cual el proveedor confirma la aceptación de la propuesta mediante un documento que tiene un folio de confirmación y detalla componentes requeridos, cantidades, tiempos estipulados y políticas de compra. Este documento es enviado mediante un mensaje `confirm` (interacción 21).

En la figura 6.30 se muestra otro escenario que se puede presentar en la ejecución del proceso colaborativo. En este caso, cuando el agente `MotoRepairPOM` propone una orden de compra mediante un mensaje `propose` (interacción 19), el agente `MotoPartsPOM` responde con una contrapropuesta de la orden de compra. Para ello genera un documento de negocio `PurchaseOrderChange` que contiene las condiciones en que puede cumplir con los requerimientos de compra. Este documento es enviado mediante un mensaje `propose` (interacción 20), con lo cual se inicia una nueva negociación entre los agentes.

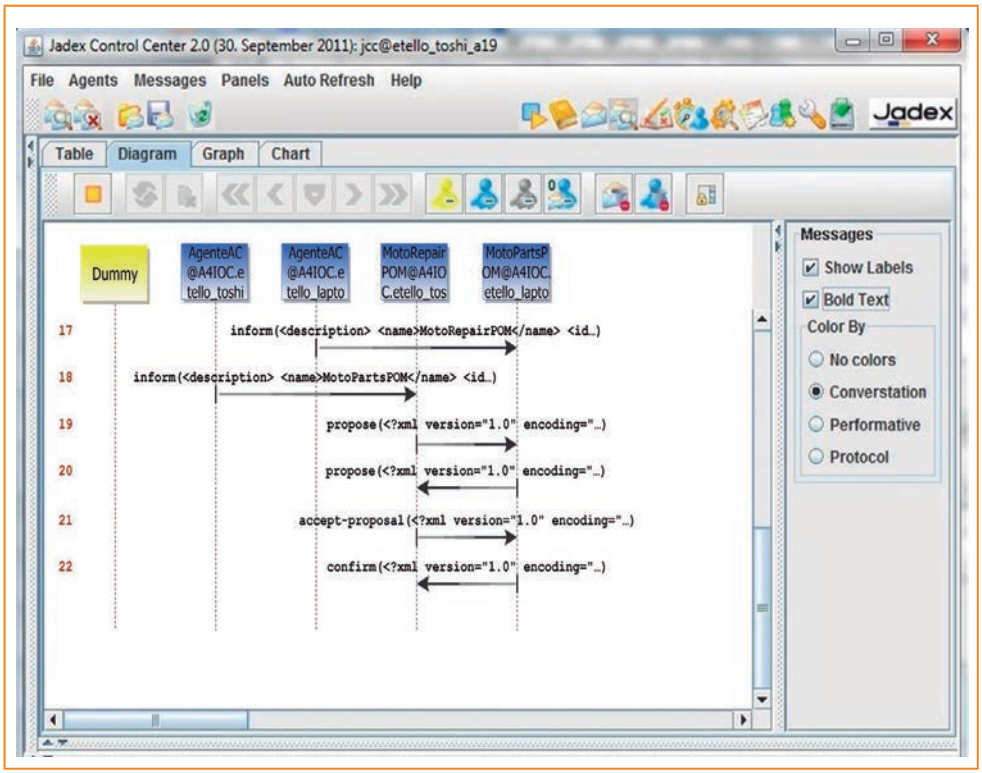



Figura 6.30. *Ejecución del proceso colaborativo* Gestión de Orden de Compra, *escenario-2*.

El agente `MotoRepairPOM` evalúa el documento propuesto y puede responder aceptando, rechazando y proponiendo cambios a la orden de compra recibida. En este caso, el agente `MotoRepairPOM` responde con un mensaje `accept-proposal` (interacción 21), con lo cual se aceptan los cambios propuestos en la orden de compra. Cuando el agente `MotoPartsPOM` recibe el mensaje de aceptación, lo procesa y genera un documento de negocio confirmando la aceptación de la orden de compra propuesta. Este documento de negocio `PurchaseOrderConfirmation` es enviado como contenido de un mensaje `confirm` (interacción 22), con lo cual la ejecución del proceso colaborativo finaliza.

Finalmente, en la figura 6.31 se muestra un tercer escenario del proceso colaborativo ejecutado entre agentes `MotoRepairPOM` y `MotoPartsPOM`, en donde la negociación de la propuesta de orden de compra finaliza de manera no satisfactoria. El agente `MotoRepairPOM` envía un mensaje `propose` que contiene el documento de negocio `PurchaseOrder` (interacción 19).

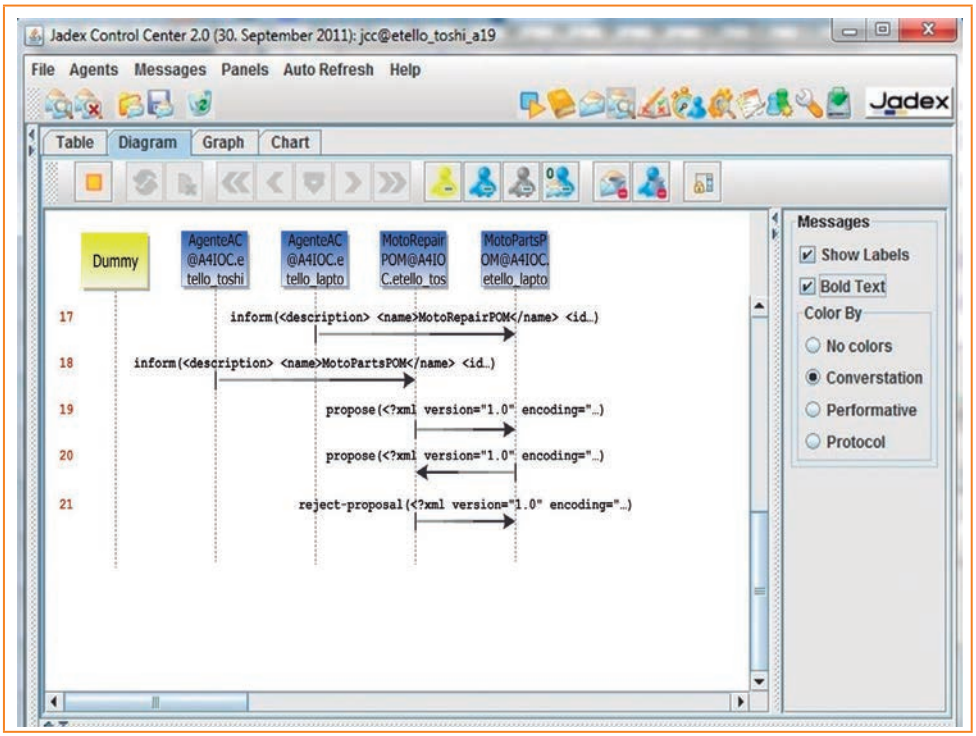



Figura 6.30. *Ejecución del proceso colaborativo* Gestión de Orden de Compra, *escenario-3.*

El agente `MotoPartsPOM` evalúa el documento de negocio y determina que no está en condiciones de cumplir con la propuesta. Por tal motivo, genera un documento de negocio que contiene una contrapropuesta. En este documento se describen las condiciones que el `Supplier` puede cumplir.

El documento de negocio `PurchaseOrderChange` generado por el agente `MotoPartsPOM` es enviado en un mensaje `propose` (interacción 20). Cuando el agente `MotoRepairPOM` recibe el mensaje con la contrapropuesta la evalúa y decide responder con un mensaje `reject-proposal` (interacción 21) mediante el cual se indica que no acepta las condiciones planteadas en el documento de negocio `PurchaseOrderChange`, finalizando la ejecución del proceso entre los agentes.

## 6.3. Análisis de resultados de los casos de estudio

En esta sección se analizan los resultados de la implementación de la plataforma A4IOC para la gestión de colaboraciones interorganizacionales dinámicas y la ejecución de procesos colaborativos en los casos de estudio presentados. En primer lugar se describen los resultados que de manera general ofrece la plataforma para ambos casos de estudio. Luego se detallan los resultados específicos para las organizaciones involucradas en cada uno de los casos de estudio.

Resultados de ambos casos de estudio:

- La implementación de la plataforma A4IOC ha posibilitado a las organizaciones negociar en forma electrónica el establecimiento de una colaboración interorganizacional dinámica. Esto les ha permitido disminuir considerablemente los tiempos que conlleva establecer un acuerdo de colaboración entre las organizaciones para ejecutar procesos colaborativos. Además, la plataforma les ha posibilitado que la negociación de los acuerdos entre las partes finalice correctamente, mediante procesos de negociación bien definidos.

- La generación en forma automática de la solución tecnológica (es decir, los artefactos de implementación) en tiempo de ejecución de la colaboración interorganizacional ha permitido a las organizaciones disminuir los tiempos y costos de desarrollo comparado con el desarrollo tradicional de sistemas, así como también ha permitido prevenir errores en la etapa de diseño y desarrollo de los artefactos de implementación. Adicionalmente, la solución tecnológica generada da soporte a los requerimientos de negocio definidos en los procesos colaborativos, garantizando la alineación de la solución interorganizacional con la solución tecnológica.

- La gestión descentralizada de los procesos mediante la plataforma A4IOC ha posibilitado que cada organización gobierne el rol que desempeña en los procesos colaborativos, facilitando el monitoreo y control de las actividades de la organización en la ejecución de dichos procesos, sin requerir de un control centralizado o de una tercera parte a nivel de las interacciones globales de un proceso colaborativo.

- La interoperabilidad de los procesos de integración y de los sistemas de las organizaciones que ejecutan los procesos colaborativos es garantizada por la plataforma a través de la solución tecnológica generada. Los agentes AP y sus modelos de procesos ejecutables permiten ejecutar los procesos colaborativos en forma descentralizada, manteniendo la sincronización en el envío y recepción de los mensajes de dichos procesos. Además, desde el punto de vista de la interoperabilidad técnica, los mensajes entre los agentes de las organizaciones son transportados a través de protocolos de internet, lo que ha posibilitado una comunicación transparente entre los sistemas multiagentes de las partes. De esta manera, las organizaciones no debieron preocuparse por garantizar este requerimiento de interoperabilidad.

- La plataforma A4IOC también les ha permitido a las organizaciones integrar sus sistemas internos para dar soporte a la ejecución de los procesos colaborativos, eliminando problemas de interoperabilidad entre dichos sistemas y los agentes AP que ejecutan los procesos.

En el caso específico de la colaboración interorganizacional dinámica implementada para dar soporte a los servicios electrónicos integrados y coordinados de atención a la salud entre el Hospital General y el Hospital de Especialidades, la plataforma les ha permitido a los hospitales realizar un intercambio eficiente de documentos clínicos en forma electrónica. Esto ha permitido a los médicos especialistas tomar mejores decisiones relacionadas con el tratamiento médico requerido por el paciente, con base en información confiable y con un mínimo de tiempo requerido para disponer de la información clínica del paciente. Además les ha posibilitado disminuir los tiempos que conlleva la transferencia y admisión de un paciente entre los hospitales. Los principales beneficios para los hospitales son:

- La plataforma A4IOC les ha permitido a los hospitales automatizar los procesos relacionados con la Gestión de Referencias de Pacientes mediante la generación de agentes AP que representan a los hospitales y ejecutan los procesos colaborativos.

- Las interacciones entre los sistemas (agentes) de los hospitales y el intercambio de los documentos clínicos requeridos para llevar a cabo un servicio de derivación de un paciente son realizados por la plataforma de acuerdo con el comportamiento de los procesos colaborativos definidos en los protocolos de interacción.

- La calidad del servicio integrado de atención a la salud que se ofrece entre el Hospital General y el Hospital de Especialidades puede ser evaluado a través del monitoreo y análisis de la ejecución de los procesos colaborativos realizados con la plataforma, con el propósito de realizar mejoras a los procesos.

- La ejecución del proceso colaborativo "Orden de Transferencia de Paciente" ha posibilitado a los hospitales a ejecutar negociaciones para acordar la derivación de un paciente. Esto ha permitido mejorar considerablemente la coordinación de las transferencias de pacientes entre los hospitales.

- La ejecución del proceso colaborativo "Gestión de Referencia Médica" les ha permitido a las organizaciones administrar en forma conjunta y coordinada la derivación de un paciente. Con ello las interacciones en forma paralela mejoran el rendimiento entre los hospitales.

- La rápida generación de los modelos de procesos de integración por los hospitales les ha permitido conocer las actividades públicas y privadas que estos hospitales tienen que implementar y la integración requerida con sus sistemas internos para llevar a cabo la ejecución de los procesos colaborativos.

- Finalmente, la plataforma les ha permitido a los hospitales adaptarse rápidamente a cambios en los modelos de procesos colaborativos acordados, con el propósito de hacer frente a nuevos requerimientos en la gestión de referencias médicas, sin requerir cambios en los sistemas que soportan la ejecución de los procesos colaborativos. La plataforma les permitió definir una nueva versión de un modelo de proceso colaborativo a través de un proceso de negociación y generar los nuevos modelos de procesos ejecutables y los nuevos agentes AP que soportan esta versión del proceso colaborativo y que remplazan la anterior versión. De esta manera, la adaptación a cambios pudieron realizarla en forma ágil, flexible y casi transparente debido a que no requirieron realizar tareas de desarrollo e implementación, disminuyendo también los costos y tiempos de desarrollo ante nuevas versiones de los procesos colaborativos acordados.

Con respecto al caso específico del dominio de la industria de las telecomunicaciones entre las empresas MotoRepair y Motoparts, la gestión de los procesos colaborativos a través de la plataforma A4IOC ha permitido un proceso de negociación en forma electrónica que habilita a MotoRepair colocar órdenes de compra a MotoParts, acelerando el proceso de adquisición de productos y reduciendo los tiempos de gestión de la provisión de componentes entre las empresas. Los principales beneficios de la gestión de la colaboración interorganizacional para estas empresas son:

- La plataforma A4IOC ha posibilitado a MotoRepair proponer y negociar una colaboración interorganizacional dinámica con la empresa MotoParts, proponiendo a esta última la ejecución del proceso colaborativo "Gestión de Orden de Compra".

- El proceso colaborativo "Gestión de Orden de Compra" permite negociar las condiciones de la orden de compra en forma electrónica, lo que posibilita acelerar el proceso de adquisición de los componentes electrónicos. Además de incluir el comportamiento requerido para habilitar a las partes a proponer cambios en las órdenes de compra mediante procesos de negociación bien definidos.

- La ejecución del proceso "Gestión de Orden de Compra" habilita la generación de la confirmación de la orden compra en forma electrónica, lo cual contribuye a reducir los tiempos que conlleva la adquisición de productos.

- La plataforma A4IOC ha permitido a las empresas automatizar la gestión de órdenes de compra a través de la derivación de procesos ejecutables y agentes AP que ejecutan el comportamiento definido en dicho proceso colaborativo.

# 7. Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones del trabajo de investigación (sección 7.1), destacando las principales contribuciones realizadas (sección 7.2) y las oportunidades de futuros trabajos de investigación identificados (sección 7.3).

## 7.1. Conclusiones

En el presente libro se propuso una solución tecnológica basada en agentes de software que permite dar soporte a la gestión de procesos colaborativos en ambientes de colaboración interorganizacionales dinámicos. Para lo cual, por un lado, se propuso una plataforma de agentes de software, la que habilita a las organizaciones a negociar acuerdos de colaboración en forma electrónica acerca de los procesos colaborativos a ejecutar o nuevas versiones a ejecutar de dichos procesos; ejecutar en forma descentralizada los procesos colaborativos acordados, y generar los artefactos de implementación que permiten ejecutar los procesos colaborativos.

Por otro lado, se propuso una metodología de desarrollo basada en los principios del desarrollo dirigido por modelos, que posibilita generar los artefactos de implementación (modelo de proceso ejecutable y código de agentes orientados a procesos de las organizaciones) a partir de modelos conceptuales de procesos colaborativos y de integración. Esta metodología y sus métodos son implementados y automatizados por agentes de software provistos en la plataforma, los cuales habilitan la generación de dichos artefactos en tiempo de ejecución de la plataforma. Posibilitando de esta manera que la plataforma soporte el aspecto dinámico de las colaboraciones interorganizacionales, permitiendo que los sistemas de las organizaciones se adapten rápidamente en tiempo de ejecución para gestionar nuevos procesos colaborativos o nuevas versiones de dichos procesos.

La generación automática, en tiempo de ejecución de la plataforma, de los artefactos de implementación requeridos por las organizaciones para dar soporte a la ejecución de procesos colaborativos conlleva varios beneficios:

- Disminución de los tiempos y costos de desarrollo, comparado con el desarrollo tradicional de sistemas.

- Permite prevenir errores en la etapa de diseño y de desarrollo de los artefactos de implementación.

- Posibilita que los sistemas (agentes) de las organizaciones funcionen de acuerdo con lo especificado en los modelos conceptuales de procesos colaborativos definidos en

un nivel organizacional, garantizando la alineación de la solución interorganizacional con la solución tecnológica.

- Posibilita que los sistemas de las organizaciones sean flexibles y se adapten rápidamente a cambios en los requerimientos organizacionales o de negocio de las colaboraciones interorganizacionales, que implican ejecutar nuevos procesos colaborativos o nuevas versiones de los procesos acordados.

La solución tecnológica propuesta es consistente con el enfoque de integración entre organizaciones orientado a procesos, en donde la colaboración (incluyendo la coordinación de actividades para la toma de decisiones en forma conjunta y el intercambio de información entre organizaciones) es definida, ejecutada y gestionada en términos de procesos colaborativos.

## 7.2. Principales contribuciones

En este libro se propuso una plataforma flexible, adaptable y configurable basada en agentes de software para la gestión de colaboraciones interorganizacionales dinámicas y de los procesos colaborativos que son acordados en las mismas. En la plataforma se propusieron y definieron diferentes tipos de agentes, los cuales por un lado se clasifican en agentes estáticos y dinámicos, y por otro lado en agentes interorganizacionales y agentes intraorganizacionales.

Los agentes estáticos son aquellos cuyo comportamiento es definido en tiempo de diseño y la implementación de los mismos es provista en la plataforma. A través de los agentes estáticos interorganizacionales (agentes *AdministradorDeColaboraciones*) y de sus mecanismos de interacción predefinidos, las organizaciones negocian acuerdos de colaboración en forma electrónica para la implementación y ejecución de procesos colaborativos, intercambiando los modelos de dichos procesos. Estos agentes posibilitan que la negociación de los acuerdos entre las partes finalice correctamente, mediante la realización de los protocolos de interacción propuestos que proveen procesos de negociación bien definidos. Se proveen además agentes estáticos para recuperar modelos de procesos desde repositorios locales y gestionar los mismos en la plataforma.

Los agentes dinámicos son aquellos en los cuales tanto su implementación como sus instancias son generadas en tiempo de ejecución. Éstos tienen como propósito dar soporte a la gestión de los procesos colaborativos que se acuerdan en forma electrónica y se ejecutan durante el transcurso de la colaboración. Por lo tanto, estos agentes no están predefinidos en la plataforma. La implementación con la estructura y el comportamiento de estos agentes (llamados agentes *AdministradorDeProceso*) se generan mediante métodos de transformación de modelos. La generación en forma automáti-

ca del comportamiento de estos agentes dinámicos permite la ejecución de diferentes instancias de un proceso colaborativo en la plataforma, facilitando el reúso de estos componentes de software.

La ejecución descentralizada de los procesos colaborativos es posibilitada a través de los agentes *AdministradorDeProceso* que realizan la ejecución distribuida y sincronizada de los procesos de integración de las organizaciones. Estos agentes dinámicos son orientados a procesos. Los planes, acciones e interacciones de estos agentes son gobernados por un motor de procesos que interpreta un modelo de proceso ejecutable (el cual representa la implementación de un proceso de integración), siguiendo los conceptos de los sistemas de información orientados a procesos. El motor ejecuta una instancia de un modelo de proceso ejecutable (proceso de integración) que es definido como una meta del agente.

A través de los agentes orientados a procesos dinámicos (agentes *AdministradorDeProceso*) la plataforma de agentes da soporte a la gestión y ejecución descentralizada de procesos colaborativos, manteniendo la autonomía de las organizaciones, las cuales tienen la capacidad de decidir cuándo, cómo y con quién se integran y cooperan al mismo tiempo que ocultan sus actividades y decisiones internas.

De esta manera, el trabajo de investigación introduce aspectos novedosos, tanto en la gestión de procesos colaborativos en ambientes de colaboraciones interorganizacionales dinámicas, como en el desarrollo de sistemas de agentes de software.

Además, la plataforma propuesta, a través de los agentes estáticos y dinámicos que la conforman, provee características de interoperabilidad e integración de sistemas, tales como comunicación, coordinación, interacciones *peer-to-peer*, cooperación, autonomía y descentralización; las cuales son funcionalidades requeridas por los sistemas de información interorganizacionales para implementar y ejecutar procesos colaborativos en ambientes de colaboración interorganizacionales dinámicos.

Otra contribución importante en el enfoque presentado en este libro es la metodología de desarrollo de soluciones tecnológicas basadas en agentes para ejecutar procesos colaborativos. Esta metodología, la cual aplica los principios del desarrollo dirigido por modelos, posibilita satisfacer un requerimiento importante en los entornos de colaboración interorganizacionales, el cual refiere a proporcionar métodos y herramientas que soporten la transición desde un nivel organizacional a un nivel tecnológico. Esto presenta beneficios sustanciales al garantizar que la solución tecnológica está alineada y cumple efectivamente con los requerimientos y comportamiento de la colaboración definidos en los modelos de procesos colaborativos que representan la solución interorganizacional. Además, la metodología permite la disminución de los tiempos del desarrollo y la reducción de errores en la programación.

Mediante la metodología propuesta se ofrece una guía explícita del proceso de desarrollo a seguir para generar los agentes orientados a procesos dinámicos, que integran la plataforma tecnológica, a partir de modelos de procesos colaborativos que las organizaciones acuerdan al establecer una colaboración interorganizacional. Como parte de

esta metodología se propusieron métodos de transformaciones de modelos que posibilitan generar modelos de procesos ejecutables y el código de los agentes dinámicos que soportan la ejecución de los procesos colaborativos.

Un método de transformación de modelos propuesto da soporte a la generación de un modelo de proceso ejecutable, que es derivado de un modelo de proceso de integración independiente de la plataforma expresado en el lenguaje BPMN. Este método contiene las reglas de transformación necesarias para la anotación de los elementos del modelo de proceso ejecutable con la semántica de ejecución requerida en la plataforma de implementación, lo cual habilita su implementación y ejecución a través del motor de procesos embebido en el agente *AdministradorDeProceso*.

Para la generación del agente *AdministradorDeProceso* se han propuesto métodos de transformaciones de modelos que generan el código XML del agente, para que el mismo pueda ser implementado y ejecutado en la plataforma propuesta. Un primer método genera un modelo del agente que contiene la lógica interna del agente (metas, planes y eventos) basado en el enfoque de agentes BDI (*Belief-Desire-Intention*), a partir de un modelo conceptual de un proceso de integración. Un segundo método de transformación que utiliza al modelo del agente derivado previamente permite la generación de un documento con el código XML que define un agente *AdministradorDeProceso*, así como también el comportamiento de este agente formado por metas, planes, eventos y elementos adicionales de configuración, habilitando su ejecución en la plataforma.

Un aporte adicional referente a la metodología es la flexibilidad para ser aplicada en ambientes de colaboración interorganizacionales tanto estáticos como dinámicos.

Por otra parte, otro aspecto importante y un aporte significativo de la investigación es la generación en forma automática de la solución tecnológica, esto es, los artefactos de implementación (modelos de procesos ejecutables y código de agentes *AdministradorDeProceso* de las organizaciones), en tiempo de ejecución de la colaboración interorganizacional. Esto es llevado a cabo mediante la implementación, automatización e integración de la metodología y de sus métodos en la plataforma de agentes a través del tipo de agente *GeneradorDeImplementaciones*. El comportamiento en este agente permite generar las transformaciones de modelos y verificar la consistencia de los modelos y código generado en tiempo de ejecución de la plataforma. Esto permite dar soporte en forma completa a los aspectos dinámicos de las colaboraciones interorganizacionales.

La plataforma de agentes propuesta se implementó con el *framework* y plataforma de implementación de agentes Jadex. En particular se utilizaron los componentes de agentes BDI y BPMN *Kernel* de Jadex, este último para implementar los motores de procesos de los agentes *AdministradorDeProceso*. La plataforma desarrollada e implementada a nivel de prototipo, la cual fue utilizada en los casos de estudio presentados, permitió demostrar la utilidad, factibilidad y beneficios de usar la plataforma propuesta en colaboraciones interorganizacionales. Además, el prototipo de la plataforma puede ser utilizado para realizar simulaciones sobre escenarios configurables correspondientes a diferentes dominios de las colaboraciones interorganizacionales.

En síntesis, los principales aportes de este trabajo de investigación son:

1. Una plataforma basada en agentes de software que les permite a las organizaciones la gestión de colaboraciones interorganizacionales y la ejecución de los procesos colaborativos.

    - Se proponen agentes estáticos (definidos en tiempo de diseño y configurados en la plataforma) que permiten a las organizaciones entablar colaboraciones interorganizacionales en forma electrónica para acordar los procesos colaborativos a ejecutar.

    - Se definieron protocolos de interacción entre agentes estáticos que explicitan el proceso de negociación entre las partes garantizando que el mismo finalice en forma correcta.

    - Se aplicó la metodología Tropos al dominio de los sistemas de información interorganizacionales, describiendo cómo la misma fue utilizada para definir la arquitectura de la plataforma y de los agentes que la componen.

    - Se aplicó la teoría de agentes BDI al dominio de sistemas multiagentes interorganizacionales.

    - Se propone un nuevo tipo de agente de software dinámico para ejecutar los procesos colaborativos, cuya estructura y comportamiento se genera en forma automática en tiempo de ejecución de la plataforma a partir de modelos de procesos colaborativos.

    - Se proponen agentes orientados a procesos dinámicos para la gestión de los procesos colaborativos, en donde el comportamiento de los mismos es gobernado por un motor de procesos que a su vez interpreta un modelo de proceso de negocio que debe ejecutar el agente. Se integran en estos agentes los conceptos de los agentes BDI con el de los sistemas de información orientados a procesos.

    - Se implementó un prototipo de la plataforma propuesta a través del *framework* y plataforma de agentes Jadex.

    - Se propuso cómo automatizar métodos MDD dentro de la plataforma a través de agentes de software.

2. Metodología y métodos MDD guían y posibilitan el proceso de desarrollo para la generación de los artefactos de implementación requeridos por las organizaciones para la ejecución de los procesos colaborativos que acuerdan ejecutar.

    - A través de la metodología se define cómo es posible generar los artefactos de implementación de los agentes a partir de modelos conceptuales de procesos colaborativos.

    - Se definió un método de transformación modelo-a-modelo para generar modelos de procesos ejecutables, basados en Jadex-Processes, a partir de modelos conceptuales de procesos de integración basados en BPMN.

    - Se definió un método de transformación modelo-a-modelo que genera el modelo de un agente orientado a procesos dinámico, basado en los conceptos de agentes BDI de Jadex, a partir de un modelo conceptual de proceso de integración basado en BPMN.

    - Se definió un método de transformación modelo-a-código que genera el archivo XML de un agente orientado a proceso dinámico, el cual define en forma declarativa la estructura y comportamiento de un agente BDI de Jadex.

    - Se formalizaron e implementaron los métodos de transformaciones de modelos usando el lenguaje y herramienta ATL.

    - Se generaron los motores ATL que contienen estas transformaciones implementadas y se integraron los mismos a la plataforma a través del agente *GeneradorDeImplementaciones*.

3. Se demostró la utilidad, la factibilidad y los beneficios que aporta la plataforma a través del desarrollo de casos de estudio representativos de diferentes dominios de las colaboraciones interorganizacionales.

Finalmente, a través de los aportes del trabajo de investigación se ha contribuido a diferentes áreas de la gestión de procesos de negocio en entornos interorganizacionales, la ingeniería de sistemas de información y la ingeniería de software.

En el área de tecnología de agentes de software se contribuye con una nueva manera de estructurar el comportamiento de los agentes BDI en términos de procesos, integrando la teoría de agentes BDI (creencias, metas y planes) con los conceptos de los sistemas de información orientados a procesos (modelos de procesos). También, en el área de agentes de software, se propone un nuevo tipo de agente, denominado agente dinámico, cuyo comportamiento no está predefinido y es generado en tiempo de ejecución del sistema a partir de modelos de alto nivel de abstracción, como lo son los modelos de procesos colaborativos.

En el área del desarrollo dirigido por modelos se muestra la factibilidad de automatizar métodos para generar un sistema con la mínima intervención del diseñador de sistemas, permitiendo generar una solución tecnológica que da soporte a los requerimientos organizacionales o de negocio definidos en los procesos colaborativos, garantizando la alineación de la solución interorganizacional con la solución tecnológica.

En el área de sistemas de información orientados a procesos y sistemas de gestión de procesos de negocio, a través del uso de agentes de software BDI con motores de procesos embebidos, se propone un nuevo tipo de sistema de información orientado a procesos que hace factible la gestión distribuida y descentralizada de procesos colaborativos.

## 7.3. Trabajos futuros

La plataforma propuesta se proyecta integrar con un sistema de gestión de repositorios orientado a servicios para la búsqueda de modelos de procesos colaborativos desde un agente *AdministradorDeModelos*, mejorando las funcionalidades de descubrimiento y búsqueda de modelos de procesos colaborativos almacenados en repositorios públicos o globales. Esto puede ofrecer grandes ventajas a las organizaciones a la hora de entablar colaboraciones interorganizacionales dinámicas y permitir el reúso de modelos de procesos existentes en forma flexible.

Una limitación de la plataforma de agentes propuesta se refiere al soporte ofrecido para llevar adelante las negociaciones que permiten establecer las colaboraciones interorganizacionales y acordar los procesos colaborativos a ejecutar. En la plataforma se ha definido para ello un protocolo de interacción con una extensión al mismo. Las actividades internas de los agentes *AdminsitradorDeColaboraciones* requeridas para dicho protocolo tienen implementados algoritmos de razonamiento básicos para determinar si una organización acuerda o no ejecutar ciertos procesos colaborativos. No obstante, se requieren de mecanismos más avanzados para dar soporte a estas negociaciones. Para resolver esta limitación se plantean dos trabajos futuros.

Por un lado, en el agente *AdministradorDeColaboraciones* se planea incorporar un algoritmo basado en reglas que permita mejorar los mecanismos de razonamiento del agente para determinar la aceptación o rechazo de una solicitud de acuerdo de colaboración. En el proceso de negociación del acuerdo puede influir una diversidad de variables, tales como la colaboración propuesta, las metas de negocio de la colaboración propuesta, las metas de negocio de las organizaciones involucradas, los procesos colaborativos que se proponen, así como también se debe considerar la experiencia previa de colaboración con una organización, los resultados alcanzados con la ejecución de procesos colaborativos similares y las actividades internas involucradas en dichos procesos. Esta información puede estar almacenada en una ontología que represente el conocimiento del agente y que se pueda acceder a la ontología mediante las creencias del agente para ejecutar las reglas de razonamiento.

Por otro lado, otro trabajo que se proyecta realizar es la formalización del documento del acuerdo de colaboración que los agentes *AdministradorDeColaboraciones* utilizan para solicitar y establecer colaboraciones. El acuerdo de colaboración puede ser formalizado con la implementación de acuerdos a nivel de servicio (*Service Level Agreement*, SLA) (53, 79), en el cual se reflejen las necesidades de las organizaciones involucradas en la colaboración interorganizacional. En SLA se especifica un contrato con las obligaciones y responsabilidades de las partes. También se deben incluir aspectos de los servicios requeridos, medición del rendimiento, gestión de problemas, garantías y finalización del acuerdo. Además se requeriría incorporar un agente de software a la plataforma con la responsabilidad de gestionar los SLA implementados en las colaboraciones interorganizacionales dinámicas existentes en cada organización.

# Referencias


(1) Aalst, W. van der, Hofstede, A. T., y Weske, M. (2003). "Business process management: A survey". En *Proceedings of the 1st International Conference on Business Process Management, 2678*, 1-12.

(2) Aalst, W. van der, y Stahl, C. (2011). *Modeling Business Processes: A Petri Net-Oriented Approach.* Massachusetts: The MIT Press.

(3) Acceleo (2012). *Transforming models into code. Specification release 3.3.0, Object Management Group (OMG) MOF Model to Text Language (MTL)*. Disponible en http://www.eclipse.org/acceleo/.

(4) Allweyer, T. (2010). *BPMN 2.0 Introduction to the Standard for Business Process Modeling,* (2ª ed.). Alemania: BoD–Books on Demand GmbH.

(5) ATL (2011). *ATL-ATLAS transformation language. Specification release 3.2.0, OBEO and AtlanMod*. Disponible en http://www.eclipse.org/atl/.

(6) Aversano, L., Grasso, C., y Tortorella, M. (2009). "A framework for measuring the alignment between business processes and software systems". En D. Slezak, T. H. Kim, A. Kiumi, T. Jiang, J. Verner, S. Abraho (eds.). *Advances in Software Engineering, 59* (pp. 213-220).

(7) Aversano, L., Grasso, C., y Tortorella, M. (2011). "ALBIS-Aligning Business Processes and Information Systems: A Case Study". En M. Cruz-Cunha, J. Varajo, P. Powell, R. Martinho (eds.). *ENTERprise Information Systems, 220* (pp. 286-296).

(8) Bauer, B., Müller, J. P., y Odell, J. (2001). "Agent UML: A formalism for specifying multiagent software systems". En P. Ciancarini, M. J. Wooldridge (eds.). *Agent-Oriented Software Engineering. Lecture Notes in Computer Science,* vol. 1957 (pp. 91-103). Berlín-Heidelberg: Springer.

(9) Bauer, B., Müller, J. P., Roser, S. (2007). "A decentralized broker architecture for collaborative business process modelling and enactment". En G. Doumeingts, J. Müller, G. Morel, B. Vallespir (eds.). *Enterprise Interoperability,* pp. 115-125. Londres: Springer.

(10) Bauer, B., Roser, S., y Müller, J. P. (2005). "Adaptive design of cross-organizational business processes using a model-driven architecture". En O. K. Ferstl, E. J. Sinz, S. Eckert, T. Isselhorst (eds.). *Wirtschaftsinformatik* (pp. 103-121). Alemania: Springer-Physica.

(11) Bellifemine, F. L., Caire, G., y Greenwood, D. (2007). *Developing multi-agent systems with JADE.* Inglaterra: Wiley.

(12) Bouchbout, K., y Alimazighi, Z. (2011). "Inter-organizational business processes modelling framework". En *Proceedings of the 15th East-European Conference on Advances in Databases and Information Systems (ADBIS)* (pp. 45-54).

(13) Bratman, M. E. (1987). *Intention, Plans and Practical Reason: Center for the Study of Language and Information.* Inglaterra: Cambridge University Press.

(14) Braubach, L., Pokahr, A., y Lamersdorf, W. (2005). "Jadex: A BDI-agent system combining middleware and reasoning". En R. Unland, M. Calisti, M. Klusch (eds.). *Software Agent-Based Applications, Platforms and Development Kits, Whitestein. Series in Software Agent Technologies* (pp. 143-168). Basilea: Birkhäuser.

(15) Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., y Mylopoulos, J. (2004). "Tropos: An Agent-Oriented Software Development Methodology". *Autonomous Agents and Multi-Agent Systems, 8*(3) (pp. 203-236).

(16) Brown, A. W., Conallen, J., y Tropeano, D. (2005). "Introduction: Models, modeling, and model-driven architecture (MDA)". En S. Beydeda, M. Book, V. Gruhn (eds.). *Model-Driven Software Developmen* (pp. 1-16). Berlín-Heidelberg: Springer.

(17) Buhler, P. A., y Vidal, J. M. (2005). "Towards adaptive workflow enactment using multiagent systems". *Information Technology and Management, 6*(1) (pp. 61-87).

(18) Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., y Massonet, P. (2002). "Agent oriented analysis using Message/UML". En M. J. Wooldridge, G. Wei, P. Ciancarini (eds.). *Agent-Oriented Software Engineering II. Lecture Notes in Computer Science,* vol. 2222 (pp. 119-135). Berlín-Heidelberg: Springer.

(19) Camarinha-Matos, L. M., Afsarmanesh, H., Galeano, N., y Molina (2009). "Collaborative networked organizations-concepts and practice in manufacturing enterprises". *Computers & Industrial Engineering, 57*(1) (pp. 46-60).

(20) Chituc, C. M., Azevedo, A., y Toscano, C. (2009). "A framework proposal for seamless interoperability in a collaborative networked environment". *Computers in Industry*, *60*(5) (pp. 317-338).

(21) DeLoach (2004). "Methodologies and Software Engineering for Agent Systems". *The MaSE Methodology* (pp. 107-125). Boston: Kluwer Academic Publishers.

(22) Dorn, J., Grün, C., Werthner, H., y Zapletal, M. (2009). "From business to software: a b2b survey". *Information Systems and e-Business Management,* 7 (pp. 123-142).

(23) Dumas, M., Aalst, W. van der, y Hofstede, A. T. (2005). *Process-Aware Information Systems: Bridging People and Software through Process Technology.* Nueva Jersey: John Wiley & Sons, Inc.

(24) Dumas, M., Rosa, M. L., Mendling, J., y Reijers, H. A. (2012). *Fundamentals of Business Process Management.* Nueva York: Springer.

(25) Eclipse (2012). *Eclipse modeling project.* Diponible en http://www.eclipse.org/modeling/. (Consulta: 2012, 5 de febrero).

(26) Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., Mckay, D., McGuire, J., Pelavin, R., Shapiro, S., y Beck, C. (1993). *DRAFT specification of the KQML Agent-Communication Language. Specification.* DARPA Knowledge Sharing Initiative External Interfaces Working Group. Disponible en http://www.csee.umbc.edu/csee/research/kqml/kqmlspec/spec.html.

(27) FIPA (2002). Agent Communication specifications deal with Agent Communication Language (ACL). Specification SC00061, Foundation for Intelligent Physical Agents (FIPA). Disponible en http://www.fipa.org/repository/aclspecs.html.

(28) Frankel, D. S. (2003). *Model Driven Architecture-Applying MDA to Enterprise Computing.* Estados Unidos: Wiley.

(29) Girault, C., y Valk, R. (2002). *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications.* Alemania: Springer.

(30) Grefen, P., Mehandjiev, N., Kouvas, G., Weichhart, G., y Eshuis, R. (2009). "Dynamic business network process management in instant virtual enterprises". *Computers in Industry, 60*(2) (pp. 86-103).

(31) Hahn, C. (2008). "A domain specific modeling language for multiagent systems". En *Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (pp. 233-240).

(32) Hahn, C., Madrigal-Mora, C., y Fischer, K. (2007). "Interoperability through a platform independent model for agents". En R. J. Goncalves, J. P. Müller, K. Mertins y M. Zelm (eds.). *Enterprise Interoperability,* vol. 2 (pp. 195-206). Londres: Springer.

(33) Hahn, C., Zinnikus, I., Warwas, S., y Fischer, K. (2011). "Automatic generation of executable behavior: A protocol-driven approach". En M. P. Gleizes, J. J. Gomez-Sanz (eds.). *Agent-Oriented Software Engineering X. Lecture Notes in Computer Science,* vol. 6038 (pp. 110-124). Alemania: Springer.

(34) Henderson-Sellers, B., y Giorgini, P. (2005). *Agent-Oriented Methodologies.* Estados Unidos: Idea Group Publishing.

(35) Hirsch, B., Konnerth, T., y Heler, A. (2009). "Merging agents and services-The JIAC Agent Platform". En A. El Fallah Seghrouchni, J. Dix, M. Dastani, R. H. Bordini (eds.). *Multi-Agent Programming* (pp. 159-185). Estados Unidos: Springer.

(36) Hofreiter, B., y Huemer, C. (2010). "Flexible workflow management in service oriented environments". En N. Griffiths, K. M. Chao (eds.). *Agent-Based Service-Oriented Computing, Advanced Information and Knowledge Processing* (pp. 81-111). Londres: Springer.

(37) Howden, N., Rönnquist, R., Hodgson, A., y Lucas, A. (2001). "JACK intelligent agents-summary of an agent infrastructure". En Proceedings of the 5th ACM International conference on autonomous agents.

(38) Iglesias, C. A., y Garijo, M. (2005). "Agent-Oriented Methodologies". *The Agent-Oriented Methodology MAS-CommonKADS* (pp. 46-78). Pensilvania: Idea Group Publishing.

(39) Jennings, N. R. (2001). "An agent-based approach for building complex software systems". *Communications of the ACM, 44*(4) (pp. 35-41).

(40) JET (2011). *JET-Java Emitter Templates, Model to Text Transformation. Specification release 1.1.1, Eclipse Modeling Project.* Disponible en http://www.eclipse.org/modeling/m2t/?project=jet.

(41) Jiang, P., Shao, X., Gao, L., y Yang, Z. (2007). "A multi-agent system for cross-organizational workflows management based on process-view". En Y. Shi, G. Albada, J. Dongarra, P. M. Sloot (eds.). *Computational Science-ICCS 2007. Lecture Notes in Computer Science,* vol. 4489 (pp. 212-215). Berlín-Heidelberg: Springer.

(42) Jouault, F., Allilaire, F., Bézivin, J., y Kurtev, I. (2008). "ATL: A model transformation tool". *Science of Computer Programming,* 72(1-2) (pp. 31-39).

(43) Kahl, T., Zinnikus, I., Roser, S., Hahn, C., Ziemann, J., Müller, J. P., y Fischer, K. (2007). "Architecture for the design and agent-based implementation of cross-organizational business processes". En R. J. Goncalves, J. P. Müller, K. Mertins, M. Zelm (eds.). *Enterprise Interoperability,* vol. II (pp. 207-218). Londres: Springer.

(44) Karaenke, P., Schuele, M., Micsik, A., y Kipp, A. (2012). "Inter-organizational interoperability through integration of multiagent, web service, and semantic web technologies". En K. Fischer, J. Müller, R. Levy (eds.). *Agent-Based Technologies and Applications for Enterprise Interoperability. Lecture Notes in Business Information Processing.* vol. 98 (pp. 55-75). Berlín-Heidelberg: Springer.

(45) Kleppe, A., Warmer, J., y Bast, W. (2003). *MDA Explained. The Model Driven Architecture: Practice and Promise.* Boston: Addison Wesley.

(46) Kouvas, G., Grefen, P., y Juan, A. (2010). "Business process enactment". En N. Mehandjiev, P. Grefen (eds.). *Dynamic Business Process Formation for Instant Virtual Enterprises. Advanced Information and Knowledge Processing* (pp. 113-132). Londres: Springer.

(47) Küster, T., Lützenberger, M., Heler, A., y Hirsch, B. (2012). "Integrating process modelling into multi-agent system engineering". *Multiagent and Grid Systems,* 8(1) (pp. 105-124).

(48) Lazarte, I. M., Chiotti, O., Villarreal, P. D., Thom, L. E., y Iochpe, C. (2011). "An MDA based method for designing integration process models in B2B collaborations". En *Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS 2011)* (pp. 55-65). Beijing: SciTePress.

(49) Lazarte, I. M., Tello-Leal, E., Roa, J., Chiotti, O., y Villarreal, P. D. (2010). "Model-driven development methodology for B2B collaborations". En *Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 2010)* (pp. 69-789). IEEE Computer Society.

(50) Li, Q., Zhou, J., Peng, Q. R., Li, C. Q., Wang, C., Wu, J., y Shao, B. E. (2010). "Business processes oriented heterogeneous systems integration platform for networked enterprises". *Computers in Industry*, *61*(2) (pp. 127-144).

(51) Lu, R., Sadiq, S. (2007). "A survey of comparative business process modeling approaches". En W. Abramowicz (ed.). *Business Information Systems. Lecture Notes in Computer Science*, vol. 4439 (pp. 82-94). Berlín-Heidelberg: Springer.

(52) Luo, J., Li, W., Liu, B., Zheng, X., y Dong, F. (2010). "Multi-agent coordination for service composition". En N. Griffiths K. M. Chao (eds.). *Agent-Based Service-Oriented Computing.* Advanced Information and Knowledge Processing (pp. 47-80). Londres: Springer.

(53) Mehandjiev, N., y Grefen, P. (2010). *Dynamic Business Process Formation for Instant Virtual Enterprises.* Londres: Springer.

(54) Meyer, N., Feiner, T., Radmayr, M., Blei, D., y Fleischmann, A. (2011). "Dynamic catenation and execution of cross organisational business processes-the jCPEX! Approach". En A. Fleischmann, W. Schmidt, R. Singer, D. Seese (eds.). *Subject-Oriented Business Process Management,* vol. 138, pp. 84-105. Berlín-Heidelberg: Springer.

(55) Müller, J. P., Bauer, B., Friese, T., Roser, S., y Zimmermann, R. (2006). "Software agents for electronic business: Opportunities and challenges (2005 re-mix)". En B. Chaib-draa y J. Müller (eds.). *Multiagent based Supply Chain Management*, vol. 28 (pp. 63-102). Berlín-Heidelberg: Springer.

(56) Object Management Group-OMG (OMG-BPMN) (2009). Business Process Model and Notation (BPMN) version 1.2. Specification formal/2009-01-03. Disponible en http://www.omg.org/spec/BPMN/1.2/PDF/.

(57) Object Management Group-OMG (OMG-BPMN) (2011). *Business Process Model and Notation (BPMN) version 2.0. Specification formal/2011-01-03.* Disponible en http://www.omg.org/spec/BPMN/2.0/PDF/.

(58) Object Management Group-OMG-CWM (2003). *Common Warehouse Metamodel (CWM) specification, v1.1. Specification formal/2003-03-02.* Disponible en http://www.omg.org/spec/CWM/1.1/.

(59) Object Management Group-OMG-MDA. (2003). *Model Driven Architecture-MDA guide version 1.0.1. Specification omg/2003-06-01.* Disponible en http://www.omg.org/cgi-bin/doc?omg/03-06-01.

(60) Object Management Group-OMG (2011). *Omg meta object facility (MOF) core specification. Specification formal/2011-08-07.* Disponible en http://www.omg.org/spec/MOF/2.4.1/.

(61) Object Management Group-OMG-MOFM2T (2008). *MOF Model to Text Transformation Language, v1.0. Specification formal/2008-01-16.* Disponible en http://www.omg.org/spec/MOFM2T/1.0/PDF.

(62) Object Management Group-OMG-QVT (2011). *Meta object facility (MOF) 2.0 query/view/transformation (QVT) specification version 1.1. Specification formal/2011-01-01.* Disponible en http://www.omg.org/spec/QVT/1.1.

(63) Object Management Group-OMG-UML (2011). *The unified modeling language-UML. Specification formal/2011-08-05.* Disponible en http://www.omg.org/spec/UML/2.4.1/.

(64) Padgham, L., y Winikoff, M. (2002). "Prometheus: A pragmatic methodology for engineering intelligent agents". En *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies* (pp. 97-108).

(65) Penserini, L., Perini, A., Susi, A., y Mylopoulos, J. (2007). *High variability design for software agents: Extending Tropos. ACM Transactions Autonomous and Adaptive Systems, 2*(4) (pp. 1-27).

(66) Pokahr, A., y Braubach, L. (2010). "The notions of application, spaces and agents-new concepts for constructing agent applications". En M. Schumann, L. Kolbe, M. Breitner, A. Frerichs (eds.). *Multi-agent Systems: Decentral approaches for designing, organizing, and operating information systems* (pp. 159-160). Gotinga: Universitätsverlag.

(67) Pokahr, A., Braubach, L., y Jander, K. (2010). "Unifying agent and component concepts". En J. Dix y C. Witteveen (eds.). *Multiagent System Technologies. Lecture Notes in Computer Science*, vol. 6251 (pp. 100-112). Berlín- Heidelberg: Springer.

(68) Pokahr, A., Braubach, L., y Lamersdorf, W. (2005). "Jadex: A BDI reasoning engine". En R. H. Bordini, M. Dastani, J. Dix, A. Fallah Seghrouchni (eds.). *Multi-Agent Programming*, *15,* pp. 149-174. Estados Unidos: Springer.

(69) Pons, C., Giandini, R., y Pérez, G. (2010). *Desarrollo de software dirigido por modelos: conceptos teóricos y su aplicación práctica* (1ª ed.). Argentina: Universidad Nacional de La Plata: McGraw-Hill.

(70) Poslad, S., Buckle, P., y Hadingham, R. (2000). "The FIPA-OS agent platform: Open source for open standards". En *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multiagents.*

(71) Rao, A. S., y Georgeff, M. P. (1995). "BDI agents: From theory to practice". En *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)* (pp. 312-319).

(72) Roser, S., y Bauer, B. (2005). "A categorization of collaborative business process modelling techniques". En *7th IEEE International Conference on E-Commerce Technology Workshops* (pp. 43-54). IEEE Computer Society.

(73) Searle, J. R. (1969). *Speech Acts.* Estados Unidos: Cambridge University Press.

(74) Shehory, O., Sturm, A. (2014). *Agent-Oriented Software Engineering.* Nueva York: Springer.

(75) Shen, W., Hao, Q., Wang, S., Li, Y., y Ghenniwa, H. (2007). "An agent-based service oriented integration architecture for collaborative intelligent manufacturing". *Robotics and Computer-Integrated Manufacturing*, *23*(3) (pp. 315-325).

(76) Shoham, Y. (1997). "An overview of agent-oriented programming". En J. M. Bradshaw (ed.). *Software agents* (pp. 271-290). Massachusetts: MIT Press.

(77) Stahl, T., y Völter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management.* Wiley.

(78) Steinberg, D. S., Budinsky, F., Paternostro, M. y Merks, E. (2009). *EMF Eclipse Modeling Framework* (2a ed.). Estados Unidos: Addison-Wesley.

(79) Stelmach, M., Kryza, B., Slota, R., y Kitowski, J. (2011). "Distributed contract negotiation system for virtual organizations". *Procedia Computer Science 4*(0) (pp. 2206-2215).

(80) Tello-Leal, E., Chiotti, O., y Villarreal, P. D. (2010). "An agent-based B2B collaboration platform for executing collaborative business processes". En W. Cellary y E. Estevez (eds.). *Software Services for e-World, IFIP Advances in Information and Communication Technology, 341* (pp. 40-50).

(81) Tello-Leal, E., Chiotti, O., y Villarreal, P. D. (2011). "Agents for Managing Business-to-Business Interactions". En *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)*. vol. II, *Agents* (pp. 238-244). Roma: SciTePress.

(82) Tello-Leal, E., Chiotti, O., y Villarreal, P. D. (2012). *Process-oriented integration and coordination of healthcare services across organizational boundaries. Journal of Medical Systems, 36*(6) (pp. 3713-3724).

(83) Tello-Leal, E., Chiotti, O., y Villarreal, P.D. (2014). "Software agent architecture for managing inter-organizational collaborations". *Journal of Applied Research and Technology, 12*(3) (pp. 514-526).

(84) Tello-Leal, E., Chiotti, O., y Villarreal, P. D. (2014). "Software agents for manage-

ment dynamic inter-organizational collaborations". *IEEE LatinAmerica Transactions*, *12*(2) (pp. 330-341).

(85) Thom, L. H., Reichert, M., y Iochpe, C. (2009). "Activity patterns in process-aware information systems: basic concepts and empirical evidence". *International Journal of Business Process Integration and Management (IJBPIM), 4(*2) (pp. 93-110).

(86) VICS (2011). *Collaborative planning, forecasting, and replenishment. Tech. rep., Voluntary guidelines, V 2.0.* Disponible en http://www.vics.org/committees/cpfr/voluntary.

(87) Villarreal, P. D. (2005). Método para el modelado y especificación de procesos de negocio colaborativos (tesis de doctorado). Argentina: Universidad Tecnológica Nacional-Facultad Regional Santa Fe.

(88) Villarreal, P. D., Lazarte, I., Roa, J., y Chiotti, O. (2010). "A modeling approach for collaborative business processes based on the UP-ColBPIP language". En S. Rinderle-Ma, S. Sadiq y F. Leymann (eds.). *Business Process Management Workshops. Lecture Notes in Business Information Processing*, vol. 43 (pp. 318-329). Berlín-Heidelberg: Springer.

(89) Villarreal, P. D., Salomone, E., y Chiotti, O. (2006). "Transforming collaborative business process models into web services choreography specifications". En J. Lee, J. Shim, S.g. Lee, C. Bussler, S. Shim (eds.). *Data Engineering Issues in E-Commerce and Services. Lecture Notes in Computer Science*, vol. 4055 (pp. 50-65). Berlín-Heidelberg: Springer.

(90) Villarreal, P. D., Salomone, E., y Chiotti, O. (2007). "Modeling and specifications of collaborative business processes using a MDA approach and a UML profile". En P. Rittgen (ed.). *Enterprise Modeling and Computing with UML* (pp. 13-44). Pensilvania: Idea Group Inc.

(91) Weske, M. (2007). *Business Process Management. Concepts, Languages, Architectures.* Berlín: Springer.

(92) Weske, M. (2012). Business Process Management (2a ed.). Nueva York: Springer.

(93) Weyns, D. (2010). *Architecture-Based Design of Multi-Agent Systems.* Berlín: Springer.

(94) White, S. A., Derek, M. (2008). *BPMN Modeling and Reference Guide: Understanding and Using BPMN.* Florida: Future Strategies Inc.-Lighthouse Point.

(95) Wooldridge, M. (2009). *Introduction to Multiagent Systems* (2 ed.). Inglaterra: John Wiley & Sons.

(96) Wooldridge, M., y Jennings, N. R. (1995). "Intelligent agents: Theory and practice". *Knowledge Engineering Review, 10*(2) (pp. 115-152).

(97) Wooldridge, M., Jennings, N. R., y Kinny, D. (2000). "The GAIA methodology for agent-oriented analysis and design". *Journal of Autonomous Agents and Multi-Agent Systems*, *3*(3) (pp. 285-312).

(98) WS-BPEL (2007). *Web Services Business Process Execution Language*, version 2.0. Standard, OASIS. Disponible en http://docs.oasisopen.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

(99) Xpand (2011). *Xpand- Model to Text Language. Specification release 1.2.1, Eclipse Modeling Project.* Disponible en http://wiki.eclipse.org/Xpand.

(100) Xu, L., de Vrieze, P., Bouguettaya, A., Liang, P., Phalp, K., y Jeary, S. (2012). "Service-oriented collaborative business processes". En V. Cardellini, E. Casalicchio, K. R. C. Branco, J. C. Estrella, F. J. Monaco (eds.). *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions* (pp. 116-132). Pensilvania: IGI Global.

(101) Yu, E. S. K. (1995). *Modelling strategic relationships for process reengineering* (tesis de doctorado). Canadá: University of Toronto-Department of Computer Science.

(102) Zinnikus, I., Hahn, C., y Fischer, K. (2008). "A model-driven, agent-based approach for the integration of services into a collaborative business process". En *Proceedings of the 7th international joint conference on Autonomous agents and multi-agent systems (AAMAS 2008)*, vol. I (pp. 241-248). Portugal: International Foundation for Autonomous Agents and Multiagent Systems.