VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"HERRAMIENTA DE GUÍA Y SUPERVISIÓN PARA EL USO AUTOMATIZADO DEL MODELO DE PROCESOS MOPROSOFT"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

ELENA CÁRDENAS VARGAS

DIRECTORA DE TESIS: DRA. HANNA OKTABA.

México, D.F. **2006.**

ASESORAS DE TESIS

M. en C. Silvia Guardati Buemo
Instituto Tecnológico Autónomo de México

Dra. Ana Lilia Concepción Laureano Cruces
Universidad Autónoma Metropolitana – Azcapotzalco y
Posgrado en Ciencia e Ingeniería de la Computación, UNAM.

A mi abuelita Naty, a mi mamá y a mi papá,
porque han sido, son y serán,
¡mi mejor ejemplo a seguir!

Agradecimientos

A la Dra. Hanna Oktaba por su valiosa asesoría, su paciencia, y sobre todo, porque con su ejemplo me inspira a seguir trabajando.

A la M. en C. Silvia Guardati y a la Dra. Ana Lilia Laureano, ya que además de brindarme su conocimiento y tiempo, recibí un trato cálido de cada una de ellas.

A la M. en C. Lupita Ibargüengoitia y al Dr. Christian Lemaître, por su importante participación en mi proyecto de tesis. Agradezco su tiempo, comentarios y sugerencias.

De manera muy importante doy gracias a mis padres Estela Vargas L. y Raymundo Cárdenas H., por su gran apoyo en todos los aspectos, a pesar de sus importantes ocupaciones. Su guía y presencia han sido determinantes en mi vida y son fuente de constante inspiración y referencia.

A mis hermanos Estela, Edith y Raymundo, por su apoyo, solidaridad y brindarme su cariño cada que nos podíamos ver. Gracias también por escucharme a pesar de nuestros distintos intereses. ¡Suerte a todos!

A Oscar por su cariño, ayuda y optimismo, que me devolvían la fuerza para seguir adelante en los momentos en que creía imposible la culminación de este proyecto.

A los amigos y familiares que han contribuido de alguna manera para que esto fuera una realidad. De manera especial agradezco a mi amiga Nely, quien ha estado conmigo siempre y cuya amistad espero conservar toda la vida. Y a Carlos Morales, ya que su ayuda fue decisiva para la culminación de esta tesis.

A los compañeros de la maestría, y a Amalia, Lulú, Diana y Juanita. Todos ayudaron a que esta etapa concluyera y fuera algo para recordar. Así mismo agradezco los apoyos académicos y económicos al Dr. Boris Escalante, al Dr. Sergio Rajsbaum y al CONACYT.

Por último, pero no por ello menos importante, agradezco a Dios todas sus bendiciones y por permitirme haber formado parte de la Universidad Nacional Autónoma de México.

Elena Cárdenas Vargas
Abril del 2006

Índice general

Índice de figuras

Índice de tablas

No hay viento favorable para el que no sabe hacia donde va.

–SÉNECA

Capítulo 1

PLANTEAMIENTO DEL PROBLEMA Y PROPUESTA DE SOLUCIÓN

El presente capítulo describe ampliamente el planteamiento del problema y la solución propuesta con la finalidad de que el lector cuente con una idea clara del contenido y propósito de este proyecto de tesis. Se introduce una arquitectura basada en agentes que se desarrolló utilizando conocimiento de las áreas de Ingeniería de Software e Inteligencia Artificial: Agentes y Razonamiento Basado en Casos.

Al ser parte importante de la solución del problema también se describen las metodologías de trabajo, así como las tecnologías utilizadas. Al final se encuentra el contenido de la tesis por capítulos.

1.1 ANTECEDENTES

En la actualidad varias de las empresas dedicadas al desarrollo de sistemas computacionales y entidades relacionadas con la Ingeniería de Software (IS) tienen conocimiento de la existencia de estándares y modelos de procesos, creados con la finalidad de transformar la generación de software en una actividad regida por técnicas y procesos probados, para sumar calidad a los productos generados.

Por consiguiente, trabajar en una empresa de desarrollo de software que sigue, o pretende seguir, alguno de estos estándares o modelos de procesos de software, involucra para un empleado entre otras cosas, tener un rol asignado (o varios) y con ello una serie de tareas y responsabilidades. Además las actividades se deben realizar de manera ordenada en coordinación con otros participantes, cumpliendo con las especificaciones indicadas por el modelo o estándar, lo cual en un proceso de implementación inicial o de aprendizaje no es fácil. Para el caso de muchas pequeñas y medianas empresas de México la situación no es más optimista, ya que la industria es apenas incipiente en nuestro país y no se cuenta con experiencia en el área de procesos de software. Una forma de aminorar la problemática es utilizando algún sistema de software diseñado específicamente para apoyar tales estándares o modelos de procesos.

El presente proyecto de tesis se centra en el estudio del Modelo de Procesos para la Industria de Software (MoProSoft), versión 1.1 [Oktaba01], que es explicado ampliamente en el capítulo siguiente.

Para mediados del 2003, en la Maestría en Ciencia e Ingeniería de la Computación, de la Universidad Nacional Autónoma de México (UNAM), se comenzó con el desarrollo de una Herramienta Integral para MoProSoft (HIM), un año después se terminó la primera versión funcional. Brevemente se puede describir a la HIM como un sistema *Web* diseñado para apoyar a las empresas en la adopción y seguimiento del MoProSoft. En el sistema cada usuario tiene uno o varios roles asignados y se le muestran las actividades que tiene que realizar (todo dependiendo de sus posibilidades de acuerdo al rol y al proyecto en que está dado de alta y conforme al modelo de procesos MoProSoft). Luego de elegir alguna actividad, se despliegan en la pantalla los elementos necesarios para llevar a cabo las tareas que indica dicha actividad.

Durante la implementación de la HIM se eligió como plataforma de desarrollo la Plataforma Java 2, Edición de Empresa (*Java 2 Platform, Enterprise Edition, J2EE*), y se utilizaron aquellos patrones de diseño que más se adecuaron a las necesidades de la herramienta [Vázquez01]. En cuanto a la parte de base de conocimiento, es importante mencionar que la lógica del funcionamiento de la HIM se basa en un *marco de trabajo,* compuesto por archivos creados utilizando la tecnología del marco de trabajo para la descripción de recursos (*Resource Description Framework, RDF*). RDF es una infraestructura que permite la codificación, intercambio y re-uso de *metadatos* estructurados para que puedan ser leídos por humanos y procesables por las máquinas.

Los archivos RDF modelan el MoProSoft junto con algunos aspectos adicionales, y pueden ser explotados por medio de consultas estilo SQL (Lenguaje Estructurado de Consultas, *Structured Query Language*).

1.2 PLANTEAMIENTO GENERAL DEL PROBLEMA

Para la creación de la HIM se fijaron varios requerimientos que el sistema debía cumplir y se completó la primera versión cumpliendo con algunos de ellos[1], otros quedaron como trabajos futuros de entre los cuales destacan el que no se explota completamente el marco de trabajo mencionado y que no se brinda ayuda al usuario inexperto en el MoProSoft.

Como se mencionó, el marco de trabajo generado en RDF modela todos los procesos del MoProSoft, considerando entre otras cosas, las actividades que se tienen que llevar a cabo, dependencias entre ellas, roles involucrados y sus permisos, productos generados en cada una de las actividades, y validaciones y verificaciones que se les tienen que hacer a estos últimos. Además se consideran datos importantes de los proyectos y de las organizaciones.

La información disponible es bastante completa y se utiliza para el funcionamiento de la HIM. Por ejemplo, cuando un usuario ingresa al sistema, su rol es reconocido junto con sus proyectos y con esa información se despliegan las actividades que debe llevar a cabo y ninguna más. Esa es una funcionalidad útil, sin embargo con la cantidad de información que se tiene disponible en el marco de trabajo, se podrían tener muchas más funcionalidades, como tomar en cuenta las dependencias y estado de los productos, esto es, que si un producto2 necesita de un producto1, y el producto1 no se ha creado, que el sistema lo indique; o dar información de utilidad. El sistema tampoco informa de las actividades realizadas y las pendientes, siendo el usuario el que debe de llevar registro de ello.

La motivación inicial para el desarrollo de la tesis fue brindar una solución a los dos problemas expuestos: se necesitaba de una adecuación a la HIM, que utilizara el marco de trabajo ya existente para brindar nuevas funcionalidades y que ayudara al usuario en el manejo de la herramienta, todo ello para contribuir con el objetivo inicial del proyecto, facilitar el aprendizaje, implantación y seguimiento del MoProSoft dentro de las empresas.

Para cumplir con las expectativas se propuso generar una herramienta semi-independiente a la HIM, que cumpliera con algunos requerimientos importantes y que además hiciera uso de tecnologías actuales, de manera que fuera una contribución importante al desarrollo de sistemas de este tipo y con ello a la exitosa implantación del MoProSoft en nuestro país.

Por consiguiente, el objetivo principal del proyecto es desarrollar una herramienta que incorporada a un sistema de apoyo al MoProSoft, HIM en este caso, funja como guía y supervisor del seguimiento de los procesos de dicho modelo y maximice la efectividad de los usuarios participantes, así como facilite el aprendizaje del mismo.

El sistema se llama *herramienta de guía y supervisión para el uso automatizado del modelo de procesos MoProSoft*, a la que nos referiremos de ahora en adelante como *AsistenteHIM*.

1.3 SOLUCIÓN PROPUESTA

Se decidió desarrollar una herramienta de software semi-independiente a la HIM, pero que se le pudiera integrar sin mayores dificultades. Que contara con las características del software de

[1] Lo que se comprueba en las partes de trabajos futuros de las tesis de los desarrolladores de HIM, ver bibliografía.

calidad: funcional, confiable, eficiente, usable, mantenible y portátil (ISO01). Que aprovechara la tecnología desarrollada en otras tesis relacionadas con la HIM, y que hiciera una contribución importante en el área de Ingeniería de Software en conjunto con la Inteligencia Artificial (Sistemas Multi-Agente y Razonamiento Basado en Casos), ya que son temas de mi interés particular, y como se explicará más adelante, proveen los medios para generar un sistema innovador que satisfaga las características deseadas.

Se trata de darle al usuario un asistente que le ayude con el trabajo, brindándole información de sus responsabilidades y posibilidades, tareas a realizar y responsables de ellas, sugerencias de la manera de llevarlas a cabo, recordatorios de tareas pendientes y coordinación del trabajo con otros usuarios; todo de acuerdo al MoProSoft.

Como el AsistenteHIM debía estar basado totalmente en el documento de MoProSoft, se optó por utilizar el marco de trabajo en RDF como base de conocimiento para el funcionamiento del sistema. Como se verá más a detalle en los capítulos de análisis, diseño y desarrollo de la herramienta (3 y 5), el marco de trabajo permitió obtener a través de consultas a los archivos RDF la información requerida, proveyendo la base de información del AsistenteHIM.

Otro aspecto central de la tesis tiene que ver con la implementación del sistema. Se resolvió utilizar agentes de software y el Razonamiento Basado en Casos debido a que sus características[2] permitieron alcanzar los objetivos más fácilmente. Esto es importante ya que implica la cohesión de varias disciplinas con la finalidad de obtener sistemas mucho más potentes, lo que incluye utilizar tanto nuevas tecnologías así como metodologías. A lo largo de todo el documento se integran las diferentes áreas de manera tal que el resultado es el sistema deseado y su proceso de construcción una guía para lograr desarrollar sistemas similares.

Básicamente un agente es *un sistema computacional situado dentro de un entorno, capaz de realizar acciones flexibles y autónomas con la finalidad de alcanzar sus objetivos de diseño* [Jennings01]. Un sistema multi-agente (SMA) es aquel que contiene una colección de dos o más agentes en el que al menos uno de ellos es autónomo y tienen la capacidad de interactuar en un entorno común [Laureano02].

Las diferentes propiedades que puede poseer un agente han permitido la definición de varios tipos de agentes. Los agentes generados para el AsistenteHIM involucran características de agentes comunicativos, adaptables, reactivos, y de interfaz, aunque cabe mencionar que cada agente posee sólo algunas de ellas y no todas como se verá con más detalle en el capítulo 3.

El AsistenteHIM es un sistema multi-agente, en el que cada agente busca en todo momento interactuar en su ambiente para llevar a cabo sus tareas a través del intercambio de conocimiento. La lógica es que los agentes de la herramienta ofrecen algunas de sus capacidades a los demás agentes, de tal forma que otros agentes puedan solicitarlos para cumplir con sus metas particulares. Para ello cada agente puede establecer comunicación con los demás a través de la plataforma de agentes que sirve de base para la interacción entre ellos.

El sistema multi-agente fue enriquecido con el enfoque de Razonamiento Basado en Casos (RBC), con el fin de potenciar los consejos dados por el AsistenteHIM. Esta técnica es utilizada en Inteligencia Artificial (IA) para la resolución de problemas. Se basa en comparar un problema

[2] Las características se describen en el marco teórico, capítulo 2.

presente con problemas similares (denominados casos) ocurridos en el pasado, para encontrar soluciones, modificar soluciones existentes o prevenir soluciones incorrectas.

Las experiencias o conocimiento previo pueden provenir del conocimiento en la bibliografía, de algún experto humano, o del uso regular del sistema [Laureano01], lo que permite que la herramienta sea adaptable a las organizaciones en donde es implantada y a su manera particular de solucionar los problemas.

Al utilizar la combinación de técnicas de RBC con sistemas multi-agente se consigue una herramienta que facilita el aprendizaje del modelo de procesos MoProSoft. Así mismo, al contar los usuarios con un asistente de actividades se espera que la curva de aprendizaje disminuya y por consiguiente el rendimiento y simpatía hacia el modelo aumenten.

1.4 JUSTIFICACIÓN

Durante el proceso de implantación de cualquier modelo o estándar de procesos de software, es de suma importancia contar con sistemas que realmente apoyen a las actividades involucradas, y no sólo que constituyan un medio para visualizar el modelo o estándar en computadora. Por consiguiente, se trata de proporcionar un sistema que asista y guíe en cada una de las tareas para que el proceso sea más sencillo y rápido para el usuario. Con ello no sólo se facilita la adopción de las normas en las organizaciones, sino que se automatizan las actividades que ahorran esfuerzo, trabajo y consecuentemente dinero.

En el presente proyecto de tesis se decidió trabajar con la HIM, porque es una herramienta conocida (tema de clases durante los cursos de la maestría), está disponible para cualquier tipo de investigación por ser de código libre, y cuenta con un *marco de trabajo* que modela al MoProSoft. Además para contribuir a que el proyecto HIM sea una realidad y se pueda distribuir en la Industria de Software mexicana como apoyo en la adopción del MoProSoft.

Pero ¿por qué no se eligió desarrollar las mismas funcionalidades enunciadas anteriormente, a través de un programa siguiendo el paradigma orientado a objetos que ya utilizaba HIM?, ¿porqué generar un sistema multi-agente y además utilizar el enfoque de Razonamiento Basado en Casos? Las justificaciones son más que el simple objetivo de involucrar tecnologías novedosas al desarrollo de la presente tesis:

1. Modelar con agentes tiene ventajas en el sentido de que este paradigma es bastante más cercano a nuestra manera de ver el mundo: muchas organizaciones se definen con un conjunto de roles y relaciones entre ellos, que luego son asumidos por personas (agentes).

2. La tecnología de agentes es suficientemente madura y muchas áreas de la ciencia de la computación han adoptado rápidamente este simple y poderoso concepto. Actualmente hay diferentes tipos de agentes, desde los agentes autónomos genéricos, agentes de software, agentes inteligentes, hasta los más específicos: agentes de interfaz, agentes virtuales, agentes de información y agentes móviles [Laureano01], lo que ha dado lugar al desarrollo de muchísimas aplicaciones en diferentes áreas: Control de tráfico aéreo, comercio electrónico, diagnóstico médico, resolución de problemas complejos por cooperación, etc.

3. El enfoque orientado en agentes es benéfico en situaciones en las cuales diversos/complejos tipos de información son requeridos, por ejemplo comunicación orientada a humanos [Caire01].

4. Los agentes son benéficos en situaciones que involucran negociación, cooperación y competencia entre diferentes entidades, o cuando el sistema tiene que actuar de manera autónoma (*Ídem p 26*).

5. Los agentes pueden ser pro-activos, es decir, actuar automáticamente cuando es necesario, así pueden interactuar con los usuarios jugando un papel de asistente personal.

6. Los agentes pueden manejar información local y distribuida, lo que es importante porque permite tener una base de conocimiento en una o varias máquinas, brindando muchas más posibilidades a cualquier sistema.

7. La información que es representada a través de los lenguajes semánticos (marco de trabajo en RDF), puede ser compartida entre aplicaciones como los agentes.

8. Dado que las fuentes de información y la manera de solucionar los problemas, varía de acuerdo a las organizaciones, el enfoque de Razonamiento Basado en Casos aplicado en los agentes, resulta muy útil ya que permite que el proceso se vaya aumentando, refinando y adaptando.

9. El RBC es un enfoque comprensible ya que se asemeja a la forma en la que resolvemos los problemas en la vida real, basamos nuestras decisiones en experiencias (buenas y malas) pasadas, y permite que el sistema vaya aumentando su eficiencia al pasar el tiempo.

10. Finalmente los agentes pueden ser tan sencillos o complejos como se quiera, lo que brinda la posibilidad de construir sistemas con características sencillas como intercambio de mensajes o delegación de tareas; hasta funcionalidades tan interesantes como el aprendizaje, el seguimiento de flujo de tareas, la planeación, personalización con cada usuario, autonomía, etc.

1.5 METODOLOGÍAS Y TECNOLOGÍAS

Metodología se define como *la aplicación de un método* y a su vez, método como *un conjunto de operaciones ordenadas con que se pretende obtener un resultado* [Larousse01]. En la actualidad existen infinidad de metodologías para llevar a cabo casi cualquier cosa, en todas las áreas del conocimiento podemos encontrar pasos a seguir que nos ayudan a obtener algún fin y el área de Ingeniería de Software no es la excepción. A continuación se explica la aplicación de cada una de ellas, dependiendo del área de conocimiento en que se utilizó.

1.5.1 Metodologías para el desarrollo de la tesis

A lo largo del desarrollo de este proyecto de tesis, se utilizaron algunos aspectos del Proceso Unificado de Desarrollo de Software (PU) [Jacobson01] y el Proceso Personal de Software (*Personal Software Process, PSP*) [Humphrey01].

Alternando varias de las actividades del PSP, como el reporte de actividades, el cuaderno de registro de tiempos y compromisos y el cuaderno de ingeniería, se siguieron algunas de las prácticas recomendadas por el PU para las fases de inicio, elaboración, construcción y transición del AsistenteHIM. Además se consideraron los flujos de trabajo de administración del proyecto, definición de requerimientos, administración de la configuración, análisis, diseño, implementación y pruebas

Con ello se consiguió llevar un orden en las actividades, contar con documentación confiable para futuras consultas o modificaciones al trabajo, tener la seguridad de cumplir con el ciclo de desarrollo de todo sistema de software de calidad, y sobre todo la conclusión de la tesis.

En el aspecto tecnológico se continuó trabajando con la plataforma de desarrollo de la arquitectura J2EE (la misma que se utilizó para el desarrollo de la HIM), la cual es un conjunto de especificaciones que tienen el lenguaje Java como su principal producto y cuenta con el apoyo de fabricantes como *Sun Microsystems* e *IBM*.

También se utilizó la biblioteca *Jena* [Jena01], que permitió leer, recorrer y sobre todo consultar, los archivos RDF del marco de trabajo desde el programa en Java, lo que se explica a detalle en el capítulo 5.

1.5.2 Metodologías y tecnología para el desarrollo del sistema multi-agente

1.5.2.1 Introducción

La ingeniería de software orientada a agentes, (*Agent Oriented Software Engineering, AOSE*) se está desenvolviendo actualmente como un nuevo paradigma para la creación de aplicaciones de agentes. Sin embargo, para que esto ocurra se tienen que implementar metodologías y herramientas robustas y fáciles de usar que proporcionen (1) la utilización de estándares orientados a las áreas de agentes y SMA, (2) metodologías para asistir en el ciclo de vida de la construcción de los sistemas, (3) lenguajes de comunicación y protocolos, y (4) entornos de desarrollo que permitan programar agentes [Jennings01].

En esa línea, se han generado en los últimos años numerosos trabajos relacionados, que se basan en su mayoría en una visión de un sistema como una organización computacional consistente en diferentes entidades interactuando. Entre las aproximaciones existentes podemos citar entre otros, los trabajos de: MAS-CommonKADS, GAIA, AUML, MaSE y la Metodología para la Ingeniería de Sistemas de Agentes de Software (*Methodology for Engineering Systems of Software AGents, MESSAGE*) [Vicente01].

La relativa juventud de estos trabajos conlleva que no exista un estándar de facto para el desarrollo de sistemas multi-agente. Sin embargo, de acuerdo a un análisis comparativo entre las metodologías anteriores (*Ídem*), se decidió utilizar MESSAGE [EURESCOM01] ya que sus características resultan más convenientes para el desarrollo del AsistenteHIM: MESSAGE abarca de una manera más completa las importantes etapas de análisis y diseño de un sistema de software, emplea el Lenguaje Unificado de Modelado (*Unified Modeling Language, UML*) en su notación, cuenta con ejemplos desarrollados, y tiene guías para orientar en las demás etapas de desarrollo.

1.5.2.2 Descripción general

MESSAGE surge en junio de 1999 como el proyecto P907-GI del Instituto Europeo para la Investigación y Estudios Estratégicos en Telecomunicaciones (*European Institute for Research and Strategic Studies in Telecommunications, EURESCOM*) y fue terminado dos años después. Entre los participantes en el proyecto se encuentran *British Telecommunications plc, France Télécom, Telecom Italia S.p.A., Portugal Telecom S.A., Belgacom, S.A. de Droit Public y Telefónica S.A* [EURESCOM01].

La metodología incorpora técnicas de Ingeniería de Software ya que organiza sus actividades con base en el Proceso Unificado de Desarrollo de Software [Jacobson01], provee un lenguaje, un método y unas guías de cómo aplicarla, centrándose en las fases de análisis y diseño y dando ideas sobre el resto de etapas como implementación, pruebas e implantación.

- *Análisis*: Su propósito es producir un conjunto de modelos en los que estén de acuerdo el analista y el usuario. Está constituida fundamentalmente por cuatro pasos, éstos son, la identificación de objetivos, el modelado de la organización, la detección de los agentes necesarios y el establecimiento de las interacciones precisas.

- *Diseño*: En esta fase se trata de modelar el sistema y encontrar su forma, de tal manera que de soporte a todos los requisitos que se le suponen y que fundamentalmente son resultado de la fase previa de análisis. Se centra en los siguientes pasos: el modelado de la arquitectura del sistema mediante refinamiento de los modelos del análisis, el modelado de los componentes internos de los diferentes agentes y el modelado de sus interacciones.

Tanto los pasos de la etapa de análisis, como los del diseño, serán desarrollados en el capítulo 3, dedicado fundamentalmente a esos dos aspectos.

Además MESSAGE modela los conceptos orientados a agentes con el mismo marco de trabajo semántico usado por el *UML*.

En cuanto a la existencia de herramientas de desarrollo asociadas, MESSAGE propone hacer uso de herramientas para UML, como *Rational Rose* o la herramienta metacase *MetaEdit+* (http://www.metacase.com/).

Finalmente, en lo que se refiere al empleo de una arquitectura de agente concreta, MESSAGE permite un diseño independiente de una arquitectura determinada o bien un diseño orientado. En el diseño orientado, que es el que se utiliza en el desarrollo del AsistenteHIM, los autores de MESSAGE emplean la arquitectura FIPA[3] y en específico el Marco de trabajo en Java para el Desarrollo de Agentes (*Java Agent Development Framework, JADE*) [Bellifemine01].

JADE es un entorno de desarrollo de software construido completamente en Java, que utiliza la programación orientada a objetos para la creación de sistemas multi-agente y aplicaciones conforme los estándares de la FIPA. Proporciona funcionalidades básicas como son la definición de agentes, funciones para paso de mensajes y un lenguaje de programación de los agentes. Sus características y manejo se describen en los capítulos 3 y 5 de este documento.

[3] FIPA (*Foundation for Intelligent Physical Agents*): Fundación para Agentes Inteligentes que promueve la tecnología basada en agentes y la interoperabilidad de sus estándares con otras tecnologías [FIPA01].

1.5.3 Metodologías y tecnología para el desarrollo del razonador basado en casos

1.5.3.1 Introducción

Aunque un sistema de Razonamiento Basado en Casos esté diseñado para investigación, para aplicación o para propósitos educativos, existe una divergencia entre los verdaderos objetivos del esfuerzo y la necesidad de desarrollar la infraestructura básica requerida para un sistema de RBC. Dicha infraestructura requiere componentes fundamentales para soportar procesos esenciales del enfoque RBC y una vez que el sistema ha sido construido, el desarrollador debe codificar también los procedimientos requeridos para probar el desempeño del sistema. Dichas actividades son independientes de la tarea y sin embargo ocupan bastante tiempo para su realización.

Desgraciadamente no se le ha dado la suficiente importancia al desarrollo de los sistemas que implementan las teorías desarrolladas en el área de IA y los esfuerzos más significativos se centran en el desarrollo de metodologías para la construcción de estos sistemas (por ejemplo, MESSAGE).

Como un esfuerzo para solucionar esa problemática se han generado varios marcos de trabajo o herramientas de desarrollo auxiliares en la construcción de un razonador basado en casos, cuya funcionalidad principal consiste en reutilizar y ofrecer al usuario las herramientas para construir un sistema razonador basado en casos, y que se pueda centrar en el dominio de su problema. Aamodt y Plaza [Aamodt01] hacen referencia a ReMind, CBR Express, Esteem, y en [Bello-Tomás01] se menciona el proyecto IBROW y la plataforma CAT-CBR, sólo por mencionar algunos.

Para el desarrollo del razonador del presente proyecto se decidió utilizar el marco de trabajo JColibri, ya que es un marco de trabajo orientado a objetos desarrollado en Java y para Java, con una interfaz gráfica para su configuración y auspiciado por el Comité Español de Ciencia y Tecnología y por el grupo para Aplicaciones de Inteligencia Artificial (*Group for Artificial Intelligence Applications*, *GAIA*) [JColibri01].

1.5.3.2 Descripción general

JColibri es un marco de trabajo para la construcción de sistemas de RBC desarrollado en el Departamento de Sistemas Informáticos y Programación de la Universidad Complutense de Madrid, España [JColibri01]. Es una evolución de la arquitectura COLIBRI [Bello-Tomás01], que consiste en una biblioteca de métodos solucionadores de problemas (*Problem Solving Method, PSM*), para solucionar las tareas de un sistema de RBC.

Está basado en el paradigma tarea/método propuesto por Aamodt y Plaza [Aamodt01] para construir un sistema que lleve a cabo las cuatro fases del RBC: recuperar, reutilizar, revisar y retener (capítulo 2). Dicho paradigma consiste en que cada una de las cuatro tareas, involucra un número de sub-tareas más específicas, y que existen métodos para solucionar las tareas, ya sea descomponiendo la tarea en más sub-tareas (métodos de descomposición), o solucionándola directamente (métodos de resolución).

Desde el punto de vista práctico, *las tareas describen la funcionalidad a ser resuelta por el programador, definiendo cuál código debe ser ejecutado*. *Los métodos resuelven estas tareas y ejecutan el código RBC* [JColibri01].

JColibri provee la mayoría del código necesario para representar la estructura de casos, las tareas, métodos y funciones de similitud, además de una plantilla de interfaz de usuario. Su *instanciación* es apoyada por una interfaz gráfica que guía la configuración de cada sistema RBC en particular, disminuyendo la curva de aprendizaje típica de estos sistemas. Adicionalmente, el marco de trabajo permite construir nuevas tareas y métodos si es necesario.

La versión que se utilizó en este proyecto es JCOLIBI 1.0 beta, liberada en noviembre de 2005.

1.6 ORGANIZACIÓN DE LA TESIS

La presente disertación de tesis consta de seis capítulos y cuatro apéndices que abarcan el proyecto en su totalidad. A continuación se describe brevemente el contenido de cada uno de ellos.

El capítulo 2 trata de registrar cuáles son los orígenes de las contribuciones de este proyecto, dando una vista sucinta de la situación actual en las áreas relacionadas con el tema: Modelos de procesos de software, El Modelo de Procesos para la industria del Software (MoProSoft), la Herramienta Integral para MoProSoft (HIM), el Marco de Trabajo para la Descripción de Recursos (RDF), Agentes y Sistemas Multi-Agente (SMA) y el enfoque de Razonamiento Basado en Casos (RBC).

El capítulo 3 comprende las fases de análisis y diseño aplicadas al desarrollo del AsistenteHIM por medio de la metodología MESSAGE (*Methodology for Engineering Systems of Software Agents*).

En el capítulo 4 se aborda la construcción del razonador basado en casos. Se incluyen conceptos importantes del Razonamiento Basado en Casos y se desarrollan las actividades de recolección de casos y creación de la base de casos. Consecutivamente se describe como se construyó el sistema razonador con la ayuda del marco de trabajo JColibri. Finalmente se presenta el sistema desarrollado y algunas pruebas que se realizaron a la base de casos.

En el capítulo 5 se define una arquitectura general del sistema AsistenteHIM y se desarrolla cada uno de sus componentes. Esto último implica la utilización de la tecnología RDF, el sistema Razonador Basado en Casos, así como la integración con la HIM. Finalmente se muestra el funcionamiento de la herramienta generada y las pruebas de sistema realizadas.

En el capítulo 6 se presentan los resultados y contribuciones obtenidos durante el desarrollo del trabajo, así como las conclusiones, limitaciones y los trabajos a futuro.

El apéndice 1 describe la notación específica de la metodología MESSAGE.

El apéndice 2 contiene esquemas complementarios de las distintas etapas de la fase de análisis del AsistenteHIM desarrollada en el capítulo 3, así como datos de la etapa de diseño.

El apéndice 3 contiene los resultados de las entrevistas realizadas para la recolección de casos (capítulo 4) y la lista completa de dichos casos.

Finalmente, el apéndice 4 contiene parte del código importante de la aplicación, y la documentación necesaria para poder llevar a cabo la integración del AsistenteHIM con la HIM. También se registraron los cambios que se hicieron en el código de la HIM y el JColibri, así como a los archivos y ontologías RDF. Por último se explica el contenido del disco compacto anexo a este documento de tesis.

Los escritos de los sabios son las únicas riquezas,
riquezas que nuestra posteridad no podrá malgastar.

–LANDOR

Capítulo 2

MARCO TEÓRICO

El presente capítulo trata de registrar cuáles son los orígenes de las contribuciones de este proyecto de tesis, dando una vista sucinta de la situación actual en las áreas relacionadas con el tema y que fueron mencionadas en el capítulo anterior: Modelos de procesos de software, El Modelo de Procesos para la industria del Software (MoProSoft), la Herramienta Integral para MoProSoft (HIM), el Marco de Trabajo para la Descripción de Recursos (RDF), Agentes y Sistemas Multi-Agente (SMA) y el enfoque de Razonamiento Basado en Casos (RBC).

2.1 MODELOS DE PROCESOS DE SOFTWARE

A mediados de los 60's y hasta principios de los 80's la creación de un producto de software era una tarea angustiosa, la programación se veía como un arte y existían pocas metodologías formales y casi nadie las usaba [Fuggetta01]; se trataba de una industria costosa, intensiva en mano de obra y con problemas de calidad, costos, plazos y capacidad de producción. Era necesario introducir una serie de herramientas y procedimientos que facilitaran por un lado, la labor de creación de nuevo software y por otro, la comprensión y el manejo del mismo. Esos fueron los inicios de la Ingeniería del Software, que hoy en día es reconocida como una disciplina legítima, digna de tener una investigación seria y un estudio concienzudo.

De acuerdo con el Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers, IEEE*), la Ingeniería de Software *es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento de software* [SWEBOK01].

A principios de los 80's surgió un área importante de la Ingeniería de Software, *Procesos de Software* [Cugola01], la cual estudia los procesos a través de los cuales un producto de software es desarrollado. Su importancia radica en que es necesario comprender y mejorar la calidad del software a través de nuevos enfoques y técnicas que estudien los procesos de desarrollo del mismo, ya que se asume que existe una correlación directa entre la calidad de los procesos y la calidad del software desarrollado.

La primera contribución importante del área de investigación de Procesos de Software, fue que el desarrollo de software es un *proceso* complejo. Fugguetta define a un proceso de software como *un conjunto de personas, estructuras de organización, reglas, políticas, actividades y sus procedimientos, componentes de software, metodologías, y herramientas utilizadas o creadas específicamente para conceptualizar, desarrollar, ofrecer un servicio, innovar y extender un producto de software* [Fuggetta01].

Con el surgimiento del área, se comenzaron a llevar a cabo una serie de talleres y eventos (Taller Internacional de Procesos de Software, *International Software Process Workshop*) y para finales de la misma década, varios organismos preocupados por la forma artesanal en la que se generaba el software comenzaron a desarrollar modelos de procesos de software para que fueran adoptados como estándares dentro de las organizaciones de la industria.

Para apoyar al Departamento de Defensa de los EE.UU se fundó en la Universidad de Carnegie Mellon el Instituto de Ingeniería de Software (*Software Engineering Institute, SEI*), que desarrolló un popular modelo de procesos para empresas en el área del software, el Modelo de Madurez de Capacidades (*Capability Maturity Model, CMM*) [SEI01].

Al CMM le siguieron más modelos y estándares de procesos (el PSP, *Personal Software Process*, el ISO 9001:2000, etc.), que hasta nuestros días se han adoptado con éxito en el amplio espectro de las aplicaciones de tecnologías de la información (TI).

En el siguiente esquema tomado de [Oktaba02] se ve de manera sencilla los principales modelos de procesos y estándares, indicando el año en que fueron desarrollados.

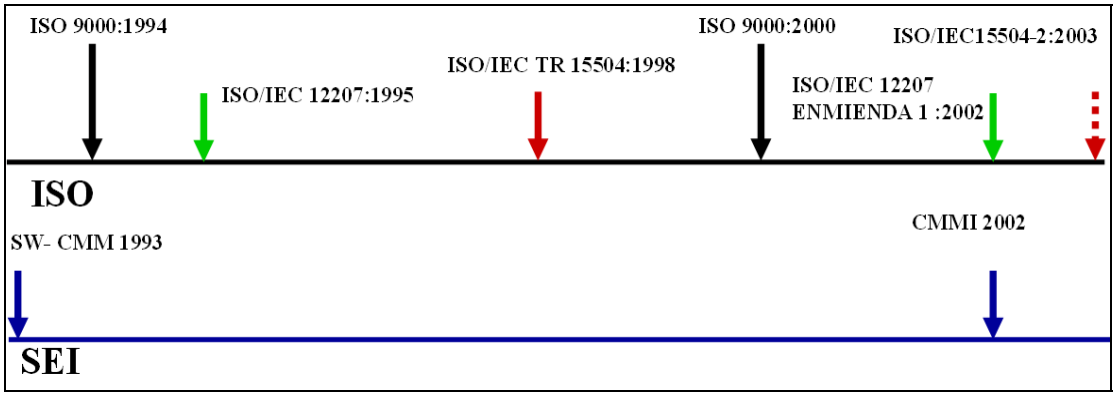

Figura 2.1: Línea de tiempo de algunos modelos de procesos y estándares para la Industria de Software [Oktaba02]

2.2 MODELO DE PROCESOS PARA LA INDUSTRIA DE SOFTWARE

2.2.1 Antecedentes

La industria del software forma parte del grupo de actividades económicas que componen a las tecnologías de información, las cuales se integran además por la industria de hardware y los servicios, y junto con las comunicaciones componen lo que se conoce como TIC (Tecnologías de Información y Comunicación).

El mercado de las TIC representa el 6.6% del valor de la producción económica mundial [SE01] y el mercado global de productos de software rebasa los 153,000 millones de dólares anuales. Estados Unidos es el principal consumidor con un gasto superior a los 75,000 millones de dólares anuales y una participación de 48.8% en el total mundial. México por su parte, tiene un nivel de gasto en las TIC de 3.2% del Producto Interno Bruto (PIB) que lo ubica en el quinto lugar a nivel mundial, dato que aunque pareciera favorable en primera instancia para la industria del software, la realidad es distinta ya que el gasto se divide con las tecnologías de comunicación (radio, televisión, telefónicas, etc.), ocasionando un rezago en el área, donde el gasto en México es 6 veces inferior al promedio mundial [SG01].

Si bien los países desarrollados continúan siendo líderes en esta materia, la demanda creciente no puede ser satisfecha con su oferta interna. Una proporción creciente de la producción mundial de software se realiza en países en desarrollo, India, Irlanda y Singapur representan casos exitosos de creación y crecimiento de industrias nacionales basadas en la exportación [SE01].

México cuenta con una posición favorable para convertirse en un competidor de talla mundial en este ramo, gracias a su ubicación geográfica (cercanía con EUA), perfil demográfico y estado de desarrollo tecnológico. No obstante la industria del software es apenas incipiente en nuestro país: participa con tan sólo el 0.10% del PIB (cifras de 2000) [DGP01].

Aunque no existe un padrón exhaustivo de esta industria que proporcione información exacta, una muestra de 206 empresas desarrolladoras de software muestra que el perfil actual de la industria es mayoritariamente micro y pequeña (90%). Aunado a esto, las empresas suelen ser volátiles, cuentan con pocos recursos y tienen procesos no estandarizados que dependen del personal que los ejecuta.

El Plan Nacional de Desarrollo 2001–2006 para México, plantea el objetivo de elevar y extender la competitividad del país mediante la estrategia de promover el uso y aprovechamiento de la tecnología y de la información. Basados en los datos anteriores, la Secretaría de Economía (SE) definió el 9 de Octubre del 2002 el Programa para el Desarrollo de la Industria de Software, PROSOFT [SE01], como uno de los medios para concretar este objetivo. Se espera situar a México como líder de esta industria en Latinoamérica para 2013 y convertirlo en líder desarrollador de soluciones de TI de alta calidad y uso de software.

Para responder a la problemática identificada, se propuso trabajar en siete estrategias básicas:

1. Promover exportaciones y la atracción de inversiones
2. Educación y formación de personal competente
3. Contar con un marco legal promotor de la industria
4. Desarrollar el mercado interno
5. Fortalecer a la industria local
6. Alcanzar niveles internacionales en capacidad de procesos
7. Promover la construcción de infraestructura física y de telecomunicaciones

A su vez, la estrategia 6 define siete líneas de acción de las cuales la segunda, *Definición de modelos de procesos y evaluación apropiados para la industria de software mexicana*, se comenzó a implementar evaluando la adopción de tres de los distintos modelos y estándares de procesos existentes (ISO 9000:2000, ISO/IEC TR 15504:1998:, SW-CMM 1993), en búsqueda de las siguientes características deseadas para el modelo.

1. Específico para el desarrollo y mantenimiento de software.
2. Fácil de entender (comprensible).
3. Definido como un conjunto de procesos.
4. Práctico y fácil de aplicar, sobre todo en organizaciones pequeñas.
5. Orientado a mejorar los procesos para contribuir a los objetivos del negocio y no simplemente ser un marco de referencia de certificación.
6. Debe de tener un mecanismo de evaluación o certificación, que indique un estado real de una organización durante un periodo de vigencia específico.
7. Aplicable como norma mexicana.

El resultado de la evaluación determinó que ninguno de los estándares o modelos cumplía con los requisitos expresados por la industria de software nacional, por lo que se decidió que se elaboraría un modelo nacional basado en los esquemas evaluados. La SE encargó a la Asociación Mexicana para la Calidad de Ingeniería de Software (AMCIS) y a la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) la elaboración de la Estrategia de Normalización para la Industria del Software en México.

Se propuso que la norma estuviera dividida en tres partes: modelo de procesos (qué procesos), modelo de capacidades de procesos (qué evaluar) y método de evaluación (cómo evaluar). Para la primera parte se desarrolló el Modelo de Procesos para la Industria de Software (MoProSoft) [Oktaba01] y la primera versión se terminó de elaborar en diciembre de 2002.

En mayo del 2003 fue terminada la siguiente versión (1.1) y para el 15 de agosto de 2005 [GOB01] fue publicada en el Diario Oficial de la Federación la declaratoria de la Norma Mexicana NMX-059/01-NYCE-2005 referente a MoProSoft.

Para la segunda y tercera parte de la norma, se elaboraron el modelo de capacidades basado en la norma ISO/IEC 15504-2 y el método de evaluación *EvalProSoft* respectivamente.

2.2.2 Descripción del modelo

2.2.2.1 Descripción General

El Modelo de Procesos para la Industria de Software (MoProSoft), versión 1.1 [Oktaba01], está dirigido a las pequeñas y medianas empresas o áreas internas dedicadas al desarrollo y/o mantenimiento de software en México y su propósito principal es fomentar la estandarización en la Industria de Software, a través de la incorporación de las mejores prácticas en gestión e ingeniería de software. Es aplicable tanto para las organizaciones que tienen procesos establecidos, así como para las que no cuentan con ellos.

El MoProSoft está fundamentado en los siguientes modelos: Modelo de Madurez de Capacidades (CMM) v1.1, ISO 9001:2000, ISO/IEC TR: 15504-2:1998, Cuerpo de Conocimiento para la Administración de Proyectos (*Guide to Project Management Body of Knowledge, PMBOK)*, Cuerpo de Conocimiento para la Ingeniería de Software *(Guide to the Software Engineering Body of Knowledge, SWEBOK),* Proceso Personal de Software (PSP) y Proceso de Software en Equipo (TSP).

Con la adopción del modelo se puede permitir elevar la capacidad de las organizaciones para ofrecer servicios de calidad y alcanzar niveles internacionales de competitividad, además de apoyar en la estandarización de sus prácticas, en la evaluación de su efectividad y en la integración de la mejora continua. Para lograr dicha estandarización MoProSoft propone un esquema de elementos divididos por categorías que servirán para la documentación de los procesos, este esquema es llamado patrón de procesos.

Así, el modelo es implementado por medio de procesos y considera los tres niveles básicos de la estructura de una organización (la Alta Dirección, la Gestión y la Operación). Estos niveles, llamados categorías, son divididos en seis procesos de alto nivel, y los procesos a su vez en actividades, y en su caso, sub-actividades. Cada actividad involucra una serie de pasos a realizar por uno o varios roles, que son usuarios con capacidades y características particulares definidas dentro del entorno del MoProSoft.

Esta organización da como resultado que dentro de un contexto general de organización, se tengan perfectamente definidos tanto los pasos de las actividades y la manera de llevarlas a cabo, como las características y responsabilidades de los usuarios que las deben de ejecutar, dejando claro las propiedades de cada objeto, su participación y su relación con los demás.

En la figura 2.2 se muestra la estructura de procesos del MoProSoft.

Figura 2.2: Arquitectura de MoProSoft

2.2.2.2 Alta Dirección

La categoría de Alta Dirección (DIR) sólo contiene el proceso de Gestión de Negocio (GN) cuyo propósito es ayudar a las organizaciones a establecer su razón social, objetivos y estrategias, así como prepararlas para situaciones de cambio. Se propone acciones para mantener una mejora continua, relación con los clientes y empleados, y tener una relación estrecha con la categoría de Gestión.

2.2.2.3 Gestión

Las actividades de los procesos de la categoría de Gestión (GES), deben proporcionar los elementos para el funcionamiento de los procesos de la Categoría de Operación, recibir y evaluar la información generada por éstos y comunicar los resultados a la Categoría de Alta Dirección.

La categoría está integrada por tres procesos: Gestión de Procesos (GP), en el que se pretende establecer los procesos de la organización, así como sus actividades de mejora. Gestión de Proyectos (GP), cuyo propósito principal es asegurar que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la empresa. Y Gestión de Recursos (GR), que se encarga de las cuestiones respecto a recursos humanos, infraestructura, ambiente de trabajo y proveedores, así como crear y mantener la *Base de Conocimiento* [Oktaba01] de la organización. Este último proceso está constituido por los subprocesos de Recursos Humanos y Ambiente de Trabajo (RHAT), Bienes, Servicios e Infraestructura (BSI) y Conocimiento de la Organización (CO).

2.2.2.4 Operación

La categoría de Operación (OPE) está integrada por los procesos de Administración de Proyectos Específicos (APE) y de Desarrollo y Mantenimiento de Software (DMS). La categoría trata las prácticas necesarias para desarrollar los proyectos de la organización y el mantenimiento de los productos de software. Realiza las actividades de acuerdo a los elementos proporcionados por la Categoría de Gestión, con la que se debe mantener una relación constante.

El propósito del proceso de Administración de Proyectos Específicos es establecer y ejecutar las actividades que permitan cumplir con los objetivos de un proyecto en el tiempo y costo esperados. Desarrollo y Mantenimiento de Software se encarga de todas las actividades que involucra el diseño de software, por ejemplo recolección de requerimientos, análisis, diseño, construcción, integración y pruebas.

Como se puede observar, MoProSoft es útil ya que abarcan todos los niveles de una organización a pesar de que se constituye por pocos procesos. Mediante flujos de trabajo, se integran las actividades que se tienen que llevar a cabo, los responsables de ello y los objetivos, indicadores y metas en relación. También se indican los productos involucrados así como sus actividades de verificación, validación y pruebas para mejorar su calidad.

Adicionalmente, MoProSoft propone la utilización de una *Base de Conocimiento* (repositorio de almacenamiento) para acumular todos los productos y documentos de la organización. Así mismo, sugiere la creación de *lecciones aprendidas,* que consisten en registrar las experiencias de la empresa y que servirán de fuente de información futura, por ejemplo para la creación de guías de ajuste.

2.3 HERRAMIENTA INTEGRAL PARA MOPROSOFT

2.3.1 Haciendo historia

Dado que el área de Ingeniería de Software es relativamente nueva, en nuestro país son pocas las empresas desarrolladoras de software que implementan procesos para su funcionamiento. Desde el nivel licenciatura de muchas de las carreras relacionadas con el área, no se toman en cuenta lo modelos de proceso y mucho menos su aplicación y seguimiento, consecuentemente en la empresa las cosas siguen igual y tratar de implementarlos cuesta aún más.

Gracias al Modelo de Procesos de Software, MoProSoft, México va en el buen camino en la solución a dicha problemática, ahora que ha sido declarado norma nacional [GOB01] se espera que sea adoptado por muchas medianas y pequeñas empresas de nuestro país, brindándoles los beneficios que proporciona el modelo.

No obstante, la implantación de cualquier modelo de procesos conlleva tiempo y esfuerzo y éste no es la excepción. Es necesario coordinar las actividades propuestas, así como los responsables y productos que se generarán. Es verdad que se cuenta con procesadores de texto, hojas de cálculo y demás documentos que ayudan en las tareas de planeación, pero es mucho mejor cuando se cuenta con alguna herramienta integral que proporcione todas las funcionalidades requeridas para la implantación del modelo. En particular el MoProSoft surgió para simplificar y

automatizar las prácticas de ingeniería de software, y una herramienta tiene que facilitar aún más este objetivo y no lo contrario.
Conscientes de lo anterior, el 19 de agosto del 2003 surge el proyecto de desarrollo de la *Herramienta Integral para MoProSoft (HIM),* la primera en ser desarrollada para el modelo, bajo la dirección de la Dra. Hanna Oktaba y la M. en C. Guadalupe Ibargüengoitia, y la participación de Ernesto Hdez. Uribe, Araceli E. Mercado Fdez., Hafiz Zurita R., Marcos O. Vázquez M., Karín Valdivieso C. y Jorge C. Vázquez, todos alumnos del curso de Ingeniería de Software Orientada a Objetos de la generación 2003 de la Maestría en Ciencia e Ingeniería de la Computación, de la Universidad Nacional Autónoma de México (UNAM). Para mediados de octubre de 2004, se entregó la primera versión funcional de la herramienta desarrollada siguiendo el mismo modelo de procesos.

2.3.2 Descripción general

HIM se basa en los procesos propuestos por el MoProSoft y tuvo como requisitos principales formar un ambiente integrado para soportar las actividades de MoProSoft, usar tecnología que permitiera su distribución como software libre, soportar diversos formatos de documentos, funcionar como entorno Web y tener las siguientes cualidades: portabilidad, soporte de diversos formatos de documentos, seguridad, usabilidad, soporte de grupos de trabajo y mantenibilidad .

Para el desarrollo de la HIM se siguieron las actividades del mismo MoProSoft, así como el Proceso Unificado de Desarrollo de Software [Jacobson01], el Proceso Personal de Software (PSP) [Humphrey01] y el Proceso de Software en Equipo (TSP) [Humphrey02].

En el aspecto tecnológico se eligió como plataforma de desarrollo la arquitectura J2EE, la cual es un conjunto de especificaciones que utiliza el lenguaje Java como su principal producto y cuenta con el apoyo de fabricantes como *Sun Microsystems* e *IBM.*

Se utilizaron aquellos patrones que más se adecuaron a las necesidades de la herramienta: *Model View Controller, Front Controller, Intercepting Filter, Validator, Composite View, Command Factory, Command* y *Facade* [Gamma01] y [Stelting01]. Además, se manejó un conocido marco de trabajo llamado *Struts* para implementar la lógica del negocio de la herramienta [Vázquez01].

La funcionalidad completa de la Herramienta Integral, se detalla en las tesis de cada uno de los alumnos desarrolladores [Hernández01], [Mercado01], [Zurita01], [Vázquez01] y [Valdivieso01].

Dado que en la construcción de la Herramienta se buscaba proporcionar apoyo a todos los procesos del MoProSoft, se integró un entorno de trabajo que da seguimiento a las actividades a realizar por cada rol específico de acuerdo a procesos y proyectos (descripción, objetivos, etc.), así como el almacenamiento o recuperación de los productos generados considerando las capacidades de creación, modificación o consulta dependiendo de cada rol, todo respetando las dependencias y relaciones establecidas en el modelo.

Se intentó que la HIM no sólo auxiliara a los usuarios en la generación de productos, sino que también los apoyara en el seguimiento a los procesos planteados por el modelo y se facilitara la comprensión e implantación del mismo, lo que se logró en parte gracias a la base de conocimiento generada utilizando el marco de trabajo para la descripción de recursos (*Resource*

Description Framework, RDF).

Base de Conocimiento

La base de conocimiento es el núcleo de funcionamiento de HIM, es en ella donde se guardan el modelo de procesos, los productos y se construyen los objetos del negocio, proveyendo todos los datos necesarios al resto de la aplicación [Hernández01].

Para desarrollar la base de conocimiento se tenían dos opciones, la primera era almacenar todo el modelado del MoProSoft en una base de datos, y la segunda utilizar la tecnología RDF. Como lo indica Hernández en su tesis, debido a las características de tamaño y número de integrantes de las pequeñas y medianas empresas a las que va dirigida la herramienta, no se consideró necesaria la utilización de una base de datos y se determinó suficiente el empleo de la tecnología RDF para implementar el modelo.

La base del conocimiento se compone de un *repositorio*, que es donde se almacenan todos los documentos generados por la organización que utilice el sistema, y un *marco de trabajo*, el cual mediante la utilización de la tecnología RDF, representa el modelo de procesos MoProSoft, modelando sus entidades y relaciones y proporcionando una estructura que permite la adaptación de los procesos en las organizaciones.

2.3.3 Oportunidades de mejora

Al final del desarrollo de la herramienta, se alcanzaron algunos de los objetivos propuestos, sin embargo hubo aspectos que se dejaron para trabajos futuros, de los cuales nos interesa mencionar los siguientes:

- No se explota completamente la maquinaria de inferencia proporcionada por la tecnología RDF, para hacer consultas por contenido de los Productos de Información Especializada (PIE's) y el Marco de Trabajo [Mercado01].

- No se tiene soporte para el acceso concurrente.

- No se notifica el cambio de estado de los productos, que consiste en enviar correos electrónicos (u otro tipo de mensaje) a los usuarios que de acuerdo a sus roles, deban conocer el cambio de estado de un producto [Zurita01].

Los puntos anteriores son importantes para este proyecto debido a que como se pudo ver en el capítulo anterior, constituyen partes importantes de la problemática que se pretende resolver. Cabe mencionar que el soporte para acceso concurrente en la HIM no se implementó como parte de esta tesis, sin embargo la tecnología de agentes desarrollada tiene soporte completo para aplicaciones de ese tipo y no debería presentar problemas adecuar el sistema desarrollado a la herramienta con acceso concurrente.

2.4 MARCO DE TRABAJO PARA LA DESCRIPCIÓN DE RECURSOS (*RDF*)

2.4.1 Introducción, la *Web Semántica*

Actualmente la Internet es un espacio preparado para el intercambio de información exclusivamente para el consumo humano. Las páginas electrónicas son creadas por personas para ser entendidas por personas, no existe un formato común para mostrar la información y mucho menos para el intercambio de la misma a través de las máquinas.

Para tratar de dar solución a estos problemas, surge la idea de la Web semántica, que tiene como objetivo crear un medio universal para el intercambio de información basado en representaciones del significado de los recursos de la Web, de una manera inteligible para las máquinas. Con ello se pretende ampliar la interoperabilidad entre los sistemas informáticos y reducir la mediación de operadores humanos en los procesos inteligentes de flujo de información.

El precursor de la idea, Tim Berners-Lee miembro fundador del W3C (*World Wide Web Consortium*), se esfuerza en que su propuesta de la *Web Semántica* sirva para ampliar la capacidad de la *World Wide Web* mediante estándares, lenguajes de marcado y otras herramientas aplicables a su tratamiento [SEMANTIC01].

Entre dichas herramientas y como resultado del esfuerzo de varias comunidades dedicadas al desarrollo de arquitecturas para el manejo de *metadatos* en la red, en 1999 se publicó la primera versión del Marco de Trabajo para la Descripción de Recursos (*Resource Description Framework, RDF*) [Miller01].

2.4.2 Descripción General

El RDF, es un lenguaje para representar información acerca de recursos en la red. Permite que las aplicaciones operen e intercambien información en lenguaje de máquina a través de la red [Manola01].

El RDF se basa en la idea de identificar cosas usando identificadores de red (llamados *Uniform Resource Identifiers, URI's*), y describir los recursos en términos de atributos y valores. Esto permite al RDF representar afirmaciones simples acerca de recursos como una gráfica de nodos y arcos representado los recursos, sus propiedades y valores.

2.4.3 Modelo Gráfico

El RDF utiliza una terminología particular para hablar acerca de las partes de las afirmaciones. Específicamente la parte que identifica el recurso del cual se trata la afirmación es llamada *sujeto.* La parte que identifica la propiedad o característica que identifica al sujeto es el *predicado*, y la parte que identifica el valor de esa propiedad es llamada *objeto* [Miller01]. El objeto puede ser un recurso o una literal (cadena de caracteres). Por ejemplo, tomando la siguiente oración en español: `http://www.geremy.org/index.html tiene un creador cuyo valor es Alejandro Cárdenas`

Los términos RDF para las partes de la afirmación son:

- El *sujeto* es la url `http://www.geremy.org/index.html`
- El predicado es la palabra "`creador`"
- El objeto es la frase "`Alejandro Cárdenas`"

El RDF modela las afirmaciones como nodos y arcos en el modelo gráfico [Manola01], en el cual una afirmación es representada por un nodo para el sujeto, un nodo para el objeto, y un arco para el predicado, dirigido desde el nodo sujeto hacia el nodo objeto. Así la oración del ejemplo anterior se puede representar como la figura 2.3.

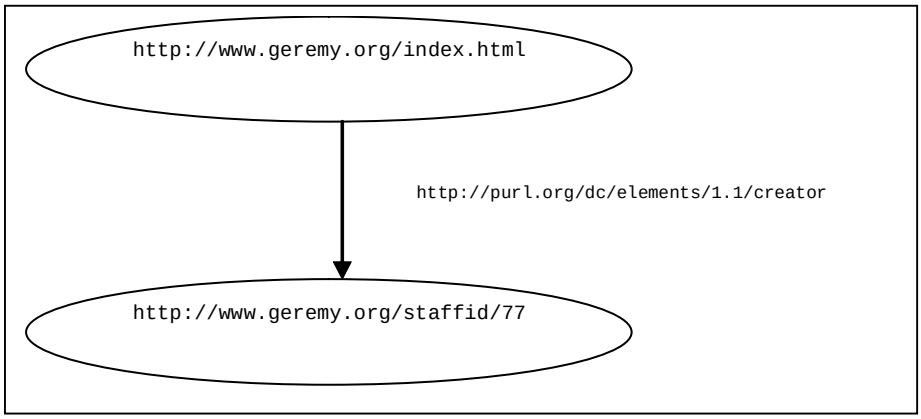

Figura 2.3: Afirmación simple en RDF

Los objetos en RDF pueden ser identificadores o valores constantes representados por cadenas de caracteres, para representar ciertos tipos de valores de propiedades, que se representan como cajas.

Para ejemplificar lo anterior y comenzar a introducir los conceptos del MoProSoft, se toma un ejemplo de [Hernández01]:

El Proceso de Administración de Proyectos Específicos de MoProSoft tiene la actividad APEA3 y se representó en el archivo AdministracionProyectosEspecificos.rdf.

A partir de esto se obtienen varias afirmaciones de las cuales sólo se toma la siguiente:

AdministracionProyectosEspecificos.rdf es del tipo *Proceso*.

Donde AdministracionProyectosEspecificos.rdf es un recurso, la propiedad es *tipo* y el valor de la propiedad es el recurso *Proceso*.

Los términos RDF para las partes de la afirmación son:

- El *sujeto* es `AdministracionProyectosEspecificos.rdf`
- El predicado es la palabra "`tipo`"
- El objeto es la palabra "`proceso`"

La gráfica RDF correspondiente es la de la figura 2.4.

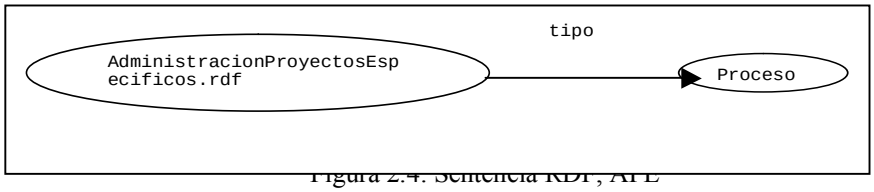

Figura 2.4. Sentencia RDF, API

2.4.4 Sintaxis XML para RDF: RDF/XML

Desarrollado bajo la guía del W3C, el RDF utiliza el Lenguaje de Marcado Extensible (*eXtensible Markup Language, XML*) como sintaxis común para el intercambio y procesamiento de *metadatos*. Con dicha sintaxis XML se escriben y convierten las gráficas en documentos XML/RDF.

Los documentos XML/RDF se forman escribiendo las oraciones que representan las gráficas anteriores,en forma de *tripletas* formadas por *sujeto, predicado y objeto*, en ese orden. Por ejemplo, la oración del ejemplo anterior se traduce en la siguiente notación de tripleta, que corresponde a la gráfica:

```
<AdministracionProyectosEspecificos> <tipo> <Proceso>
```

La notación de tripletas no es tan sencilla como en el ejemplo anterior, debido a que requiere que en el texto XML se hagan *referencias URI* (que son una forma de identificador más general que los URL que se utilizan en la Internet), lo que puede resultar en líneas muy largas, como se aprecia en el siguiente código que representa la misma tripleta anterior:

```
<file:///C:/HIM/BC/Configuracion/MoProSoft/AdministracionProyectosEspecificos
.rdf# AdministracionProyectosEspecificos.rdf>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Type>
<file:///C:/HIM/BC/Configuracion/Ontologias/proceso.owl#Proceso>
```

Por conveniencia, se utilizan los *nombres calificados (Qnames)* de XML que hacen posible que la tripleta anterior resulte finalmente de la siguiente manera:

```
ape:AdministracionProyectosEspecificos
rdf:type
proceso:Proceso
```

2.4.5 Ontologías o Vocabularios RDF: *RDF Schema*

Otro aspecto importante es el de ontología (vocabulario). La definición más consolidada según [Castells01] es la propuesta por Gruber, que la describe como *una especificación explícita y formal sobre una conceptualización compartida*. Lo que es interpretado por el mismo autor (Castells), como que *las ontologías definen conceptos y relaciones de algún dominio, de forma compartida y consensuada, y que ésta conceptualización debe ser representada de una manera formal, legible y utilizable por las computadoras*.

Por consiguiente, las ontologías se utilizan para describir, por medio de propiedades definidas, los tipos específicos de clases o recursos que se pretenden describir con el RDF. Tales clases y propiedades son declaradas como un vocabulario RDF utilizando una extensión del mismo RDF: El *Esquema RDF* (*RDF Schema o RDFS)*.

De esta manera, se tendrán dos tipos de archivos, unos de tipo ontología que es donde se definen cada uno de los conceptos del dominio del tema a describir, y otros de tipo RDF que utilizan las ontologías para describir los recursos.

En el proyecto de la Herramienta Integral para MoProSoft, se generaron varias ontologías de los conceptos identificados en el análisis: ontología de actividad, de proceso, de producto, de proyecto, etc. A continuación un fragmento de la ontología actividad con el fin de aterrizar los conceptos anteriores.

En el código del ejemplo siguiente se identifica en negrita la definición de la clase *Actividad*, más abajo se definen todas las propiedades de dicha clase, de las cuales se muestran *nombreMPS* (indica el nombre que tiene en el documento de MoProSoft y es de tipo *String*), *tieneSubActividad* y *dependedeActividad* (las dos de tipo *Actividad*), que como su nombre lo dice, representan las propiedades correspondientes a sub-actividades y actividades con las que tiene dependencia la actividad principal, definida en la clase.

```
<rdfs:Class rdf:ID="Actividad">
…
</rdfs:Class>
<rdf:Property rdf:ID="nombreMPS">
      <rdfs:domain rdf:resource="#Actividad"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="tieneSubActividad">
      <rdfs:domain rdf:resource="#Actividad"/>
      <rdfs:range rdf:resource="#Actividad"/>
</rdf:Property>
<rdf:Property rdf:ID="dependeActividad">
      <rdfs:domain rdf:resource="#Actividad"/>
      <rdfs:range rdf:resource="#Actividad"/>
</rdf:Property>
```

La descripción completa de la sintaxis RDF/XML y del Esquema RDF es extensa y requiere de una explicación que queda fuera del ámbito de este trabajo, sin embargo, era importante mostrar un ejemplo práctico para que el lector tuviera una idea clara del contenido de las ontologías y los archivos RDF, que comprenden el marco de trabajo que utiliza el AsistenteHIM como fuente de información.

Cabe mencionar que la elaboración del marco de trabajo de HIM fue el objetivo central de la tesis de Ernesto Uribe [Hernández01], el cual se alcanzó y está disponible para su estudio y uso en la bibliografía, y de manera práctica en los archivos que están incluidos en el proyecto de HIM.

En síntesis el marco de trabajo comprende varios archivos RDF que componen todo el modelo y algunos aspectos extras no contemplados en él. Primeramente se definieron vocabularios u

ontologías de los conceptos claves y posteriormente se crearon todos los archivos RDF que representan a cada uno de los procesos del MoProSoft.

2.4.6 Lenguaje de consulta de datos RDF, *RDQL*

Hasta ahora se han explicado modelos, gráficas y archivos RDF/XML, sin embargo, el RDF también cuenta con un mecanismo que permite realizar consultas de forma eficiente, para recuperar los datos almacenados en los archivos y explotar todo el potencial del RDF: El Lenguaje de Consulta de Datos RDF (*RDF Data Query Lenguaje, RDQL*).

El RDQL es un lenguaje de consulta de datos sobre archivos RDF, que recibe una solicitud en forma de consulta que sigue un patrón en forma de tripletas, y devuelve un conjunto de registros o renglones donde cada uno contiene la misma serie de pares con la forma etiqueta o nombre–valor, correspondiente a las variables solicitadas en la consulta, de manera similar a como se realizan en SQL [Hernández01].

La forma general de las consultas en RDQL es la siguiente:

```
SELECT variables a recuperar
WHERE tripletas
AND expresiones de filtrado
USING declaración de prefijos
```

En el `SELECT` se listan las variables a recuperar, separadas por una coma y precedidas por un signo de interrogación.

En el `WHERE` se utilizan las tripletas necesarias para especificar las condiciones con las que deberá cumplir cada registro que devuelva la consulta. Cada tripleta va escrita entre paréntesis y separadas por comas.

Con `AND` se pueden especificar expresiones booleanas para llevar a cabo filtros sobre los valores de los objetos. Además se pueden combinar con el uso de "`&&`", `AND` y "`||`", `OR`, o expresiones de igualdad (`EQ`) y desigualdad entre cadenas (`NE`).

En `USING` se declaran los prefijos que son utilizados dentro de las cláusulas `WHERE` y `AND` para que su lectura sea más fácil.

Un ejemplo de consulta realizada para una funcionalidad de la HIM es la siguiente, en dónde se busca obtener la ayuda de determinada actividad y su nombre MoProSoft (variables del `SELECT` en la línea 1`).` La actividad a recuperar es de tipo *Actividad* (línea 2) y la ayuda y el nombre MoProSoft se encuentran en las variables *ayuda* y *nombreMPS* de la ontología (líneas 3 y 4). En la línea 5 se hace la comparación del nombreMPS con el contenido de la variable *nomActividad* para que el resultado sólo sea de la actividad requerida.

Por último, en las líneas 6 y 7 se declararon los prefijos *RDF* y *actividad* que fueron utilizados en las cláusulas `WHERE` y `AND`.

```
1      SELECT ?actividad, ?nombreMPS, ?ayuda
2      WHERE ( ?actividad , <RDF:type>, <actividad:Actividad>),
3      ( ?actividad , <actividad:ayuda>, ?ayuda),
4      ( ?actividad , <actividad:nombreMPS>, ?nombreMPS)
5      AND ?nombreMPS EQ "+nomActividad+"
6      USING RDF FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ,
7      actividad FOR <file:///C:/HIM/BC/Configuracion/Ontologias/actividad.owl#>
```

Suponiendo que el contenido de la variable *nomActividad* es , el resultado son los valores de la tabla 2.1:

?actividad	?nombreMPS	?ayuda
GNA1.1	A1.1	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la Política de Calidad.

Tabla 2.1: Resultados de la consulta con RDQL

Es importante comprender el funcionamiento del RDQL debido a que constituye una parte importante del sistema de este proyecto de tesis ya que los objetivos principales del AsistenteHIM son dar guía y supervisión al usuario en el uso del MoProSoft, y ello sólo es posible basándose directamente en el propio modelo de procesos. Dado que el MoProSoft ya está modelado en el marco de trabajo, la parte medular son las consultas y el manejo de la información que se obtenga de ellas, ya que representa la misma información de guía que se le puede proporcionar al usuario. Las consultas RDQL estarán implementadas en un agente buscador, como se explica en el capítulo 5.

2.4.7 Jena

Para desarrollar aplicaciones basadas en RDF o lenguajes similares, se necesitan bibliotecas para leer y procesar las ontologías y archivos RDF definidos en estos lenguajes. El *parser (*programa que lleva a cabo el proceso de analizar una secuencia de entrada*)* de RDF y OWL[1] (*Web Ontology Languaje*) más popular es *Jena* [Jena01], desarrollado por Hewlett Packard, que permite leer, recorrer y modificar grafos tanto RDF como OWL desde un programa Java.

Jena se utilizó en el proyecto de HIM para manejar los vocabularios y archivos RDF construidos desde la herramienta. Así se pudo concretar el uso de los archivos RDF generados y basar parte importante de su funcionamiento en eso y en las consultas a los mismos archivos. También se utilizó Jena en el AsistenteHIM principalmente para las consultas del *agenteBuscador* (capítulo 3 y 5).

[1] Lenguaje de marcado para publicar y compartir datos usando ontologías en la red [SEMANTIC01].

2.5 FUNDAMENTOS DE AGENTES Y SISTEMAS MULTI-AGENTE

2.5.1 Introducción

Son conocidas y mencionadas en varios artículos, las metáforas referenciadas por Nicholas Negroponte [Negroponte01] respecto a la capacidad de relación entre las personas y las computadoras, y sobre todo en la capacidad de éstas (las computadoras) de manejar la información y filtrarla según los intereses y necesidades de cada usuario. Lo que antes era ciencia ficción es ahora una realidad y un área en constante desarrollo: los agentes y sistemas multi-agente (SMA).

Esta área está siendo utilizada en diversas aplicaciones, desde comparativamente pequeños sistemas tales como filtradores de correo electrónico, hasta sistemas grandes, complejos y de misión crítica como los de control de tráfico aéreo. Tales sistemas de tipos totalmente diferentes, comparten la abstracción de *agente* y su posibilidad de aplicación es lo que ha llevado a muchos investigadores y desarrolladores en interesarse en el tema.

El enfoque basado en agentes involucran varias áreas del conocimiento como lo son la Filosofía, la Sociología, la Economía y la Informática: Inteligencia Artificial, Sistemas Distribuidos, Computación Orientada a Objetos y más recientemente la Ingeniería de Software.

Al investigar información sobre el tema inmediatamente nos damos cuenta que el campo de la computación basado en agentes cuenta con mucha bibliografía y parece un poco confusa por la variedad de opiniones entre los autores. Para disminuir esa apreciación se han hecho innumerables esfuerzos por definir, conceptualizar y ordenar temas de importancia así como conceptos fundamentales[2]. A continuación se presenta información esperando lograr una buena comprensión del tema en relación con el presente proyecto de tesis.

2.5.2 Agentes

2.5.2.1 Teoría de Agentes

El término agente ha sido definido en innumerables ocasiones y desgraciadamente prevalece la falta de consenso. Entre las definiciones más citadas están las de los autores Russell, Norving, Pattie Maes, Barbara Hayes-Roth, Katia Sycara, Nicholas R. Jennings, Michael Wooldridge y Hyacinth S. Nwana; las cuales fueron analizadas en su mayoría por Stan Franklin y Art Graesser en su artículo *¿es un agente o sólo un programa?* [Franklin01], concluyendo con su propia definición que abarca una gran variedad de clases de agentes, apropiada para nuestros fines:

> *Un agente autónomo es un sistema situado dentro y como parte de un entorno, que percibe ese entorno y actúa en él a través del tiempo, en favor de su propia agenda así como del efecto que percibe en el futuro.*

La definición anterior se puede adaptar al proyecto AsistenteHIM ya que este último es un sistema multi-agente (SMA) situado dentro del entorno HIM y como parte del mismo, percibe ese entorno y actúa en él a través del tiempo (cuando el usuario realiza las actividades en la

[2] En la bibliografía están registradas las fuentes consultadas para el tema.

HIM). Así mismo, cada uno de los componentes del SMA perciben el entorno conformado por los demás agentes y actúan en el.

Por otro lado, Michael Wooldridge y Nick R. Jennings [Jennings01] propusieron una clasificación que se ha vuelto un estándar desde 1992 y que también nos resulta de utilidad para especificar de manera más clara la naturaleza del AsistenteHIM. Se proponen dos nociones de agencia. Una noción débil en la cual los agentes exhiben cuatro propiedades básicas:

1. *Autonomía* (los agentes actúan sin intervención humana).
2. *Sociabilidad* (los agentes interactúan con otros agentes).
3. *Reactividad* (los agentes perciben el "mundo" y reaccionan a estímulos específicos).
4. *Situación* (los agentes exhiben un comportamiento orientado a objetivos).

Y una noción más fuerte o restrictiva, donde los agentes suelen ser considerados como sistemas intencionales, esto es, sistemas cuya conducta puede ser predecida atribuyendo creencias, deseos y una conducta racional.

De acuerdo a las definiciones anteriores, los agentes del AsistenteHIM se clasifican dentro de la noción débil de agente. Los agentes tienen autonomía porque actúan de manera reactiva con base en sus competencias y los estímulos del entorno que corresponden a dichas competencias. Son sociables porque se comunican entre ellos para conseguir sus fines. Son reactivos porque responden a acciones del usuario dentro del entorno que es la HIM, y tienen objetivos bien definidos. Los agentes no se pueden ubicar dentro de la noción más fuerte de agentes ya que no involucran comportamientos intencionales.

De acuerdo a las cuatro características anteriores, Jennings, Sycara y Wooldridge definen a un agente como [Jennings01]

...un sistema computacional, situado dentro de un entorno, que es capaz de acción flexible y autónoma, con la finalidad de alcanzar sus objetivos de diseño.

Que es la definición más adecuada para los agentes utilizados en el AsistenteHIM ya que cada uno de ellos lleva a cabo sus acciones dentro del entorno, conforme sus objetivos de diseño explicados ampliamente en el capítulo 3.

2.5.2.2 Arquitecturas de Agentes

Las arquitecturas de agentes describen la interconexión de los módulos software/hardware que permiten a un agente tener la conducta especificada en las teorías de agentes. Las tecnologías de objetos y de sistemas expertos cuentan con arquitecturas básicas fijas, en el caso de los sistemas multi-agente se cuenta con distintos tipos de arquitecturas. Wooldridge clasifica las arquitecturas atendiendo al tipo de procesamiento empleado en deliberativas, reactivas e híbridas [Wooldridge02].

2.5.2.2.1 Arquitecturas deliberativas

Las arquitecturas deliberativas siguen la corriente de la IA simbólica, que se basa en la hipótesis de los sistemas de símbolos físicos, según la cual un sistema de símbolos físicos es capaz de

manipular estructuras simbólicas y a través de ellos puede exhibir una conducta inteligente. Las arquitecturas de agentes deliberativos suelen basarse en la teoría clásica de planificación de inteligencia artificial: dado un estado inicial, un conjunto de operadores/planes y un estado objetivo, la deliberación del agente consiste en determinar qué pasos debe encadenar para lograr su objetivo, siguiendo un enfoque descendente (*top-down*).

Las arquitecturas deliberativas pueden clasificarse como horizontales (dentro de otra clasificación de arquitecturas de agentes) porque los estímulos recibidos del exterior son procesados en varias capas de diferente nivel de abstracción y al final el nivel superior decide qué acciones hay que llevar a cabo [Wooldridge02].

2.5.2.2.2 Arquitecturas reactivas

Las arquitecturas reactivas proponen una arquitectura que actúa siguiendo un enfoque conductista, con un modelo estímulo-respuesta. No tienen un modelo del mundo simbólico como elemento central de razonamiento y no utilizan razonamiento simbólico complejo, sino que siguen un procesamiento ascendente (*bottom-up*), para lo cual mantienen una serie de patrones que se activan bajo ciertas condiciones de los sensores y tienen un efecto directo en los actuadotes (figura 2.5). Como es citado en [Laureano02], las arquitecturas reactivas empiezan a ser investigadas en los 80's por Pattie Maes y Brooks. Las principales arquitecturas de este tipo son las reglas situadas, las arquitecturas incluidas (*subsumption*), los autómatas de estado finito, las tareas competitivas y las redes neuronales [Mas01].

Las arquitecturas reactivas pueden clasificarse como verticales porque los estímulos recibidos del exterior son procesados por capas especializadas que directamente responden con acciones a dichos estímulos y pueden inhibir las capas inferiores (figura 2.5).

Los agentes del AsistenteHIM son reactivos, cuentan con vistas parciales del entorno, lo que les permite la distribución de objetivos.

Existe una excepción con el agente que utiliza el enfoque RBC (capítulos 3 y 4). Los módulos reactivos se encargan de procesar los estímulos que no necesitan deliberación (como los demás agentes), mientras que el módulo deliberativo está formado por el razonador basado en casos.

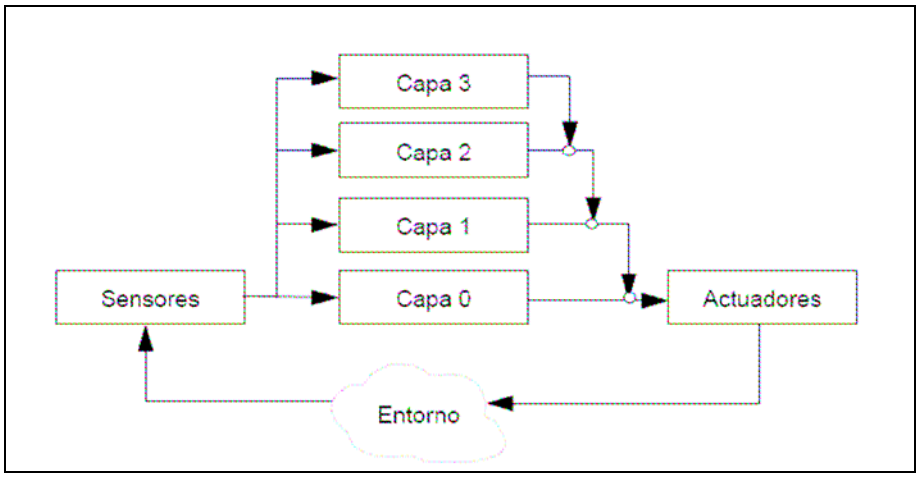

Figura 2.5: Arquitectura incluida de un agente reactivo

2.5.2.3 La naturaleza del entorno

En la definición del término de agente se tiene la palabra clave *ambiente* o *entorno,* y es importante abarcar el tema ya que determina, hasta cierto punto, el diseño más adecuado para el agente y la utilización de cada una de las familias principales de técnicas de implementación de agentes [Russell01].

- *Totalmente observable vs. Parcialmente observable*: Si los sensores del agente le proporcionan acceso al estado completo del medio en cada momento, entonces de dice que el entorno de trabajo es totalmente observable.

- *Determinista vs. Estocástico*: Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es determinista; de otra forma es estocástico. Si el medio es determinista, excepto para las acciones de otros agentes, decimos que el medio es *estratégico*.

- *Episódico vs. Secuencial*: En un entorno de trabajo episódico, la experiencia del agente se divide en episodios atómicos. Cada episodio consiste en la percepción del agente y la realización de una acción posterior. En entornos secuenciales, por otro lado, la decisión presente puede afectar a decisiones futuras.

- *Estático vs. Dinámico*: Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es dinámico para el agente; de otra manera se dice que es estático. Si el entorno no cambia con el paso del tiempo, pero el rendimiento del agente cambia, entonces se dice que el medio es *semi-dinámico*.

- *Discreto vs. Continuo*: La distinción entre discreto y continuo se puede aplicar al *estado* del medio, a la forma en la que se maneja el *tiempo* y a las *percepciones* y *acciones* del agente. Por ejemplo, un medio con estados discretos como el del juego del ajedrez tiene un número finito de estados distintos y un taxista conduciendo define un estado continuo.

- *Agente individual vs. Multi-agente.*

El caso más complejo es el *parcialmente observable, estocástico, secuencial, dinámico, continuo y multi-agente.*

El entorno del AsistenteHIM se puede clasificar como *totalmente observable, determinista, secuencial, estático, continuo y multi-agente*, lo que se explica en el siguiente capítulo por ser parte del análisis del sistema.

2.5.2.4 Lenguajes de Agentes

Wooldridge define a los lenguajes de agentes como lenguajes que permiten programar agentes con los términos desarrollados por los teóricos de agentes, y distingue los siguientes tres niveles de abstracción en la programación de agentes [Wooldridge02]:

- *Lenguajes de programación de la estructura del agente:* permiten programar las funcionalidades básicas para definir a un agente: funciones de creación de procesos y

funciones de envío y recepción de mensajes. Los lenguajes empleados suelen ser lenguajes de propósito general o lenguajes específicos (C++, Java o Prolog).

- *Lenguajes de comunicación de agentes:* definición del formato de los mensajes intercambiados, de las primitivas de comunicación y de los protocolos disponibles. Podemos distinguir dos tipos de lenguajes de comunicación:

    - *Procedimentales:* se basan en el intercambio de directivas procedimentales, es decir, un agente recibe un mensaje que implica la ejecución de un procedimiento. Suelen emplear lenguajes de intérpretes de órdenes (*scripts*) como Perl, Tcl, etc.

    - *Declarativos:* se basan en el intercambio de actos comunicativos, es decir, un agente recibe un mensaje con un acto comunicativo que le permite interpretar el contenido del mensaje. Los ejemplos más extendidos de este enfoque son el FIPA-ACL (*Agent Communication Languaje*) y el KQML (*Knowledge Query and Manipulation Lenguaje)* [Mas01].

- *Lenguajes de programación del comportamiento del agente:* permiten definir el conocimiento del agente: conocimiento inicial (modelo de entorno, creencias, deseos, objetivos), funciones de mantenimiento de dicho conocimiento (reglas, planes, etc.), funciones para alcanzar sus objetivos (planes, reglas, etc.) y funciones para desarrollar habilidades (programación de servicios). Se distinguen cuatro niveles de descripción:

    - *Lenguajes de descripción de agente.*
    - *Lenguajes de programación basados en el estado mental.*
    - *Lenguajes basados en reglas de producción.*
    - *Lenguajes de especificación.*

Para el desarrollo completo del asistente HIM se utilizaron algunos de los lenguajes mencionados: En cuanto a los *lenguajes de programación de la estructura del agente*, se utilizó Java para implementar el sistema. El FIPA-ACL fue utilizado para la generación de mensajes de los agentes y el SL0 como lenguaje de contenido. De los *lenguajes de programación del comportamiento del agente*, se utilizó un *lenguaje de descripción de agente* (donde los agentes se derivan de una clase de agente genérica, permitiendo la definición de los elementos básicos del modelo de agente) llamado JADE [Bellifemine06]. Todo lo anterior se verá más claramente a lo largo de los siguientes capítulos.

2.5.2.5 Tipologías de Agentes

Las definiciones discutidas en el apartado de teorías de agentes involucran una serie de propiedades de un agente, las cuales pueden ayudar a clasificar los agentes de diversas maneras útiles. Por ejemplo, los agentes se pueden clasificar de acuerdo a las tareas que llevan a cabo o de acuerdo a sus arquitecturas de control, por la capacidad y sensibilidad de sus sentidos, por el rango y la efectividad de sus acciones, por su estado interno o de acuerdo a su entorno.

Jacques Ferber sugiere una primera clasificación que divide a los agentes en reactivos y cognitivos, dando origen a dos escuelas de pensamiento [Ferber01]. En la *escuela cognitiva* se dice que los agentes son *intencionales,* esto es, que tienen objetivos y planes explícitos que les permiten alcanzar sus metas. En la *escuela reactiva* por el contrario, establece que no es

necesario que los agentes sean inteligentes de manera individual para que el sistema demuestre comportamiento inteligente, ya que éste se consigue con el desempeño de todos los agentes reactivos.

En la tabla 2.2 se listan varias propiedades que definen diferentes tipos de agentes [Jennings01].

Propiedad	*Significado*
Reactivo (percibe y actúa)	Responde en tiempo a los cambios en el ambiente.
Autónomo	Tiene control sobre sus propias acciones.
Orientado a objetivos (pro-activo, con propósitos)	No actúa simplemente en respuesta al ambiente, sino a objetivos.
Temporalmente continuo	Es un proceso continuo.
Comunicativo (sociable)	Se comunica con otros agentes y tal vez incluyendo otras personas.
Que aprende (adaptable)	Adapta su comportamiento de acuerdo a experiencias previas.
Móvil	Capaz de transportarse de una máquina a otra.
Flexible	Las acciones no están preestablecidas.
Con carácter	Cuenta con personalidad y estado emocional.

Tabla 2.2: Propiedades de los agentes [Jennings01]

En la misma bibliografía, se menciona que Franklin y Graesser [Jennings01] utilizan el subconjunto de propiedades de la tabla anterior y expresan que cada agente satisface las primeras cuatro propiedades y agregando otras se producen distintas clases de agentes, de las cuales surgen los *Agentes de Software,* que son los utilizados para el proyecto del AsistenteHIM, por lo que se tratan de manera particular en el siguiente apartado.

Agentes de Software

Un agente de software es autónomo, comunicativo, posee recursos propios, tiene una representación parcial de otros agentes, posee servicios que puede ofrecer a otros agentes y su comportamiento atiende sus objetivos, tomando en cuenta sus recursos y habilidades disponibles y dependiendo de sus representaciones y las comunicaciones que recibe [Ferber01].

Los tipos de agentes de software más conocidos son [Nwana01]:

- *Agentes de interfaz:* los agentes de interfaz, también denominados *asistentes personales* o *agentes de usuario*, tienen como objetivo simplificar las tareas rutinarias que realiza un usuario. Son agentes flexibles que exhiben capacidades de adaptación al medio. Ayudan al usuario en tareas repetitivas habituales y se basan en la idea de delegación, además pueden actuar por iniciativa propia. Pueden, por ejemplo, aprender a filtrar el correo electrónico fijándose en el comportamiento habitual del usuario; planificar encuentros, negociando con los asistentes personales de los otros miembros del encuentro; o detectar que una noticia puede ser relevante para un usuario y comunicárselo. Dentro de esta clasificación se ubica el *agenteCoordinador* del AsistenteHIM (capítulo 3).

- *Agentes móviles:* Pueden transitar entre varias máquinas, para evitar una sobrecarga de comunicación o utilizar recursos de los que no dispone en su máquina.

- *Agentes de Internet/información:* Su misión es navegar por la red recolectando información, indexarla y ofrecer la información que puede interesar al usuario cuando realiza una consulta.

2.5.3 Sistemas multi-agente

2.5.3.1 Introducción, la Inteligencia Artificial Distribuida

La inteligencia artificial distribuida (IAD) se considera un sub-campo de la Inteligencia Artificial, que se centra en los comportamientos inteligentes colectivos que son producto de la cooperación de diversas entidades denominadas agentes [Noriega01]. En la disciplina se distinguen tres periodos cronológicos bien diferenciados [Nwana01]:

- La IAD "clásica" se centra en el estudio de la conducta colectiva, en oposición a la IA, que estudia la conducta individual.

- La IAD "autónoma" se centra en el estudio de los agentes individuales situados en un mundo social. Se modifica la visión inicial de la IAD en que la "perspectiva social" significa que la sociedad prima sobre los individuos y se centra en estudiar agentes autónomos en un mundo multi-agente.

- IAD "comercial" se centra en la aplicación de la IAD clásica y autónoma, desarrollando agentes (denominados genéricamente *agentes software*) con características muy diferenciadas, que están siendo explotados de forma comercial.

La IAD "clásica" es la que nos interesa, debido a que se divide en dos áreas principales de investigación: Resolución Cooperativa de Problemas Distribuidos (*Cooperative Distributed Problem Solving, CDPS*) y Sistemas Multi-Agente (*Multi-Agent Systems, MAS*), que estudia la coordinación de la conducta entre un conjunto de agentes inteligentes autónomos, que es parte importante de esta tesis.

2.5.3.2 Concepto de sistema multi-agente

En [Laureano01] se especifica que de acuerdo a Luck y d'Inverno "un sistema multi-agente es aquel que contiene una colección de dos o más agentes. Como los objetivos no pueden existir sin haber sido generados por agentes autónomos, es imposible para los agentes existir sin autonomía en un sistema. Así, la definición de un sistema multi-agente es cualquier sistema que contiene dos o más agentes, al menos un agente autónomo y al menos también, una relación entre dos agentes dónde uno satisfaga los objetivos del otro".

Las características de los sistemas multi-agente son las siguientes [Jennings01].

- Cada agente tiene información incompleta y un punto de vista limitado;

- No hay control global del sistema;
- Los datos, o la información, están descentralizados, y
- La computación es asíncrona

Algunas razones para el creciente interés en la investigación de los SMA incluyen la habilidad que tienen para proveer eficiencia a los sistemas, la habilidad para permitir inter-operación de sistemas antiguos, y la habilidad de resolver problemas en los que los datos, los expertos o el control, están distribuidos.

A pesar de que los SMA proveen numerosas ventajas potenciales, también encaran muchas dificultades. A continuación se presentan los problemas inherentes en el diseño e implementación de los SMA y que se dice, son los problemas básicos que estudia la IAD clásica, y serán comunes a todos los sistemas (*Ídem pp. 4*).

1. Cómo formular, describir, descomponer y asignar problemas, y sintetizar los resultados entre un grupo de agentes inteligentes.

2. Cómo capacitar a los agentes para que se comuniquen e interactúen: qué lenguajes de comunicación o protocolos deben utilizarse, qué y cuándo deben comunicarse, etc.

3. Cómo asegurar que los agentes actúan coherentemente al tomar decisiones o realizar acciones, cómo acomodar los efectos globales de las decisiones locales y prevenir interacciones no deseadas.

4. Cómo capacitar a los agentes para representar y razonar sobre acciones, planes y conocimiento de otros agentes para coordinarse; cómo razonar sobre el estado de su proceso de coordinación (inicio o terminación).

5. Cómo reconocer y reconciliar puntos de vista e intenciones conflictivas entre un conjunto de agentes para coordinar sus acciones; cómo sintetizar los puntos de vista y los resultados.

6. Cómo utilizar técnicas de ingeniería y desarrollar sistemas con IAD. Cómo diseñar plataformas de SMA y metodologías de desarrollo con técnicas de IAD.

A lo largo de los siguientes capítulos, se desarrolla el SMA AsistenteHIM tratando de cubrir la mayoría de los puntos anteriores de una manera práctica.

2.5.3.3 Arquitecturas Multi-Agente

Existen dos arquitecturas importantes de los SMA:

- *Arquitectura RETSINA:* RETSINA (*Reusable Environment for Task-Structured Intelligent Network Agents*) es una arquitectura multi-agente independiente del dominio, que puede aplicarse para el desarrollo de agentes cooperativos y que ha sido creada en el Instituto de Robótica de la Universidad de *Carnegie Mellon* [RETSINA01]. El entorno de RETSINA se ha desarrollado sobre la base de que los agentes forman una comunidad en la que todos ellos están al mismo nivel y sin imponer restricciones, a nivel de

infraestructura, en las comunicaciones. La infraestructura de servicios que proporciona esta arquitectura impide la existencia de un control centralizado.

- *Arquitectura FIPA:* Proporciona la infraestructura para el desarrollo y uso de agentes, y contiene los recursos de hardware y software necesarios para poner en marcha esta infraestructura. La idea es que en cada implementación concreta se tomen las decisiones relativas a las componentes usadas para desarrollarla, pero manteniendo siempre un conjunto de interfaces externas que permitan la interoperabilidad entre plataformas.

Actualmente, la mayoría de los entornos de desarrollo y ejecución de agentes tienden a adoptar FIPA, razón por la que se utilizó dicha arquitectura en el desarrollo del AsistenteHIM. Esto se abordará con mayor detalle en el capítulo 5.

2.5.4 Aplicaciones de los agentes y sistemas multi-agente

La tecnología de agentes está saliendo rápidamente de las universidades y laboratorios para comenzar a ser utilizada en la resolución de problemas reales, en un rango que va desde aplicaciones industriales hasta las comerciales. Una buena forma de ejemplificar su utilidad es analizando la gran variedad de aplicaciones que los SMA tienen.

A continuación las principales áreas donde el enfoque basado en agentes está siendo utilizado junto con algunos ejemplos de sistemas mencionados en [Muscettola01] y [Jennings01].

- Aplicaciones Industriales
  - Manufactura y Robots clasificadores de piezas
  - Control de procesos
  - Telecomunicaciones y Control de tráfico aéreo
- Aplicaciones comerciales
  - Manejo de información
  - Comercio electrónico
  - Administración de procesos de negocio
  - Buscadores e identificadores de recursos humanos
  - Asistente de correo electrónico

- Aplicaciones de entretenimiento
  - Juegos y entretenimiento móvil
  - Teatro y cine interactivo
  - Asistentes personales de viaje

- Aplicaciones médicas
  - Diagnóstico médico
  - Monitoreo de pacientes

- Aplicaciones variadas
  - En sociología, el comportamiento de una sociedad
  - En economía, los sistemas de negociación
  - En Inteligencia Artificial, resolución de problemas complejos por cooperación
  - Construcción de aplicaciones concurrentes

2.6 RAZONAMIENTO BASADO EN CASOS

2.6.1 Introducción

Un campo trascendental para el ser humano es el de la toma de decisiones, ya que cada vez que tenemos un problema hay que tomar una decisión y los problemas abundan. Con ayuda de la memoria (recordando experiencias exitosas y desafortunadas), elegimos una alternativa que nos parezca suficientemente racional y que nos permita maximizar el resultado esperado luego de hecha nuestra acción.

De manera análoga a ese proceso natural para la resolución de problemas o prevención de posibles errores, se plantea que funcione un sistema utilizando el enfoque de Razonamiento Basado en Casos (RBC). El RBC es un área de la Inteligencia Artificial, que se preocupa por el estudio de los mecanismos mentales necesarios para repetir lo que se ha hecho o vivido con anterioridad, ya sea por uno mismo o ya sea por casos concretos recopilados en la bibliografía o en la sabiduría popular [Aamodt01].

El RBC toma problemas similares ocurridos en el pasado, a los que denomina casos (estructuras del tipo *si x, entonces y*), para encontrar soluciones a los mismos, modificar soluciones existentes y explicar situaciones anómalas [Delany01]. Captura las características de dicho problema, busca casos pasados con valores similares para dichas características, analiza las soluciones de estos casos y propone una solución al problema. Se puede aprender del problema actual para problemas futuros.

El aprendizaje de los casos en un sistema de Razonamiento Basado en Casos requiere métodos para extraer el conocimiento relevante de la experiencia, integrar el caso en la estructura del conocimiento, e indexar el problema para ser seleccionado en casos similares. Cuando es implementado con éxito incrementa la eficiencia y reduce sustancialmente los costos, automatizando procesos como diagnósticos médicos, calendarización y diseño [Shepperd01]. El aprendizaje no se implementa en el Razonador Basado en Casos del presente proyecto debido a que es una labor que requiere tiempo y era necesario delimitar el alcance del proyecto.

Cabe mencionar que diversas organizaciones como IBM, VISA Internacional, Volkswagen, British Airways y la NASA, han utilizado el RBC en aplicaciones tales como soporte a clientes, aseguramiento de la calidad, mantenimiento aéreo, planeación y soporte en la toma de decisiones [Schank01].

2.6.2 Haciendo historia

El Razonamiento Basado en Casos fue concebido a finales de los años 70 (1977) con los trabajos de Roger Schank y se basa en la premisa fundamental de que los problemas similares son mejor resueltos con soluciones similares [Leake01]; la idea es aprender de la experiencia.

A principios de los 80's Janet Kolodner y sus estudiantes comenzaron a construir el primer sistema que utilizara este enfoque: CYRUS [Kolodner01]. Luego, como lo citan Agnar Aamodt y Enric Plaza [Aamodt01], la investigación del RBC no se restringió sólo a los EUA: surgieron trabajos europeos como el del grupo de Derek Sleeman de Aberdeen en Escocia y casi al mismo tiempo Keane de la universidad de la Trinidad de Dublín emprendió la investigación de la ciencia cognitiva en el razonamiento analógico que ha influenciado al RBC.

Hasta la fecha se han desarrollado numerosos sistemas que utilizan el RBC, como lo demuestran los casi 70 sistemas mencionados por Janet Kolodner [Kolodner01] y la extensa bibliografía en Internet.

2.6.3 ¿Qué es el Razonamiento Basado en Casos?

> *El Razonamiento Basado en Casos es una técnica para manejar y utilizar el conocimiento, que puede estar organizado como abstracciones discretas de eventos o entidades, que están limitados en tiempo y espacio (casos)* [Shepperd01].

En el Razonamiento Basado en Casos, un razonador recuerda situaciones previas, similares a la que tiene en el momento y las utiliza para resolver el nuevo problema. Los casos recordados son usados para sugerir una manera de solucionar el nuevo problema, para sugerir una manera de adaptar una solución que no es adecuada del todo, para advertir de posibles fallas o para interpretar una situación.

Difiere de la mayoría de las otras técnicas de la Inteligencia Artificial (IA) ya que no es basada en modelos. Esto quiere decir que al contrario de los enfoques basados en conocimiento que usan reglas, el desarrollador no tiene que definir explícitamente causalidades ni relaciones entre el dominio de interés [Kolodner01].

Si se observa la manera en que las personas solucionan los problemas, es fácil darse cuenta de que el RBC se utiliza en todo nuestro alrededor (por ejemplo al ordenar un platillo en un restaurante, probablemente basemos nuestra decisión para la comida a partir de experiencias pasadas en ese, o en restaurantes del mismo tipo). En general, la segunda vez que resolvemos algún problema o desempeñamos una tarea, es más fácil que la primera porque recordamos o repetimos la solución previa. También somos más competentes la segunda vez porque recordamos nuestros errores y generalmente tomamos otra vía para evitar cometerlos nuevamente.

Las siguientes premisas delinean brevemente el modelo RBC [Kolodner01].

- Hacer referencia a casos pasados provee ventajas, sobre todo en situaciones recurrentes.

- Entender e interpretar una situación es parte necesaria del ciclo de razonamiento, además son prerrequisitos para solucionar un problema y co-requisitos durante la resolución del mismo. Cualquier forma de razonamiento requiere que la situación sea elaborada con suficiente detalle y representada con suficiente claridad y con vocabulario apropiado, para permitir al razonador reconocer el conocimiento que necesita para razonar acerca de eso.

- Como los casos viejos no son exactamente iguales al nuevo, es necesario *adaptar* la situación antigua para que se ajuste a la nueva.

- El aprendizaje ocurre como una consecuencia natural del razonamiento. Si un procedimiento nuevo es derivado en el curso de solucionar un problema complejo y todo va bien durante su ejecución, entonces, en efecto, se ha aprendido un nuevo procedimiento para lidiar con esta nueva clase de situaciones. Si al contrario, se

encontraron problemas al utilizar este caso nuevo en una situación nueva, el razonador estará advertido de que el procedimiento en el caso es fallido o que sus índices (que representan su rango de aplicación) son imprecisos. Se hace un esfuerzo para analizar los resultados de la nueva situación y para solucionar sus problemas. La nueva situación almacenada en la biblioteca de casos envuelve un refinamiento o modificación del conocimiento encontrado en el caso original, y sus índices designan cuándo es útil.

- La retroalimentación y su análisis a través de procedimientos consecuentes y razonamiento explicativo, son partes necesarias del ciclo completo de razonamiento/aprendizaje. Los procedimientos consecuentes incluyen explicaciones de fallas y el intento de repararlas.

Los sistemas de Razonamiento Basado en Casos se utilizan cuando los problemas se representan mejor en forma de conocimiento del medio que en forma de reglas, cuando es más fácil extraer conocimiento mediante casos ya ocurridos que mediante reglas, cuando es necesario incorporar creatividad y sentido común para resolver el problema, y cuando se repiten los problemas cada cierto tiempo.

La intuición en el Razonamiento Basado en Casos es que las situaciones recurren con regularidad, lo que fue hecho en alguna situación probablemente puede ser aplicado en una situación similar y si sabemos que funcionó en esa situación previa similar a una nueva, tendremos con qué empezar el razonamiento acerca de la nueva situación. Al pensar en los procesos y actividades que sugiere el MoProSoft encontramos que, como todo modelo, son todas iguales para cualquier organización, obviamente se pueden encontrar excepciones, pero la premisa es que con el Razonamiento Basado en Casos se pueden almacenar experiencias para cualquiera de las actividades del modelo, tomando en cuenta roles, responsabilidades y demás, y ser sugeridas en cualquier empresa que pretenda seguir al modelo de procesos con la seguridad de que serán de utilidad.

Asimismo, el Razonamiento Basado en Casos también puede servir como memoria corporativa que almacena todas las experiencias que ocurran dentro del negocio, lo que constituye una gran base de conocimiento valiosa para cualquier organización.

2.6.4 El ciclo del RBC

Según Agnar Aamodt y Enric Plaza, en el nivel más alto de generalidad el ciclo de un sistema RBC general se describe por los cuatro procesos siguientes (llamado las 4R's) [Aamodt01]:

1. RECUPERAR (*Retrieve*) el o los casos más parecidos al problema actual. Para ello el sistema utiliza la biblioteca de casos.

2. REUTILIZAR (*Reuse*) la información y el conocimiento de dicha biblioteca de casos para intentar resolver el problema.

3. REVISAR (*Revise*) la solución propuesta.

4. RETENER (*Retain*) la parte útil de esta experiencia para ser utilizada en la resolución de futuros problemas.

Un nuevo problema es resuelto *recuperando* uno o más de los casos almacenados, *reutilizando* el caso (o casos) de alguna manera, *revisando* la solución basada en la reutilización de un caso anterior, y *reteniendo* la nueva experiencia incorporándola en la base de conocimiento existente. Cada uno de los cuatro procesos involucra más pasos específicos, pero eso será descrito en el capítulo 4, dedicado a la construcción del razonador basado en casos. En la figura 2.6 se muestra gráficamente el ciclo RBC.

La habilidad para comprender un problema nuevo en términos de experiencias viejas tiene dos partes: *recuperación* de experiencias pasadas/anteriores e *interpretar* o *reutilizar* la nueva situación en términos de las experiencias recuperadas. Al primero se le llama *el problema de indexación*, que en términos generales significa encontrar en la memoria la experiencia más cercana a la nueva situación. La apropiada recuperación de los casos es el corazón del RBC.

La *reutilización* es el proceso de comparar la situación nueva con las experiencias recuperadas. Cuando las soluciones a los problemas son comparadas con las soluciones antiguas, el razonador gana un entendimiento de los pros y contras de hacer algo de una manera particular.

La *revisión* o *adaptación* es el proceso de modificar una solución anterior para que satisfaga las demandas de la nueva situación. Se han identificado numerosos métodos de adaptación [Kolodner01] y que se pueden usar para insertar, eliminar o sustituir una parte de la solución recuperada.

Una de las claves de un razonador basado en casos es su habilidad de aprender de sus experiencias, y para llevarlo a cabo el razonador requiere de retroalimentación, de modo que pueda interpretar qué estuvo bien y qué estuvo mal en sus soluciones y *retener* los casos útiles. Sin retroalimentación probablemente el razonador sea más rápido al solucionar los problemas, pero repetirá sus errores y nunca incrementará sus capacidades [Plaza01].

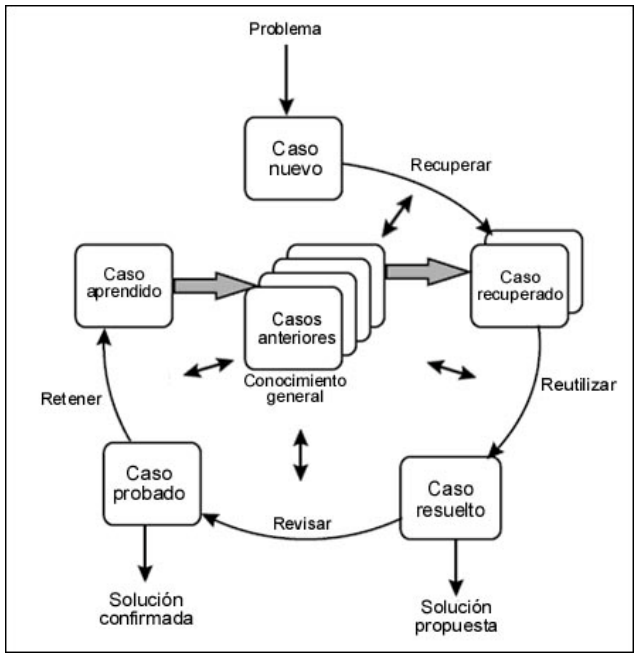

Figura 2.6: El ciclo del Razonamiento Basado en Casos, adaptación de [Aamodt01]

2.6.5 Tipos de Sistemas con Razonamiento Basado en Casos

Hay diferentes tipos de sistemas de Razonamiento Basado en Casos que se pueden construir. Por un lado tenemos a los sistemas completamente automatizados y por el otro, a los sistemas de sólo-recuperación.

Los primeros son aquellos sistemas que resuelven los problemas por si solos y tienen medios para interactuar con el mundo exterior y recibir retroalimentación. Por su parte, los sistemas de sólo-recuperación trabajan interactivamente con el usuario para resolver los problemas, sirven para aumentar la memoria de las personas, proveyéndoles casos de los que probablemente no estarían al tanto, pero son ellas, las personas, las responsables de las decisiones importantes. Entre esos dos extremos hay variaciones de las tareas que los sistemas pueden realizar: dar información en respuesta a requisiciones, ser un navegador de casos y proveedor de información, jugar el papel de entrenador, supervisar las tareas de los humanos y proporcionarles guía y sugerencias, o actuar como un colega [Plaza01].

De acuerdo a los objetivos para la herramienta se seleccionaron las características del tipo de sistema que mejor se adecuó a nuestra situación y necesidades, resultando en un sistema de sólo-recuperación que provee información y proporciona guía y sugerencias a los usuarios, lo que se describe ampliamente en el capítulo 4.

Si quieres alcanzar lo más alto, empieza con lo más bajo.

–SYRUS

Capítulo 3

ANÁLISIS Y DISEÑO DEL AsistenteHIM

Este capítulo comprende las importantes fases de análisis y diseño aplicadas al desarrollo del AsistenteHIM por medio de la metodología MESSAGE (*Methodology for Engineering Systems of Software Agents*). Ello constituye la base para la implementación, pruebas e implantación del sistema, etapas siguientes del ciclo de desarrollo de software y que se abarcan en el capítulo 5.

3.1 MESSAGE (*Methodology for Engineering Systems of Software Agents*)

3.1.1 Introducción

Como se mencionó en el primer capítulo, el enfoque orientado a agentes proporciona distintas capacidades a los sistemas y es apropiado para una amplia gama de aplicaciones. Debido a eso en los últimos años no sólo está siendo utilizado por investigadores del área de Inteligencia Artificial, sino también por ingenieros de software encargados de desarrollar aplicaciones industriales y empresariales.

El desarrollo de sistemas multi-agente (SMA) por los ingenieros de software, involucra un notable aumento de complejidad y resulta evidente que necesita de la existencia de métodos y herramientas que faciliten la obtención de productos finales fiables. De tal suerte que en los últimos años se han emprendido numerosos esfuerzos para generar metodologías que contribuyan a la adopción del enfoque orientado a agentes y aprovechen todo el conocimiento generado en el área de Ingeniería de Software y particularmente en la de Procesos de Software. Tal es el caso de MESSAGE, metodología seleccionada para la realización de las etapas de análisis y diseño del ciclo de vida de desarrollo del SMA AsistenteHIM.

La riqueza de esta metodología radica en que incorpora técnicas de Ingeniería de Software (se basa en el Proceso Unificado de Desarrollo de Software [Jacobson01] y un enfoque de refinamiento), con los puntos clave de la teoría de IA en cuanto a agentes y SMA. Además, provee un lenguaje de modelado basado en UML, un método, y guías de cómo aplicar todo, centrándose en las fases de análisis y diseño.

La metodología está constituida por una serie de documentos que ayudaron a llevar a cabo las etapas de desarrollo del AsistenteHIM (Requerimientos, Análisis, Diseño, Implementación, Pruebas y Desarrollo): 1) Metodología Inicial, 2) Líneas guía, 3) La metodología, y 4) Recomendaciones y herramientas de soporte. Cada documento está constituido por información importante que se podría tratar ampliamente en el desarrollo del presente proyecto de tesis, sin embargo, el objetivo principal no es la aplicación de esta metodología, sino el desarrollo del SMA de manera que se incorporen algunas prácticas de la Ingeniería de Software, y demostrar que el enfoque orientado a agentes solucionó el problema inicial descrito en el capítulo 1. Así pues, se puede resumir la utilización de MESSAGE como sigue:

En la etapa inicial la metodología apoyó con guías para identificar si la necesidad que teníamos podía ser resuelta por el enfoque orientado a agentes o no, concluyendo en que sí, tema debidamente explicado en el primer capítulo. Posteriormente, se llevaron a cabo las actividades que marcan las etapas de análisis y diseño en la metodología, las cuales son tratadas a continuación en este mismo capítulo. Por último, MESSAGE proporcionó guías para el desarrollo, implementación y pruebas del sistema multi-agente.

Para apoyar a la generación de cada uno de los modelos indicados en las fases de análisis y diseño con la notación adecuada, MESSAGE propone hacer uso de herramientas para diagramación con UML (por ejemplo *Rational Rose*) o la herramienta *MetaEdit+* (http://www.metacase.com/). En este proyecto se utilizó MetaEdit+ 4.0 ya que se proveen bibliotecas para hacerla una herramienta CASE (Ingeniería de Software ayudada por computadora, *Computer-Aided Software Engineering*) orientada a agentes que soporta la notación de MESSAGE y su manejo es bastante sencillo. En la figura 3.1 se muestra la interfaz

de usuario de MetaEdit+, para más información (características, instalación, manejo, etc.) se puede consultar el cuarto documento que conforma la metodología [Caire02].

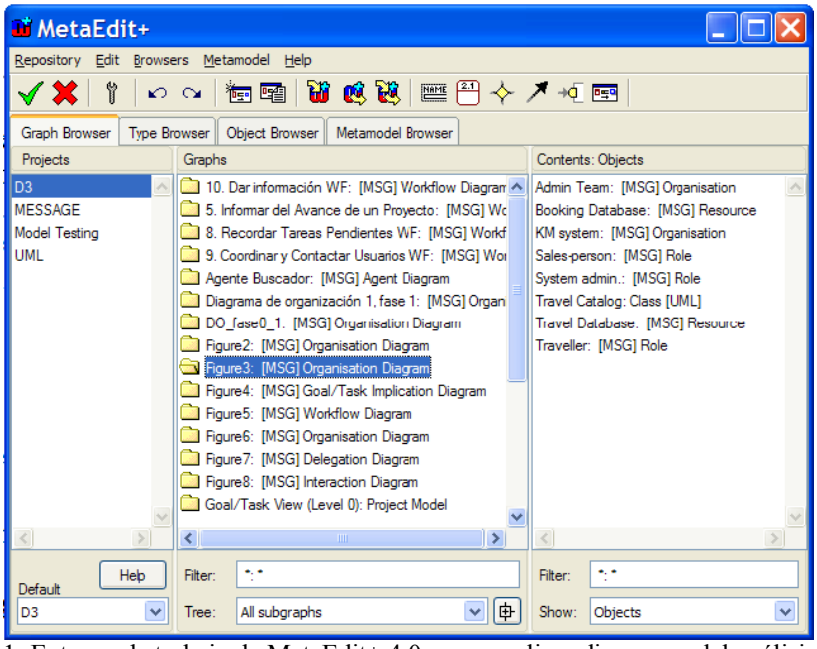

Figura 3.1: Entorno de trabajo de MetaEdit+ 4.0, para realizar diagramas del análisis y diseño

3.1.2 Conceptos fundamentales en MESSAGE

Conceptualmente muchas propuestas introducen a lo largo de su desarrollo términos muy similares. La contribución de MESSAGE es la agregación de los conceptos de la teoría de agentes a nivel práctico y en diagramas, con la finalidad de plasmar dichos conceptos en un modelo de análisis basado en UML, comprensible para los ingenieros de software encargados del desarrollo de sistemas.

La mayoría de los conceptos a nivel de conocimiento se ubican en tres categorías principales: *EntidadConcreta, Actividad* y *EntidadDeEstadoMental.*

Los tipos principales de la categoría *EntidadConcreta* son agente, organización, rol y recurso.

En MESSAGE, un **agente** es una entidad atómica autónoma, capaz de realizar alguna función útil. La capacidad funcional es referida como el servicio del agente, el cual es el análogo de *operación* en un objeto (Programación Orientada a Objetos, POO). La autonomía se refiere a que las acciones del agente no sólo son dictadas por eventos o interacciones externas, sino también por su propia motivación, a la cual se le llama *propósito*.

Una **organización** es un grupo de agentes trabajando para un propósito común. Es una entidad virtual ya que en el sistema no se encuentra dentro de ninguna entidad computacional tal como organización. Se estructura a través de relaciones de poder entre las partes constituyentes (superior-subordinado).

Rol es un concepto empleado sobre todo para indicar posibles cambios de conducta de un agente. La distinción entre rol y agente es análoga a la de interfaz y clase (POO): rol (interfaz) describe las características externas de un agente en un contexto particular, mientras que el concepto de agente (clase) describe las características internas de ese agente. Un agente puede ser capaz de jugar varios roles a la vez, o un rol puede ser jugado por varios agentes.

Un **recurso** es usado para representar entidades no autónomas como bases de datos o programas externos usados por los agentes.

Los tipos principales de la categoría *Actividad* son tarea, interacción y protocolo de interacción.

Una **tarea** es una unidad de actividad a nivel de conocimiento con un realizador principal. Tiene un conjunto de pares de situaciones que describen las precondiciones y poscondiciones necesarias para su realización. Además, las tareas pueden estar compuestas por sub-tareas, que están ligadas entre ellas y constituyen dichas condiciones. El diagrama de actividades de UML puede ser utilizado para modelar este tipo de concepto.

Una **interacción** es empleada para modelar el comportamiento social de los agentes. En MESSAGE existen modelos o vistas específicos para modelar las interacciones existiendo bastante similitud en el significado otorgado a este término: Una interacción tiene más de un participante y un propósito que se busca alcanzar por dichos participantes. Un **protocolo de interacción** define un patrón de intercambio de mensajes, asociado con una interacción.

El **objetivo** es un tipo de la categoría *EntidadDeEstadoMental*, que asocia un agente con una situación (estímulos que se presentan en el entorno), y algunas veces son derivados de sus propósitos (el entorno cambia debido a las acciones del agente en el). Es útil expresar los propósitos en términos de una función de utilidad que asocia "valores buenos" con situaciones.

Otros dos conceptos importantes son la *EntidadDeInformación*, que se refiere a un objeto que encapsula un trozo de información, y el **mensaje**, que es un objeto comunicado entre agentes. Los atributos de un mensaje especifican el remitente, el destinatario, un acto del habla (que sitúa dentro de una categoría, las intenciones del remitente) y un contenido (*EntidadDeInformación*).

La figura 3.2 reúne los conceptos anteriores en lo que se llama *metamodelo* de MESSAGE, el cual es un modelo que sintetiza los conceptos más importantes de la metodología y sus relaciones, lo que se espera sea comprendido al término del capítulo.

En las siguientes dos secciones del capítulo se llevan a cabo las etapas de análisis y diseño de MESSAGE, aplicadas al desarrollo del sistema AsistenteHIM. Se utilizaron los conceptos definidos anteriormente y la notación indicada por MESSAGE (apéndice 1).

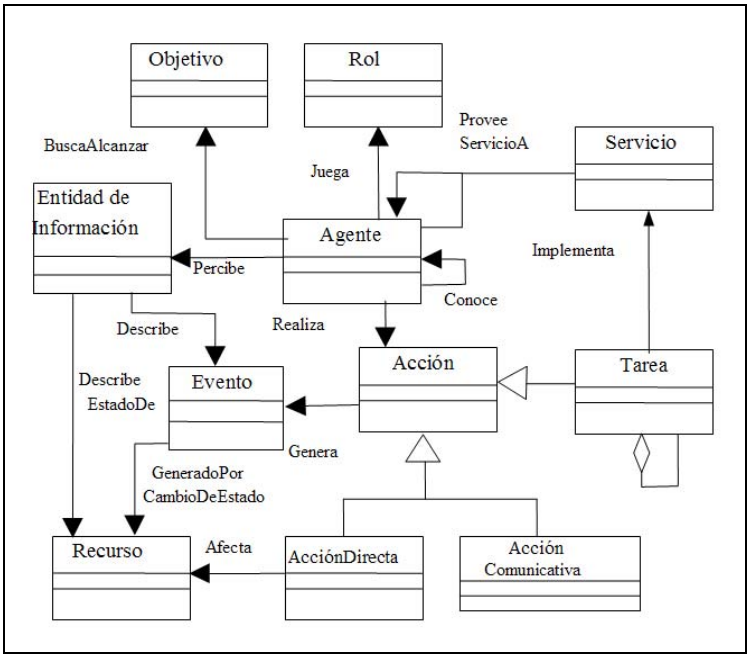

Figura 3.2: *Metamodelo* de MESSAGE, adaptado de [Caire01]

3.2 ANÁLISIS

El propósito del análisis es producir una especificación del sistema en desarrollo. Es una interpretación del problema a ser resuelto representada como un modelo abstracto cuyo propósito es [Caire02]:

- Comprender mejor el problema, ya que al inicio del proceso de desarrollo los requerimientos no son claros, son incompletos y redundantes. Es muy importante entonces clarificar y formalizar los requerimientos funcionales (lo que el sistema tiene que hacer) y los no funcionales (el desempeño que deberá tener el sistema) antes de diseñar una solución.

- Para confirmar cuál es el problema a resolver (validación). Una vez que los requerimientos del sistema han sido clarificados, es posible identificar si un sistema que cumpla con dichos requerimientos podrá satisfacer las necesidades de las diferentes personas que utilizarán el sistema.

- Para facilitar el diseño de la solución. El modelo del análisis constituye el punto de inicio para la actividad de diseño, proveyendo una descripción detallada de la funcionalidad del sistema y una vista completa de su estructura. Utilizar el análisis orientado a agentes de MESSAGE, nos lleva a un diseño orientado a agentes y por consiguiente a una programación orientada a agentes que da como resultado un sistema multi-agente.

3.2.1 Definición de requerimientos

El primer paso en el análisis es la definición de requerimientos (en lenguaje natural o diagramas) del sistema a desarrollar, y documentación que incluya una breve descripción de cómo están

estructuradas las organizaciones donde será implantado el sistema (descripción de objetivos, actividades llevadas a cabo en términos de flujos de trabajo, terminología y tipo de información utilizada).

En los temas desarrollados en el capítulo 1 se especificaron ampliamente los conceptos principales para entender el presente proyecto de tesis: antecedentes y descripción del problema, solución propuesta, algunas justificaciones y los objetivos que se buscan con el desarrollo del AsistenteHIM.

En síntesis, el proyecto involucra el desarrollo de un sistema multi-agente que incorporado a la HIM, sirva como guía y supervisor del seguimiento de los procesos del MoProSoft. El SMA debe explotar el marco de trabajo generado en RDF para la HIM, utilizando a su vez el enfoque de Razonamiento Basado en Casos (RBC) para la solución de problemas.

Es muy importante mencionar que por cuestiones de tiempo, el desarrollo de este proyecto de tesis se ha limitado en alcance. Se consideró brindar apoyo sólo al primer proceso del modelo, Gestión de Negocio (GN), por ser el más completo en la HIM hasta la fecha, además de tener características que facilitan el desarrollo de un SMA de tamaño adecuado para probar la solución propuesta. Así pues, los requerimientos, análisis, diseño, desarrollo, implementación y pruebas del AsistenteHIM, sólo abarcan dicho proceso de GN.

El objetivo de la herramienta de guía y supervisión es maximizar la efectividad de los usuarios participantes en los procesos de MoProSoft dentro de una empresa que cuente con un sistema de apoyo al modelo de procesos. Para alcanzar dicho objetivo y de acuerdo a lo especificado anteriormente, se identificaron los siguientes requerimientos para el sistema AsistenteHIM.

1. Integrar en un SMA aspectos de la Ingeniería de Software, la Ingeniería de Procesos de Software, la Teoría de Agentes y el enfoque de Razonamiento Basado en Casos, de acuerdo a lo especificado en el capítulo 1.

2. De acuerdo al MoProSoft, la herramienta debe ser capaz de llevar un control (al menos parcial) del flujo de trabajo de los usuarios.

3. Proveer información acerca de los roles que tienen asignados los usuarios, sus capacidades y responsabilidades.

4. Indicar al usuario las actividades que tiene que realizar, es decir, su lista de tareas pendientes.

5. Introducir algunas buenas prácticas de la Ingeniería de Software brindando sugerencias de la manera en que se deben realizar algunas actividades y dar soporte a problemas, dando ideas preventivas o sugerencias (RBC).

6. Ser más eficiente al contar con un elemento que le permita adaptarse a las diferentes formas de realizar las actividades de cada empresa u organización (RBC).

7. Dar información de las actividades y los productos.

8. Proporcionar ayuda para contactar usuarios.

9. Realizar cada uno de los puntos anteriores siempre tomando en cuenta el rol del usuario al que se le está brindando el servicio y las capacidades que tiene asociadas de acuerdo al modelo de procesos MoProSoft.

10. Tener interacción con el usuario a través de la interfaz de usuario de HIM.

11. Que pueda acoplarse con HIM para poder llevar a cabo todos los requerimientos mencionados, aprovechando el trabajo de generaciones pasadas.

12. Utilizar el marco de trabajo desarrollado en RDF.

13. Ser un software libre, abierto y que cumpla con las características de calidad: correcto, confiable, robusto, eficiente, usable, verificable, mantenible, reparable, evolucionable, reusable, portable, entendible e interoperable [Oktaba02].

3.2.2 Análisis

La actividad del análisis involucra el desarrollo y mantenimiento del modelo del sistema multi-agente y su entorno a un nivel de abstracción alto. Su propósito es comprender el problema en términos de distribución de la solución [Evans01], además de producir un modelo del sistema a ser desarrollado y su entorno, que esté de acuerdo entre el analista y el cliente.

Utilizando el *metamodelo* de MESSAGE y dado que el análisis orientado a agentes emplea un amplio conjunto de conceptos, la metodología indica generar un número de sub-modelos distribuidos en 2 niveles, que pongan énfasis en diferentes aspectos del modelo completo para comprenderlo mejor. Este conjunto de modelos es el siguiente:

- Modelo del Dominio (MD)
- Modelo de Agente (MA)
- Modelo de la Organización (MO)
- Modelo de Objetivo/Tarea (MO/T)
- Modelo de Interacción (MI)

Hay distintos enfoques para desarrollar los modelos anteriores y se utilizan de acuerdo al objetivo que se busque. Los enfoques a elegir son a) descomposición basada en objetivos, b) descomposición basada en tareas y c) descomposición orientada a responsabilidades [Evans01]. El modelo de descomposición que se siguió es el basado en objetivos y se detalla a continuación.

1. Se desarrollaron primero el modelo de la organización y el modelo de objetivo/tarea, ligando sus contenidos de tal manera que hubiera una correspondencia entre los sub-objetivos y los propósitos de los agentes, así como las sub-tareas y los servicios que los agentes pudieran proveer. Este proceso identificó los agentes y sus características principales.

2. Los detalles de los agentes fueron elaborados en el Modelo del Agente.

3. Para alcanzar objetivos de más alto nivel los agentes debían cooperar, lo cual derivó en el desarrollo del Modelo de Interacción.

4. En paralelo a las actividades anteriores, se fue desarrollando el Modelo del Dominio.

La metodología especifica que el diseño de los diagramas de cada modelo se realiza en fases, las cuales conforman un proceso de refinamiento ya que se pueden repetir hasta alcanzar el nivel de detalle requerido por medio de ciclos de mejora continua y deben ser consistentes entre ellas.

En el proyecto se llevaron a cabo dos fases: La fase 0 es donde se utilizó la información básica como descripción de requerimientos, contexto, procesos de negocio de la empresa u organización donde se utilizará el sistema, etc. Los modelos de esta fase dan una vista general del sistema, su entorno y su funcionalidad global. El detalle del nivel 0 se centra en la identificación de entidades y sus relaciones de acuerdo al *metamodelo* [Caire01].

En la fase 1 se detallaron cada uno de los diagramas desarrollados en la fase 0 y se generaron algunos otros que involucran entidades más complejas. La estructura y comportamiento de entidades como organización, agentes, tareas y objetivos fueron definidos.

3.2.2.1 Construcción de los modelos, fase 0

3.2.2.1.1 Modelo de la Organización (MO)

El Modelo de la Organización busca definir la estructura y el comportamiento de un grupo de agentes (en el sistema multi-agente y en la organización externa) trabajando juntos para alcanzar objetivos comunes [Caire01]. El MO generado representa la estructura de las organizaciones (donde se prevé se implante el AsistenteHIM), en términos de departamentos, divisiones, participantes, etc. En este modelo se ve al sistema a desarrollar como una caja negra y se enfoca en sus relaciones con otras entidades y su ambiente.

Lo primero a tener en cuenta en el diseño de este modelo son las entidades externas significativas para el cumplimiento de los requerimientos, esto es, el entorno, los usuarios y los recursos [EURESCOM02].

Entorno

Para la especificación correcta del entorno del sistema, es necesario diferenciar entre el aspecto referido como entorno ya que se considera que hay dos enfoques a definir:

La herramienta será utilizada en cualquier empresa desarrolladora de software que lleve a cabo las actividades y procesos del MoProSoft, por lo que se puede considerar como entorno a dicha empresa, la cual por estar siguiendo el modelo de procesos citado, deberá tener una estructura igual o muy similar a la que se observa en la figura 3.3, es decir, responsables divididos en las áreas de Dirección, Gestión y Operación, con sus respectivas sub-áreas, cada uno con responsabilidades particulares.

Por otro lado, se tiene como entorno a los sistemas de apoyo al seguimiento del MoProSoft, ya que son el puente de interacción con el usuario. En este caso se define a la HIM como el entorno del sistema.

Tomando los conceptos descritos en el capítulo 2, se identificaron las siguientes características útiles en la especificación del enfoque de agentes.

El entorno es:

- *Totalmente Observable* dado que el sistema es capaz de percibir el estado completo del entorno en todo momento.
- *Determinista* en mayor parte porque, casi siempre, el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el usuario y el agente.
- *Secuencial* porque las acciones que se lleven a cabo en determinado momento tienen repercusiones para la realización de otras actividades del mismo u otros usuarios.
- *Estático,* el entorno no cambia con el paso del tiempo.
- *Continuo* ya que conforme va pasando el tiempo, las percepciones y acciones del sistema van cambiando.

Usuarios

Los usuarios son cualquier persona que labore en la empresa de desarrollo de software que siga los procesos y actividades del MoProSoft, que tenga asignado uno o varios roles y con ello capacidades, restricciones y tareas específicas.

Debido a que se consideró el proceso de Gestión de Negocio, los usuarios específicos para la herramienta son los que tengan asignados los roles de Responsable de Gestión de Negocios, Grupo Directivo (pueden ser varios) y Grupo de Gestión (pueden ser varios).

Recursos

Se identificaron tres recursos que el sistema utiliza, controla o recibe alguna entrada:

- El repositorio que contiene los archivos RDF, para describir el MoProSoft y que sirven de fuente de información para la HIM.
- La base de conocimiento de la organización, donde se almacena toda la información relacionada con el modelo de procesos y los productos desarrollados, a excepción del código.
- Repositorio donde se tienen almacenados los casos (biblioteca de casos), información utilizada por el enfoque de Razonamiento Basado en Casos.

A partir de lo anterior se generaron los siguientes diagramas de organización que modelan el AsistenteHIM al nivel más alto de abstracción.

En el diagrama de organización de la figura 3.3, se modela la empresa de desarrollo de software donde se implantará el sistema. Como dicha empresa estará siguiendo el MoProSoft, se espera que tenga una estructura similar a la del diagrama (con más elementos probablemente). La

empresa estará conformada por tres departamentos, Alta Dirección, Gestión y Operación, cada uno dividido en áreas de procesos, que a su vez estarán conformadas ya sea por sub-áreas de procesos o por uno o varios roles participantes. Se espera además que la empresa tenga al menos una base de conocimiento o repositorio para almacenar información y productos.

Nótese que se modela a HIM también como organización porque para implantar la herramienta de guía y supervisión, antes debe estar siendo utilizado algún sistema de apoyo a MoProSoft. Cabe mencionar nuevamente que se utiliza HIM como ejemplo en el presente proyecto pero la intención es que se pueda sustituir por cualquier otro sistema diseñado para el mismo fin, obviamente haciendo los ajustes necesarios.

En la figura 3.4 se simplifica la estructura ya que sólo se trabajará con la categoría de Alta Dirección. La base de conocimiento y HIM siguen presentes y se detalla más la composición de la organización de Gestión de Negocio, que está conformada por tres tipos de roles, Responsable de Gestión de Negocio (RGN), Grupo Directivo (GD) y Grupo de Gestión (GG). Dentro del Grupo Directivo pueden participar una o varias personas y dentro del Grupo de Gestión se consideran a tres roles importantes: Responsable de Gestión de Procesos (RGP), Responsable de Gestión de Proyectos (RGPY) y Responsable de Gestión de Recursos (RGR).

En el diagrama de la figura 3.5 se aprecia una vista más centrada en el sistema. El AsistenteHIM tiene conocimiento de la HIM, de los roles y asociaciones con los repositorios: recupera información de los archivos RDF y ontologías, almacenados en la base de conocimiento (Configuración y Repositorio). El AsistenteHIM también tiene asociación con la biblioteca de casos, que contiene los casos recolectados para utilizar el enfoque de Razonamiento Basado en Casos que servirá para dar sugerencias de la manera en que se deben llevar a cabo ciertas actividades y para prevención de problemas.

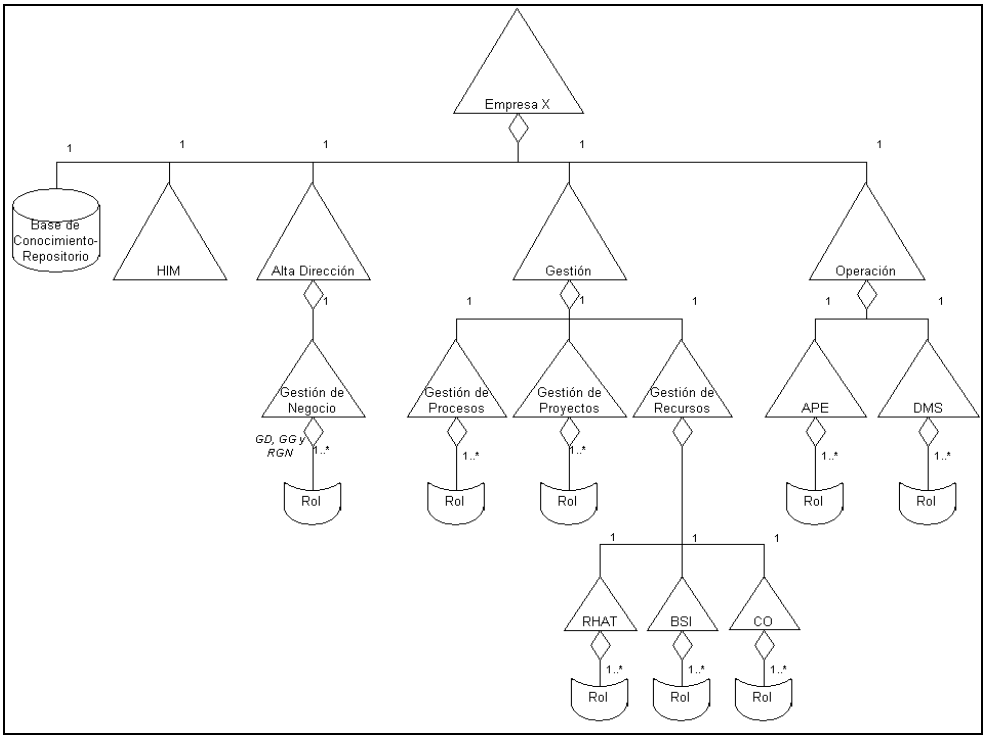

Figura 3.3. Diagrama de Organización, Fase 0 (relaciones estructurales)

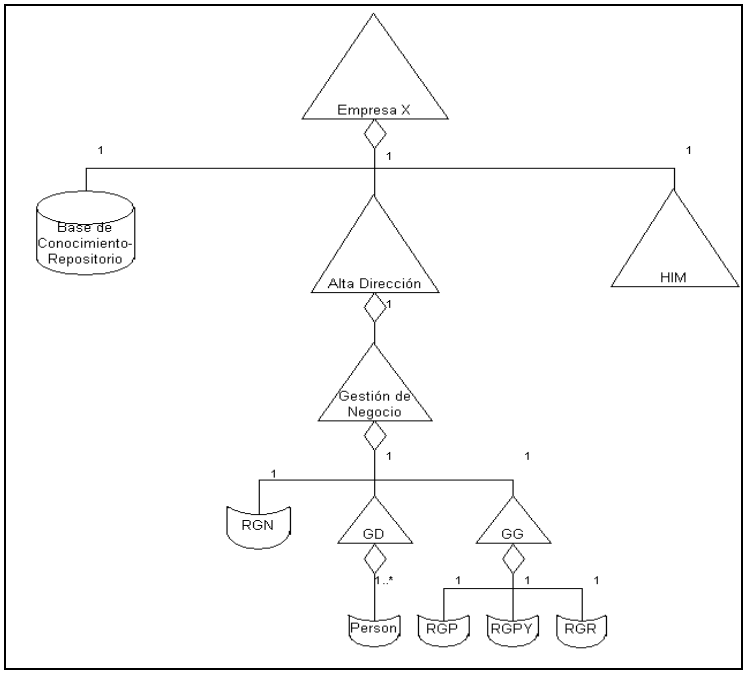

Figura 3.4. Diagrama de Organización, Fase 0, simplificado

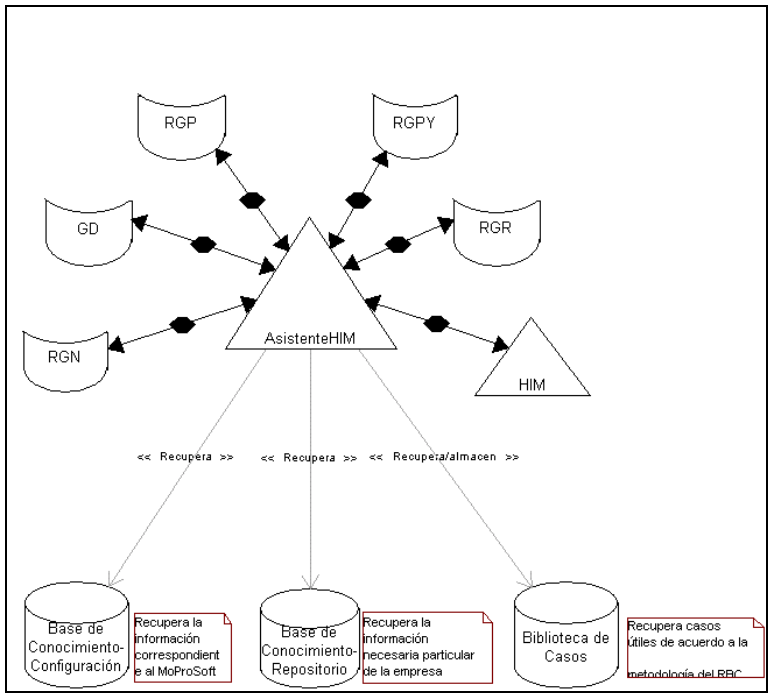

Figura 3.5. Diagrama de Organización, Fase 0 (relaciones de conocimiento)

3.2.2.1.2 Modelo Objetivo/Tarea (MO/T)

En el nivel 0 el modelo Objetivo/Tarea representa el contexto en el cual operará el SMA y contiene un conjunto completo de objetivos y tareas de la organización en la que será implantado.

El modelo Objetivo/Tarea describe cómo los objetivos de alto nivel (por ejemplo, las responsabilidades de un agente) son descompuestos en objetivos de bajo nivel (aquellos que se pueden asignar a agentes constitutivos). Se comienza a desarrollar tomando como nodo raíz el objetivo principal del sistema y descomponiéndolo en sub-objetivos hasta que éstos puedan ser asignados a un agente en particular, es decir, asignados como su propósito de agente.

La descomposición de objetivos es una técnica que puede ayudar a expresar el propósito del sistema de una manera más clara y concreta, e identificar de manera natural los agentes involucrados o necesarios a desarrollar. El resultado es un árbol cuyas raíces son los objetivos de alto nivel y las hojas, los de bajo nivel.

El diagrama de la figura 3.6 muestra el objetivo principal del AsistenteHIM: maximizar la efectividad de los usuarios y su nivel de aprendizaje y satisfacción con el MoProSoft.

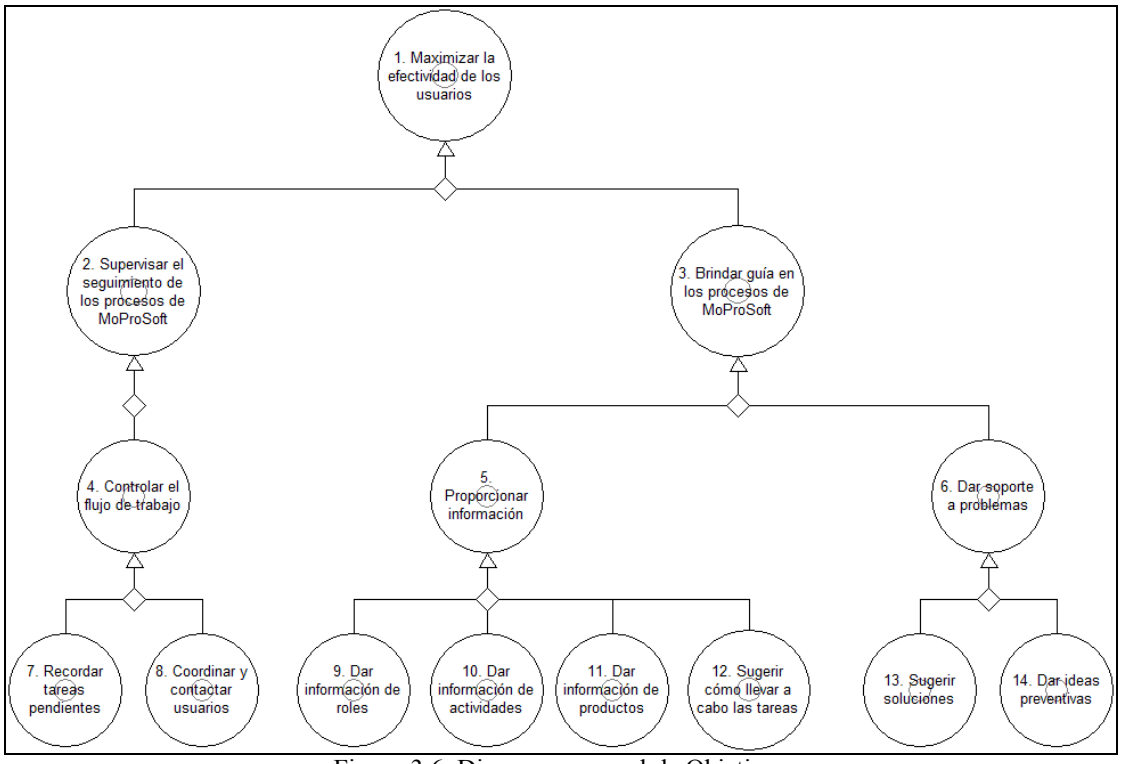

Figura 3.6. Diagrama general de Objetivos

3.2.2.1.3 Modelo de Agente (MA)

El modelo de agente consiste en modelar los agentes (o roles) identificados en el modelo de objetivos, los cuales tienen atributos como el propósito, relaciones, comportamiento, etc. En la fase 0 se representa con descripciones sencillas de los agentes, generados a partir de la descomposición de objetivos hecha en el modelo Objetivo/Tarea, donde los sub-objetivos del sistema localizados en las hojas del árbol general pueden ser asignados a los agentes. Se toma en cuenta el diagrama de organización de la fase 0 y sirve de entrada al mismo, pero de la fase 1.

Se identificaron cuatro agentes para el sistema: *agenteCoordinador, agenteSupervisor, agenteRBC y agenteBuscador*.

Agente Coordinador:

- Es el que proporciona la interfaz entre el usuario y el sistema multi-agente, mediante la inclusión de algunos elementos en la capa de vista de la HIM (capítulo 5).

- Observa y está atento a las necesidades del usuario y cuando se requiere llama a los agentes *Supervisor, Buscador y agenteRBC.*

En la figura 3.7 se muestra el diagrama de agente del agente coordinador, donde los círculos externos representan sus objetivos y los polígonos los servicios que utiliza de otros agentes.

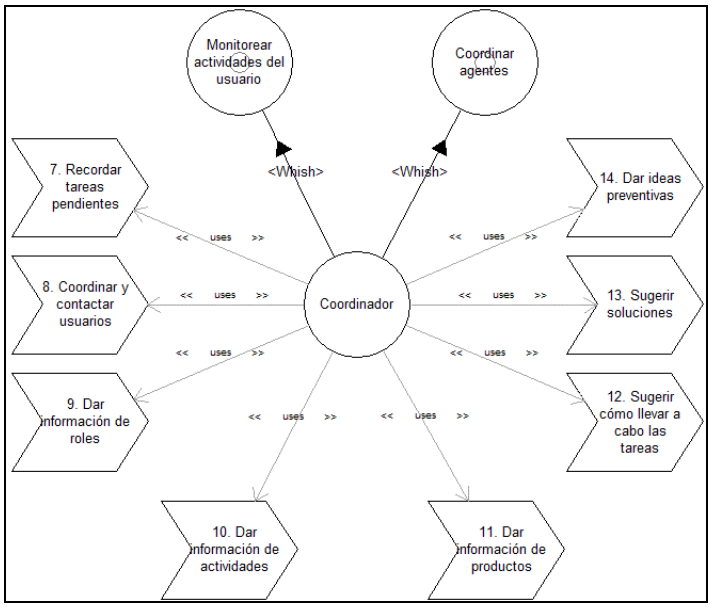

Figura 3.7. Diagrama de agente (*agenteCoordinador*)

Agente Supervisor:

- Controla el flujo de trabajo e implementa el servicio de notificación de tareas pendientes.

- Implementa el servicio de comunicación para proveer los medios de comunicación entre los usuarios.

- Junto con el servicio de comunicación, abarca las dependencias de las actividades que se están llevando a cabo, con sus roles relacionados.

- Utiliza el servicio de información solicitada del *agenteBuscador.*

En la figura 3.8 se observa el diagrama de agente del agente supervisor, donde también están representados los objetivos del agente y los servicios que utiliza (el de brindar la información solicitada y que es provisto por el agente buscador). Los servicios 7 y 8 son los que provee el agente supervisor al agente coordinador.

En los diagramas siguientes se maneja la misma notación, en la que los servicios pueden ser provistos (relación de provisión) o utilizados (relación de uso).

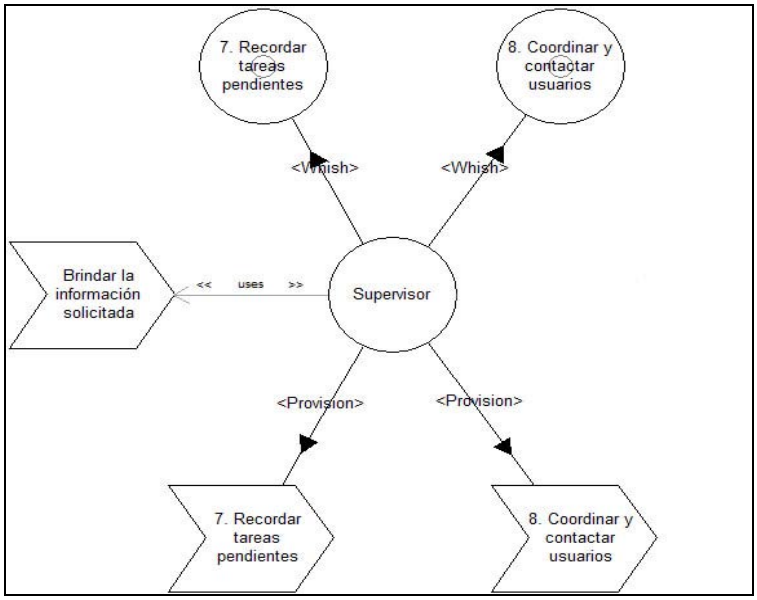

Figura 3.8. Diagrama de agente (*agenteSupervisor*)

Agente RBC:

- Utiliza el enfoque de RBC para proporcionar los servicios de sugerir la manera de llevar a cabo tareas, sugerencias y prevención de problemas (figura 3.9).
- Involucra todo lo relacionado con la construcción de un razonador basado en casos, tema a tratar en el capítulo siguiente.
- Utiliza el servicio de información solicitada del *agenteBuscador* (figura 3.9).

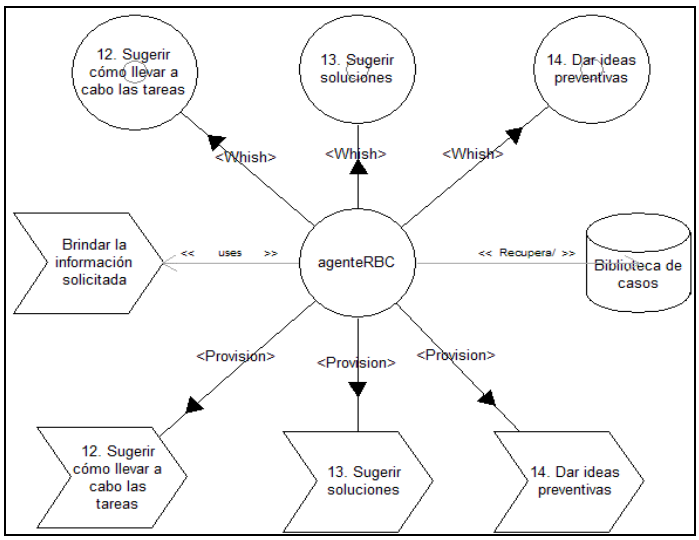

Figura 3.9. Diagrama de agente (*agenteRBC*)

Agente Buscador:

- Haciendo uso de la base de conocimiento en RDF, proporciona de manera eficiente el servicio de consulta de información requerida por los demás agentes (figura 3.10).

- Permite implementar el servicio informativo para proporcionarles a los usuarios información importante de sus roles, actividades y productos (figura 3.10).

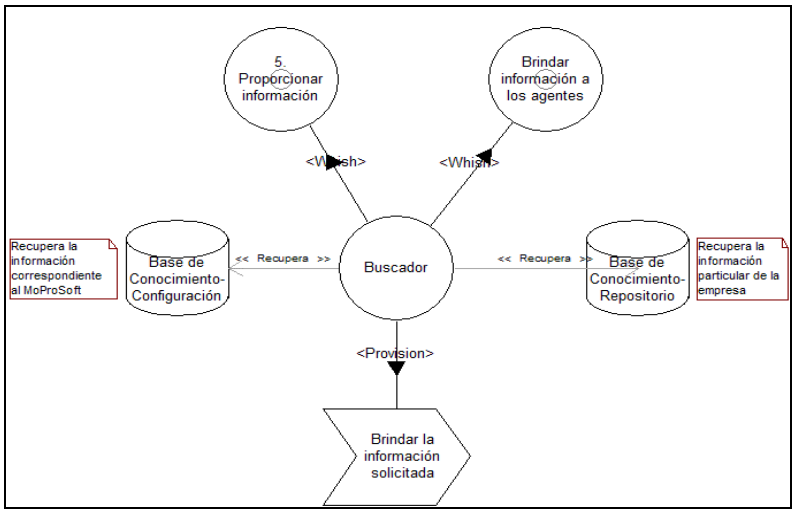

Figura 3.10. Diagrama de agente (*agenteBuscador*)

3.2.2.1.4 Diagramas de flujo de trabajo

Después de generar el modelo objetivo/tarea, el diagrama de flujo de trabajo ayuda a relacionar un objetivo con su respectivo servicio y posteriormente descomponer a éste último en un conjunto de tareas realizables por los agentes. La descomposición de tareas debe comenzar cuando el objetivo pueda ser alcanzado por un agente del sistema.

Un diagrama de flujo de trabajo se puede ver como un diagrama de actividades de UML donde las tareas se muestran en vez de actividades. El diagrama también muestra las clases que son entradas/salidas de las tareas usando objetos de flujo [Evans01].

A continuación se muestran los diagramas de flujo de trabajo asociados a los diversos servicios que ofrecen los agentes. Los diagramas de las figuras 3.11 y 3.12 se refieren a los servicios de *Recordar tareas pendientes* y *Coordinar y contactar usuario,* que brinda el *agenteSupervisor*. Al igual que todos los diagramas de este tipo, se encuentran divididos en tres partes:

- En la primera parte se indica la notación utilizada y el servicio que trata el diagrama.

- En la segunda parte está el flujo de tareas que son realizadas secuencialmente por los agentes indicados en la tercera parte. Además se muestran como clases UML las entidades de información que sirven como entrada o salida en cada tarea.

- En la tercera parte se ubican los agentes involucrados y con flechas se indican las tareas de las cuales son responsables.

Cabe mencionar que todos los diagramas de los agentes (menos el coordinador) parten del supuesto de que ya fueron llamados por el *agenteCoordinador* y su tarea ya ha sido asignada.

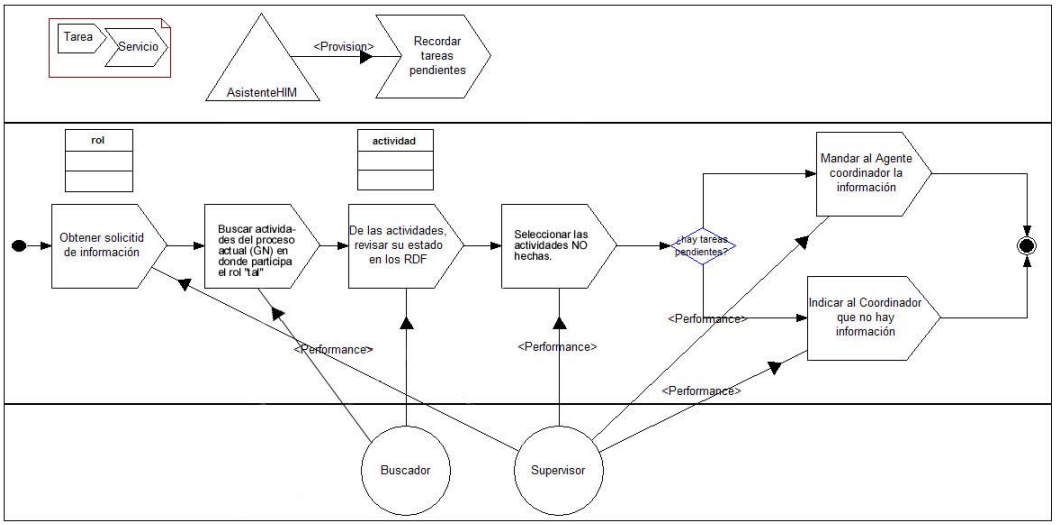

Figura 3.11: Diagrama de flujo de trabajo para el servicio de *Recordar tareas pendientes*

Como se puede observar en estos diagramas, se van identificando los objetos de información que se van a manejar y que son del dominio del problema (rol, actividad y usuario).

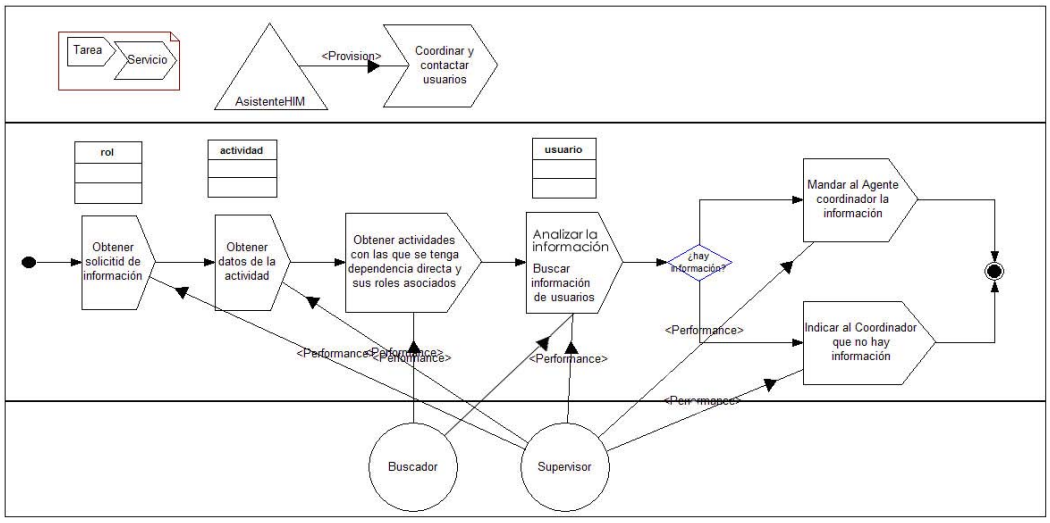

Figura 3.12: Diagrama de flujo de trabajo para el servicio de *Coordinar y Contactar usuarios*

Es importante indicar que para que se realicen las intervenciones del *agenteBuscador*, tiene que existir una solicitud especial por parte del agente que requiere el servicio (en este caso el *agenteSupervisor*), proporcionando el tipo de información que requiere y los parámetros necesarios. Al terminar con las operaciones exclusivas de este agente para proporcionar el

servicio, se tienen que enviar los resultados al *agenteCoordinador,* o en su caso, la notificación de resultados nulos.

El diagrama de la figura 3.13 se refiere a todos los servicios del *agenteRBC* que involucra el enfoque de Razonamiento Basado en Casos. Se presenta un solo diagrama ya que las tareas del razonador no se cubren en este tipo de diagramas y requieren de más especialización. Todo el desarrollo del sistema razonador basado en casos que tiene este agente se trata en el capítulo 4.

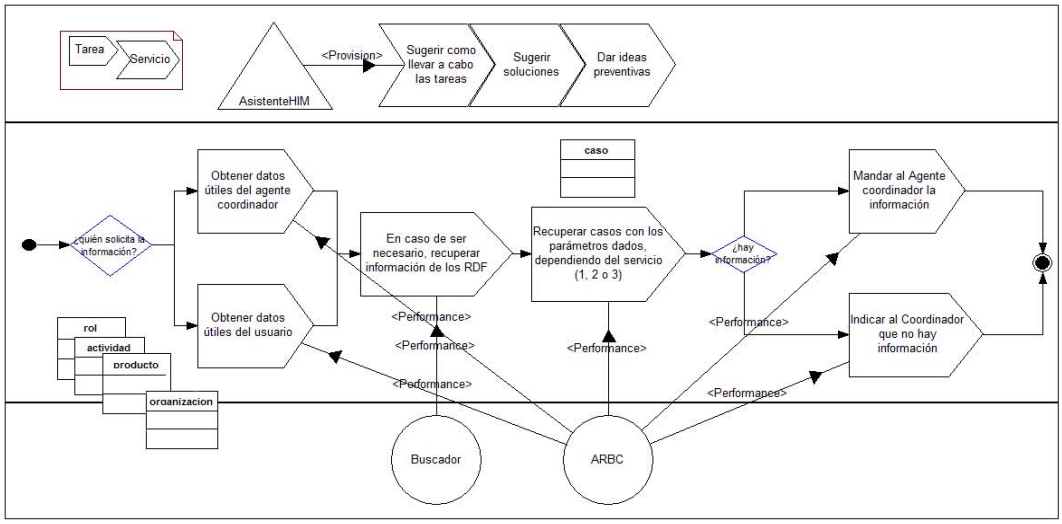

Figura 3.13: Diagrama de flujo de trabajo para los servicios del *agenteRBC*

En las tareas del *agenteRBC* también están involucradas algunas entidades del dominio del problema: el rol, la actividad, el producto y la organización. Toda la información que contienen sirve para formar los casos.

Los diagramas de las figuras 3.14 y 3.15 tratan el importante servicio del *agenteBuscador.* El *agenteBuscador* es el único que tiene acceso a la Base de Conocimiento y realiza las consultas a los archivos en RDF.

En la figura 3.14 se muestra el diagrama de flujo del servicio de brindar la información solicitada pero *del usuario,* esto es, que es el servicio que utiliza el *agenteCoordinador* para proporcionarle al usuario información acerca de su rol, actividad y producto. Y en el diagrama de la figura 3.15 se muestra el servicio de brindar la información solicitada pero a *los agentes,* esto es, el servicio de búsquedas en la base de conocimiento que le solicitan los demás agentes para poder completar sus tareas.

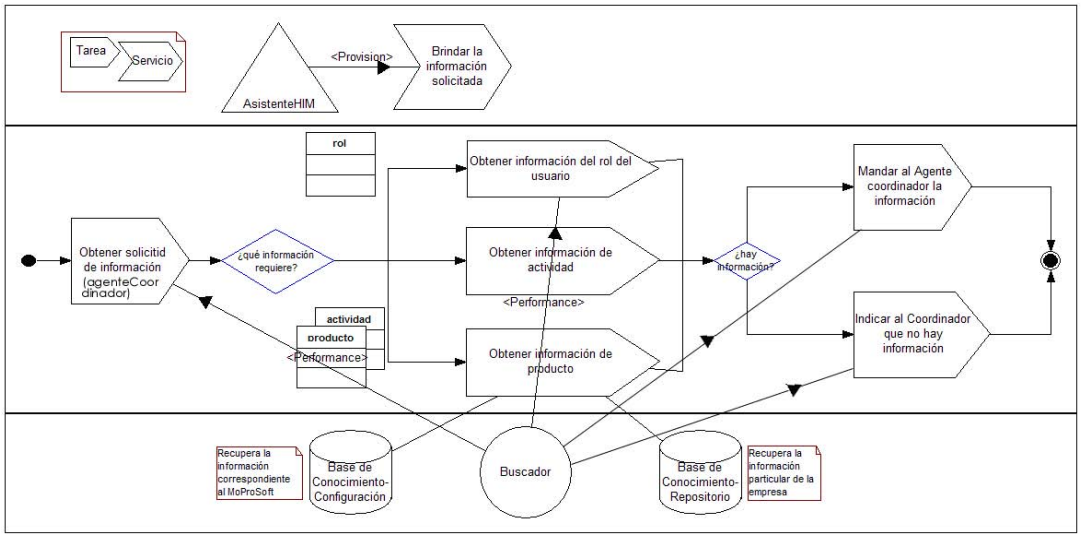

Figura 3.14: Diagrama de flujo de trabajo para el servicio de *Brindar la información solicitada al usuario*

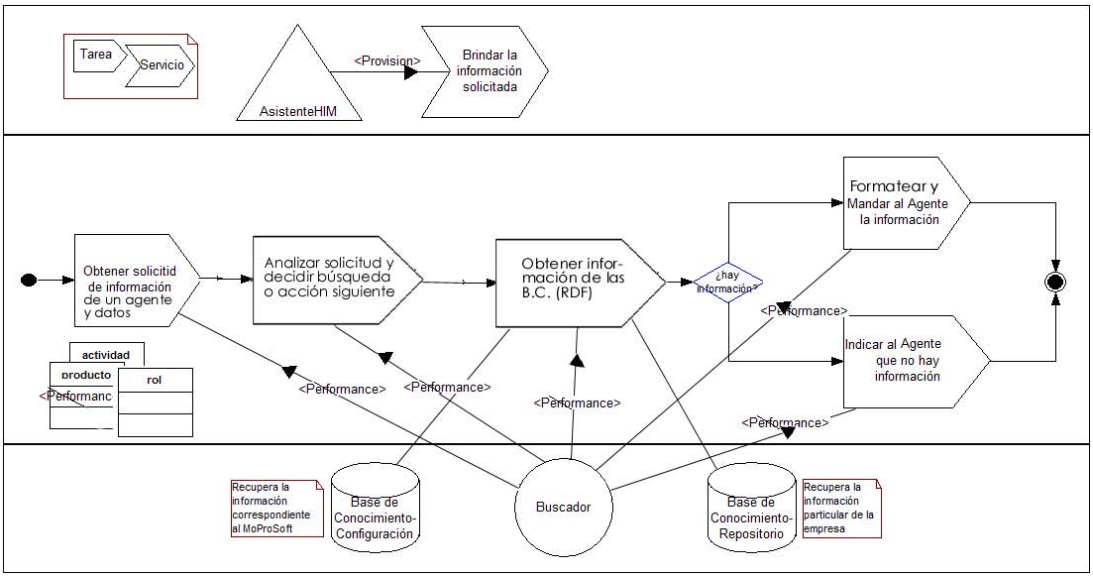

Figura 3.15: Diagrama de flujo de trabajo para el servicio de *Brindar la información solicitada a los agentes*

Por último, en el diagrama de la figura 3.16 se muestran los servicios proporcionados por el *agenteCoordinador*, que es el que está al pendiente de las actividades del usuario y orquesta los servicios de los demás agentes para alcanzar los objetivos del AsistenteHIM.
El *agenteCoordinador* utiliza los servicios de todos los agentes y por supuesto, también todas las entidades manejadas en el sistema de guía AsistenteHIM.

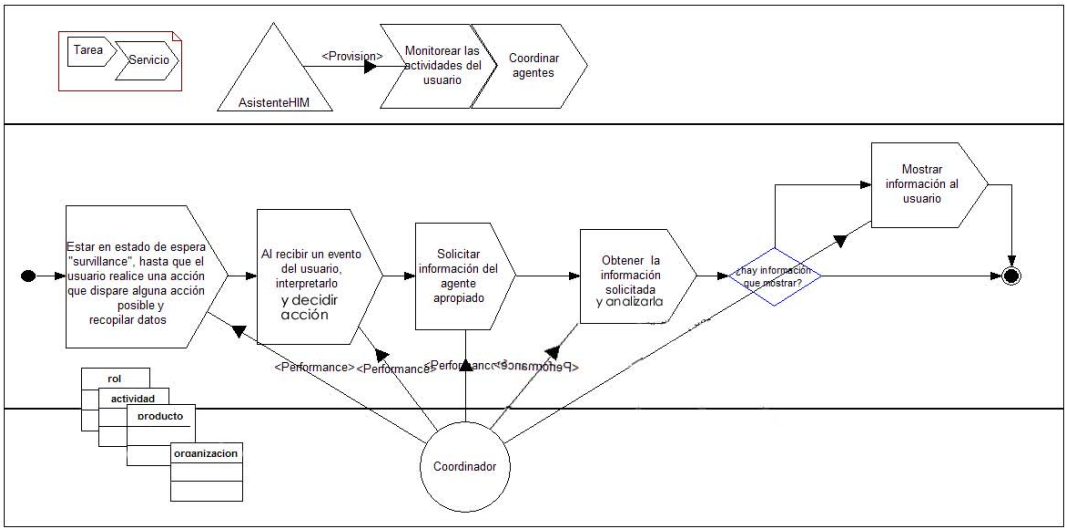

Figura 3.16: Diagrama de flujo de trabajo para los servicios del *agenteCoordinador*

3.2.2.2 Construcción de los modelos, fase 1

La fase 1 consiste en refinar los modelos generados en la fase 0 y definir la estructura y comportamientos de las entidades (organización, agentes, interacción y objetivos/tareas).

3.2.2.2.1 Modelo de la organización (MO)

Al moverse de nivel 0 a nivel 1, el análisis se centra en el sistema, identificando las piezas principales de funcionalidad requeridas (vistas tipos de agentes) a partir de los diagramas de organización anteriores, la descripción de requerimientos y el diagrama de objetivos. El proceso es utilizar los agentes identificados en la fase 0 para exponer el funcionamiento del sistema, sus componentes e interacciones de una manera general.

Como se puede ver en la figura 3.17, el AsistenteHIM queda constituido de la siguiente manera:

- Un *agenteCoordinador* que sirve de puente entre la interfaz gráfica de la HIM y el AsistenteHIM, y coordina a los agentes *Supervisor, Buscador y RBC* para alcanzar su objetivo. El *agenteCoordinador* es el único que tiene contacto directo con los usuarios.

- Un *agenteSupervisor*, ampliamente descrito en el modelo de agente fase 0 y 1 (al igual que los siguientes agentes).

- Un *agenteRBC.*

- Un *agenteBuscador.*

Con base en lo anterior y de acuerdo a la definición de sistema multi-agente dada por Luck y d'Inverno (sección 2.5.3.2 del capítulo 2), el AsistenteHIM es un SMA en el que el *agenteCoordinador* es el agente autónomo que tiene relación con los agentes *Supervisor, RBC y Buscador,* para satisfacer sus objetivos.

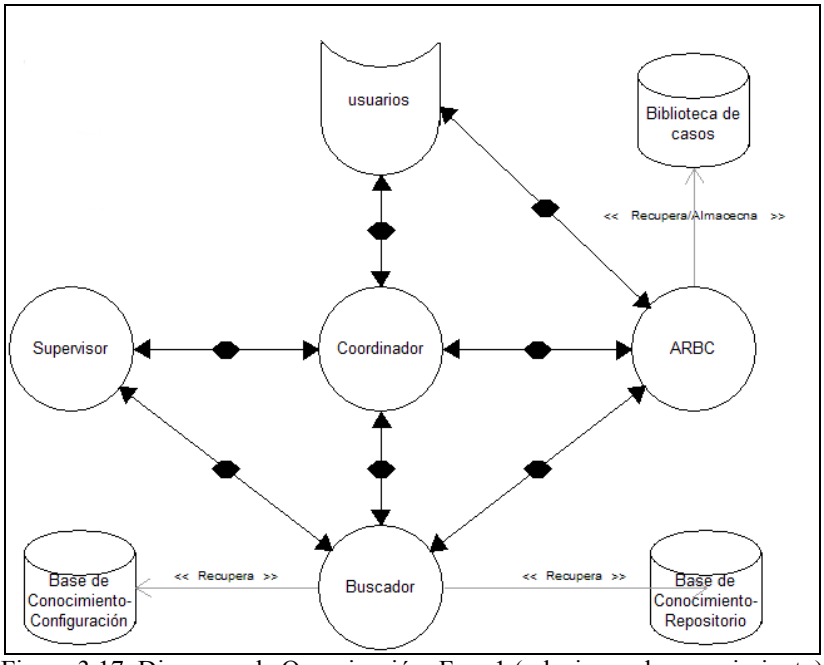

Figura 3.17. Diagrama de Organización, Fase 1 (relaciones de conocimiento)

El objetivo principal del sistema es *maximizar la efectividad de los usuarios* y se aplica a *todos* los roles, por ello los servicios que sirven para alcanzar tal objetivo y los agentes que los proporcionan, serán para *todos* los usuarios.

Se contemplan tres recursos, la base de conocimiento-configuración que contiene los archivos RDF que modelan todo el MoProSoft, la base de conocimiento-repositorio, que son los archivos que se generan a través del uso de la HIM y la biblioteca de casos que servirá para implementar el enfoque de RBC.

3.2.2.2.2 Modelo de Agente (MA)

El modelo del agente en la fase 1 consiste en generar un esquema de agente por cada agente identificado en el sistema, tomando en cuenta los diagramas de la fase 0 del análisis. El contenido de dichos esquemas utiliza algunos conceptos delineados anteriormente que forman parte de la teoría de agentes.

Como se identificaron cuatro agentes para el SMA, se generaron cuatro esquemas de agente, cada uno indicando la identidad del agente, sus servicios, su objetivo, sus interacciones con el entorno, su estado mental, comportamiento, y alguna información adicional. Nótese que la descripción que proporcionan estos esquemas de los agentes, es bastante detallada y deja a un paso el diseño y la implementación de los agentes.

En la tabla 3.1 se muestra el esquema del *agenteCoordinador*. Los demás esquemas se pueden consultar en el apéndice 2 de este documento.

Agente: Coordinador
Identidad: El agente es identificado como el responsable de proporcionar la interfaz entre el usuario y el sistema multi-agente, mediante la su inclusión dentro de la capa de control de la HIM (*servlet*, capítulo 5), para que cuando sea necesario solicite los servicios de los demás agentes.
Servicios: No brinda servicios a otros agentes, sino al usuario: monitorear las actividades del usuario y coordinar los agentes.
Objetivo: activo (significa que es autónomo)
Interacciones con el entorno: *Entrada receptora:* La interfaz gráfica de HIM, que es donde el agente está en espera (*survillance*), para brindar ayuda y guía a los usuarios. Utilizando los servicios del *agenteBuscador* tiene acceso a información del marco de trabajo en RDF. *Relaciones:* Se necesita tener contacto con los agentes *Supervisor, Buscador y RBC* para utilizar sus servicios en favor del usuario. *Propiedad de recurso y acceso:* 1. Acceso a la información del usuario (rol, actividad que realiza, etc.) por medio de la interfaz gráfica de HIM, además del seguimiento de actividades. 2. Acceso de lectura a toda la información que pueden brindar los demás agentes a través de sus servicios, descrita en los esquemas anteriores. *Acciones:* 1. (interna) Mantener un estado de espera monitoreando las actividades del usuario. 2. (interna) Interpretar las acciones del usuario en la interfaz de HIM, y traducirlas en acciones que se tienen que llevar a cabo utilizando los servicios de los demás agentes. 3. (interna) Formular y enviar mensaje de petición de servicio. 4. (comunicar) Recibir información solicitada. 5. (interna) Evaluar la información obtenida y decidir la siguiente acción. 6. (comunicar) Indicar al agente que brindó el servicio, que ya se recibió la información. 7. (comunicar) Mostrar información al usuario.
Estado mental y comportamiento: *Propósito:* Los objetivos de este agente son (1) monitorear las actividades del usuario y (2) coordinar los agentes, de manera que todos los servicios que brindan sean utilizados para alcanzar los objetivos del sistema AsistenteHIM. *Comportamiento:* El agente espera monitoreando las actividades del usuario en la interfaz de la HIM, y llama a los demás agentes cuando sea necesario proporcionar sus servicios. Al recibir la respuesta de los agentes, se concentra en realizar las tareas indicadas en los diagramas de flujo de trabajo para este agente, y mantener la relación con el usuario. Puede realizar las acciones internas, comunicativas o externas (de las listadas anteriormente) que sean necesarias. Es preciso conocer el rol del usuario que se está atendiendo y la interfaz gráfica de la HIM, de manera que se identifiquen las acciones que se realizan y se conozca la forma de intervenir con la ayuda de los demás agentes. Por ejemplo si se acaba de entrar al sistema sería bueno llamar al *agenteSupervisor* para que le indique al usuario sus tareas pendientes. *Conocimiento y creencias:*

1. Tiene conocimiento de la interfaz de la HIM y cómo interpretar las acciones del usuario en ella. 2. Tiene conocimiento de las tareas (y su orden) necesarias para brindar sus servicios. 3. Sabe cómo encontrar la información que necesita (mensajes a los demás agentes). 4. Sabe cómo interpretar los mensajes o entidades de información que recibe de los agentes. 5. Sabe cómo transferir los resultados al usuario
Información adicional: Nótese que el agente utiliza todos los servicios de los agentes *Supervisor, Buscador y RBC* en pro del usuario y no proporciona servicios a ninguno. Es el puente entre la HIM y el AsistenteHIM.

Tabla 3.1: Esquema del *agenteCoordinador*

3.2.2.2.3 Modelo de Interacción (MI)

Para alcanzar objetivos de más alto nivel, los agentes deben cooperar, lo cual deriva en el desarrollo del Modelo de Interacción. Éste señala cuáles, porqué y cuándo los agentes necesitan comunicarse, dejando al proceso de diseño los detalles de cómo toma lugar la comunicación. El modelo captura la manera en la que los agentes intercambian información entre ellos y con su ambiente, así como los protocolos que se utilizan.

Los diagramas de interacción muestran el agente que inicia la interacción, los que responden, el motivador de la interacción (comúnmente un objetivo del iniciador u objetivo común entre los participantes), más otra información opcional como la condición de disparo y la información suministrada y conseguida por cada uno de los participantes [Caire01].

El modelo de interacción que se desarrolló en este proyecto consiste en un conjunto de *interacciones* de alto nivel (intercambio de información entre entidades con un propósito [EURESCOM01]), así como una representación más detallada en términos de *protocolos de interacción*.

Los protocolos de interacción (notación originaria del trabajo hecho en FIPA [FIPA02]) consisten en una especificación detallada de los mensajes intercambiados entre los agentes participantes, así como las diferentes secuencias permitidas en las que los mensajes pueden ser intercambiados. Son una extensión a los diagramas de interacción de UML con construcciones adicionales para facilitar el modelado de interacciones más complejas. FIPA mantiene una biblioteca de protocolos de interacción (Interaction Protocol, *IP*) predefinidos y comúnmente requeridos [FIPA03].

En el apéndice 1 se muestra un ejemplo de diagrama de interacción, sin embargo la metodología MESSAGE indica que es opcional utilizar ese tipo de diagramas o esquemas que contengan los mismos datos. En su lugar, se generaron varios esquemas de interacción para modelar todas las interacciones identificadas en el AsistenteHIM. A continuación se muestran algunos, los demás pueden ser consultados en el Apéndice 2.

En la tabla 3.2 se muestra la interacción que especifica que el *agenteCoordinador* realiza una acción apropiada de acuerdo a las acciones del usuario en la interfaz de HIM. El agente tiene que estar en espera de cualquier evento que indique una actividad del usuario en la que se le pueda brindar información útil, decidir qué tipo de información es oportuna, y así poder seguir con la siguiente interacción, que se muestra en la tabla 3.3.

Interacción 1	Realizar acción
Motivador	Monitorear las actividades del usuario
Iniciador	Interfaz de HIM
Colaborador(es)	*agenteCoordinador*
Entradas	Acción del usuario en la interfaz de HIM y datos necesarios
Salidas	Información proporcionada al usuario de acuerdo a su acción
Procesamiento	El *agenteCoordinador* debe saber qué información es oportuna para el usuario, dependiendo de las acciones de éste último en la interfaz de HIM (por ejemplo si cambia de rol, indicarle sus posibilidades con ese nuevo rol)
Restricciones	Todo depende de la comunicación que se establezca entre la interfaz de HIM y el *agenteCoordinador*

Tabla 3.2: Esquema de interacción con motivador *Monitorear las actividades del usuario*

La interacción 2 muestra de manera general la petición de información que hace el *agenteCoordinador* a los demás agentes. Para cada uno de los agentes este esquema varía en las entradas, salidas y motivadores, por lo que también fueron desarrollados y pueden ser revisados con la numeración 2.1, 2.2, 2.3, etc., en la tabla 3.4 y en el apéndice 2.

Interacción 2	Petición de información
Motivador	Coordinar agentes
Iniciador	*agenteCoordinador*
Colaborador(es)	Todos los agentes
Entradas	Acción del usuario y datos (por ejem. rol, actividad, proceso y producto)
Salidas	Acción e información recopilada por el agente
Procesamiento	El *agenteCoordinador* debe saber qué información es oportuna para el usuario, dependiendo de las acciones de éste último en la interfaz de HIM, y así solicitar dicha información al agente indicado
Restricciones	Cuando no se encuentre la información requerida, el agente encargado de recopilarla tendrá que notificar al coordinador de la situación
IP	FIPA-query Protocol

Tabla 3.3: Esquema de interacción con motivador *Coordinar agentes*

Las acciones para recuperar información de la base de conocimiento, también son interacciones y se muestran en esquemas de interacción (tablas 3.4 y 3.5).

Interacción 2.1	Petición de información
Motivador	Recordar tareas pendientes
Iniciador	*agenteCoordinador*
Colaborador(es)	*agenteSupervisor*
Entradas	Acción del usuario, rol, proceso (GN)
Salidas	Lista de las actividades pendientes del usuario
Procesamiento	El *agenteSupervisor* revisa las actividades asignadas al rol del usuario, en el proceso de GN y le indica cuáles no están hechas
Restricciones	El *agenteSupervisor* notifica si no se encontró información
IP	FIPA-query Protocol

Tabla 3.4: Esquema de interacción con motivador *Recordar tareas pendientes*

Interacción 3	Petición de información
Motivador	Brindar información solicitada
Iniciador	*agenteBuscador*
Colaborador(es)	Base de conocimiento en RDF
Entradas	Consulta apropiada con los parámetros necesarios
Salidas	Información requerida
Procesamiento	El agente busca la información solicitada por los demás agentes y que está almacenada en la base de conocimiento en RDF

Tabla 3.5: Esquema de interacción con motivador *Brindar información solicitada*

Todas las peticiones de información al *agenteBuscador*, tienen el mismo formato (interacción 6, apéndice 2), aunque varían en muchos aspectos: motivador, iniciador, entradas y salidas, por lo que se desarrollaron los esquemas por separado. En la tabla 3.6 se muestra la primera interacción que se refiere a la petición de información del *agenteSupervisor* al *agenteBuscador*, con el motivador de *recordar tareas pendientes* al usuario.

Interacción 4	Petición de información
Motivador	Recordar tareas pendientes
Iniciador	*agenteSupervisor*
Colaborador(es)	*agenteBuscador*
Entradas	Acción usuario, rol y proceso
Salidas	Lista de actividades no hechas
Procesamiento	El *agenteSupervisor* busca tener una lista de actividades pendientes para recordarle al usuario. El *agenteBuscador* hace la consulta en los archivos en RDF
Restricciones	Cuando no se encuentre la información requerida, el *agenteBuscador* tendrá que notificar al *agenteSupervisor* de la situación
IP	FIPA-query Protocol

Tabla 3.6: Esquema de interacción con motivador *Recordar tareas pendientes (3)*

De los protocolos de interacción de FIPA sólo se utiliza el FIPA-query Protocol, que se describe a continuación con ayuda de la figura 3.18 obtenida del sitio web de FIPA [FIPA03].

- FIPA-query Protocol: El agente que inicia la conversación tiene la intención de que el receptor le informe de algo concreto. En el primer mensaje, el agente iniciador tiene la posibilidad de enviar un mensaje con una preformativa *query-if* o *query-ref,* a lo que el agente receptor, tras un tipo de proceso que se representa con el rectángulo vertical o hilo de interacción, responde con uno de los mensajes alternativos si no entiende la pregunta (*not-understood*), si no quiere contestar (*refuse*), si ha sufrido algún error al procesar la pregunta (*failure*) o con el resultado que espera el iniciador [Mas01].

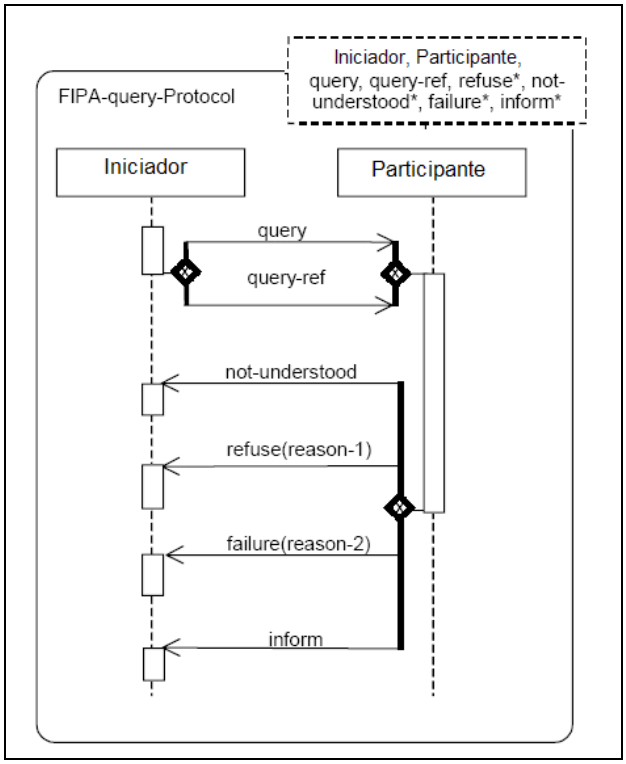

Figura 3.18: Protocolo de interacción *FIPA-query Protocol,* [FIPA03]

3.2.2.2.4 Modelo de Dominio (MD)

El modelo del dominio captura los conceptos de dominio importantes para el sistema y el usuario, que servirán de base para la construcción de la ontología de comunicación entre los agentes y para derivar representaciones internas de conocimiento para los agentes basados en conocimiento.

El modelo es representado por medio de un diagrama de clases típico de UML, donde las clases representan los conceptos específicos de dominio y las asociaciones representan las relaciones específicas de dominio.

Se construyó en paralelo a los demás modelos agregando conceptos nuevos y relaciones conforme se fueron identificando en los otros diagramas. Además fue comparado con las ontologías que se utilizaron en HIM, de manera que se facilita el uso de los recursos en RDF.

Cada una de las clases representa una entidad que tendrá ciertas características e información de utilidad para los agentes y que permite realizar las acciones correspondientes a cada uno. Por medio de estas entidades se comparte la información y la interpretación de los datos debe ser igual por todos los agentes.

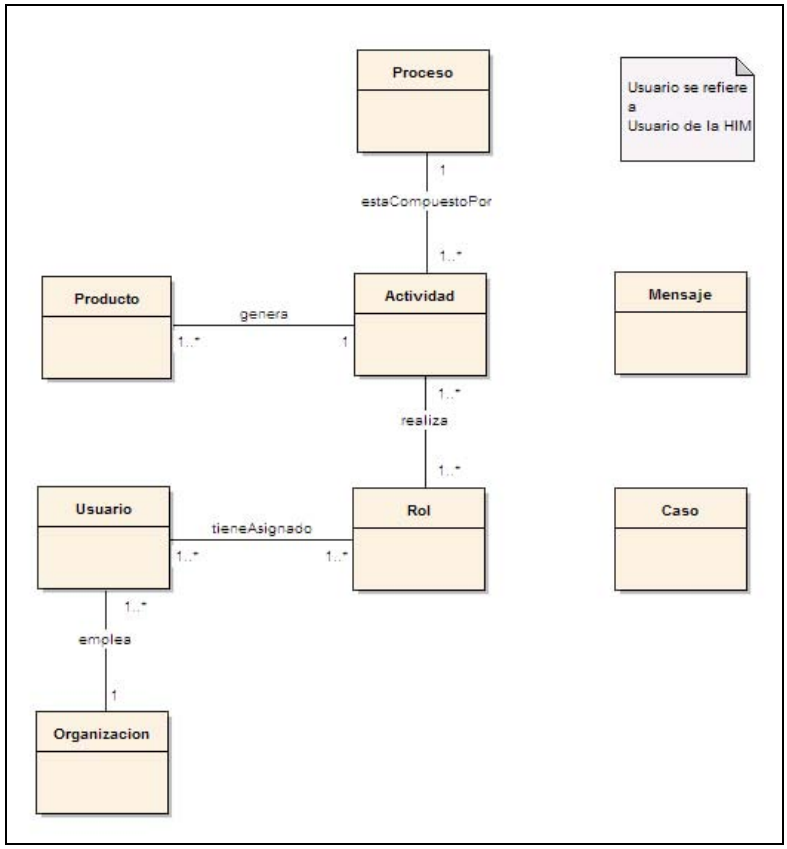

Figura 3.19: Modelo de dominio del AsistenteHIM

3.3 DISEÑO

El diseño de los sistemas multi-agente en MESSAGE, consiste en definir la arquitectura general de un sistema de software e identificar los componentes principales del sistema, especificando sus tareas y estableciendo las interfaces entre los componentes.

Asimismo, el diseño incluye un nivel de trabajo en el que los diagramas y esquemas producidos en la etapa de análisis son transformados en entidades computacionales, en términos de subsistemas, interfaces, clases, operaciones, algoritmos, objetos, diagramas de objetos y otros conceptos computacionales [Caire02].

El propósito del diseño es definir la solución del problema, describiendo las realizaciones físicas que cubren todos los requerimientos y limitaciones definidos en la fase de análisis. Esta descripción computacional debe permitir la implementación del sistema con un refinamiento directo, agregando detalles acerca de los lenguajes de programación y plataformas de desarrollo, pero no cambiando la estructura del sistema [EURESCOM02]. El diseño de SMA debe tomar en cuenta opciones de plataformas y adherirse a estándares tales como los de FIPA [FIPA01].

Aunque existen varios enfoques diferentes para el diseño de agentes y SMA, hay poca experiencia que indique cuál es mejor. Consecuentemente el diseño en MESSAGE se presenta en dos enfoques diferentes que se adecuan según el caso del diseñador:

- Enfoque 1, donde el diseño es dirigido por la organización del sistema multi-agente y una arquitectura de agente.

- Enfoque 2, orientado a una plataforma de desarrollo de agentes concreta.

El enfoque 1 puede verse como una aproximación de alto nivel, donde un agente es más que una clase: un agente es visto como subsistema, con una arquitectura interna que define las relaciones de los distintos componentes del agente. Estos componentes son entidades computacionales que son identificadas y construidas por transformación y proceso de refinamiento de los modelos de análisis. Esta aproximación trata de ser lo más independiente posible de la arquitectura de agente que se elija para un agente concreto, y está basado en el empleo de patrones arquitectónicos.

En contraste, el segundo enfoque puede verse como un proceso de diseño a bajo nivel, ya que es específico de una plataforma de agente. Considera que cada agente puede ser convertido a una clase, lo cual deriva principalmente de la mayoría de las herramientas para construir agentes, en las que hay una clase *Agente* de la cual se derivan los tipos específicos de agentes. En este enfoque los autores de MESSAGE emplean una plataforma FIPA y en concreto la implementación JADE (*Java Agent Development Framework*) [Bellifemine01], basada en un modelo de Programación Orientada a Objetos (POO) y empleando Java como lenguaje de programación.

Para el diseño del AsistenteHIM se decidió utilizar el segundo enfoque manejando JADE para la etapa de desarrollo, gracias a que se basa en un modelo de POO, que es familiar al desarrollo de sistemas en la comunidad de Ingeniería de Software, y sobre todo con el desarrollo de la HIM. Además de que al trabajar con un entorno de desarrollo especializado para SMA, nos liberamos de desarrollar servicios básicos de agentes (brindados por el entorno de desarrollo), para concentrarnos en implementar la lógica de la aplicación. Las bondades de este enfoque se evidencian de manera práctica en el siguiente tema, al resultar un diseño muy cercano a la implementación, limpio y sintetizado.

Es importante mencionar que la plataforma para el desarrollo de agentes JADE provee un entorno de ejecución de agentes, que incluye mecanismos que soportan el nombramiento de los agentes y servicios de directorio, transporte de mensajes entre agentes, protocolos de coordinación y soporte de ontologías. Además, la plataforma tiene un conjunto de herramientas para hacer más rápido el desarrollo de los SMA, e incluye como parte fundamental especificaciones FIPA para la administración, comunicación, arquitectura y aplicación de los agentes.

La tabla 3.7 resume las características de JADE [Bellifemine01]:

Nombre	JADE (*Java Agent Development Framework*)
Proveedor	TILAB (http:jade.tilab.com)
Lenguajes	Java: J2EE, J2SE, J2ME CLDC/MIDP1.0 platforms
Disponibilidad	*Open Source,* Licencia LGPL
Características técnicas/ funcionales	▪ Distribuido, aplicación multi-plataforma con comunicación punto a punto ▪ Basado en un modelo de programación orientado a objetos ▪ Modelo de ejecución basado en el concepto de *comportamiento* ▪ Apegado a los estándares FIPA ▪ Administración del ciclo de vida de los agentes

	▪ Servicios de páginas blancas y amarillas con la oportunidad de crear gráficas de federación en tiempo de ejecución ▪ Herramientas gráficas que soportan las fases de compilación, administración y monitoreo ▪ Soporte para generar código de agentes y migración en estado de ejecución ▪ Soporte para protocolos de interacción complejos (ej. contract-net) ▪ Soporte para la creación de contenido de mensajes y administración incluyendo XML y RDF. ▪ Soporta la integración de los agentes en páginas JSP por medio de una biblioteca de etiquetas ▪ Soporte para la aplicación de niveles de seguridad (sólo en J2SE) ▪ Protocolos de transporte para seleccionar en tiempo de ejecución
Entorno de red	Probado en los campos de *bluetooth, GPRS, W-LAN* e Internet

Tabla 3.7: Resumen de las características principales de JADE [Bellifemine06]

Es significativo mencionar lo anterior ya que la plataforma de desarrollo elegida para un proyecto (en particular sus servicios), tiene un impacto directo en la manera que se lleva a cabo la etapa de diseño del mismo. Conforme a ello, se desarrolla la etapa de diseño del AsistenteHIM. Más explicación del uso de JADE se ve en el capítulo 5.

3.3.1 Descripción del proceso de diseño de bajo nivel

El proceso de diseño que se describe en la siguiente figura, sintetiza la manera en la que se utilizaron los modelos de la fase de análisis, para construir el modelo de diseño orientado a la implementación del SMA en la plataforma de agentes JADE.

En el área superior se ubican los modelos resultantes del análisis que sirven como entradas al diseño. En la parte inferior, el diagrama de actividades de la figura 3.20 muestra las tareas necesarias para producir un diseño del sistema. Cabe mencionar que los diagramas de clases y de objetos generados a continuación fueron utilizados ampliamente para llevar a cabo el desarrollo del AsistenteHIM, tema tratado en el capítulo 5.

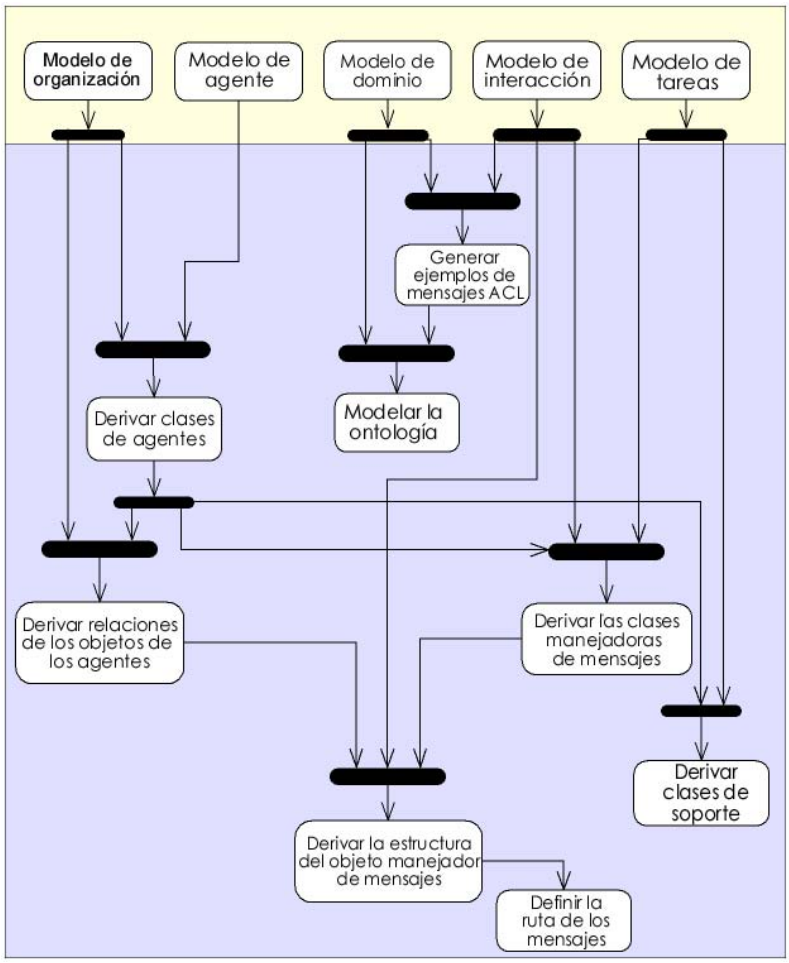

Figura 3.20: Diagrama de actividades del proceso de diseño de bajo nivel [Evans01]

3.3.1.1 Generación de mensajes ACL preliminares

En este paso se produjo un mensaje ACL preliminar por cada mensaje intercambiado en el modelo de interacción. Antes de crear los mensajes, se tomó la decisión de utilizar el lenguaje de contenido FIPA-ACL/SL0, dado que es soportado por JADE y es adecuado para el tipo de comunicación que se requiere entre los agentes del AsistenteHIM.

Los mensajes intercambiados entre agentes se construyeron con base en los componentes que especifica el estándar ACL [Mas01]:

- El *objetivo* de cada mensaje indica el servicio que se pretende atender con la generación de los mensajes. Cabe mencionar que no es parte del mensaje.

- La *preformativa* indica el tipo de acto comunicativo. Para el presente proyecto se utilizaron las preformativas del estándar FIPA CAL (*Comunicative Act Library*) [FIPA01].

- El *receptor* indica la entidad que recibe el mensaje.

- El *contenido* como su nombre lo indica, comprende la información que se desea enviar en el mensaje, ya sean oraciones, variables u otros datos.

- El *lenguaje* indica el lenguaje de contenido utilizado en los mensajes ACL.

- El *identificador* de la comunicación sirve para llevar un control de los mensajes que componen una conversación. En este caso es un número compuesto por tres partes: la primera indica el agente que envía el mensaje, el segundo el agente receptor y el tercero es un contador en caso de que existan varios mensajes diferentes con los mismos remitentes y receptores.

La tabla 3.8 muestra algunos de los mensajes generados, la totalidad de ellos puede ser consultada en el apéndice 2.

Entidad que envía el mensaje	Mensaje
interfazHIM	:performative inform :receiver: agenteCoordinador :content: accion, datos :language fipa-sl0 NOTA: Este propiamente no es un mensaje que será traducido a ACL, sólo ejemplifica que por medio de las acciones del usuario en la interfaz gráfica de HIM, se contactará al agenteCoordinador para que lleve a cabo una acción.
agenteCoordinador 0	:performative Inform :reveiver interfazHIM :content "información para el usuario, dependiendo de lo provisto por los demás agentes" :language fipa-sl0 NOTA: Tampoco es un mensaje ACL en sí, y sólo termina de ejemplificar la relación de información que debe haber entre la interfaz y el agenteCoordinador. Objetivo: recordar al usuario sus tareas pendientes :performative request :reveiver agenteSupervisor :content: accion, datos (rol, proceso) :language fipa-sl0 :conversation-id 011 NOTA: La preformativa request(a) indica que el receptor debe realizar una acción a y además responder al emisor el resultado de dicha acción con un acto comunicativo inform [Mas01]. La mayoría de los mensajes de este sistema son de este tipo, y no involucran conversaciones complejas.
agenteSupervisor 1	Objetivo: recordar al usuario sus tareas pendientes :performative inform :reveiver agenteCoordinador :content accion, datos (rol, proceso, actividadesPendientes) :language fipa-sl0 :conversation-id 011
agenteBuscador 2	Objetivo: brindar información solicitada-rol :performative inform :reveiver agenteCoordinador :content accion, datos (rol, Información útil del rol) :language fipa-sl0 :conversation-id 022
agenteRBC 3	Objetivo: tareas RBC :performative request :reveiver agenteCoordinador :content accion, datos (Oración con la sugerencia, prevención o instrucción que surja del RBC). :language fipa-sl0 :conversation-id 031

	Objetivo: solicitud de información :performative request :reveiver agenteBuscador :content accion, datos :language fipa-sl0 :conversation-id 321

Tabla 3.8: Mensajes de los agentes del AsistenteHIM

3.3.1.2 Refinar el modelo de dominio u ontología

En este paso se refinó el modelo de dominio generado en el análisis (figura 3.21), ya que se detectaron relaciones y propiedades adicionales en la generación de mensajes ACL. Nótese que las clases ya incluyen los nombres de las propiedades y que desapareció la clase de proceso.

Cabe mencionar que no están las clases de agente y comportamiento (*behaviour*), debido a que el modelo de dominio es un diagrama de clases que incluye solamente los conceptos que sirven para la comunicación entre los agentes, y no dichos agentes ni sus acciones.

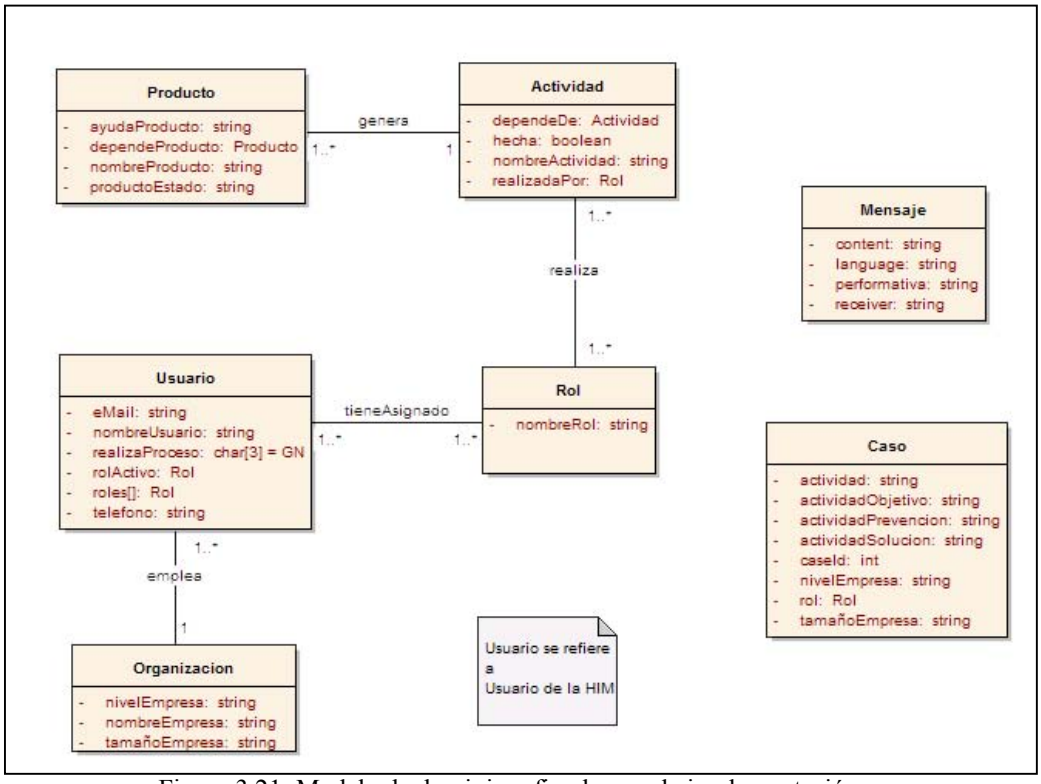

Figura 3.21: Modelo de dominio refinado para la implementación

3.3.1.3 Derivar las clases de los agentes

Se produjeron diagramas de clases UML (figura 3.22) definiendo cada tipo de clase de agente. Jade provee dos clases *Agent* disponibles para derivar las clases de agentes específicas para cada aplicación. La clase *Agent* es la opción más básica para hacer sub-clases de agentes, mientras que

la clase *GUIAgent* soporta mejor agentes que son requeridos como interfaz para eventos de interfaces gráficas de usuario (*Graphical User Interfaces, GUI's*). Aunque al agenteCoordinador le corresponde estar situado en la interfaz del usuario, su clase no se extendió de *GUIAgent* debido a que la interfaz de la HIM es Web y por lo tanto la integración es distinta, como se verá en el capítulo 5.

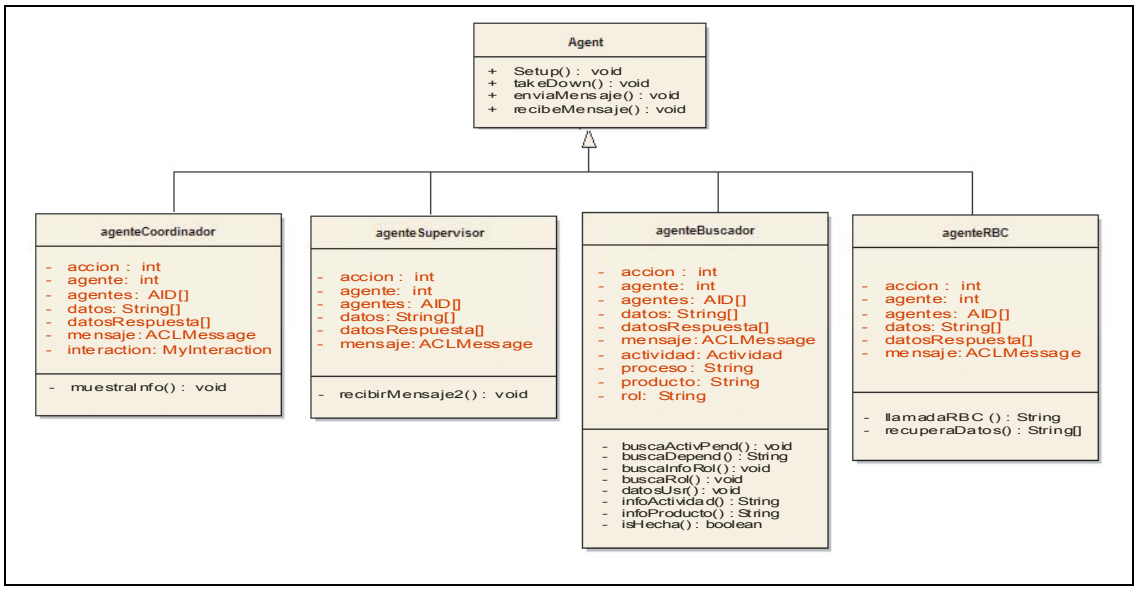

Figura 3.22: Diagrama de clases de los agentes en el AsistenteHIM

3.3.1.4 Derivar las relaciones de conocimiento de los objetos agente

En este paso se identificaron las instancias de los agentes de cada tipo, así como las relaciones de conocimiento entre cada agente por medio de un diagrama de objetos UML, que es muy parecido al diagrama generado en el modelo de organización en la etapa de análisis, pero aquí vemos clases en lugar de agentes.

En el diagrama de la figura 3.23 se observa que existe una instancia por cada tipo de agente definido y que existe relación entre el *agenteCoordinador* y el *agenteBuscador* con todos los agentes, pero el *agenteSupervisor* y el *agenteRBC* no la tienen entre si.

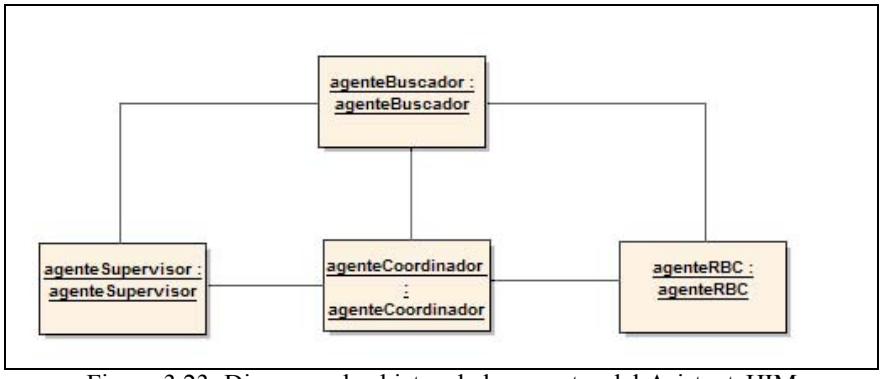

Figura 3.23: Diagrama de objetos de los agentes del AsistenteHIM

Adicional a esto, es necesario especificar cómo se conocen cada uno de los agentes para poder interactuar: En la plataforma de desarrollo JADE, esto se alcanza mediante la localización del agente DF (*Directory Facilitator*), que proporciona el servicio de *páginas amarillas* de un SMA.

3.3.1.5 Definir clases de soporte

Este paso consiste en producir un conjunto de diagramas de clases de soporte, como los componentes de la interfaz gráfica de usuario (GUI).

Como el AsistenteHIM es una herramienta que se incluye desde otro sistema, no se diseña una interfaz de usuario específica, sino que se adecua a dicho sistema. En este caso se utilizó la GUI de HIM.

El proceso de inserción del AsistenteHIM en la propia HIM, se aborda en el capítulo 5 ya que queda fuera de los límites de la metodología MESSAGE y tiene que ver más con la arquitectura de la HIM que con el sistema multi-agente.

3.3.1.6 Derivar clases manejadoras de mensajes

El objetivo de esta tarea es desarrollar una estructura de herencia, formada con clases tipo comportamiento (*behaviour*, provistas por JADE), para manejar la comunicación de la aplicación.

Un comportamiento envuelve todas las tareas u operaciones de un agente, con sus rutas de ejecución al interconectarlos. Son unidades lógicas de actividad que pueden estar compuestas de distintas maneras, JADE provee un conjunto de clases básicas de comportamientos (figura 3.24), las cuales se instancian para desarrollar comportamientos específicos de la aplicación; además existen otros comportamientos predefinidos que soportan los protocolos de interacción de FIPA. Los comportamientos pueden ser agregados y removidos dinámicamente dentro de un agente y pueden ser descompuestos en secuencia o en paralelo formando comportamientos complejos.

La estructura de herencia que se desarrolló como diagrama de clases (figura 3.25) muestra cuáles de las clases *comportamiento* específicas se instanciaron para manejar las tareas y comunicación de los agentes del AsistenteHIM.

Los comportamientos *recibeMensajeAgente* son propios de cada agente y se encargan de recibir y procesar los mensajes. El comportamiento *enviaMensaje* también es propio de cada agente y el del tipo *oneShotBehavior* ya que es una actividad que sólo se ejecuta una vez cuando se requiere y no más.

El comportamiento *accionUsuario* es del *agenteCoordinador* y se encarga de decidir qué acciones secuenciales se deben de llevar a cabo cuando la interfaz de la HIM le indique que el usuario ha realizado alguna acción. *MostrarInfo* es para cuando ya se tiene la información para mostrar al usuario y *rbcBehaviour* es propio del *agenteRBC* e involucra las acciones necesarias para que se pueda integrar el razonamiento basado en casos, esto último se trata con más detalle en el capítulo siguiente.

La implementación de los comportamientos se puede interpretar como los métodos de cada agente del diagrama de clases de la figura 3.22, con la observación de que no son simples métodos o funciones, sino que involucran las características de cada tipo de comportamiento mostradas en la figura 3.24.

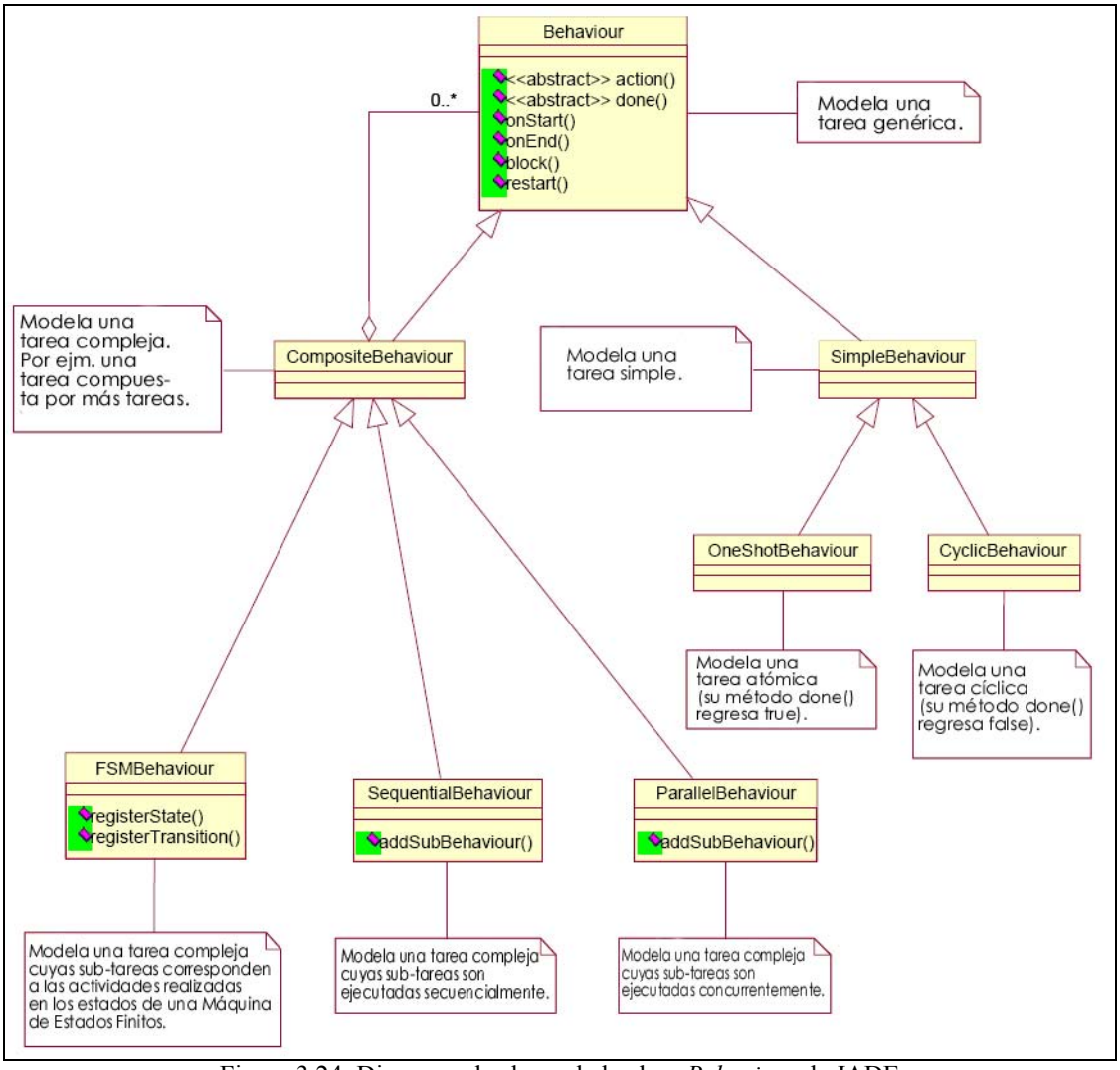

Figura 3.24: Diagrama de clases de la clase *Behaviour* de JADE

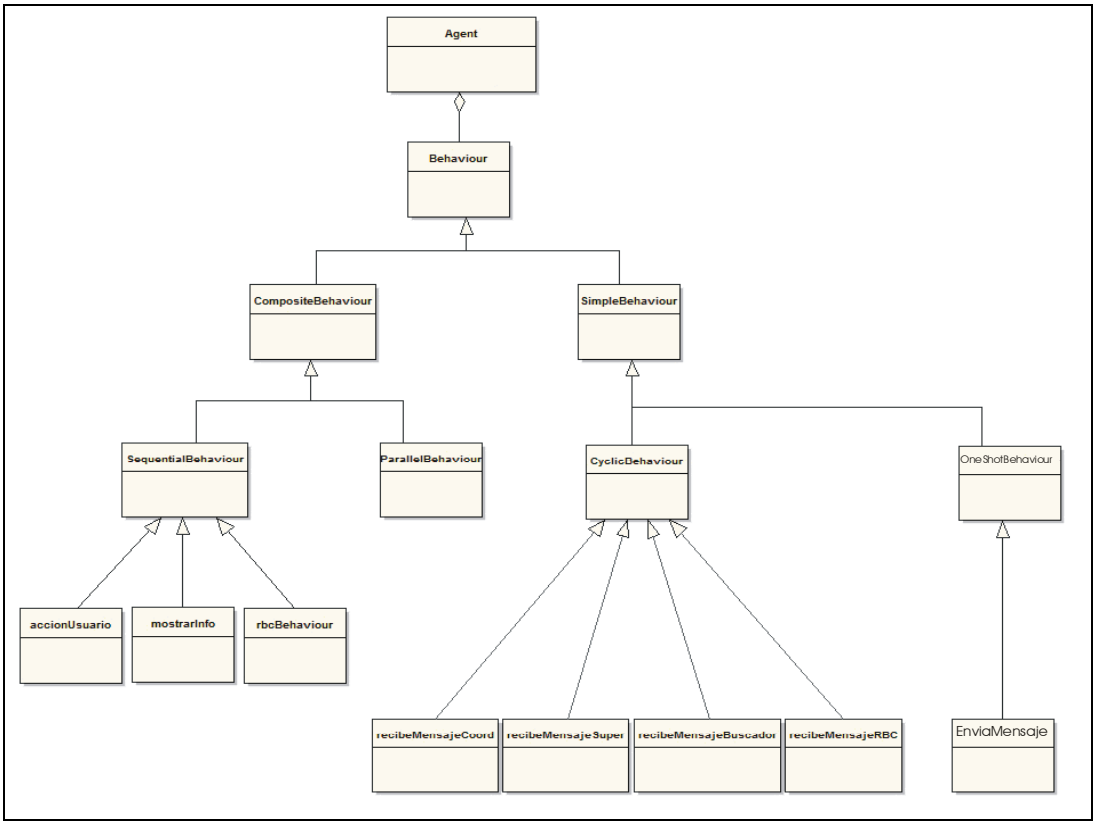

Figura 3.25: Diagrama de clases de las clases manejadoras de mensajes del AsistenteHIM

3.3.1.7 Derivar la estructura del objeto manejador de mensajes

En esta actividad se especificó una jerarquía que contiene las instancias de los comportamientos identificados en la actividad anterior, para manejar los protocolos definidos en el modelo de interacción de la etapa de análisis. Esto se ilustra utilizando un diagrama de objetos UML como se muestra en la siguiente figura.

El diagrama de la figura 3.26 muestra de manera general la correspondencia entre los comportamientos y los agentes. El comportamiento *enviaMensaje* también es parte de todos los agentes.

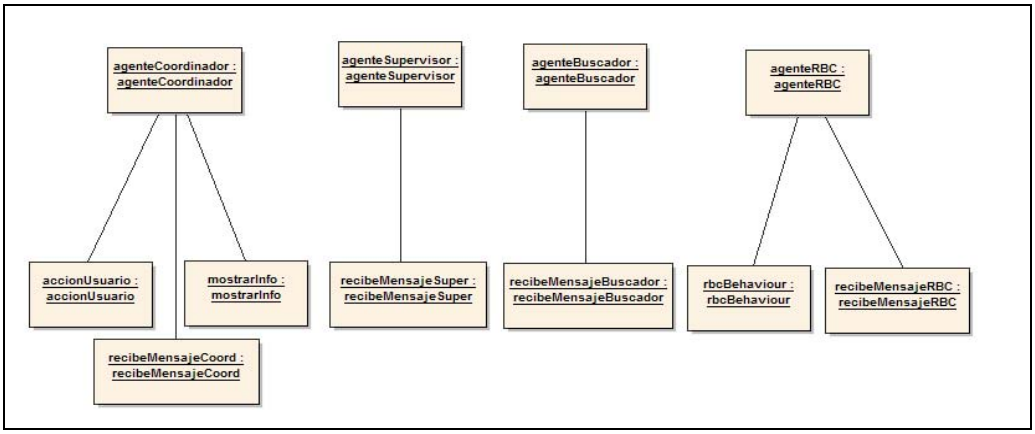

Figura 3.26: Diagrama de los objetos manejadores de mensajes del AsistenteHIM

3.3.1.8 Definir la ruta de mensajes

Esta última actividad trata de generar un árbol de decisión para especificar la manera en que los mensajes de entrada son dirigidos a manejadores individuales de mensajes (a una instancia de comportamiento individual como los de la figura 3.26).

Sin embargo, no se realizó debido a que los árboles de decisión son usados para seleccionar el mejor curso de acción en los casos en que se enfrenta incertidumbre. En el caso del AsistenteHIM son muy pocas las variantes en la manera en la que los mensajes son dirigidos, ya se encuentra bastante explícito con la generación de mensajes ACL (primer paso del diseño). Además, los comportamientos tampoco son muchos, por lo que queda claro a dónde va cada mensaje y qué comportamiento lo atenderá.

En los siguientes capítulos se abarca el tema del desarrollo del Razonador Basado en Casos y la implementación del AsistenteHIM en su totalidad, que es donde se aprovecha el valor del trabajo realizado en el presente capítulo.

Si tienes conocimiento, permite a otros encender sus velas con él.

–W. CHURCHILL

Capítulo 4

EL RAZONADOR BASADO EN CASOS

En este capítulo se aborda la construcción del razonador basado en casos, que constituye la parte medular del *agenteRBC* mencionado en el capítulo anterior.

Se incluyen conceptos importantes del Razonamiento Basado en Casos y se desarrollan las actividades de recolección de casos y creación de la base de casos. Consecutivamente se describe como se construyó el sistema razonador con la ayuda del marco de trabajo JColibri.

Finalmente, se presenta el sistema desarrollado y algunas pruebas que se realizaron a la base de casos.

4.1 INTRODUCCIÓN

A lo largo del capítulo anterior se desarrollaron varios modelos que ayudaron a obtener un diseño final del sistema. En cada uno de esos modelos se abordaron aspectos importantes que tienen que ver con el SMA, sin embargo, en el caso del *agenteRBC* no fue posible detallar algunos modelos (esquema de agente y modelo de interacción) ya que involucraban el enfoque de Razonamiento Basado en Casos (RBC), y por consiguiente tampoco se obtuvo un diseño que permitiera la implementación del *agenteRBC* como en el caso de los otros agentes.

Para solucionar lo anterior y poder implementar el AsistenteHIM en su totalidad, el presente capítulo se centra en el estudio de este agente (*agenteRBC)* y la construcción del razonador basado en casos (RaBC) que se utiliza.

4.2 EL *agenteRBC*

Como primer paso en este proceso de construcción del RaBC, se describe el *agenteRBC* ya que es el agente que lo implementa.

El *agenteRBC* surge de la necesidad identificada al realizar el modelo Objetivo/Tarea de la fase de análisis del capítulo anterior. Como se pudo apreciar en la figura 3.6, tres de los objetivos del AsistenteHIM involucran actividades de soporte a problemas: (1) sugerir como llevar a cabo las tareas, (2) sugerir soluciones a problemas, y (3) dar ideas preventivas. Dichas actividades se asignaron a un agente, el agente responsable de "ayudar a solucionar los problemas" del usuario.

Como segundo paso era necesario definir la manera en la que el agente pudiera alcanzar sus objetivos. Entonces se decidió utilizar el RBC ya que es un enfoque útil porque es fácil de comprender (se asemeja a la forma en la que resolvemos los problemas en la vida real), y está orientado a resolver justamente las necesidades que se tenían para el agente: solucionar problemas, dar ideas en la realización de actividades o prevenir de posibles errores. Como otra ventaja de este enfoque, se tiene que las experiencias o conocimiento previo pueden provenir del conocimiento en la bibliografía, de algún experto humano, o del uso regular del sistema [Laureano01], lo que permite que la herramienta sea adaptable a las organizaciones donde se implanta.

Así pues, el agente se definió formalmente como el *agenteRBC*, que para su funcionamiento implementaría el Razonamiento Basado en Casos. A continuación se presenta una lista de las características principales del agente, pues influyen en el desarrollo del RaBC.

- Es identificado como el responsable de sugerir cómo llevar a cabo las tareas que realiza el usuario, sugerirle soluciones a problemas, o darle ideas preventivas mediante la utilización del enfoque de RBC.

- Brinda sus servicios al usuario a través del *agenteCoordinador*. Además puede tener relación con el *agenteBuscador* en caso de necesitar algún dato de los archivos RDF, o directamente con el usuario para el caso de algunas actividades propias del RaBC.

- Es el único que tiene acceso a la base de casos.

- Realiza las acciones especificadas en el esquema del *agenteRBC* (apéndice 2, tabla 2).

Cabe aclarar que el desarrollo del *agenteRBC* como tal (clase agenteRBC, comportamientos, interacciones, etc.) y el de los demás agentes, se aborda en el siguiente capítulo dedicado a la implementación del AsistenteHIM y a la integración de los sistemas (RaBC con el AsistenteHIM y este último con la HIM).

4.3 RAZONAMIENTO BASADO EN CASOS

En el segundo capítulo se introdujeron los conceptos fundamentales del enfoque de Razonamiento Basado en Casos, sin embargo, es necesario ahondar más en algunos temas de importancia para la construcción del RaBC.

Como se mencionó, el enfoque de RBC pertenece al área de la IA y se utiliza para la resolución de problemas. Se basa en la utilización de problemas similares ocurridos en el pasado (denominados casos), para encontrar soluciones a otros problemas, modificar soluciones existentes y explicar situaciones anómalas.

El proceso completo que se realiza en un sistema de RBC se puede exponer de distintas maneras según el autor, sin embargo, es ampliamente aceptada la versión de Aamodt y Plaza [Aamodt01], que lo representa como un ciclo de cuatro actividades (4R's): *Recuperar, Reutilizar, Revisar y Retener.*

Una descripción inicial de un problema define un nuevo caso. Este nuevo caso es usado para *recuperar* un caso de la colección de casos previos. A través de la *reutilización* el caso recuperado es mezclado con el nuevo caso, dando como resultado un caso solucionado (por ejemplo una solución propuesta al problema inicial). A través del proceso de *revisión* la solución es probada (por ejemplo siendo aplicada al entorno del mundo real y evaluada por un experto) y reparada si falla. Durante la etapa de *retención*, las experiencias útiles son retenidas para su reutilización, y la base de casos es actualizada con un nuevo caso aprendido, o con la modificación de algunos casos existentes.

Cualquier sistema RBC debe de llevar a cabo al menos la primera de las cuatro actividades, y la realización de las demás dependerá del tipo de sistema que se vaya a implementar: sistemas completamente automatizados, sistemas de sólo recuperación y sistemas que varían entre esos dos extremos. Tal decisión es importante ya que de ello depende la construcción del sistema RaBC.

De acuerdo a los objetivos del AsistenteHIM y principalmente del *agenteRBC*, se definió el RaBC como un sistema que implementa las actividades de *Recuperación, Reutilización* y *Retención*. Este sistema es un evaluador y un adaptador de casos, es decir, que si se quiere hacer una mejora al caso esto es posible de realizar. El sistema realiza la evaluación del caso y las soluciones que pueden ser tomadas para resolverlo con base en casos previos, después permite que el usuario modifique los casos para agregarlos como nuevos o modificados a la base de casos.

Como se explica en los capítulos 5 y 6, el RaBC no pudo ser integrado en su totalidad al AsistenteHIM por cuestiones de tiempo. Sólo se incluyó la actividad de *Recuperación*, sin embargo la funcionalidad de las otras dos actividades si se desarrolló (como se verá en la

siguiente sección de este capítulo) y está disponible para su integración con el AsistenteHIM con las modificaciones pertinentes.

El proceso de las 4R's se utiliza para describir el RBC como un ciclo de pasos secuenciales. Sin embargo, cada uno de dichos pasos involucra una serie de sub-pasos que hay que tener en cuenta a la hora de construir un sistema razonador basado en casos. Conforme a ello Aamodt y Plaza [Aamodt01] sugieren un paradigma orientado en *tareas*, en donde cada paso es visto como una tarea que el RaBC tiene que realizar.

En la vista orientada en tareas cada tarea es preparada de acuerdo a los objetivos del sistema, y una tarea es realizada por medio de la aplicación de uno o varios métodos. Para que un método sea capaz de llevar a cabo una tarea es necesario que tenga conocimiento del dominio general de la aplicación, así como información acerca del problema y su contexto (*ídem pp. 9*).

En la figura 4.1 se muestra la estructura de tareas-métodos que se refiere en el trabajo citado, donde las tareas están en letra normal y los métodos subrayados. En ese nivel de descripción, se supone que el conjunto de sub-tareas que se desprenden de una tarea, es suficiente para realizar con éxito dicha tarea. Un método especifica el algoritmo que identifica y controla la ejecución de las sub-tareas y accede y utiliza el conocimiento e información requerida. La descripción de cada método y cada tarea, así como su aplicabilidad en cada caso, queda fuera del alcance de este trabajo.

Al igual que en muchos campos de la IA, no hay métodos universales de RBC para cada dominio de aplicación. Por consiguiente, el reto es estandarizar métodos adecuados para resolución de problemas y aprendizaje en dominios de temas particulares, y para entornos de aplicaciones particulares. Con relación al modelo de tareas de Aamodt y Plaza, los problemas principales que enfrenta la investigación en RBC pueden ser agrupados en cinco áreas (*Ídem pp. 11*):

1. Representación de conocimiento
2. Métodos de recuperación
3. Métodos de reutilización
4. Métodos de revisión
5. Métodos de retención

Un conjunto de soluciones coherentes a estos problemas constituye un método de RBC.

El problema de representación de conocimiento se trata de manera teórica en el siguiente apartado (los casos), ya que es un tema de suma importancia para el proceso de construcción de cualquier RaBC. El mismo tema se trata de manera práctica al inicio del apartado 4.5.

Algunos de los métodos restantes se abordan al desarrollar el razonador de este proyecto de tesis, como se verá a lo largo del capítulo.

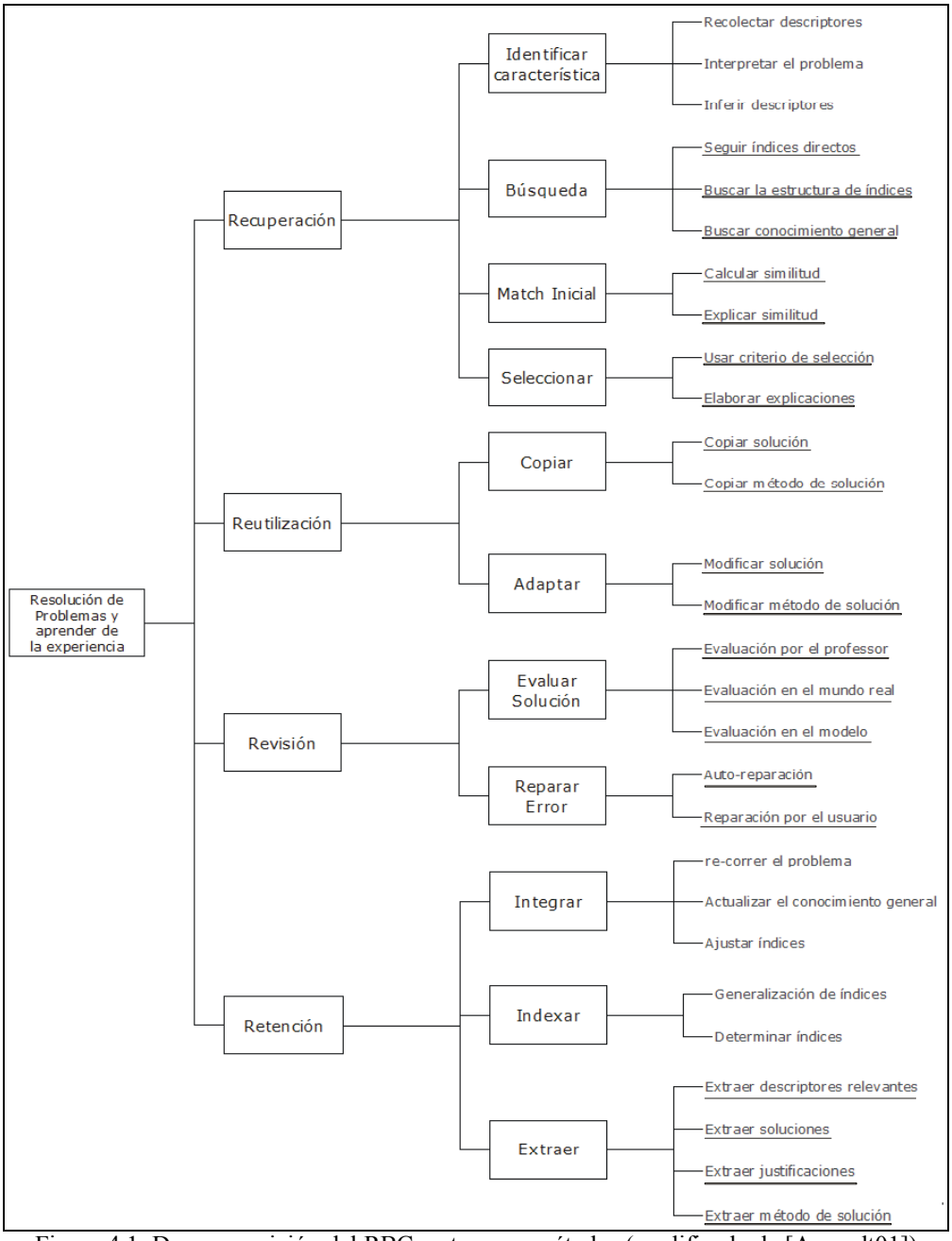

Figura 4.1: Descomposición del RBC en tareas y métodos (modificado de [Aamodt01])

4.4 LOS CASOS

Los casos que representan conocimiento específico se ligan a situaciones específicas y constituyen conocimiento a un nivel *operacional,* esto es, hacen explícito cómo una tarea fue llevada a cabo, cómo una pieza de conocimiento fue aplicada o qué estrategia en particular se usó para alcanzar un objetivo. Los casos almacenan el conocimiento que puede ser difícil de capturar en un modelo general, permitiendo razonar con cosas específicas cuando el conocimiento general no está disponible [Plaza01].

Los casos se pueden representar en varios tamaños y formas, pueden cubrir una situación que evoluciona con el tiempo (por ejemplo el seguimiento de un paciente a través de varias

enfermedades), pueden representar una vista superficial (como la elección de un tipo particular de ventana para un edificio), o pueden cubrir cualquier aspecto entre esos dos extremos. Pero lo que es común a todos los casos es que representan una situación experimentada, que cuando es recordada, forma un contexto en el cual el conocimiento embebido en el caso es presumiblemente aplicable y constituye un punto de partida para la resolución del nuevo problema dentro de la nueva situación [Bogaerts01].

Algunas cuestiones surgen de esta discusión, como la relación entre los casos y el conocimiento, si vale la pena almacenar todas las experiencias y si no, cuáles si y cuáles no. En relación a esto, Janet Kolodner [Kolodner01] indica que un caso se debe almacenar sólo si constituye una experiencia diferente de alguna manera de lo que estaba esperado, es decir, se almacenan las mayores variaciones a la norma. La idea fundamental es que si la diferencia es instructiva tal que brinde enseñar una lección para el futuro, que no se puede inferir fácilmente a partir de los casos almacenados, entonces sí se debe almacenar como un caso.

Debemos considerar qué tipos de lecciones puede enseñar un caso y el mejor indicio es que los casos almacenados en la biblioteca o base de casos (que es el repositorio donde se almacenan los casos) deben contribuir a alcanzar los objetivos del razonador. Se puede resumir que los principios de los casos son los siguientes:

- Un caso representa conocimiento específico ligado a un contexto.

- Los casos pueden venir en diferentes formas y tamaños, cubriendo espacios de tiempo largos o pequeños, asociando soluciones con problemas, consecuencias con situaciones, o ambas.

- Un caso almacena experiencias diferentes de lo que estaba esperado. Sin embargo, no todas las diferencias valen la pena para almacenarse. Los casos que valen la pena de almacenar, son los que enseñan una lección útil.

- Las lecciones útiles son aquellas que tienen el potencial de ayudar al razonador a alcanzar más fácilmente un objetivo o un conjunto de objetivos, o que advierten de la posibilidad de una falla o señalan un problema.

Así, se llega a la definición de caso:

> *Un caso es una pieza de conocimiento contextualizada, que representa una experiencia que enseña una lección fundamental para alcanzar los objetivos del razonador* [Kolodner01].

Los casos tienen dos partes:

1. La(s) lección (es) que enseña.

2. El contexto en el que puede enseñar su(s) lección(es): Es llamado *índices del caso* y nos indica bajo cuáles circunstancias es apropiado recuperar un caso.

Como proveedores de sugerencias, los casos proveen una solución propuesta que es adaptada a la nueva situación. Como proveedores de contexto, los casos proveen evidencia concreta para, o en

contra, de una solución o interpretación que puede conllevar a procedimientos de interpretación o evaluación.

Es importante que un razonador basado en casos comience con un conjunto representativo de casos que comprendan ejemplos de éxito y fallo en el tema que se trata con el sistema razonador. Los intentos exitosos son utilizados para proponer soluciones a los problemas nuevos y los intentos fallidos son usados para advertir de fallas potenciales.

Los razonadores basados en casos que implementan la fase de *retención,* se pueden volver más competentes a lo largo del tiempo, derivando mejores respuestas de las que podrían con menos experiencia.

Un razonador cuyos casos cubran más del dominio, será un mejor razonador que aquel cuyos casos cubran menos del dominio. Uno cuyos casos cubran instancias de éxito y fallo, será mejor que aquel cuyos casos sólo cubran instancias de éxito. Índices nuevos permiten al razonador ajustar su aparato de recuperación de manera que recuerde los casos en los momentos más apropiados.

4.5 CONSTRUCCIÓN DEL RAZONADOR BASADO EN CASOS

El conocimiento reunido hasta ahora es suficiente para comenzar la construcción del sistema razonador basado en casos (RaBC). Cabe mencionar una vez más que el desarrollo de cada RaBC varía de acuerdo al tipo de sistema que se desee y la forma de implementarlo. En el sistema del presente proyecto de tesis se desarrollan las actividades de *Recuperación, Reutilización* y *Retención* a través del marco de trabajo JColibri.

4.5.1 Objetivos del Razonador Basado en Casos

Como primer paso para construir un sistema razonador basado en casos, se definen los objetivos o tareas que el sistema realizará [Kolodner01]. De acuerdo a ello se define el tipo de sistema y se desarrollan todas las demás etapas como son la recolección de casos, la construcción de la biblioteca de casos, la recuperación, la reutilización y las pruebas.

Los objetivos que el sistema razonador basado en casos deberá alcanzar a través del *agenteRBC* son cinco:

1. Brindar una guía que indique la manera en que se deben llevar a cabo las actividades del modelo de procesos MoProSoft (proceso Gestión de Negocio).

2. Advertir de problemas que posiblemente surjan al realizar dichas actividades y, si es posible, sugerir soluciones.

3. Aumentar la memoria de los usuarios al proveerles casos de los que probablemente no estarían al tanto o no recordarían.

4. Dar información en respuesta a requisiciones.

5. Trabajar interactivamente con el usuario para incrementar el número de casos en la biblioteca y así mejorar sus indexaciones y resultados.

4.5.2 Recolección de casos

Para la tarea de recolección de casos, generalmente se consideran tres principios [Kolodner01]:

1. Los casos deben cubrir el rango de tareas que el sistema deberá soportar o llevar a cabo.

2. Sobre dichas tareas, los casos deben cubrir desde las soluciones conocidas hasta los errores conocidos.

3. La recolección de casos es un proceso incremental, al inicio se hace lo mejor que se puede y después, al ir probando y encontrando errores o huecos, se va modificando la biblioteca de casos.

Los siguientes pasos especifican brevemente cómo se deben de recolectar los casos:

1. Para cada tarea o sub-tarea que se realice en el sistema (HIM en nuestro ejemplo y el proceso de Gestión de Negocio específicamente), los casos deben ilustrar cómo llevar a cabo dicha tarea con métodos bien conocidos. Si hay varias formas de realizar la labor, todas se deben de registrar en los casos e indexarlos.

2. Para cada tarea o sub-tarea también se deben almacenar como casos los problemas bien conocidos, obstáculos, sus soluciones y las maneras de evitarlos. Se indexan combinando los atributos para predecir las fallas.

3. Para cada tarea o sub-tarea, se delinean las soluciones correctas e incorrectas (indicado en los dos pasos anteriores) y se almacenan en casos. Se indexan con las descripciones de sus consecuencias y los atributos que las definen.

Resumiendo, los casos deben de tener una lección a enseñar, contener al menos la información necesaria para su indexación, estar indexados de acuerdo a las situaciones en las que pueden proveer una guía, y contener suficiente información para ser entendidos y aprovechados por los humanos y las máquinas.

Para construir una biblioteca de casos se pueden considerar distintas fuentes de casos, por ejemplo una base de datos existente, registros de casos, archivos, revistas, libros de texto, sistemas para solución de problemas y expertos humanos. En el proyecto se identificó que se pueden utilizar las siguientes:

- Base de datos existente: Las lecciones aprendidas almacenadas en formato RDF (experiencias exitosas o de fracaso en relación a cualquier tema, detectadas y documentadas por los usuarios de la herramienta).

- Revistas y libros de texto.

- Expertos humanos: Las consultoras y todo el personal que participó en la implantación del modelo de procesos MoProSoft, de las cuatro empresas seleccionadas para las pruebas controladas[1].

Con respecto a los expertos humanos, se recolectó información para los casos por medio de entrevistas, pidiéndoles a los expertos que recordaran y contaran sus experiencias con el MoProSoft. Las entrevistas se realizaron con grabadora de voz, a una de las consultoras autorizadas para supervisar la implantación del MoProSoft en las empresas seleccionadas para las pruebas controladas, Angélica Su (Itera S.A. de C.V.) y a los responsables de dicha implantación en cada una de las tres empresas que contribuyeron en este proyecto de tesis:

1. ARTEC – Arquitectura en Tecnología de México S.A. de C.V: Ing. Edgardo Herrera.
2. MAGNABYTE – Magnabyte, S.A. de C.V. : Ing. Oscar Flores
3. SGA – Sistemas de Gestión Administrativas S.C.: Ing. Ángeles Naranjo.

El resultado de las entrevistas se puede ver en el apéndice 3.

A partir de la información obtenida en las entrevistas, otros documentos (MoProSoft y planes estratégicos de otras empresas) y la información contenida en los archivos RDF de la HIM, fue posible definir los casos que conforman la base de casos.

4.5.3 Construcción de la base de casos

La base de casos construida en el presente proyecto es una base de datos estructurada, que contiene los casos históricos que capturan los problemas reales y sus soluciones, las cuales surgen al momento de implantar el proceso de Gestión de Negocio de MoProSoft dentro de una empresa (sólo casos de empresas nivel 0 y 1 por ahora ya que no hay más información disponible dado que el MoProSoft es un modelo reciente).

La estructura de la base de casos que se muestra en la tabla 4.1 se definió conforme a la información disponible, y con la finalidad de que fuera eficiente para los procesos del ciclo RBC a llevarse a cabo en el RaBC. Se decidió utilizar una base de datos hecha con MySql DataServer 5.0 Community Edition [MySql01] sobre archivos de texto para que la distribución siguiera siendo libre (conforme a lo definido en el desarrollo de la HIM) y para facilitar la agregación de casos posteriores.

Nombre del campo	Tipo de datos	Descripción
caseId	Número	Sólo para identificar los casos
rol	Texto	rol: GD, RGN, RGP, RGPY, RGR o ANY
actividad	Texto	Actividad
nivelEmpresa	Número	0, 1, 2, 3, 4, y 5 (niveles de MoProSoft)
tamañoEmpresa	Texto	pequeña, mediana, grande o ANY
actividadObjetivo	Texto	Esto es parte de la respuesta del RaBC
actividadSolucion	Texto	Esto es parte de la respuesta del RaBC
actividadPrevencion	Texto	Esto es parte de la respuesta del RaBC

Tabla 4.1: Estructura de la base de casos para el AsistenteHIM

[1] Pruebas de la implantación del MoProSoft realizadas en cuatro empresas piloto como parte del Programa para el Desarrollo de la Industria de Software, PROSOFT [SE01].

Los primeros cinco campos forman los índices de los casos y los tres últimos son las soluciones de cómo realizar una tarea o qué prevenir al realizar alguna actividad, por lo que no son requeridos todos los campos en cada caso. Su utilización depende del servicio que esté brindando el *agenteRBC* (sugerir como llevar a cabo las tareas, sugerir soluciones o dar ideas preventivas).

Como en los casos están involucradas algunas entidades del dominio del problema: el rol, la actividad, el producto y la organización; se definieron tipos de datos específicos para dos de ellos: el tipo rol, que puede ser sólo GD, RGN, RGP, RGPY, RGR o ANY (ya que son los roles involucrados en el proceso de Gestión de Negocio, que es del cual nos ocupamos en este proyecto); y el tipo tamañoEmpresa que puede valer solamente “pequeña”, “mediana”, “grande” o ANY. Esos tipos de datos se utilizan en el siguiente apartado al construir el sistema RaBC.

Algunos ejemplos de casos son los que se muestran en las siguientes tablas (tabla 4.2, 4.3 y 4.4), en el apéndice 3 se pueden ver la totalidad de ellos.

Nombre del campo	Contenido
caseId	2
Rol	GD
Actividad	GNA1.1
nivelEmpresa	0
tamañoEmpresa	Pequeña
actividadObjetivo	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.
actividadSugerencia	Como GD Ud. puede sugerir la utilización del análisis FODA o lluvia de ideas entre sus colaboradores.
actividadPrevencion	

Tabla 4.2: Ejemplo del caso con identificador 2

Nombre del campo	Contenido
caseId	42
Rol	GD
Actividad	GNA1.1
nivelEmpresa	1
tamañoEmpresa	Pequeña
actividadObjetivo	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.
actividadSugerencia	Tomar en cuenta los documentos anteriores y actualizarlos o adaptarlos al MoProSoft.
actividadPrevencion	

Tabla 4.3: Ejemplo del caso con identificador 42

Nombre del campo	Contenido
caseId	44
Rol	GD
Actividad	GNA1.1
nivelEmpresa	0

tamañoEmpresa	Pequeña
actividadObjetivo	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.
actividadSugerencia	Se puede organizar un taller de planeación estratégica para definir conceptos y utilizarlos, que ayude a aprender y llevar a cabo las actividades de MoProSoft.
actividadPrevencion	

Tabla 4.4: Ejemplo del caso con identificador 44

4.5.4 Desarrollo del sistema RaBC

Como se mencionó en el capítulo 1, la construcción de un sistema razonador basado en casos es una tarea compleja donde se tienen que tomar varias decisiones importantes. El diseñador del sistema tiene que elegir cómo representar los casos, la estructura de organización de casos, cuáles métodos solucionarán las tareas RBC, cual conocimiento (aparte de los casos específicos) será utilizado por esos métodos [Bello-Tomás01], etc.

Afortunadamente ya existen varios marcos de trabajo o herramientas que ayudan con el cumplimiento de dicha tarea y que se mencionaron en el capítulo 1. Para el desarrollo del RaBC del presente proyecto se eligió el marco de trabajo JColibri [Colibri01]. En la figura 4.2 se muestra su interfaz gráfica:

Figura 4.2: Interfaz Gráfica de Usuario de JColibri 1.0 v. beta

4.5.4.1 Diseño del marco de trabajo JColibri

Un marco de trabajo es una aplicación semi-completa, que puede ser especializada para crear aplicaciones a la medida, proveyendo el diseño a sistemas en un determinado dominio [Bello-Tomás01]. Brevemente el diseño del JColibri comprende una jerarquía de clases Java y varios archivos XML. El marco de trabajo se organiza alrededor de los siguientes elementos:

- Tareas y métodos: Archivos XML que describen las tareas soportadas por el marco de trabajo así como los métodos que las resuelven.

- Base de casos: Conectores que soportan diferentes tipos de persistencia de casos, desde un archivo de texto hasta una base de datos.

- Tipos de datos*:* Elementos que permiten crear tipos de datos del usuario que facilitan las tareas del razonador.

- Casos: Numerosas interfaces y clases que son incluidas para proveer una representación abstracta de los casos que soporta cualquiera de las estructuras de casos.

- Métodos solucionadores de problemas (*PSM's*): Conformados por el código que soporta los métodos que resuelven las tareas especificadas por los archivos XML (primer punto). Cualquier usuario puede desarrollar sus propias tareas y métodos y por medio de la licencia de software libre enriquecer el marco de trabajo.

- Funciones de ayuda: Complementan a los *PSM's*. Las más importantes son las funciones de similitud pero existen otras más como las de adaptación.

- *Core JColibri*: Es el componente más importante del marco de trabajo. Se encarga de mantener la configuración del RBC y ejecutar las aplicaciones. Cuando se genera un sistema razonador, lo que se hace es configurar el *Core* con las tareas apropiadas, métodos, tipos de datos, casos, etc. Luego para correr la aplicación sólo se tienen que llamar a los métodos del *Core*. Este elemento está compuesto por dos elementos muy importantes:

    - *CBRState:* Mantiene la configuración de tareas y métodos.
    - *CBRContext*: Actúa como un pizarrón donde los métodos intercambian datos. Usualmente contiene la base de datos y los casos de trabajo.
    - Paquetes: Administran los componentes restantes: Tipos de datos, funciones de similitud, estructuras de casos, etc.

La figura 4.3 muestra la arquitectura del JColibri, es decir, la manera en que están organizados los elementos descritos.

Construir un sistema RBC con este marco de trabajo es un proceso de configuración (instanciación), asistido por una interfaz gráfica (figura 4.2) en la que se seleccionan las tareas que el sistema debe tener, y para cada tarea se asigna el método que realiza la labor.

El proceso sintetizado comprende los siguientes pasos:

- Definir la estructura de casos, la fuente de casos y la organización de la base de casos (estos dos últimos realizados en el apartado anterior).

- Seleccionar tareas y métodos hasta que el sistema esté completo.

- Una vez que el sistema está configurado, se genera el código Java del sistema RBC, que además incluye una interfaz gráfica de usuario que puede ser modificada o eliminada.

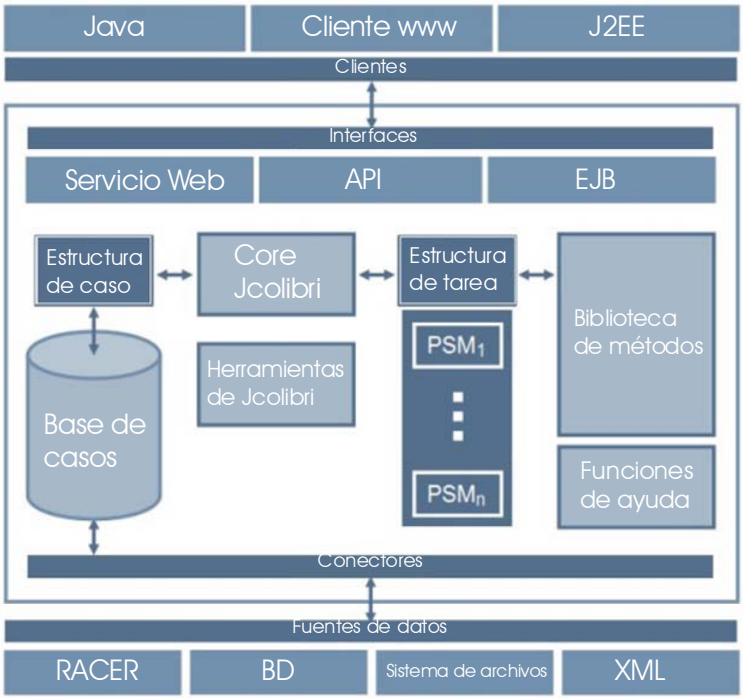

Figura 4.3: Arquitectura de JColibri, adaptado de [Bello-Tomás01]

4.5.4.2 Definición de los tipos de datos específicos

Antes de definir la estructura de casos del sistema, es necesario definir los tipos de datos específicos que se utilizarán. JColibri maneja tipos de datos estándar, como lo son *integer, double, boolean, string,* etc. Pero permite extender esa colección al crear tipos de datos específicos del usuario.

Para el RaBC se definieron dos tipos de datos específicos con la finalidad de que los atributos que fueran de esos tipos, no tuvieran la posibilidad de tener más valores que los permitidos por la aplicación:

- RolesEnum: Representa los nombres (siglas) de los roles que maneja el MoProSoft para el proceso de Gestión de Negocio: Grupo Directivo (GD), Responsable de Gestión de Negocio (RGN) y Grupo de Gestión (GG): Responsable de Gestión de Procesos (RGP), Responsable de Gestión de Proyectos (RGPY) y Responsable de Gestión de Recursos (RGR).

- EmpresaSizeEnum: Representa el tamaño de la empresa, que para nuestros casos la clasificación de pequeña, mediana y grande se consideró adecuada.

4.5.4.3 Definición de la estructura de casos

La definición de la estructura de casos es el primer paso para la definición del sistema RBC, pues es la base de todo el sistema y se utiliza a lo largo de todo su desarrollo. Consiste en definir los atributos que manejará el sistema, de acuerdo a la base de datos que contiene los casos.

Para definir y manejar la estructura de casos JColibri permite configurar los atributos de los casos por medio de tipos de datos estándares, tipos específicos de datos (previamente definidos), y funciones de similitud.

En el marco de trabajo un caso se compone por las siguientes partes:

- *Descripción:* Describe el problema por medio de varios atributos.
- *Solución*: Contiene la descripción de la solución del caso. Esta parte no está completamente implementada en la versión actual del JColibri.
- *Resultado:* Representa el resultado de aplicar el caso a una situación real (positiva o negativa).

Adicionalmente para cada atributo se tiene que definir funciones de similitud, que son funciones que evalúan la similitud entre los casos (más específicamente entre una consulta y un caso). De esa manera el método de recuperación puede elegir los casos más similares a la consulta [Bello-Tomás01].

La estructura de casos definida para el sistema razonador de este proyecto se conformó con la descripción de siete atributos que conforman un caso: rol, actividad, nivelEmpresa, tamañoEmpresa, actividadObjetivo, actividadSugerencia y actividadPrevencion. Para cada uno se definió su nombre, tipo, peso y función de similitud como se muestra en la figura 4.4.

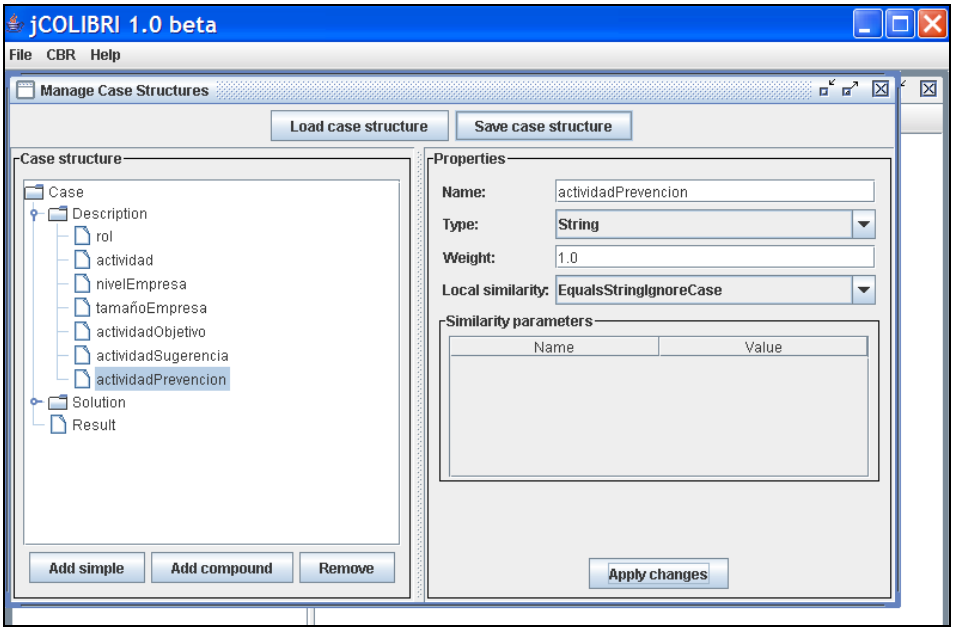

Figura 4.4: Definición de la estructura de casos para el sistema RaBC

4.5.4.4 Estableciendo el conector del sistema

Los sistemas RBC deben de tener acceso a los casos de una manera eficiente, problema que se complica conforme crece el tamaño de la base de casos. Con respecto a ello, JColibri divide el problema en dos aspectos relacionados: mecanismos de persistencia y organización en memoria.

La persistencia se consigue a través de conectores, que son objetos que *saben* como acceder y recuperar los casos desde un medio, y proporcionar esos casos al sistema RBC de una manera uniforme [JColibri01].

El conector implementa los métodos para iniciar una conexión con la base de casos (*init*), guardar casos (*storeCases*), borrar casos (*deleteCases*), recuperar todos los casos (*retrieveAllCases*), y terminar la conexión con la base de casos (*close*). Así, ofreciendo una interfaz común con la base de casos (la cual también implementa otra interfaz común con los métodos citados), permite que la organización e indexación elegidas para la base de casos no afecten la implementación de los métodos.

Tomando en cuenta lo anterior, el siguiente paso consistió en configurar el conector JDBC para cargar la base de casos previamente construida. JColibri provee conectores para archivos de texto, bases de datos MySql, u otros (figura 4.5).

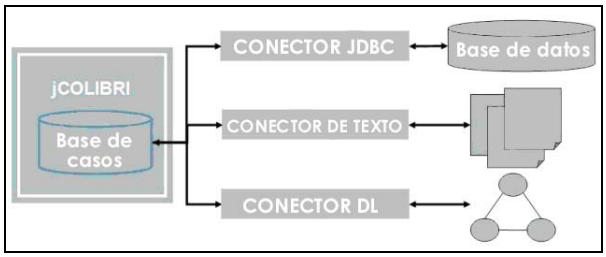

Figura 4.5: Conectores en JColibri [JColibri01]

En la figura 4.6 se puede apreciar la interfaz de configuración del conector que se utilizó (JDBC), en el que además de proporcionar parámetros como la estructura de casos, el controlador, el puerto, etc.; también se ligó cada atributo de la estructura de casos definida, con los campos de los casos de la base de datos.

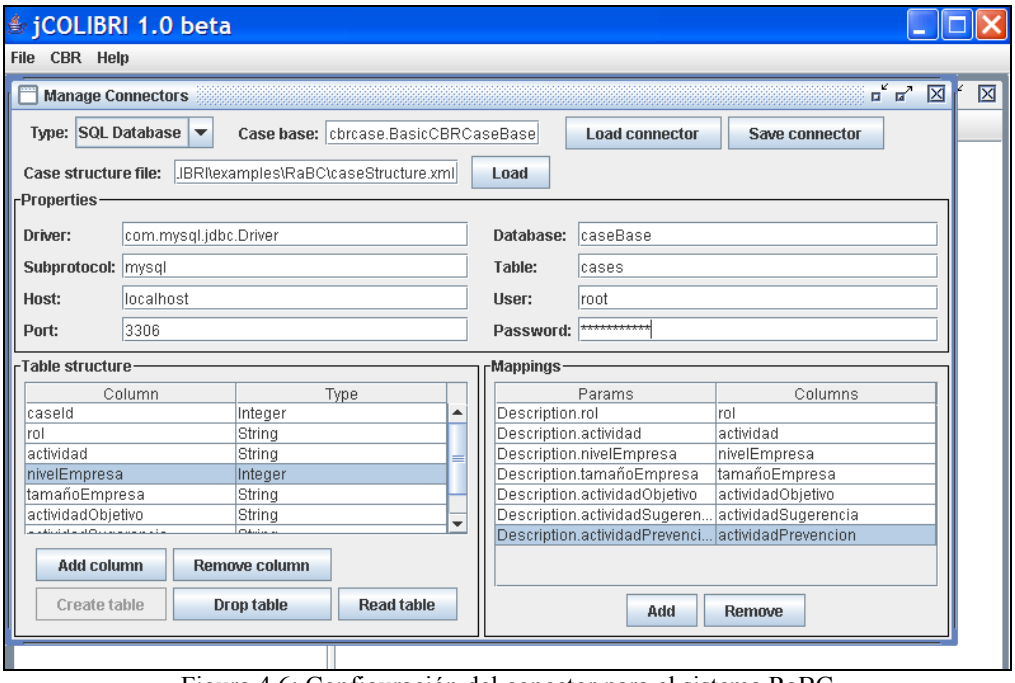

Figura 4.6: Configuración del conector para el sistema RaBC

4.5.4.5 El ciclo RBC

Como fue mencionado, el JColibri está basado en el paradigma tarea/método propuesto por Aamodt y Plaza [Aamodt01] para construir un sistema que lleve a cabo las cuatro fases del RBC (recuperar, reutilizar, revisar y retener).

Dicho paradigma consiste en que cada una de las cuatro tareas involucra un número de sub-tareas más específicas, y que existen métodos para solucionar las tareas (*problem solving methods, PSMs*), ya sea descomponiendo la tarea en más sub-tareas (métodos de descomposición), o solucionándola directamente (métodos de resolución).

El marco de trabajo utiliza dicha descomposición de tareas dentro de una infraestructura de tareas, que se va configurando al definir los métodos que las implementarán. Cada tarea puede ser resuelta por varios métodos, se elige el que se considere más apropiado y entonces se instancía.

Además de considerar las cuatro tareas del ciclo RBC, el marco de trabajo incluye dos tareas adicionales, llamadas Pre-Ciclo y Post-Ciclo, que como su nombre lo dice, la primera se ejecuta antes del ciclo RBC obteniendo los casos de la base de datos, y la segunda se ejecuta después del ciclo RBC para almacenar los casos en una capa persistente y cerrar las conexiones que se hayan hecho durante el proceso. En la figura 4.7 se ve la interfaz de configuración de tareas/métodos que provee el JColibri.

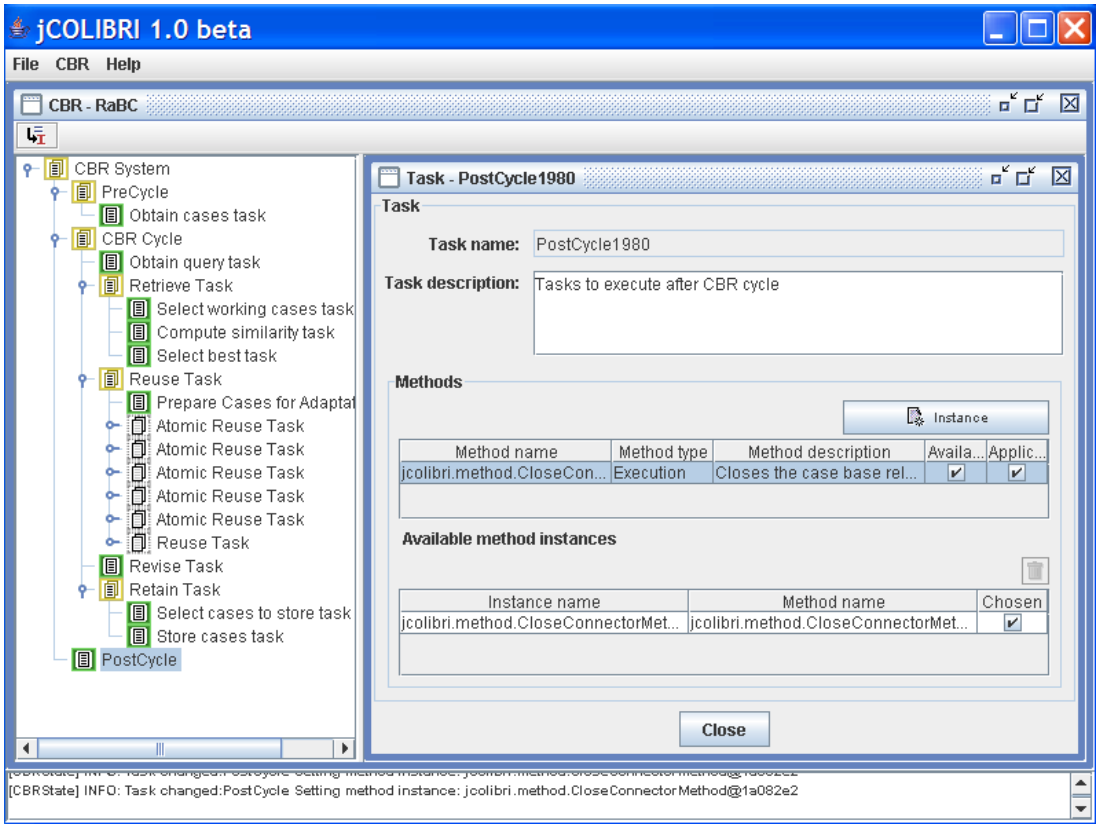

Figura 4.7: Estructura de descomposición de tareas y configuración del sistema RaBC

Está fuera del alcance de este proyecto dar una explicación completa de cada uno de los métodos solucionadores de problemas incluidos en el marco de trabajo. En su lugar se explicará su aplicación en las tareas principales del sistema RaBC, además de que en la siguiente figura se listan cada uno de ellos, de acuerdo a la tarea que implementan. Cabe mencionar que no han sido traducidas las tareas para facilitar su ubicación dentro de la herramienta.

PreCycle: jcolibri.method.PreCyleMethod
Obtain cases task: jcolibri.method.LoadCaseBaseMethod (connector.xml)
CBRCycle: jcolibri.method.CBRMethod
Obtain query task: jcolibri.method.ConfigureQueryMethod (caseStructure.xml)
Retrieve task: jcolibri.method.RetrieveComputationalMethod
Select working cases task: jcolibri.method.SelectAllCasesMethod
Compute similarity task: jcolibri.method.NumericSimCoputationMethod
Select best task:
Selects just the best of the found cases
Select de cases indicated by the user: ….method.SelectSomeCases
Reuse task: jcolibri.method.ReuseComputationalMethod
Prepare Cases for Adaptation Task: … .CopyCasesforAdaptationMethod
Atomic Reuse Task: jcolibri.method.NumericDirectProportionMethod
Atomic Reuse Task: jcolibri.method.NumericDirectProportionMethod
Revise task: jcolibri.method.ManualRevisionMethod (caseStructure.xml)
Retain task: jcolibri.method.RetainComputationalMethod
Select cases to store task: jcolibri.method.RetainChooserMethod
Store cases task: jcolibri.method.StoreCasesMethod
PostCycle: jcolibri.method.CloseConnectorMethod

Figura 4.8: Métodos solucionadores de problemas implementados en cada tarea del RaBC

4.5.4.5.1 Recuperación

Es el proceso de recuperar de la memoria un caso o conjunto de casos y en general consiste en dos sub-pasos:

Recuperación de casos anteriores: El objetivo es recuperar casos que sean útiles para llevar a cabo el razonamiento explicado en los siguientes pasos. Dichos casos son aquellos que tienen el potencial de hacer predicciones relevantes acerca del nuevo caso. La recuperación se lleva a cabo usando los atributos del nuevo caso como índices dentro de la biblioteca de casos, se recuperan aquellos casos etiquetados con dichos atributos (o con derivaciones de los mismos).

Seleccionar el mejor subconjunto: En este paso se selecciona el caso (o casos) más promisorios para razonar del conjunto de casos generados en el paso anterior. El propósito es separar de los casos relevantes un subconjunto más pequeño de casos candidatos mucho más apropiados.

Al contrario de las búsquedas a bases de datos que fijan un valor específico en un registro, la recuperación de casos debe estar provista con heurísticas que lleven a cabo coincidencias parciales, ya que en general no existen casos que coincidan exactamente con un nuevo caso.

Entre los métodos más conocidos para recuperación están el del vecino más cercano, de inducción, inducción guiada por conocimiento y recuperación de plantillas; los cuales pueden ser utilizados juntos o por separado [Bello-Tomás01]. Desgraciadamente JColibri no proporciona

información suficiente para saber cuál método se implementa, solamente se sabe que el método proporciona un procedimiento que selecciona (1) el mejor caso que considere, o (2) la cantidad de casos indicados por el usuario. En nuestro sistema razonador se utilizó la segunda opción con tres casos recuperados.

4.5.4.5.2 Reutilización

En la *reutilización* se compara el caso nuevo (*query*) con los casos recuperados, lo que suele ser complejo debido a que no existen muchos métodos genéricos para la adaptación de casos [Díaz-Agudo01]. Afortunadamente con JColibri se simplifica la tarea ya que provee un diseño abstracto donde la adaptación basada en casos puede ser alcanzada con varios métodos simples, para ajustar los valores de tipos primitivos que pueblan ese diseño abstracto. En la práctica se comprende mejor la idea:

Primero se lleva a cabo una tarea que prepara los casos para la adaptación, consiste en eliminar la información similar de los casos. Enseguida se ejecutan las sub-tareas que sean necesarias, las cuales se configuran con factores que fuerzan el resultado a ciertas especificaciones.

Para nuestra aplicación se especificó que en los casos de resultado, variaran todos los atributos menos la actividad; esto debido a que se espera que los casos puedan brindar ayuda para una actividad determinada, pero puede ser de utilidad ver las sugerencias existentes para el mismo caso pero de una empresa de nivel más alto por ejemplo (o con un rol distinto).

4.5.4.5.3 Revisión

En la etapa de *revisión* se modifica un caso anterior (recuperado) para que satisfaga las demandas de la nueva situación.

Se han identificado numerosos métodos de adaptación [Kolodner01] que se pueden usar para insertar, eliminar o sustituir una parte de la solución recuperada. Sin embargo, el método estándar de revisión de JColibri no involucra ninguno de dichos métodos de adaptación, sino que muestra al usuario el caso recuperado y le permite modificar cualquier atributo para luego almacenarlo en la biblioteca de casos, ya sea como el mismo caso modificado o uno nuevo.

4.5.4.5.4 Retención

La finalidad de la etapa de *retención* es que el razonador basado en casos aprenda de sus experiencias. Para llevar a cabo lo anterior el razonador requiere de retroalimentación, de modo que pueda interpretar qué estuvo bien y qué estuvo mal en sus soluciones y *retener* los casos útiles, ya que de otra forma repetirá sus errores y nunca incrementará sus capacidades.

Existen estrategias muy complejas para llevar a cabo esta última tarea pero al igual que la anterior, el JColibri delega la responsabilidad al usuario, permitiéndole elegir los casos a almacenar y almacenarlos.

4.6 EL RAZONADOR BASADO EN CASOS PARA EL AsistenteHIM

4.6.1 Resultado

Al terminar la fase de configuración de tareas y métodos, se concluyó con el desarrollo del sistema razonador basado en casos, lo que permitió generar una platilla de código Java que pudo ser adaptada al AsistenteHIM por medio del *agenteBuscador*.

El código java generado por JColibri incluye una interfaz gráfica de usuario temporal para la aplicación, con la que se puede ver el funcionamiento del sistema y realizar pruebas a la base de casos. La interfaz cuenta con botones para llevar a cabo el Pre-Ciclo, el CicloRBC (4R's) y el Post-Ciclo (figura 4.9).

Al dar clic en el Pre-Ciclo se recupera la base de casos. Al dar clic en el Ciclo-RBC se despliega una pantalla para introducir los datos para la consulta. Finalmente, en el Post-Ciclo se realizan los cambios indicados y se cierra la conexión con la base de datos que contiene los casos.

En el cuadro de texto grande se muestra lo que se llama contexto, que comprende los casos con los que se está trabajando, la consulta y los casos recuperados.

Como se puede ver en el recuadro *query* de la figura 4.9, los tipos de datos específicos se despliegan como lista para poder elegir sólo los valores permitidos.

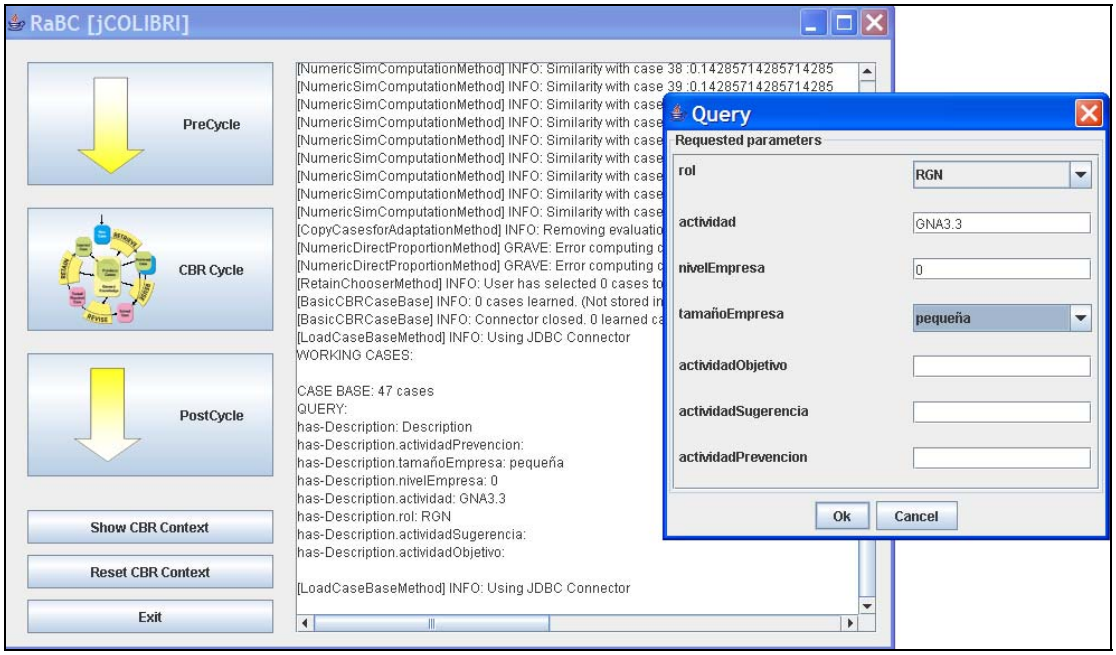

Figura 4.9: Interfaz del sistema RaBC (solicitando una consulta al usuario)

En la figura 4.10 se aprecia la manera en que se muestran al usuario los casos recuperados por el razonador, dando la oportunidad de llevar a cabo las etapas de *revisión* y *retención*, es decir, modificar y guardar los casos en la biblioteca de casos.

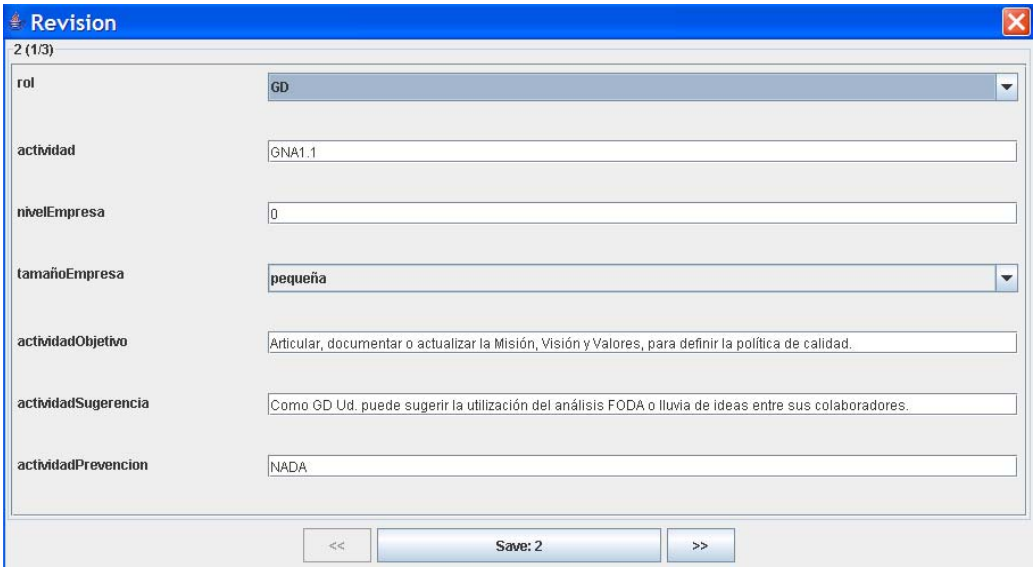

Figura 4.10: Resultado de la consulta en el sistema RaBC

4.6.2 Pruebas de sistema

Se realizaron pruebas al sistema y a la base de casos efectuando varias consultas, de las cuales se muestran dos a continuación (tablas 4.5 y 4.6).

Datos de la consulta	
Rol	GD
Actividad	GNA1.1
nivelEmpresa	0
tamañoEmpresa	Pequeña
Datos de los casos recuperados	
IdCaso	2
Rol	GD
Actividad	GNA1.1
nivelEmpresa	0
tamañoEmpresa	Pequeña
Objetivo	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.
Sugerencia	Como GD Ud. puede sugerir la utilización del análisis FODA o lluvia de ideas entre sus colaboradores.
Prevención	
IdCaso	44
Rol	GD
Actividad	GNA1.1
nivelEmpresa	0
tamañoEmpresa	Pequeña
Objetivo	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.
Sugerencia	Se puede organizar un taller de planeación estratégica para definir conceptos y utilizarlos, que ayude a aprender y llevar a cabo las actividades de MoProSoft.
Prevención	
IdCaso	0

Rol	ANY
Actividad	GNA1.1
nivelEmpresa	0
tamañoEmpresa	ANY
Objetivo	Entrar al sistema
Sugerencia	Si se tienen problemas con la definición de algunos términos de MoProSoft, consultar el documento del modelo en la página 5.
Prevención	

Tabla 4.5: Consulta de prueba a la base de casos

Como se puede observar en la tabla 4.5, la consulta especifica ciertos parámetros que el razonador toma en cuenta para recuperar los casos. Como se indicó que se recuperarían los tres mejores casos y no hay más que dos que cumplen con los parámetros especificados, el sistema recupera el tercero con variaciones a los atributos de rol, y tamaño de empresa, dando un caso que es el que considera que será de más utilidad al usuario.

Datos de la consulta	
Rol	RGN
Actividad	GNA3.3
nivelEmpresa	0
tamañoEmpresa	Pequeña
Datos de los casos recuperados	
IdCaso	23
Rol	RGN
Actividad	GNA3.3
nivelEmpresa	0
tamañoEmpresa	Pequeña
Objetivo	Generación de Propuesta de Mejoras al Plan Estratégico actual.
Sugerencia	Mantener la comunicación con el grupo de gestión y organizar sus propuestas de manera que se puedan presentar al GD de una manera eficiente.
Prevención	
IdCaso	34
Rol	GG
Actividad	GNA3.3
nivelEmpresa	0
tamañoEmpresa	Pequeña
Objetivo	Generación de Propuesta de Mejoras al Plan Estratégico actual.
Sugerencia	Como responsable de un área de la empresa, basar las sugerencias en resultados e informes para el RGN y el GD.
Prevención	
IdCaso	35
Rol	GD
Actividad	GNA3.3
nivelEmpresa	0
tamañoEmpresa	Pequeña
Objetivo	Generación de Propuesta de Mejoras al Plan Estratégico actual.
Sugerencia	Tomar en cuenta las propuestas de mejora del RGN y decidir con base en ello.
Prevención	

Tabla 4.6: Consulta de prueba a la base de casos

En este segundo ejemplo (tabla 4.6) los parámetros de la consulta sólo coinciden con un caso en la base de casos y el sistema encuentra otros casos con la variante del rol, que son los que decide mostrar. Como se dijo anteriormente, pueden ser útiles las sugerencias para un rol distinto, siempre y cuando sean para la misma actividad.

Con base en lo anterior, se puede decir que los resultados en el desarrollo del razonador basado en casos fueron satisfactorios.

En el capítulo siguiente se describe la arquitectura e implementación del AsistenteHIM en su totalidad, y por consiguiente, la manera en que se integró el RaBC a través del *agenteRBC*.

Alguien que jamás ha cometido un error, jamás ha intentado nada nuevo.

–A. EINSTEIN

Capítulo 5

CASO PRÁCTICO

En este capítulo se utiliza el trabajo generado en las secciones anteriores para definir una arquitectura del sistema asistenteHIM y desarrollar cada uno de sus componentes. Esto último implica la utilización de la *base de conocimiento* en RDF por el agenteBuscador, y el sistema Razonador Basado en Casos del agenteRBC, así como la integración con la HIM por medio del agenteCoordinador.

Finalmente, se muestra el funcionamiento del prototipo de la herramienta generada y las pruebas realizadas. Lo anterior representa la aplicación concreta y práctica, último paso del presente proyecto de tesis.

5.1 INTRODUCCIÓN

A lo largo de los capítulos anteriores se ha desarrollado la idea, primero, de un sistema que sirva al usuario como asistente y que le ayude a seguir las actividades del MoProSoft a través de la HIM, para después seguir con el análisis y el diseño de la herramienta AsistenteHIM[0]. Se ha dicho que el sistema brindará información al usuario de sus responsabilidades y posibilidades dentro de la empresa, tareas a realizar y responsables de ellas, sugerencias de la manera de llevarlas a cabo, recordatorios de tareas pendientes y coordinación del trabajo con otros usuarios; todo de acuerdo al MoProSoft.

Para implementar dicha herramienta se decidió utilizar el enfoque orientado a agentes de la Inteligencia Artificial, proponiendo un sistema multi-agente en el que cada agente busca en todo momento interactuar en su ambiente para llevar a cabo sus tareas a través del intercambio de conocimiento. Así, durante la fase de análisis en el capítulo 3, se detectaron y definieron cuatro agentes que conformarían el AsistenteHIM: *agenteCoordinador*, *agenteSupervisor, agenteBuscador* y *agenteRBC.*

En esa misma fase se desarrollaron varios modelos que ayudaron a obtener un diseño final del sistema, comprendido por los mensajes en pseudo-código que intercambiarían los agentes, su ontología y las clases de los mismos agentes, así como la descripción de su comportamiento en entidades de programación llamadas *behaviour*. De igual manera, en el capítulo 4 se cumplió con otra fase del desarrollo del AsistenteHIM, la construcción del Razonador Basado en Casos, que sería agregado al *agenteRBC* para que pudiera cumplir con sus objetivos.

Lo anterior deriva en la etapa final del proyecto de tesis, que es la implementación del sistema multi-agente, actividad que produce el sistema en términos de código fuente, scripts, archivos binarios, ejecutables, datos de configuración, etc.

La implementación está fuertemente ligada al diseño del sistema generado en el capítulo 3, dado que debe haber consistencia entre ellos. Dado un buen modelo de diseño, la implementación del sistema debe ser una traducción de módulos, interfaces, algoritmos y otras construcciones definidas en el modelo de dominio, en estructuras de datos, procedimientos y secuencias de instrucciones directamente soportadas por el lenguaje de programación seleccionado. Con respecto a esto último, cabe mencionar que la programación del SMA y sus componentes se basa en gran medida en la utilización del marco de trabajo JADE (con Java), ya que fue el que se eligió para el desarrollo del SMA.

En este capítulo se describe la arquitectura general del sistema, la implementación de cada agente y se analizan algunas pruebas y resultados del trabajo.

5.2 ARQUITECTURA GENERAL DEL SISTEMA

En el diagrama de organización fase 1, del capítulo 3 (figura 3.17) se presentó al asistenteHIM como un conjunto de agentes que interactúan con el usuario a través del *agenteCoordinador*, además se identificaron como recursos del sistema la base de conocimiento en RDF y la biblioteca de casos. Dicho diagrama fue útil para obtener el diseño del SMA, sin embargo, para comprender la implementación se requiere de un diagrama mucho más completo, en el que

queden explícitos todos los componentes relacionados para el buen funcionamiento del sistema, sean o no parte del SMA.

Como primer paso se analiza la Herramienta Integral para MoPrososft (HIM), ya que el SMA está embebido dentro de ella para cumplir con los objetivos del proyecto. Después se abordan las arquitecturas de los agentes y del SMA teniendo como base la plataforma JADE, para finalizar con una arquitectura global del asistenteHIM.

5.2.1 Arquitectura de la Herramienta Integral para MoProSoft

En el desarrollo de la HIM se tomaron como base varios patrones de diseño y un marco de trabajo (*Struts*), con el fin de mejorar la comprensión, dividir el trabajo y facilitar la integración de nuevos elementos al sistema [Vázquez01].

Para definir la arquitectura de la HIM fue utilizado el patrón de arquitectura MVC (Modelo, Vista, Controlador) que consiste en dividir un componente o un sistema en tres partes, facilitando la modificación o personalización de cada parte [Stelting01]. El *modelo* se ocupa de los cambios de estado del sistema, la *vista* de la representación del sistema, y el *controlador* de la funcionalidad del sistema, asignando las acciones a la vista según su impacto en el modelo.

Para cada componente del MVC se utilizaron diversos patrones de diseño, algunos de nuestro interés se muestran en la figura 5.1, que servirán para explicar la implementación de cada uno de los agentes en este capítulo.

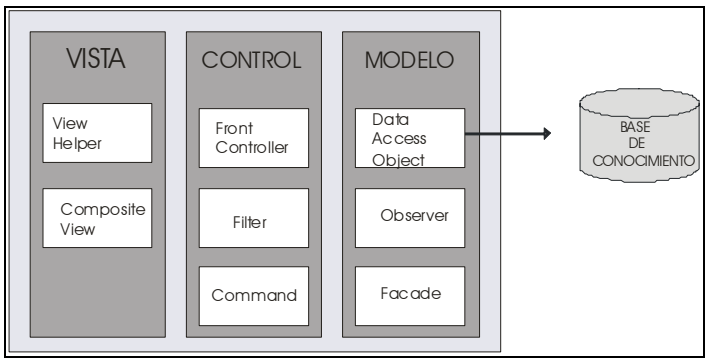

Figura 5.1: Arquitectura de HIM de acuerdo al patrón MVC

Entre otros componentes, en la capa de Vista se encuentra la interfaz gráfica de la HIM (JSP y HTML), en la capa de control las clases controladoras (*ActionMapping, ActionServlet, ActionForm*, etc.), y en la capa de modelo todo lo que tiene que ver con los datos (base de conocimiento, Jena, etc.).

El SMA se integró a la HIM en la capa de control y utiliza elementos de las capas de vista y de modelo. En las siguientes secciones se explica detalladamente este proceso, adicionalmente el apéndice 4 contiene una guía de las adiciones y cambios que se le hicieron al código de la HIM y a la estructura de archivos.

Cabe mencionar que con respecto a la configuración final de la HIM, se consideró sólo un servidor Web (Tomcat) que es el que contiene las páginas, las reglas del negocio y el acceso a datos, y cada cliente puede acceder a ellos a través de su navegador Web.

Hay mucho más que decir con respecto a este tema, sin embargo se espera que con la información proporcionada se consiga una comprensión adecuada de la implementación del AsistenteHIM. Una explicación más detallada de los componentes de la HIM queda fuera del alcance de este proyecto.

5.2.2 Arquitectura del Sistema Multi-Agente

En el capítulo 2 se menciona que existen dos arquitecturas principales para los SMA: la arquitectura *RETSINA* y la arquitectura *FIPA*. Como la implementación de nuestro sistema se apoya con la plataforma JADE, se utiliza la segunda arquitectura ya que es la que se maneja en esa plataforma.

La arquitectura *FIPA* (figura 5.2) proporciona la infraestructura para el desarrollo y uso de agentes, y contiene los recursos de hardware y software necesarios para poner en marcha esta infraestructura. Se compone de un Sistema de Administración de Agentes (*Agent Management System, AMS*), un Servicio de Directorio (*Directory Facilitator, DF*) y un Canal de Comunicación de Agentes (*Agent communication Chanel, ACC*).

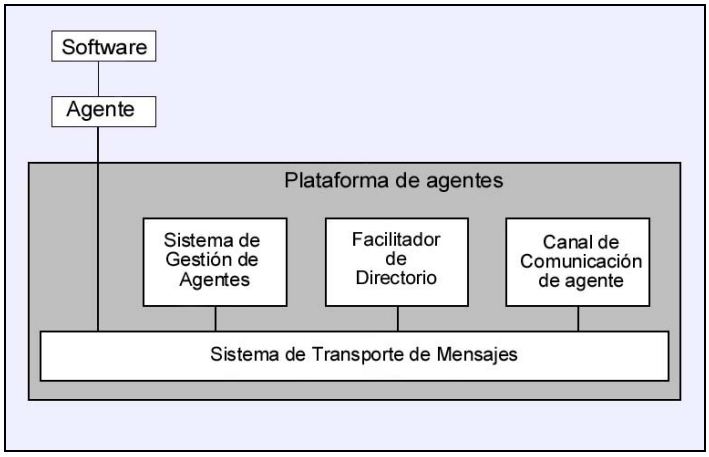

Figura 5.2: Arquitectura de la plataforma de agentes propuesta por FIPA [FIPA01]

JADE implementa la arquitectura *FIPA* y al considerarse un *middleware*[1] de agentes que provee una plataforma de agentes (entorno de ejecución) y un marco de trabajo de desarrollo (bibliotecas de clases), brinda algunas características extra. Tales características son herramientas gráficas para la gestión de agentes: Agente de Monitoreo Remoto (*Remote Monitoring Agent, RMA*), Agente de Servicio de Directorio (*DF*), así como un conjunto de agentes auxiliares para probar a los agentes desarrollados (*Dummy Agent, Sniffer Agent e Introspector Agent*). Además, proporciona una librería con la mayoría de los protocolos de interacción definidos por FIPA, gestión de ontologías y soporte para lenguajes de contenido.
A continuación se explican más detalladamente cada uno de los componentes de la plataforma JADE:

[1] Software que conecta dos aplicaciones separadas.

- El Sistema de Administración de Agentes (AMS) garantiza que cada agente en la plataforma tenga un único nombre. Es el encargado de proporcionar los servicios de páginas blancas y ciclo de vida, y de mantener el directorio de los identificadores de agentes y su estado. Cada agente debe registrarse con el AMS para tener un identificador válido.

- El Servicio de Directorio (DF) es un agente que proporciona el servicio de páginas amarillas. Un agente puede encontrar otros agentes que proporcionan los servicios que requiere para cumplir sus objetivos.

- El Canal de Comunicación de Agentes (ACC) es el elemento que controla el intercambio de mensajes entre los agentes:

    - La arquitectura de comunicación ofrece un mecanismo de paso de mensajes flexible y eficiente.
    - JADE crea y gestiona una cola de mensajes de entrada privados a cada agente.
    - Los agentes acceden a su cola de mensajes a través de una combinación de diferentes métodos (*blocking, polling, timeout* y *pattern matching based*) [Bellifemine06].
    - El protocolo de transporte se adapta a cada situación eligiendo de manera transparente el mejor protocolo disponible (Java RMI, HTTP, IIOP, etc.).
    - Los agentes envían/reciben objetos java que representan mensajes ACL, dentro del alcance de los protocolos de interacción.

- El Agente de Monitoreo Remoto (RMA) permite de manera remota la gestión, monitorización y el control del estado de los agentes, permitiendo por ejemplo iniciar la ejecución de un agente, parar y reiniciar agentes, y enviar mensajes.

- Las herramientas gráficas para el RMA y el DF ayudan a realizar las tareas arriba citadas pero de una manera gráfica. El *Dummy Agent, el Sniffer Agent* y el *Introspector Agent* son componentes que apoyan al programador a revisar y probar su sistema multi-agente.

Cada uno de los elementos anteriores forma parte del sistema multi-agente y no fue necesario desarrollarlos ya que son proporcionados por JADE. De tal suerte, el desarrollo restante del sistema consiste en la implementación de cada uno de los agentes utilizando los componentes descritos. Ese tema se trata en el siguiente apartado de este capítulo, pero antes es necesario describir la arquitectura general de ese elemento fundamental que es el agente.

Como se mencionó en la sección 2.5.2.2 del capítulo 2, el debate entre los distintos tipos de arquitecturas de agentes está aún abierto en el área de IA y de acuerdo a sus características, pueden ir desde las más simples (formadas por agentes reactivos), hasta las más complejas (compuestas por agentes que involucran actividades de aprendizaje, planeación y resolución de problemas de acuerdo a estructuras de creencias, deseos e intenciones).

En la clasificación presentada en [Wooldridge02], que divide las arquitecturas en deliberativas, reactivas e híbridas, ubicamos al AsistenteHIM como un SMA reactivo. Esto es, que actúa siguiendo un enfoque conductista con un modelo estímulo-respuesta, en el que los estímulos recibidos del exterior son procesados por capas especializadas que directamente responden con acciones a dichos estímulos.

Tomando en cuenta lo anterior, nuestros agentes son entidades de software que de acuerdo a los estímulos del entorno (interfaz de la HIM), tienen la capacidad de reaccionar, ya sea realizando algún procesamiento interno (por medio de comportamientos), o interactuando entre ellos al enviar mensajes (utilizando el lenguaje de comunicación de agentes ACL), a través de la plataforma JADE y su servicio de mensajería.

Administrando su cola privada de mensajes, los agentes deciden cuándo leer los mensajes y qué mensajes leer. En la figura 5.3 se ve la arquitectura interna de un agente en JADE.

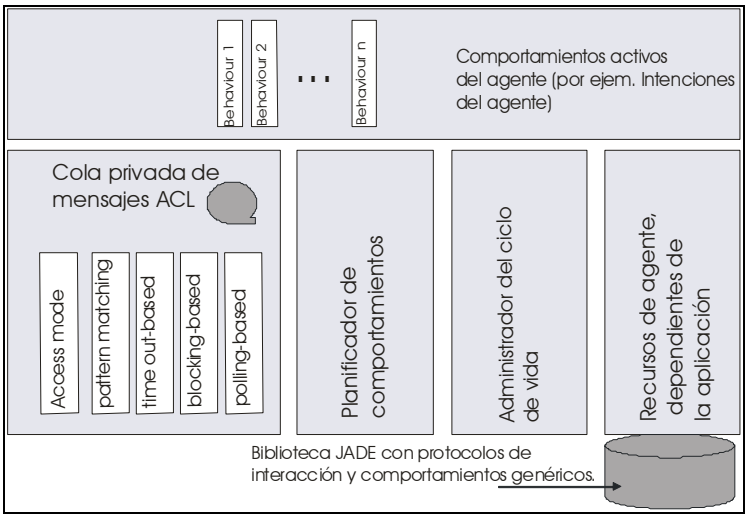

Figura 5.3: Arquitectura interna de un agente JADE

Ahora que se han explicado cada uno de los componentes de la plataforma JADE, finalizamos esta sección con la descripción de su estructura, ya que es importante para comprender la arquitectura global del AsistenteHIM.

En la figura 5.4 se muestra la arquitectura de JADE, la cual está definida para poder ser distribuida entre diferentes clientes (*host*), incluso con sistemas operativos distintos [Bellifemine01]. Cuando una plataforma es disparada, se crean automáticamente el Servicio de Administración de Agentes (AMS) y el Servicio de Directorio (DF), además el módulo de Comunicación entre Agentes (ACC) es configurado para permitir la comunicación por mensajes.

Al dividir la plataforma en varios *host*s, cada uno tiene una aplicación Java y por lo tanto una máquina virtual de java (JVM) ejecutándose en él. Así, cada una de las máquinas virtuales se convierte en un contenedor básico de agentes que provee un ambiente propicio para la ejecución de dichos agentes. El conjunto de todos los contenedores es llamado *plataforma,* el cual formará la capa JADE. Además, como se ve en la figura 5.4, es posible tener más de una plataforma.

El contenedor principal (*front-end*) es el contenedor en el cual viven los agentes AMS y DF, y donde el registro RMI es creado (utilizado internamente por JADE para el intercambio de mensajes). Los otros contenedores no contienen esos agentes, sino que se conectan al principal formando un entorno completo de ejecución para cualquier conjunto de agentes JADE distribuidos.

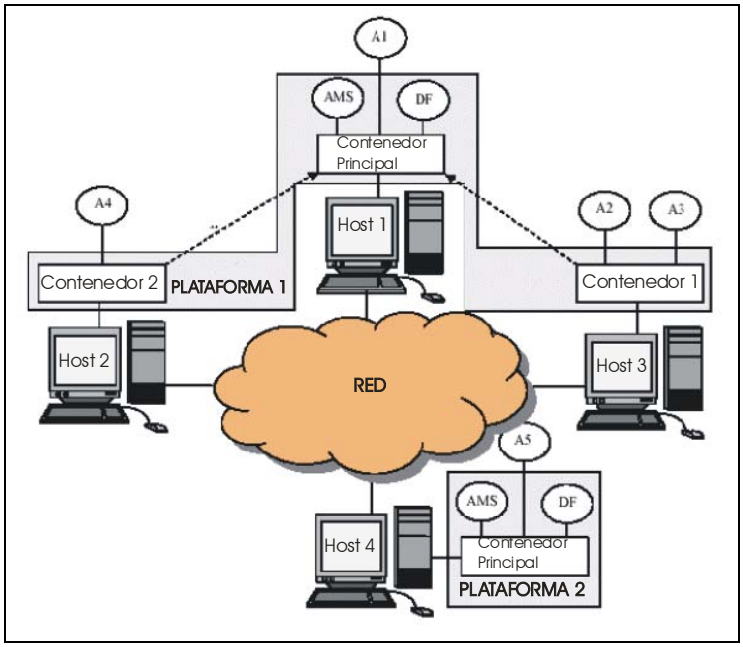

Figura 5.4: Plataforma JADE distribuida en varios contenedores

En el AsistenteHIM se definió una sola plataforma formada por dos contenedores que albergan a los agentes necesarios: Tenemos los agentes AMS y DMS definidos en el contenedor principal junto con los demás elementos de la plataforma. Luego se tienen los 4 agentes definidos en el diseño (agenteCoordinador, agenteSupervisor, agenteBuscador, agenteRBC) en un contenedor secundario llamado *Contenedor-1*.

El agenteCoordinador interactúa con el usuario a través de la interfaz gráfica de la HIM (JSP en la capa de vista) para realizar las tareas de brindarle los servicios al usuario y coordinar a los demás agentes.

5.2.3 Arquitectura del AsistenteHIM

Utilizando las arquitecturas de HIM y JADE se generó una arquitectura global (figura 5.5) donde se observa la composición del AsistenteHIM y la manera en que interactúa con los componentes independientes, ya sea la Internet, la HIM, el JColibri o los recursos.

Se ejecutan dos JVM en el servidor, una para tomcat (y la HIM funcionando) y la otra con la plataforma JADE, que a su vez contiene el contenedor principal con los componentes necesarios de un SMA.

Cuando un usuario realiza una acción en la interfaz de la HIM, es llamada una función que avisa al *AgentServlet* que ha ocurrido una acción y le manda los parámetros. El *AgentServlet* notifica al agenteCoordinador del evento y este último se encarga de realizar alguna acción para cumplir con sus objetivos.

El procesamiento de cada uno de los agentes fue explicado en el capítulo 3 y se implementa en la sección siguiente. No obstante, hay dos elementos que no han sido descritos: El *JColibri Core* y el *AgentServlet.*

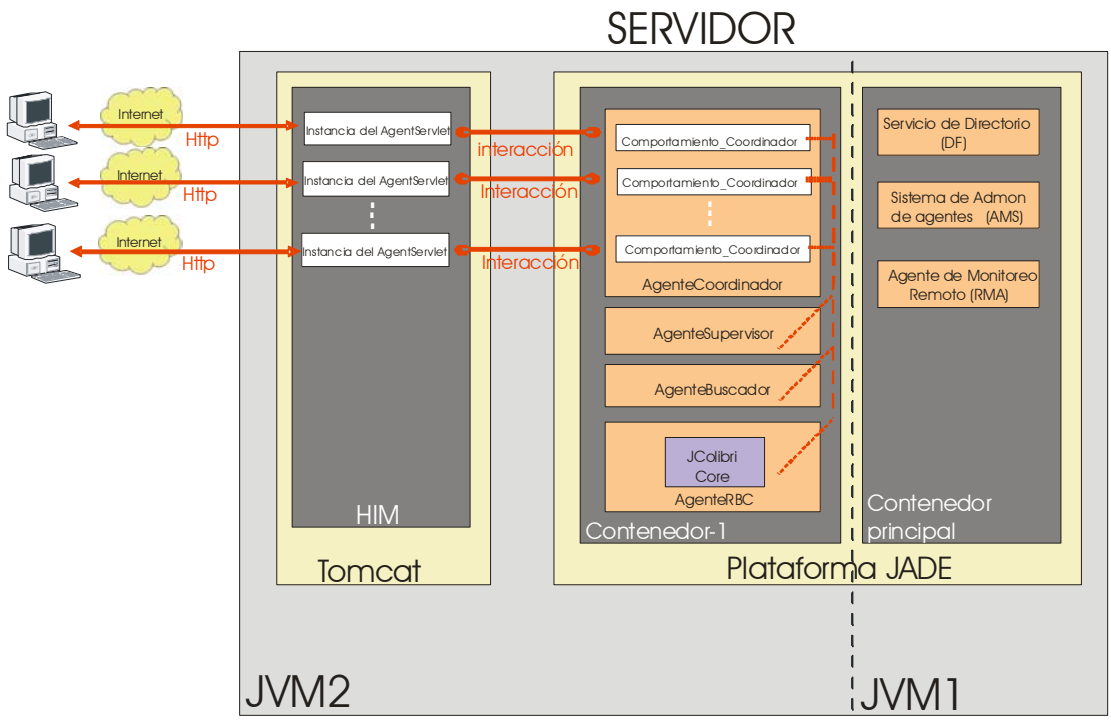

Figura 5.5: Arquitectura del AsistenteHIM

El *JColibri Core* es el componente más importante del marco de trabajo JColibri y fue descrito en la sección 4.5.4.1 del capítulo 4. Lo que hay que mencionar aquí es que (1) representa el razonador basado en casos que se implementó en el agenteRBC y (2) dentro de él se incluye la biblioteca de casos y la base de datos de donde se genera, por lo que ése recurso ya no aparece en el diagrama.

El elemento *AgentServlet* tal como su nombre lo indica, es un servlet que se implementó para poder ligar al agenteCoordinador a la interfaz de la aplicación Web HIM [Gandon01]. Se ejecuta en el servidor Tomcat cuando se llama a la función *popitup()*, esto es, cuando el usuario realiza alguna actividad en la interfaz de la HIM que tiene que ver con los servicios del SMA. Después de ejecutarse, el *AgentServlet* contesta la petición *httpRequest* proveyendo la información adecuada generada por el sistema multi-agente.

Es importante mencionar que el servlet requiere que una plataforma JADE ya esté corriendo en la máquina del servidor Tomcat antes de ejecutarse. Como se muestra en la figura 5.5, cada instancia del servlet responde a una petición *httpRequest* y es ligada a un comportamiento del agenteCoordinador. Cada instancia de dicho comportamiento realiza las acciones que considere pertinentes para poder proveer una página Web con la información generada por el AsistenteHIM. Las acciones llevadas a cabo por el agenteCoordinador pueden ser mandar mensajes a los demás agentes en solicitud de algún servicio o el procesamiento de algún dato interno.

En esta arquitectura el servidor del servlet levanta un hilo por cada petición, lo que es reflejado por el agenteCoordinador, permitiendo el manejo de múltiples peticiones en paralelo y la consecuente adaptación de los comportamientos para manejar los diferentes tipos de peticiones de distintas maneras. Con lo anterior se tiene una implementación concurrente capaz de atender las peticiones de múltiples usuarios.

A continuación se describe brevemente el código del *AgentServlet*, el cual se encuentra en el apéndice 4. No se incluye el código de los demás componentes para simplificar la comprensión, no obstante si es del interés del lector, toda la aplicación está en el disco anexo a este documento. El código se dividió en tres secciones:

Sección 1: El servlet es una extensión del *HttpServlet*, con referencias estáticas a los cuatro agentes del SMA.

Sección 2. El método de inicialización del servlet llama a un método estático sincronizado que asegura que los agentes fueron creados con su contenedor conectado a la plataforma local de JADE. Si no fue así, el método crea un contenedor no principal que se une a la instancia local de la plataforma JADE, y después crea una instancia de cada uno de los agentes y le pasa al agenteCoordinador un objeto sincronizador. El sincronizador es utilizado por el agente para notificar al servlet que ha iniciado y está listo para manejar peticiones.

Sección 3. El método *doGet(...)* es llamado cuando llega una petición *httpRequest*. Luego un objeto de interacción es usado para envolver los objetos de petición y respuesta, y sincronizar el servlet con el comportamiento del agente. El objeto de interacción es puesto en la cola de objetos del agenteCoordinador y el servlet espera entonces a que el agente le notifique que la respuesta http fue actualizada. Esto último sucede después de que el agente ha llevado a cabo los comportamientos y acciones necesarias para brindar el servicio a los usuarios. Es ahí donde entra todo el funcionamiento del SMA y que se describe detalladamente a continuación.

5.3 DESARROLLO E INTEGRACIÓN DE LOS COMPONENTES

La etapa de desarrollo del sistema multi-agente se realizó de manera satisfactoria ya que se utilizaron todos los elementos que se han generado a lo largo de la tesis. Se desarrollaron los cuatro agentes y se integró el SMA a la HIM.

Resultaron de utilidad sobre todo los esquemas de agentes de la etapa de análisis y la información generada en el diseño (clases de agentes, mensajes ACL, esquemas de interacción, etc.). Los elementos de la *base de conocimiento* en RDF y la interfaz de JADE también fueron importantes.

5.3.1 Plantilla básica de agente

Para simplificar la explicación de cada uno de los agentes, se describe en esta sección el los elementos básicos comunes a todos ellos, dejando las particularidades de cada uno para una explicación más detallada.

Se utilizó el IDE JBuilderX con Java Jsdk1.4 para el desarrollo de las clases que implementan los agentes, y las librerías de JADE que proveen las clases necesarias para su construcción. En estos agentes se utilizan dos clases principales que son parte de la librería de JADE: la clase *agent* de la que heredan los agentes y la clase *behaviour* la cual permite definir las tareas que deben realizar los agentes.

De acuerdo al análisis y diseño del capítulo 3, los agentes están construidos sobre un conjunto de estructuras de datos y paquetes de software, y manejan sus estructuras de datos internas a través de un conjunto de funciones. El estado de cada agente está dado por un conjunto de objetos que están dentro de sus estructuras de datos. Cuando el estado del agente cambia (esto puede pasar si recibe un mensaje de otro agente) un agente puede tomar acciones. Así pues, cada agente puede tener asociado un conjunto de acciones, con sus precondiciones y efectos respectivos. Puede solicitar la invocación de una acción para lograr algún propósito o puede requerir comunicarse con otro agente para lo cual necesita enviar mensajes comunicativos usando ACL.

Dentro de las características más importantes de los agentes del AsistenteHIM están las siguientes, que podemos identificar en el código de la tabla 5.1:

1. Un conjunto de variables globales para inicialización del agente. Aquí se definen las estructuras de datos para manipular los mensajes y su contenido, así como los nombres de los demás agentes si es que se conocen.

2. Eventos por medio de los cuales el agente puede percibir los mensajes ACL que están destinados para él. Consiste en uno o varios comportamientos cíclicos encargados de la recepción de mensajes, que se agregan a la ejecución de comportamientos del agente.

3. Métodos para su interacción tanto con la plataforma JADE, así como para el mantenimiento y consulta de sus propiedades y conocimiento. Lo que incluye los comportamientos *takedown()* (ejecutado cuando muere el agente), y las funciones necesarias para realizar acciones específicas de cada agente. Estos métodos y/o comportamientos son los que se describen por separado ya que varían en cada agente.

4. Módulos para el manejo del contenido de los mensajes ACL que están implementados y van de salida. Como los comportamientos para recibir mensajes y los métodos para el mantenimiento y consulta de conocimiento ya han sido implementados, se necesita un elemento que garantice el envío de los mensajes pertinentes a sus destinatarios.

1	`public class agenteSupervisor extends Agent {`
	`  private String datos[] = new String[4];`
	`  private String datosRespuesta[] = new String[4];`
	`  private int accion, agente;`
	`  private ACLMessage msgEnvia;`
	`  private ACLMessage msgRecibe, msgRecibe2;`
	
	`  //La lista de los agentes que "conoce" para que realicen las tareas`
	`  private AID[] agentes = {`
	`  new AID("agenteCoordinador", AID.ISLOCALNAME),`
	`  new AID("agenteSupervisor", AID.ISLOCALNAME),`
	`  new AID("agenteBuscador", AID.ISLOCALNAME),`
	`  new AID("agenteRBC", AID.ISLOCALNAME)};`
	
	`  // Inicializaciones del agente`
	`  protected void setup() {`
	
	`  //Se Agrega el comportamiento ciclico para recibir mensajes`
	`  addBehaviour(new recibirMensaje());`
	`  addBehaviour(new recibirMensaje2());`
	`  } //setup()`

2	//Comportamiento cíclico para recibir mensajes del agente Coordinador específicamente.
	  private class recibirMensaje extends CyclicBehaviour {
	  //aquí se pueden declarar variables
	  public void action() {
	  MessageTemplate template = MessageTemplate.MatchSender(agentes[0]);
	  msgRecibe = myAgent.receive(template);
	  if (msgRecibe != null) {
	  // Message recibido. Se procesa
	  agente = 2;
	  try {
	  System.arraycopy(msgRecibe.getContentObject(), 0, datosRespuesta, 0, 4);
	  }
	  catch (UnreadableException e) {
	  e.printStackTrace();
	  }
	  mensajeEnvia();
	
	  } //if
	
	  else {
	  block();
	  } //else
	  } //action
	  } //recibirMensaje
	
	--- más comportamientos ---

3	// operaciones de limpieza del agente
	  protected void takeDown() {
	  // Se imprime un mensaje de despedida
	  System.out.println("El agente Supervisor " + getAID().getName() +" se está muriendo... volvere!!!");
	  }//takeDown
	
	//Comportamiento para llevar a cabo una acción
	  private class miComportamiento extends OneShotBehaviour {
	  //aquí se pueden declarar variables
	  public void action() {
	
	  ---- código que implementa el comportamiento ---
	
	  } //action
	  } //Micomportamiento

4	
	
	  public void mensajeEnvia() {
	  msgEnvia = new ACLMessage(ACLMessage.INFORM);
	  msgEnvia.clearAllReceiver();
	  msgEnvia.addReceiver(agentes[agente]);
	  try{
	  msgEnvia.setContentObject(datosRespuesta);
	  }
	  catch (IOException e)
	  {
	  e.printStackTrace();
	  }
	  send(msgEnvia);
	  }//mensajeEnvia

Tabla 5.1: Código de muestra para un agente básico del AsistenteHIM

Los cuatro agentes se implementaron con un código semejante al de la tabla 5.1. La variación más significativa se da dentro de los métodos que realizan las funciones específicas de cada agente y que se describen a continuación para cada agente.

5.3.2 Agente Coordinador

Los objetivos del agenteCoordinador son monitorear las actividades del usuario y darle la información apropiada utilizando los servicios de los demás agentes.

Para alcanzar el segundo objetivo se implementaron los comportamientos que decidieran la acción a llevar a cabo, dependiendo de las acciones del usuario en la interfaz. Ello se realizó por medio de una estructura de decisión y el envío y recepción de mensajes con los demás agentes, tal como se describe en la plantilla básica de agente.

Sin embargo, alcanzar el segundo objetivo involucró acciones un poco más complejas dentro del agenteCoordinador como de la HIM.

Para poder monitorear las actividades del usuario en la interfaz gráfica de HIM (figura 5.6), fue necesario *incluir* al agenteCoordinador dentro de esa interfaz. Eso se hizo utilizando el elemento *AgentServlet* descrito en la sección anterior. Nótese que el *incluir* al agente dentro de la interfaz es más bien para facilitar la comprensión de la idea, ya que la comunicación de los eventos del usuario al agente se hace por medio de la interfaz de la HIM y el *AgentServlet.*

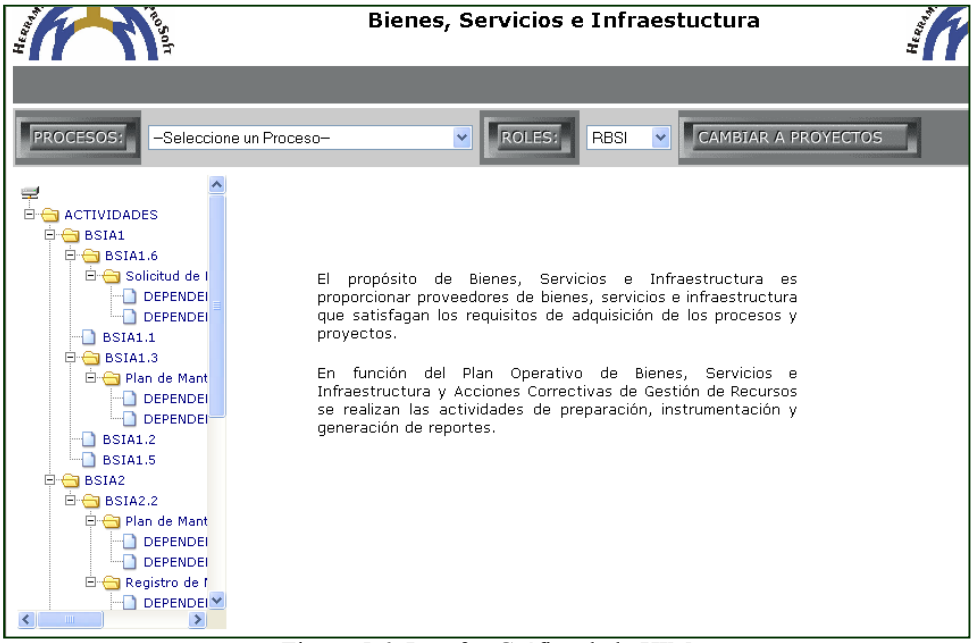

Figura 5.6: Interfaz Gráfica de la HIM

Cada petición (acción del usuario) se liga al comportamiento de recepción de acciones del usuario del agenteCoordinador, y cada instancia de dicho comportamiento ejecuta las acciones que considere pertinentes para poder proveer la información al usuario.

En el comportamiento que recibe las acciones del usuario, el agente decide (dependiendo de la acción del usuario) a cuál de los agentes le pedirá un servicio. Luego le manda un mensaje al agente elegido y cuando llegue la respuesta la procesa de tal forma que sea desplegada en una nueva página Web.

Hasta este paso se consideraba que el agenteCoordinador estaría en la interfaz de la HIM en espera de las acciones del usuario, para brindarle los servicios disponibles. Sin embargo, al integrar el SMA con la HIM se notó que había algunas cuestiones en el sistema que debían ser corregidas si es que se quería tener un sistema funcional y con ello cumplir con el objetivo principal del sistema:

> *...herramienta que incorporada a un sistema de apoyo al MoProSoft, HIM en este caso, funja como guía y supervisor del seguimiento de los procesos de dicho modelo y maximice la efectividad de los usuarios participantes, así como facilitar el aprendizaje del mismo.*

El problema se describe con las siguientes proposiciones:

1. Para entrar al sistema HIM el usuario se identifica y se despliega la pantalla principal, que es donde reside el agenteCoordinador (conceptualmente).

2. Al elegir el usuario un rol para trabajar, el agente detecta la acción y le muestra información relacionada con el elemento rol de acuerdo al MoProSoft.

3. Al elegir un proceso el agente le indica sus actividades pendientes en ese proceso y de acuerdo a su rol.

4. Al elegir una actividad el agente le puede mostrar información diversa: (1) información de la actividad y el producto que se realizan, (2) los roles relacionados con esa actividad, (3) las dependencias que existen con otras actividades y (4) sugerencias de cómo realizar la actividad.

5. Como el agente considera de utilidad los cuatro servicios, se despliega la información de todo, creando confusión por el exceso de información.

6. Al elegir una actividad que tiene dependencia con otra que no está hecha, el agente impide la realización de la actividad en turno, es decir, modifica su entorno de acuerdo a sus objetivos.

7. Si el usuario vuelve a cambiar de rol o de proceso, se vuelve a desplegar la información de los puntos 1 y 2, y así sucede cuántas veces se realice esa actividad por parte del usuario. Eso genera que la información ya no sea útil cuando el usuario ya ha recibido esa información varias veces, y más bien se torna molesto.

8. Lo mismo pasa al seleccionar una nueva actividad: se despliega la información de todos los servicios del AsistenteHIM con relación a las actividades.

Los problemas de los puntos 5, 7 y 8 se dan principalmente debido a que el agenteCoordinador se definió como reactivo y sólo reacciona a la acción del usuario sin llevar a cabo ningún procesamiento más estructurado.

En el punto 6 el comportamiento del agente es correcto, sin embargo también genera problemas: la HIM y el mismo MoProSoft están en etapa de pruebas y adaptación, por lo que no se puede ser tan restrictivo con el flujo de tareas.

Considerando lo anterior y con el objetivo de que con el AsistenteHIM se *maximice la efectividad de los usuarios participantes y facilite el aprendizaje del MoProsoft,* se decidió modificar la implementación inicial.

Se agregó un elemento intermedio entre la interfaz de la HIM y el agenteCoordinador: Un menú contextual para las actividades del árbol, y botones para las listas de roles y procesos. Estos cambios se describen en la sección de integración del SMA con la HIM.

En la figura 5.7 se muestran tales elementos y es donde se encuentra el agenteCoordinador.

Figura 5.7: Menú contextual y botones agregados a la interfaz HIM

Es importante mencionar que los cambios descritos conllevan a que el agenteCoordinador pierda parte de la reactividad y autonomía que se le había implementado. Eso no resulta satisfactorio desde el punto de vista del sistema multi-agente, sin embargo se puede encontrar justificación en los siguientes puntos:

- Con la modificación hecha, el AsistenteHIM ya está funcionando dentro de la HIM y facilita el manejo y aprendizaje del MoProSoft, objetivo importante de este proyecto de tesis.

- La HIM se encuentra en un estado de desarrollo, sin embargo cuando se consiga una versión funcional y adecuada para ofrecer a la Industria del Software, se podrá agregar el SMA con los comportamientos definidos originalmente.

- Desde el análisis y diseño, hasta la implementación, el sistema AsistenteHIM está orientado a ser un SMA, por lo que es posible la implementación de comportamientos más complejos en todos los agentes, para poder adaptarlos después a la HIM u otro sistema, todo sin modificar su estructura inicial y funcionando como SMA.

La clase del agenteCoordinador quedó como se ve en la figura 5.8:

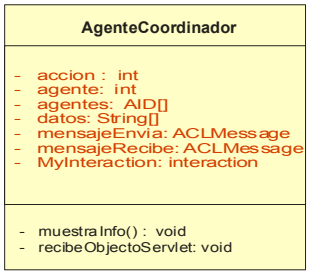

Figura 5.8: Clase agenteCoordinador del AsistenteHIM

En la sección de resultados de este capítulo se complementan los puntos anteriores, además de que son tratados en las conclusiones generales del presente proyecto de tesis.

5.3.3 Agente Supervisor

Los servicios que brinda del agenteSupervisor son la notificación de tareas pendientes, dependencia de actividades y los medios para facilitar la comunicación y coordinación entre usuarios. Para llevarlos a cabo utiliza a su vez los servicios del agenteBuscador. Este agente no tiene interacción con ningún recurso de la HIM ni del razonador basado en casos, por lo que no requirió ninguna implementación extra que las mencionadas en la plantilla de agente.

El comportamiento principal de este agente consiste en decidir qué acción realizar de acuerdo al contenido de los mensajes que recibe:

1. Si los mensajes provienen del agenteCoordinador se tiene que realizar una acción específica para regresarle la información de uno de sus servicios. Para esto se tiene que decidir cuál de los tres servicios es el que se requiere y se hace conforme a los datos del contenido del mensaje: acción que realizó el usuario, rol que tiene seleccionado, y el proceso y actividad si es el caso. Después de decidir qué servicio se va a proporcionar, se manda el mensaje apropiado al agenteBuscador, para que le regrese los datos que necesita de los archivos RDF.

2. Si el mensaje proviene del agenteBuscador, entonces se analizan los datos del contenido del mensaje, que serán respuesta a peticiones anteriores (punto 1). Luego se formatean de acuerdo al servicio que se esté brindando, para que el agenteCoordinador los pueda mostrar al usuario.

3. Finalmente, se genera un mensaje con la información del servicio y se envía al agenteCoordinador con preformativa *inform.*

El código de éste y todos los agentes se puede consultar en el disco compacto anexo a este documento.

5.3.4 Agente Buscador

La actividad fundamental de los agentes en el AsistenteHIM es el intercambio de información, por lo que un elemento importante es el agenteBuscador. Como ya se ha dicho, él provee el servicio de información a todos los agentes, así que su comportamiento se puede simplificar en

que (1) recibe mensajes, (2) los analiza, de acuerdo a ello (3) realiza una búsqueda a la base de conocimiento en RDF y finaliza (4) mandando los datos al agente que se los solicitó.

La estructura del agente corresponde a la de la plantilla de agente descrita en la sección anterior. Aquí el módulo que merece mención es el que se encarga de las búsquedas en los archivos RDF.

Para comenzar con la explicación de tal módulo, es necesario explicar la implementación de una parte de la capa del modelo de la HIM. Esta parte se ocupa del acceso a los datos almacenados en RDF y utiliza el patrón de diseño DAO (*Data Access Object*).

Los componentes del patrón son:

- *BussinessObject:* Objeto de negocio que representa al solicitante de datos. Es el objeto que requiere acceder a la fuente de datos para obtener y almacenar los datos.
- *DataAccessObject:* El DAO abstrae la implementación del acceso a datos a los objetos de negocio.
- *DataSource:* Representa la implementación de la fuente de datos.
- *TransferObject:* Objeto utilizado como un transportador de datos. Es creado y llenado por el objeto DAO y puede ser leído y modificado por el objeto de negocios.

En la figura 5.9 se aprecia el diagrama de clases del patrón DAO y en la figura 5.10 la aplicación de tal patrón en la HIM.

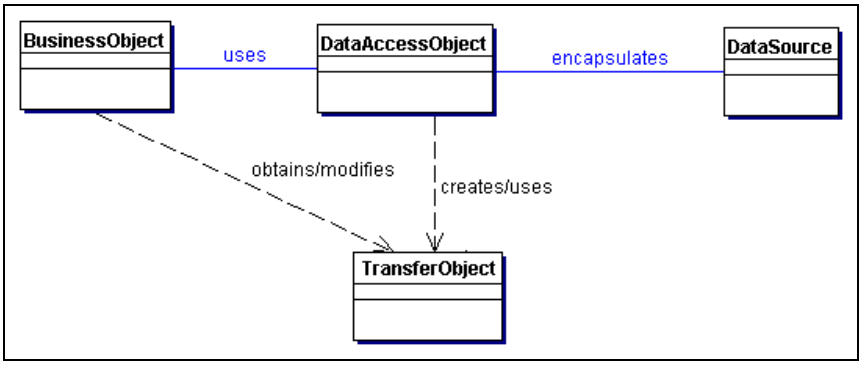

Figura 5.9: Diagrama de clases para el patrón DAO

En la primera parte de la figura 5.10 está la interfaz mediante la cual los objetos del negocio interactúan con los DAO. Así, estos últimos hacen uso de los servicios proporcionados por los DAO sin que tengan que enterarse cómo lo hacen.

En términos generales el procesamiento consiste en que el objeto de negocio instancia al objeto DAO al que le solicita los datos. Éste último accede a la fuente, recupera los datos y los deposita en un objeto *ValueObject*. El objeto de negocio trabaja con estos datos y si desea guardar los cambios manda llamar al DAO para que actualice la fuente con los datos contenidos en el *ValueObject*.

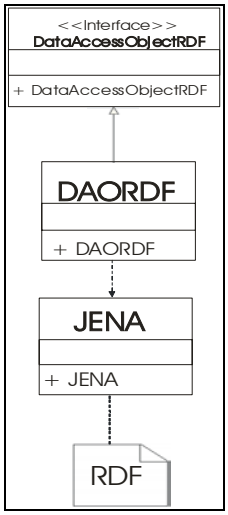

Figura 5.10: Diagrama de aplicación del patrón DAO en la HIM

Dentro de los objetos DAO se realiza la consulta de los datos a los archivos RDF mediante la utilización de la API de Jena v.1.x. Así, cada DAO contiene consultas RDQL que regresan el contenido a un vector de resultados. Se definieron varios objetos DAO, uno para cada elemento presente en las ontologías de HIM, por ejemplo DAORDFActividad, DAORDFProceso, DAORDFProducto, DAORDFOrganización, etc.

Como parte de los objetivos de este proyecto de tesis era aprovechar el trabajo hecho en la HIM por otros compañeros, y en especial el marco de trabajo en RDF; se decidió utilizar los objetos DAO para las consultas del agenteBuscador. Eso fue posible gracias a que las clases del AsistenteHIM están dentro del proyecto HIM y ahí están definidos los objetos DAO.

El agenteBuscador juega el papel del objeto de negocio, instanciando al objeto DAORDFActividad, DAORDFOrganización y DAORDFProducto, para obtener los datos que proporcionan las consultas ya definidas para la HIM.

Las consultas necesarias para el agenteBuscador son:

1. Lista de actividades pendientes en un proceso (gestión de negocio para esta etapa de implementación), correspondientes a un determinado rol.

2. Lista de roles que participan en determinada actividad y sus datos de usuario.

3. Lista de actividades de las que depende determinada actividad.

4. Información (ayuda) de determinado rol.

5. Información (ayuda) de determinada actividad.

6. Información (ayuda) de determinado producto.

La mayoría de las consultas ya estaban hechas para la HIM, sin embargo se tuvieron que definir algunas otras por lo que se modificó la clase DAORDFActividad.java. Para no generar ningún problema con el funcionamiento de la HIM, se creó otro objeto DAO con el contenido de

DAORDFActividad y las consultas extra que se necesitaron generadas utilizando el API de Jena y la tecnología RDF: el DAORDFActividad_AH.

También se hicieron pequeñas modificaciones a los archivos actividad.rdf y actividad.owl (del proyecto HIM en MetaInformación) para implementar algunas características que faltaban para el funcionamiento de las nuevas consultas. En el apéndice 4 se explican los cambios hechos a la HIM, y en el disco compacto se incluyen todos los archivos modificados.

En la tabla 5.2 se muestra un método de los agregados para el AsistenteHIM en el DAORDFActividad_AH que se encarga de recuperar la información de una actividad determinada. A continuación la descripción del código:

1. Se hacen las inicializaciones necesarias: crear un objeto tipo archivoRDF para establecer la conexión con el archivo en RDF a utilizar, se crean las rutas de los archivos RDF y se leen tales archivos. Finalmente, se crea un vector y un objeto actividad para almacenar los resultados de la consulta.

2. Se arma la consulta con formato RDQL y se manda ejecutar, almacenando el resultado en un objeto *iterator*.

3. El objeto *iterator* contiene el resultado, que es un conjunto de objetos actividad. Se recorre cada elemento completando los valores de sus atributos y se almacenan los objetos actualizados en un vector de resultados.

4. Finalmente, se regresa el objeto con los resultados al agente para el procesamiento de la información.

1	`public Vector RecuperaAyudaActividad(String idProceso, String idActividad) {`
	`archivoRDF = new ArchivoRDF();`
	`this.rutaProcesoMPS = this.rutaProcesoMPS + idProceso;`
	`this.rutaProcesoMeta = this.rutaProcesoMeta + idProceso;`
	`this.modeloMPS= this.recuperarModelo(this.rutaProcesoMPS);`
	`this.modeloMeta = this.recuperarModelo(this.rutaProcesoMeta);`
	`Vector actividadesPadre = new Vector();`
	`Actividad actividad;`

2	`char coma = '"';`
	`String nomActividad = coma + idActividad + coma;`
	`String queryString = "SELECT ?actividad, ?nombreMPS, ?ayuda " +`
	`"WHERE ( ?actividad , <RDF:type>, <actividad:Actividad>), "+`
	`" ( ?actividad , <actividad:ayuda>, ?ayuda), "+`
	`" ( ?actividad , <actividad:nombreMPS>, ?nombreMPS) "+`
	`" AND ?nombreMPS EQ "+nomActividad+" "+`
	`" USING RDF FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#> , "+`
	`" actividad FOR <file:///C:/HIM/BC/Configuracion/Ontologias/actividad.owl#> ";`
	`//ESTA CONSULTA RECUPERA LA AYUDA DE DETERMINADA ACTIVIDAD`
	`Iterator iter = archivoRDF.consulta(queryString, this.modeloMeta);`

3	`while (iter.hasNext()) {`
	`ResultBinding res = (ResultBinding) iter.next();`
	`actividad = new Actividad();`
	`actividad.setIdentificador(this.extraeLocal(res.get("actividad")));`
	`actividad.setNombre(this.extraeLocal(res.get("nombreMPS")));`
	`actividad.setDescripcion(this.extraeLocal(res.get("ayuda")));`
	`actividad.setHecha(false);`
	`actividad.setFactible(true);`

	actividad.setPermitida(true); actividadesPadre.add(actividad); }
4	return actividadesPadre;}

Tabla 5.2: Código de un método del objetoDAORDFActividad_AH que recupera la información de una actividad determinada

Las consultas le regresan al agente un *ValueObject* (vector) con el resultado, y posteriormente el agente debe procesar esa información para mandarla a los agentes que se la solicitaron.

El procesamiento involucra el análisis y manejo de los objetos (tipo actividad, producto u organización) regresados por cada uno de los DAO, para darles el formato correcto que esperan los demás agentes para su utilización. En todos los casos, el agenteBuscador regresa por medio del contenido de un mensaje de respuesta, un vector tipo *String* con los datos requeridos.

5.3.5 Agente con Razonamiento Basado en Casos

Los objetivos del agenteRBC son sugerir la manera de realizar las actividades y dar ideas preventivas que le sean de utilidad al usuario. Eso lo hace con la ayuda del Razonador Basado en Casos (RaBC) desarrollado y explicado en el capítulo 4.

La estructura general del agenteRBC corresponde a la descrita en la plantilla de agente y el método que merece mención es justamente el del RaBC.

Como se dijo en el capítulo 4, el RaBC se desarrolló y probó con la ayuda del marco de trabajo JColibri, y al final del proceso fue posible generar una clase Java con el código necesario para correr la aplicación de manera independiente, o en su caso, agregarla a otro sistema. Esto último es lo que se hizo con el agenteRBC el cual hace uso de dicha clase Java (RaBC.java) para poder brindar sus servicios.

Cabe mencionar que RaBC.java hace uso de las bibliotecas del JColibri, por lo que deben de ser agregadas al proyecto AsistenteHIM y por consiguiente a la misma HIM. El código de la RaBC.java se incluyó en el apéndice 4.

Como se indicó oportunamente en la sección 4.3 del capítulo 4, el RaBC se definió y construyó como un sistema que implementa las actividades de *Recuperación, Reutilización* y *Retención*. Sin embargo, en el AsistenteHIM no fue posible utilizarlo de esa manera por cuestiones de tiempo en la etapa de implementación y sólo se integró con la actividad de *Recuperación*, con lo que sigue siendo un sistema de razonamiento basado en casos pero de sólo recuperación.

La dificultad se debió a que el JColibri implementa sus actividades por medio de una interfaz gráfica y la ayuda del usuario (sección es 4.5.4.5.2 y 4.5.4.5.3 del capítulo 4), utilizando el paquete *java swing*[2]. Éste paquete es para desarrollar elementos gráficos en aplicaciones *stand-alone*, lo que no es compatible en absoluto con la interfaz para Web de HIM, que se basa en HTML y JavaScript por ser mediante JSP's.

Lo anterior implica que para integrar el RaBC al AsistenteHIM, se tenían que hacer las siguientes modificaciones para cada una de las actividades en las que tuviera que ver una interfaz

[2] Ver pantallas de la interfaz del RaBC que se generó en el capítulo 4.

gráfica, esto es, en las tres actividades de nuestro RaBC (*Recuperación, Reutilización* y *Retención*).

1. Modificar los métodos definidos dentro del API de JColibri, que corresponden a la actividad, para que no generen las interfaces gráficas con *java swing*, utilizadas tanto para recuperar los datos de entrada, como mostrar los datos de salida.

2. Sustituir de algún modo la manera en la que se recuperan los datos del usuario por medio de las interfaces, para hacer llegar dichos datos al RaBC y pueda trabajar sobre ellos.

3. Recuperar los datos de respuesta (casos) resultados del razonamiento basado en casos.

Las actividades anteriores se llevaron a cabo para adaptar la primera actividad (*Recuperación*), y como se indicó, no se alcanzaron a realizar para las otras dos actividades. El proceso se describe a continuación y sería el mismo en caso de querer adaptar las actividades restantes.

1. La actividad de *Recuperación* consiste en recuperar casos anteriores de utilidad conforme a un nuevo caso, que es proporcionado por el usuario mediante una interfaz gráfica de consulta (figura 4.9, capítulo 4). Se tuvo que eliminar dicha interfaz gráfica, lo que se logró modificando el código del método *ConfigureQueryMethod* del marco de trabajo JColibri, que es el que se utiliza en la actividad de *Recuperación.*

2. Luego para hacerle llegar al RaBC los datos del nuevo caso, se modificó el código de la clase RaBC generada con JColibri, para que recibiera los datos como parámetros del agenteRBC.

3. Por último, desde el agenteRBC se recuperó el primer caso de respuesta generado por el RaBC y se obtuvieron los campos de objetivo, sugerencia y prevención.

Los cambios hechos al código mencionado se documentaron en el apéndice 4.

El agenteRBC al tener los datos correspondientes a los campos de objetivo, sugerencia y prevención del caso recuperado (que son oraciones en lenguaje natural), procede a enviarlos como respuesta al agenteCoordinador para que sean mostrados al usuario proveyendo el servicio de RBC.

5.4 INTEGRACIÓN CON LA HIM

Tomando cada una de las partes descritas en las secciones anteriores de este capítulo, tenemos que el AsistenteHIM está conformado por:

- Cuatro clases que implementan los agentes: agenteCoordinador.java, agenteSupervisor.java, agenteBuscador.java y agenteRBC.java.

- Tres clases que hacen posible la integración del agenteCoordinador en la interfaz gráfica de la HIM: el servlet AgenteServlet y los componentes Interaction.java y Sinchronizer.java.

- Una clase que implementa el Razonamiento Basado en Casos: RaBC.java.
- Una clase que implementa el objeto de acceso a datos DAORDFActividad_AH.java y su respectiva interfaz IDAORDFActividad.java.
- Bibliotecas necesarias para el funcionamiento y uso de JADE y JColibri
- Los archivos modificados actividad.rdf y actividad.owl en la carpeta MetaInformación de la base de conocimiento de la HIM, no conforman el AsistenteHIM estrictamente hablando, sin embargo son necesarios para su funcionamiento (agenteBuscador).

El AsistenteHIM fue integrado a la HIM agregando casi todas las clases anteriores al proyecto en JBuilder, agrupadas en un paquete llamado *asistentehim*.

Para ser consistentes, las clases DAO se agregaron al paquete *moprosoft.him.bc.dao,* ya definido en la HIM y utilizado para almacenar todos los objetos DAO de la aplicación.

Para terminar, se agregaron las bibliotecas de JADE y JColibri a las propiedades del proyecto HIM en JBuilder.

Además de lo anterior, en la descripción del agenteCoordinador se agregó un elemento intermedio entre la interfaz de la HIM y el agenteCoordinador: Un menú contextual para las actividades del árbol y botones para las listas de roles y procesos, por lo que existen otros elementos, que si bien no forman parte del AsistenteHIM, son indispensables para su funcionamiento en la HIM:

- Script menuContextual.js que contiene el código en javaScript que genera el menú contextual.
- Script actividad.js modificado (ya formaba parte de la HIM).
- Hoja de estilo menuContextual.css con las opciones de formato del menú contextual.
- JSP PlantillaProcesos.jsp modificado para agregar la función que llama al servlet *AgentServlet*.

El código de los archivos anteriores también es incluido y descrito en el apéndice 4.

En la figura 5.11 se pueden identificar los elementos correspondientes al AsistenteHIM dentro de la estructura del proyecto HIM en JBuilder.

Figura 5.11: Elementos correspondientes al AsistenteHIM dentro de la estructura del proyecto HIM en JBuilder

5.5 PRESENTACIÓN Y PRUEBA DEL PROTOTIPO

La idea fundamental al presentar el prototipo del AsistenteHIM, es mostrar la funcionalidad de la herramienta desarrollada y la forma de utilizarla dentro de la HIM.

Se hace la demostración del sistema ejecutando la aplicación HIM desde JBuilder y visualizando la herramienta a través de un navegador de Internet, utilizando las funciones que tienen que ver con el AsistenteHIM. No se realizaron pruebas con usuarios reales dado que la HIM aún no está en condiciones de ser liberada al público.

Como se trata de determinar si el comportamiento del sistema implementado satisface su especificación, los diferentes casos de prueba se diseñaron tomando como referencia los objetivos en las hojas del diagrama general de objetivos del capítulo 3 (tema 3.2.2.1.2), donde se definió como objetivo principal del AsistenteHIM el *maximizar la efectividad de los usuarios y su nivel de aprendizaje y satisfacción con el MoProSoft.*

Antes de comenzar con los casos de prueba se ejecutó la HIM y se ingresó al sistema con un usuario determinado. Se mostró la página principal y se desplegaron los elementos requeridos para nuestras pruebas.

La interfaz es la que se aprecia en la figura 5.7 y a continuación se llevaron a cabo cada uno de los casos de prueba de acuerdo a los objetivos especificados para el AsistenteHIM.

1. Objetivo: *Recordar tareas pendientes*

Al dar clic en el botón  situado debajo de la lista de procesos se llama al agenteCoordinador y se le envían los datos, es entonces cuando el agente analiza la información y decide que la acción a realizar es la de proporcionar la información de las tareas pendientes del rol RGN en el proceso Gestión de Negocio.

Se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en una nueva ventana (figura 5.12):

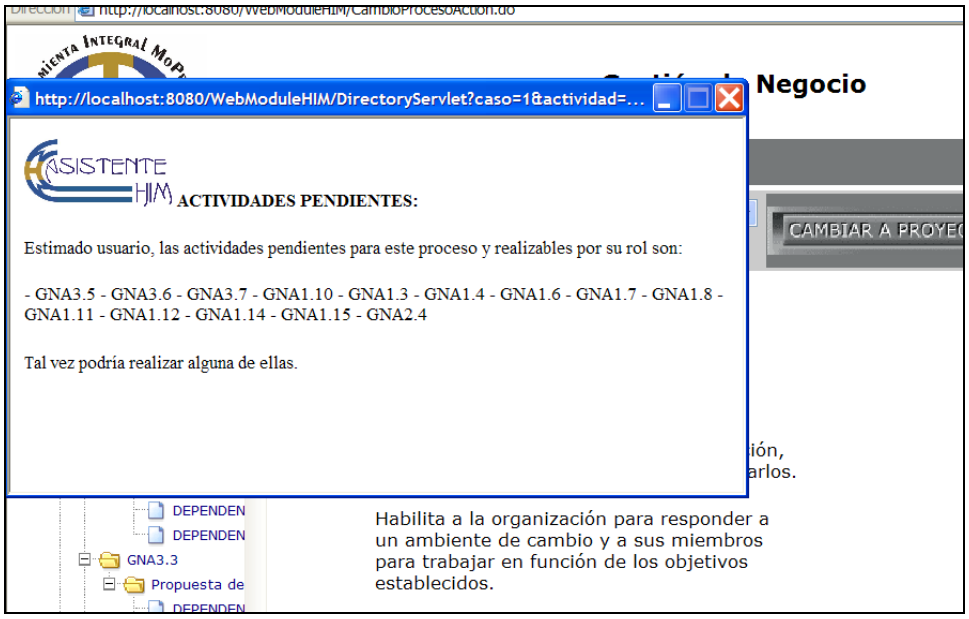

Figura 5.12: Página Web generada en respuesta del AsistenteHIM a la petición del usuario (*tareas pendientes*)

Se ha analizado el archivo GestionNegocio.rdf y se observa que las tareas no hechas en el proceso Gestión de Negocio y realizadas por el rol RGN, si coinciden con las mostradas por el AsistenteHIM.

Objetivo: cumplido.

2. Objetivo: *Coordinar y contactar usuarios*

En el árbol de actividades que se despliega en la interfaz de la HIM, al dar clic derecho sobre el elemento *Propuesta de Mejoras* de la actividad *GNA1.8*, se despliega el menú contextual. Al dar clic en la opción *Roles relacionados*, se llama al agenteCoordinador y se le envían los datos, es entonces cuando el agente analiza la información y decide que la acción a realizar es la de coordinar y contactar a los usuarios relacionados con la actividad seleccionada.

Se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en una nueva ventana (figura 5.13):

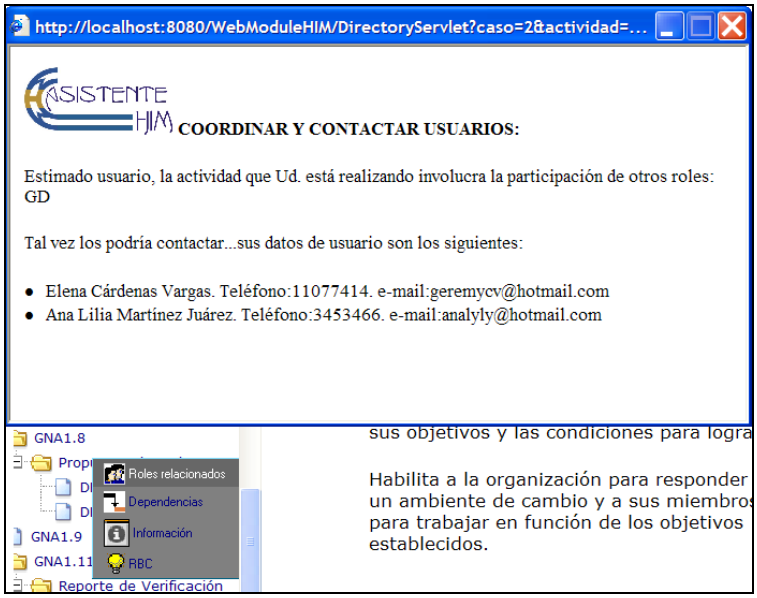

Figura 5.13: Página Web generada en respuesta del AsistenteHIM a la petición del usuario (coordinar y contactar usuarios)

Se han analizado los archivos en RDF y se observa que la información almacenada si coinciden con los datos mostrados por el AsistenteHIM.

Objetivo: cumplido.

3. Objetivo: *Dar información de roles*

Al dar clic en el botón  situado debajo de la lista de roles se llama al agenteCoordinador y se le envían los datos, es entonces cuando el agente analiza la información y decide que la acción a realizar es la de proporcionar la información del rol RGN.

Se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en una nueva ventana (figura 5.14).

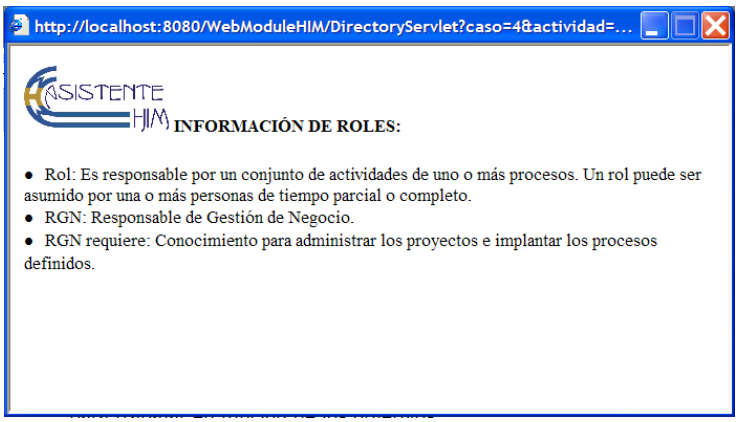

Figura 5.14: Página Web generada en respuesta del AsistenteHIM a la petición del usuario (dar información de roles)

Se ha analizado la información de acuerdo al MoProSoft y se observa que la información almacenada si coincide con los datos mostrados por el AsistenteHIM.

Objetivo: cumplido.

4. Objetivos: *Dar información de actividades y productos*

En el árbol de actividades que se despliega en la interfaz de la HIM, al dar clic derecho sobre el elemento *Lecciones Aprendidas* de la actividad *GNA3.7*, se despliega el menú contextual. Al dar clic en la opción *Información*, se llama al agenteCoordinador y se le envían los datos, es entonces cuando el agente analiza la información y decide que la acción a realizar es la de brindar la información de ayuda correspondiente a la actividad y producto seleccionados.

Se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en una nueva ventana (figura 5.15):

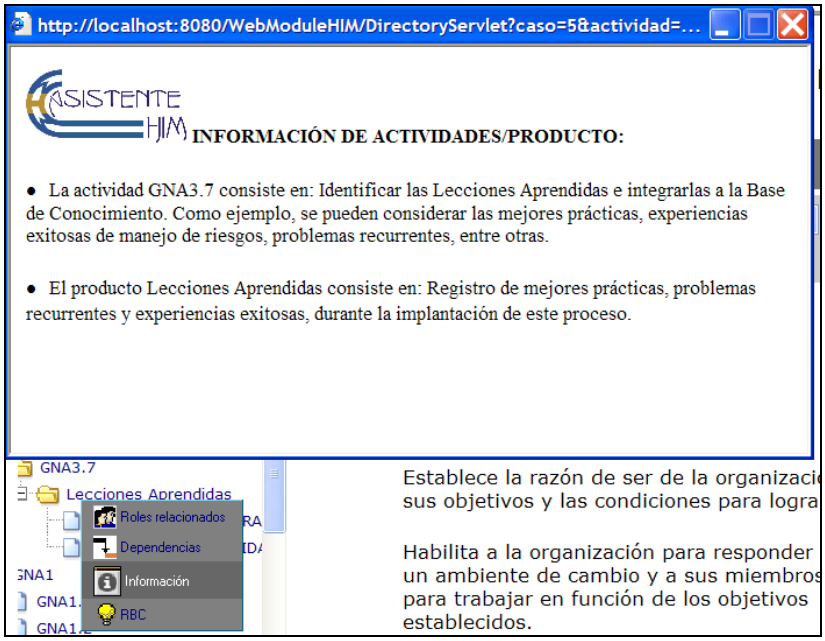

Figura 5.15: Página Web generada en respuesta del AsistenteHIM a la petición del usuario (información de actividades/producto)

Se ha analizado el archivo GestionNegocio.rdf y se observa que la información de ayuda de la actividad *GNA3.7* y del producto *Lecciones Aprendidas* si coincide con la mostrada por el AsistenteHIM.

Objetivo: cumplido.

5. Objetivos: ***Tareas RBC (sugerir cómo llevar a cabo las tareas, sugerir soluciones y dar ideas preventivas)***

En el árbol de actividades que se despliega en la interfaz de la HIM, al dar clic derecho sobre el elemento *Propuesta de Mejoras* de la actividad *GNA3.3*, se despliega el menú contextual. Al dar clic en la opción *RBC*, se llama al agenteCoordinador y se le envían los datos, es entonces cuando el agente analiza la información y decide que la acción a realizar es la de brindar los servicios del agenteRBC con los datos del rol, proceso, actividad y producto seleccionados. El agenteCoordinador completa el caso además con los campos de tamañoEmpresa=pequeña y nivelEmpresa=0|1 ya que por lo pronto no hay información real con casos de empresas con características diferentes en cuanto a esos dos atributos.

Se lleva a cabo todo el procesamiento descrito en las secciones anteriores, y se despliega la información adecuada en una nueva ventana (figura 5.16):

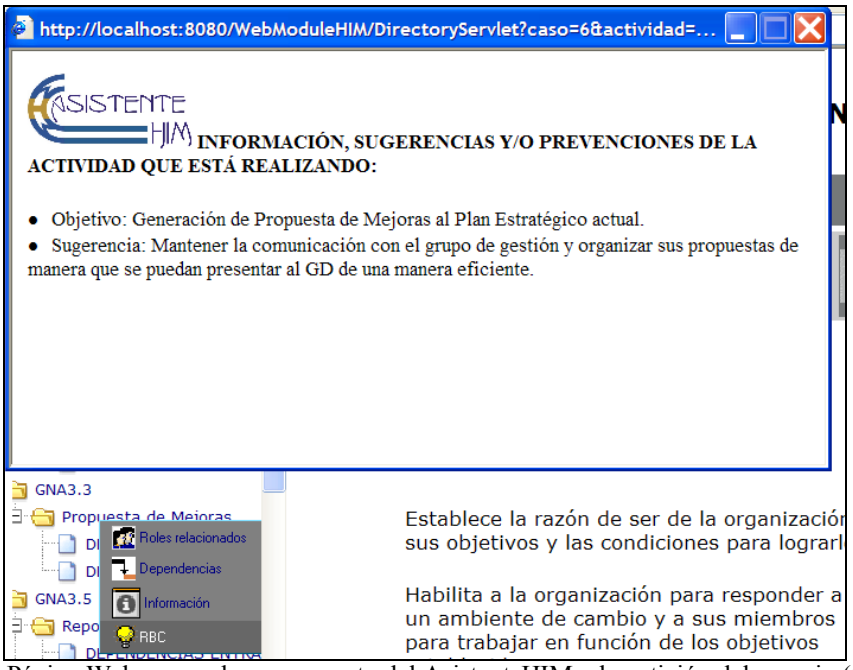

Figura 5.16: Página Web generada en respuesta del AsistenteHIM a la petición del usuario (tareas RBC)

Se ha analizado la información de acuerdo al MoProSoft y a las pruebas realizadas al razonador basado en casos y se observa que la información almacenada si coincide con los datos mostrados por el AsistenteHIM.

Objetivo: cumplido.

5.6 RESULTADOS

En el presente capítulo se describió el proceso de implementación y desarrollo de la herramienta AsistenteHIM, partiendo del análisis de las arquitecturas de los elementos relacionados, para

generar una arquitectura global del sistema. Luego se mostró la manera en que se implementaron cada una de las partes de la herramienta y la forma en que se integró con la HIM. Por último, se presentó el prototipo a manera de pruebas del sistema, con base en los sub-objetivos especificados en el diagrama general de objetivos del capítulo 3 (tema 3.2.2.1.2).

Para evaluar los resultados completos de esta etapa, en la tabla 5.3 se retoman los requerimientos especificados para la herramienta AsistenteHIM en la etapa de análisis del capítulo 3 (tema 3.2.1), donde se indica con una calificación a base de estrellas, si el objetivo fue alcanzado o no. Tres estrellas corresponden a objetivo alcanzado y ninguna estrella a objetivo no alcanzado.

REQUERIMIENTO	RESULTADO
Integrar en un SMA aspectos de la Ingeniería de Software, la Ingeniería de Procesos de Software, la Teoría de Agentes y el enfoque de Razonamiento Basado en Casos, de acuerdo a lo especificado en el capítulo 1.	★★★
De acuerdo al MoProSoft, la herramienta debe ser capaz de llevar un control (al menos parcial) del flujo de trabajo de los usuarios.	★★
Comentario: No se lleva un control del flujo de tareas debido a que en el estado actual de la HIM y el mismo MoProSoft (son relativamente nuevos para la industria), no es recomendable ser tan estricto en cuanto al orden en que se realizan las actividades.	
Proveer información acerca de los roles que tienen asignados los usuarios, sus capacidades y responsabilidades.	★★★
Indicar al usuario las actividades que tiene que realizar, es decir, su lista de tareas pendientes.	★★★
Introducir algunas buenas prácticas de la Ingeniería de Software brindando sugerencias de la manera en que se deben realizar algunas actividades y dar soporte a problemas, dando ideas preventivas o sugerencias (RBC).	★★★
Ser más eficiente al contar con un elemento que le permita adaptarse a las diferentes formas de realizar las actividades de cada empresa u organización (RBC).	★
Comentario: Parte del objetivo se cumplió ya que el sistema razonador basado en casos si se implementó con las etapas que permiten un aprendizaje y adaptación. Sin embargo, sólo se pudo integrar una de las tres etapas del RaBC al agenteRBC por cuestiones de tiempo.	
Dar información de las actividades y los productos.	★★★
Proporcionar ayuda para contactar usuarios.	★★★
Realizar cada uno de los puntos anteriores siempre tomando en cuenta el rol del usuario al que se le está brindando el servicio y las capacidades	★★★

que tiene asociadas de acuerdo al modelo de procesos MoProSoft.	
Tener interacción con el usuario a través de la interfaz de usuario de HIM.	★★
Comentario: Faltó la implementación de la interfaz gráfica para las actividades de revisión y retención del RBC.	
Que el AsistenteHIM pueda acoplarse con HIM para poder llevar a cabo todos los requerimientos mencionados, aprovechando el trabajo de generaciones pasadas.	★★★
Utilizar el marco de trabajo desarrollado en RDF.	★★★
Ser un software libre, abierto y que cumpla con las características de calidad: correcto, confiable, robusto, eficiente, usable, verificable, mantenible, reparable, evolucionable, reusable, portable, entendible e interoperable [Oktaba02].	★★
Comentario: Si es un software libre y abierto pero falta pulir algunas de las características del software de calidad.	

Tabla 5.3: Evaluación de los objetivos del AsistenteHIM

Evaluando los resultados de la tabla anterior, se puede concluir que la herramienta AsistenteHIM cumple de manera relativamente satisfactoria con la mayoría de los requerimientos especificados al inicio de este proyecto, aunque se reconoce que queda mucho trabajo y detalles por realizar.

Con respecto a los objetivos generales de la tesis, con este caso práctico se demuestra que el objetivo de *desarrollar una herramienta de software semi-independiente a la HIM (pero aprovechando sus recursos), que fungiera como guía y supervisor del seguimiento de los procesos del MoProSoft, maximizando la efectividad de los usuarios y facilitando el aprendizaje del mismo*, fue cumplido *casi* en su totalidad. Donde el *casi* se debe a las modificaciones que se hicieron al agenteCoordinador y al agenteRBC.

La victoria se alcanza al realizarse las cosas.
La satisfacción del alma estriba en lo alcanzado.

–W. SHAKESPEARE

Capítulo 6

CONCLUSIONES

En este capítulo se presentan los resultados y contribuciones obtenidos durante el desarrollo del trabajo. Además, se mencionan las conclusiones generales con respecto a los objetivos planteados.

Por último, se comentan las limitaciones que se tuvieron, y algunas líneas de investigación identificadas para continuar con el trabajo relacionado a esta tesis, para cubrir la investigación y desarrollo de los temas que no se contemplaron como parte de la solución.

1.1 RESULTADOS

Los resultados obtenidos con el desarrollo de este trabajo de investigación están fuertemente relacionados a los objetivos que se plantearon al principio de la tesis.

Para comenzar, se puede afirmar que se ha desarrollado un sistema multi-agente que cumple de manera relativamente satisfactoria con la mayoría de los requerimientos especificados al inicio de este proyecto:

- El AsistenteHIM está incorporado a la HIM de manera relativamente sencilla (apéndice 4) y hace uso de sus recursos, aprovechando así el trabajo de otras generaciones. Fueron de utilidad los trabajos para desarrollar y utilizar el *marco de trabajo* en RDF, ya que son parte del funcionamiento del sistema multi-agente que provee la información que se le muestra al usuario.

- El AsistenteHIM puede fungir como guía del seguimiento de los procesos del MoProSoft y contribuir a maximizar la efectividad de los usuarios participantes, gracias a la variedad de información (ayuda) que brinda. Con ello indudablemente se facilita el aprendizaje del modelo de procesos MoProSoft y se contribuye a su exitosa implantación en nuestro país.

- Por el contrario, el sistema no cumple con la especificación de *supervisor* de las actividades del usuario, debido a las limitaciones que se tienen tanto en la HIM como en el mismo MoProSof, y a que el manejo de las interfaces Web aún no ha sido muy estudiado por los desarrolladores de la comunidad JADE, por lo que la integración de agentes en ellas no es sencilla.

Con respecto a las metodologías utilizadas y al proceso de análisis, diseño y construcción del AsistenteHIM, se obtuvieron resultados bastante satisfactorios:

- El capítulo 3 constituye por si mismo una guía sencilla y ejemplificada, para desarrollar sistemas multi-agente a partir de la metodología MESSAGE.

- Se describió la utilización de la mayoría de las herramientas utilizadas, resultado de un análisis que buscaba obtener las más adecuadas y que fueran software libre.

Con respecto al tema del Razonamiento Basado en Casos, el resultado también fue positivo ya que fue posible desarrollar el sistema deseado, cumpliendo con las características esperadas en la etapa de análisis.

- Las expectativas fueron rebasadas ya que se consiguió implementar un sistema razonador que llevara a cabo tres de las cuatro actividades estándar. Además, se generó el código con una interfaz temporal que permitió realizar pruebas a la biblioteca de casos, formada con experiencia real obtenida de las entrevistas que amablemente proporcionaron algunos participantes (capítulo 4).

- Desgraciadamente todo el avance descrito en el párrafo anterior, no fue utilizado en su totalidad debido a problemas con el manejo de las interfaces Web nuevamente, ya que se requerían demasiados cambios en código que por cuestiones de tiempo fueron imposibles de realizar.

Evaluando los resultados anteriores, se considera que el objetivo de integrar la Ingeniería de Software con los sistemas multi-agente y el Razonamiento Basado en Casos de la Inteligencia Artificial, ha sido conseguido ya que la presente tesis constituye la cohesión y aprovechamiento de conocimiento de éstas áreas, en pro del desarrollo de sistemas mucho más capaces y que cumplan con las exigencias del presente y futuro.

1.2 CONTRIBUCIONES

Con base en los resultados obtenidos, podemos considerar que las principales contribuciones del presente proyecto de tesis son las siguientes:

1. Se utilizó y generó una guía completa de la metodología MESSAGE para desarrollar sistemas multi-agentes, que envuelve prácticas bien conocidas de la Ingeniería de Software con aspectos teóricos de la teoría de agentes de la IA.

2. Con el sistema multi-agente generado se espera que sea más fácil el manejo de la HIM y que el usuario aprenda más rápido el MoProSoft, con lo que se contribuye un poco a su implantación dentro de nuestro país.

3. La herramienta desarrollada utiliza los recursos de la HIM para llevar a cabo sus acciones, lo que contribuye a la utilización del trabajo generado por tesistas anteriores y al seguimiento del proyecto HIM.

4. Se implementó y se integró al AsistenteHIM un sistema razonador basado en casos, lo cual es importante ya que representa la integración del área de RBC de la Inteligencia Artificial, a la Ingeniería de Software y en particular el área de Procesos de Software, lo cual no ha sido explotado.

5. Finalmente, se puede decir que la aplicación de técnicas de diversas áreas de la Ingeniería de Software y la Inteligencia Artificial, con el fin de desarrollar una herramienta que cumpliera con los objetivos propuestos; constituye una contribución original al conocimiento de dichas áreas en desarrollo con importantes aplicaciones futuras.

1.3 CONCLUSIONES

Como en cualquier sistema, en el desarrollo del AsistenteHIM se siguió la metodología *MESSAGE* que ayudó a llevar a cabo las importantes fases de análisis y diseño. Con ello se hizo evidente la ventaja que representa seguir un método para realizar las actividades de construcción de un sistema de software.

Ésa es parte importante de los procesos de la Ingeniería de Software y ahora se está utilizando en otras áreas en desarrollo como la IA con resultados satisfactorios.

Siguiendo con la convergencia de áreas, es de destacar la aplicación y utilidad que brinda a los sistemas el enfoque de Razonamiento Basado en Casos, ya que con una implementación generada a partir de un marco de trabajo, se pueden conseguir sistemas para gran variedad de

áreas, como lo son la medicina, aplicaciones de entretenimiento e industriales, aplicaciones comerciales, etc.

Lo anterior son buenas noticias pero también hay que tomar en cuenta que se tienen que realizar varias actividades en la generación de un sistema razonador, lo que requiere de tiempo y disponibilidad de suficiente información para formar una base de casos útil.

En cuanto a los sistemas multi-agente (SMA) se considera que también es otra área en constante desarrollo y que ofrece muchas posibilidades a cualquier sistema. Al integrarse dentro de la HIM se ha dejado una estructura básica de agentes relativamente sencilla, lista para añadir nuevas capacidades incrementando la funcionalidad de la herramienta.

Tomando en cuenta lo anterior y todo el trabajo de la tesis, se considera que se han alcanzado la mayoría de los objetivos del proyecto gracias a la convergencia de las áreas ya mencionadas. Cada parte contribuyó a mejorar un aspecto del sistema, lo que lo hace más innovador y eficiente. Sin embargo, la complejidad también aumentó ocasionando dificultades para profundizar en cada componente y desarrollarlos como se hubiera querido.

Se obtuvo una herramienta que, con algunas limitaciones, cumple con la mayoría de los requisitos identificados (capítulo 3). Además, para llevar a cabo las etapas de desarrollo del sistema, se utilizaron diversas metodologías y herramientas de ayuda, procedimiento documentado que sirve de guía al construir este tipo de sistemas multi-agente y es buena referencia para trabajos futuros.

En conclusión, el resultado obtenido constituye una pequeña contribución a las áreas de Ingeniería de Software, Sistemas Multi-Agentes y Razonamiento Basado en Casos, así como al proyecto de la HIM; dejando abiertas muchas opciones para trabajos futuros.

1.4 LIMITACIONES

A lo largo de la presente tesis se desarrollaron los temas de introducción, investigación y referencia, análisis, diseño, desarrollo, integración y pruebas. En la mayoría de las etapas se tuvieron resultados satisfactorios, no obstante, también se presentaron algunas limitaciones que es necesario mencionar:

1. Desarrollar un sistema que integra las áreas de IS e IA implica el procesamiento de mucha información, lo que es difícil de lograr en un proyecto de tesis de maestría, por lo que cada componente quedó relativamente sencillo.

2. Al depender de la HIM, la herramienta generada cuenta con sus capacidades pero también con sus restricciones (sistema complejo y aún en etapa de desarrollo, manejo complicado de interfaz de usuario, versiones específicas de bibliotecas, etc.), lo que conlleva a tener limitaciones con respecto a ello:

    a. Se invirtió demasiado tiempo en la integración con la HIM debido a su complejidad.

b. No se pudo implementar completamente el comportamiento especificado para el agenteCoordinador, dado que en el estado que se encuentra tanto la HIM como el mismo MoProSoft, no se puede ser tan restrictivo en la información.

c. Se tuvo problemas con la integración del razonador basado en casos (RaBC) debido a la utilización de bibliotecas antiguas en la HIM, lo que ocasionó incompatibilidades. Además de las especificaciones que utilizan *java swing* en JColibri.

3. Al depender del marco de trabajo JColibri para el desarrollo del RaBC, se limitó a la utilización de sus métodos específicos para implementar las tareas del RBC, y por lo tanto el conocimiento de la manera en que se llevan a cabo es muy limitado.

1.5 TRABAJOS A FUTURO

Debido a que este proyecto de tesis involucra varias áreas de la computación y por las limitaciones de tiempo no fue posible explotarlas, se identifican varios trabajos a futuro, que incluso podrían ser temas completos de otros proyectos de tesis.

1. Desarrollar los cambios necesarios en todo el marco de trabajo de JColibri para que también soporte las interfaces Web.

2. Con ayuda de lo anterior, terminar de implementar el RaBC como se generó en el capítulo 4, en el AsistenteHIM.

3. Mejorar todo el razonador para que sea posible la implementación de una herramienta (con o sin agentes) que promueva y facilite el intercambio de lecciones aprendidas en la Industria del Software.

4. Enriquecer cada uno de los agentes del SMA con características de agentes inteligentes, de planeación, etc. Para mejorar el desempeño de toda la herramienta.

5. Integrar conocimiento del área de Interfaces Gráficas Humano-Computadora para mejorar la interfaz del SMA, de tal forma que sea más usable.

Bibliografía

[Aamodt01] Agnar Aamodt and Enric Plaza. *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* Págs. 1-27. AI Communications. IOS Press, Vol. 7:1, pp. 39-59. 1994.

[Bellifemine01] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa. *JADE Tutorial: JADE Tutorial Introduction*. Telecom Italia Laboratory, 2004. http://jade.tilab.com.

[Bellifemine02] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa. *JADE Tutorial: JADE Programming for Beginners*. Telecom Italia Laboratory, 2004. http://jade.tilab.com.

[Bellifemine03] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa. *JADE Programmer's Guide*. Telecom Italia Laboratory, 2004. http://jade.tilab.com.

[Bellifemine04] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa. *JADE Administrator's Guide*. Telecom Italia Laboratory, 2004. http://jade.tilab.com.

[Bellifemine05] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. *JADE – A FIPA Compliant Agent Framework*. Telecom Italia Laboratory, 2004. http://jade.tilab.com.

[Bellifemine06] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. *JADE A White Paper*. Págs. 1-14. Exp. In Search of Innovation. Volume 3. No.3, September 2003.

[Bello-Tomás01] Juan José Bello-Tomás, Pedro A. González-Calero, Belén Díaz-Agudo. *JColibri: An Object-Oriented Framework for Building CBR Systems*. Págs. 15. Dep. Sistemas Informáticos y Programación Universidad Complutense de Madrid, España.

[Bogaerts01] Steven Bogaerts and David Leake. *IUCBRF: Indiana University Case-Based Reasoning Framework. Technical Report 608. A Framework For Rapid and Modular Case-Based Reasoning System Development. Report Version 1.0.* Págs. 1-63. Computer Science Department, Indiana University, Indiana, U.S.A., Mayo de 2005.

[Castells01] Pablo Castells. *La Web Semántica*. Págs. 13. Escuela Politécnica Superior, Universidad Autónoma de Madrid.

[Caire01] Giovanni Caire, Wim Coulier, Francisco Garijo, Jorge Gómez, Juan Pavón, Francisco Leal, et. al. *Agent Oriented Análisis using MESSAGE/UML*. Págs. 1-17. EURESCOM, Project P907. http://www.eurescom.de/Public/Projects/p900-series/P907/P907.htm.

[Caire02] Giovanni Caire, Francisco Leal and Joao Rodríguez. *MESSAGE: Methodology for Engineering Systems of Software Agents. Recommendations on supporting tools.* Págs. 1-27. EURESCOM, Project P907-GI Septiembre, 2001. http://www.eurescom.de/Public/Projects/p900-series/P907/P907.html

[Cockburn01] D. Cockburn and J. Nick R. ARCHON: A distributed artificial intelligence system for industrial applications. In G. M. P. O'Hare and J. N. R, editors, *Foundations of Distributed Artificial Intelligence*, pages 319–344. John Wiley & Sons, 1996.

[Cugola01] Gianpaolo Cugola and Carlo Ghezzi. *Software processes: a retrospective and a path to the future*. Págs. 26. Politecnico di Milano, Departamento di Elettronica e Informaciones, Italia, 1994.

[Delany01] Sarah J. Delany and Pádraig Cunningham. *The Application of Case-Based Reasoning to Early Software Project Cost Estimation and Risk Assessment*. Págs. 1-20. Department of Computer Science, Trinity Collage Diblin.

[DGP01] Digital Planet: *The Global Information Economy*. WITSA. Noviembre de 2000.

[Díaz-Agudo01] Belén Díaz-Agudo, Pedro A. González-Calero, Pedro P. Gómez-Martín, Marco A. Gómez-Martín. *On Developing a Distributed CBR Framework through Semantic Web Services*. Págs. 10. Dep. Sistemas Informáticos y Programación Universidad Complutense de Madrid, España.

[EURESCOM01] EURESCOM participants in Project P907-GI. *MESSAGE: Methodology for Engineering Systems of Software Agents. Deliverable 1, Initial Methodology.* Págs. 1-75. 2001. http://www.eurescom.de/Public/Projects/p900-series/P907/P907.htm.

[EURESCOM02] EURESCOM participants in Project P907-GI. *MESSAGE: Methodology for Engineering Systems of Software Agents. Final Guidelines for the Identification of Relevant Problem areas where Agent Technology is Appropiate.* Págs. 1-34. September, 2001. http://www.eurescom.de/Public/Projects/p900-series/P907/P907.htm.

[Evans01] Richard Evans, Paul Kearney, Jamie Stark, Giovanni Caire, Francisco J. Garijo, et. al. *MESSAGE: Methodology for Engineering Systems of Software Agents. Methodology for Agent-Oriented Software Engineering.* Págs. 1-75. EURESCOM, Project P907-GI September, 2001. http://www.eurescom.de/Public/Projects/p900-series/P907/P907.htm.

[Ferber01] Jacques Ferber. *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*. Addison-Wesley. Great Britain, 1999

[FIPA01] Foundation for Intelligent Physical Agents. 1996. http://www.fipa.org

[FIPA02] Response to the OMG Analysis and Design Task Force UML 2.0 Request for Information: Extending UML for the specification of Agent Interaction Protocols, OMG 16/12/1999, Bauer, B. et al. ftp://ftp.omg.org/pub/docs/ad/99-12-03.pdf

[FIPA03] http://www.fipa.org/repository/ips.html

[Franklin01] Stan Franklin and Art Graesser. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. Págs. 1-11. Institute for Intelligent Systems, University of Memphis. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[Fuggetta01] Alfonso Fuggetta. *Software Process: A Roadmap*. Págs. 8. Politecnico di Milano, Departamento di Elettronica e Informaciones, Italia.

[Gamma01] E. Gamma, R. Helm, R. Jonson y J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison Wesley.

[Gandon01] Fabien Gandon. *Linking a Servlet to a JADE Agent.* Junio 2003. http://jade.tilab.com/community-3rdpartysw.htm.

[GOB01] http://www.gobernacion.gob.mx/dof/2005/agosto/dof_15-08-005.pdf

[Gómez01] Jorge J. Gómez Sanz. *Metodologías para el desarrollo de sistemas multi-agente*. Págs. 1-12. Departamento de Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense. Madrid, España.

[Hernández01] Ernesto Hernández Uribe. *Tesis de maestría: Uso de la tecnología RDF para presentar y manejar los procesos MoProSoft y su aplicación en HIM*. Págs. 121. Universidad Nacional Autónoma de México, 2005.

[Humphrey01] Watts Humphrey. *Introduction to the Personal Software Process (SEI Series in Software Engineering)*. Addison-Wesley, 1996.

[Humphrey02] Watts Humphrey. *Introduction to the Team Software Process*. Addison-Wesley, 2000.

[ISO01] ISO/IEC 9126:2001.

[Jacobson01] Ivar Jacobson, Grady Booch y James Rumbaugh. *El Proceso Unificado de Desarrollo de Software*. Pearson Education, 2000.

[Jena01] http://www.hpl.hp.com/semweb/jena2.htm

[Jennings01] Nicholás R. Jennings, Katia Sycara and Michael Wooldridge. *A roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems*. Págs. 1, 275-306. Kluwer Academia Publishers. Boston, 1998.

[JColibri01] http://gaia.fdi.ucm.es/grupo/projects/jcolibri/index.html

[Kolodner01] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc. 1993.

[Lassila01] Ora Lasssila. *Introduction to RDF Metadata*. W3C NOTE 1997, 11-13. http://www.w3.org/TR/NOTE-rdf-simple-intro-971113.html

[Larousse01] *Larousse, Diccionario de la Lengua Española*. Págs. 434. Larousse Editorial. 2001.

[Laureano01] Ana Lilia Laureano-Cruces, Gilberto Espinosa-Paredes. *Behavioral design to model a reactive decisión o fan expert in geothermal wells*. Págs. 1-4. Internacional Journal of Approximate Reasoning, 2004. www.sciencedirect.com.

[Laureano02] Ana Lilia Laureano Cruces. *Tésis Doctoral: Interacción Dinámica en Sistemas de Enseñanza Inteligentes*. Universidad Nacional Autónoma de México. México D.F., 2000.

[Leake01] David Leake and Steven Bogaerts. *IUCBRF: A Framework For Rapid And Moduar Case-Based Reasoning System Development, Ver. 1.0.* Págs. 63. Computer Science Department, Indiana University, 11 de Mayo de 2005.

[Lemaître01] Christian Lemaître L. *Curso de Sistemas Multi-Agente*. Maestría en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México. México D.F., 2005.

[Manola01] Frank Manola and Eric Miller, Editors, W3C Recommendation, 10 de Febrero de 2004. http://www.w3.org/TR/rdf-primer/.

[Mas01] Ana Mas. *Agentes Software y Sistemas Multiagente. Conceptos, Arquitecturas y Aplicaciones*. Págs. 296. Pearson Education, S.A. Madrid, 2005.

[Maes01] Pattie Maes, Henry A. Kautz, Bart Selman, Michael Coen and Doug Riecken. *Agents that reduce work and information overload*. Págs. 1-10. Communications of the ACM, vol. 37, No. 7, Julio de 1994.

[Mercado01] Araceli E. Mercado Fernández. *Tesis de maestría: La sincronización de los elementos de una base de conocimiento para MoProSoft y su aplicación en una herramienta integral*. Págs. 73. Universidad Nacional Autónoma de México, 2005.

[Miller01] Eric Miller. *An Introduction to the Resource Description Framework*. Págs. 11. D-Lib Magazine. Dublín, Ohio, Mayo de 1998.

[Müller01] J. P. Müller, M. Pischel, and M. Thiel. A pragmatic approach to modeling autonomous interacting systems - preliminary report -. In M. Wooldridge and N. Jennings, editors, *Agent theories, architectures, and languages*, Págs. 226–240. ECAI, Springer-Verlag: Heidelberg, Germany, 1994.

[Muscettola01] Incola Muscettola, Gregory A. Dorais, Check Fry R. Levinson and Christian Plaunt. *IDEA:Planning at the Coreo f Autonomous Reactive Agents*. Págs. 1. NASA Ames Research Center. Moffett Field, California, U.E.A.

[MySql01] http://www.mysql.com

[Negroponte01] Nicholas Negroponte. *Ser digital*. Hodder and Stoughton, 1995.

[Noriega01] Pablo Noriega. *Tesis Doctoral: Agent Mediated Auctions: The FishMarket Metaphor*. Universidad Autónoma de Barcelona, Facultad de Ciencias, Diciembre de 1997.

[Nwana01] H. S. Nwana. *Software Agents:An overview. Knowledge Engineering Review*. Págs. 1-40. Septiembre, 1996. http://www.sce.carleton.ca/netmanage/docs/AgentsOverview/ao.html.

[Oktaba01] Hanna Oktaba, Claudia Alquicira Esquivel, Angélica Su Ramos, et al. *Modelo de Procesos para la Industria de Software, MoProSoft. Versión 1.1*. Págs. 1-121. México D.F., 2003.

[Oktaba02] Hanna Oktaba. *Presentación para Curso de Ingeniería de Software*. Maestría en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México. México D.F., 2004.

[Plaza01] Enric Plaza, Joseph Lluís Arcos and Francisco Martín. *Cooperative Case-Based Reasoning*. Págs. 1-22. Lectures Notes in Artificial Intelligence, Springer Verlag, 1997.

[RAE01] http://www.rae.es

[Recio01] Juan A. Recio, Belén Díaz-Agudo, Marco A. Gómez-Martín, Nirmalie Wiratunga. *Extending JColibri for Textual CBR*. Págs. 15. Dep. Sistemas Informáticos y Programación Universidad Complutense de Madrid, España.

[RETSINA01] http://www.cs.cmu.edu/~softagents/

[Rodríquez01] Oscar M. Rodríguez. *Supporting Knowledge Management by Intelligent Agents in Software Maintenance Groups*. CICESE-Computer Science Department, 2004. orodrigu@cicese.mc.

[Rosenchein01] J. S. Rosenchein and M. Genesereth. *Deals among rational agents.* Págs. 91-99. IJCAI-85, 1985.

[Russell01] Stuart Russell and Peter Norving. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[Schema01] The W3C RDF Schema Working Group. http://www.w3.org/TR/WE-RDF-Schema/

[SE01] Secretaría de Economía. *Programa para el desarrollo de la industria de software*. Secretaría de Economía, 2001. http://www.software.net.mx/desarrolladores/prosoft/Programa/prosoft.htm.

[SEI01] http://www.sei.cmu.edu/cmm/

[SEMANTIC01] http://www.w3schools.com/semweb/default.asp

[SEMANTIC02] http://iswc.semanticweb.org/

[SEMANTIC03] http://www.elsevier.com/locate/websem/

[SEMANTIC04] http://www.etaij.org/seweb/

[SG01] Revista Software Guru. Oct-Dic 2004

[Schank01] R. Schank. *Dynamic Memory: A theory of reminding and learning in computers and people*. Cambridge, UK: Cambridge University Press, 1982.

[Shepperd01] Martin Shepperd. *Case-based Reasoning and Software Engineering*. Págs. 1-18. Empirical Software Engineering Research Group, Bournemouth University, UK. mshepper@bmth.ac.uk.

[Stelting01] Stephen Stelting, Olav Maassen. *Patrones de diseño aplicados a JAVA*. Págs. 210-213. Pearson Education, S.A. Madrid, 2003.

[SWEBOK01] The Institute of Electrical and Electronics Engineers, Inc. *Guide to the Software Engineering Body of Knowledgr: SWEBOK*. Págs. 202. U.S.A, 2004.

[Sycara01] Katia P. Sycara. *Multiagent Systems*. Págs. 1-14. AI Magazine Volume 19, No. 2 Intelligent Agents, Summer 1998.

[Valdivieso01] Karin Valdivieso Castillo. *Tesis de maestría:Utilización de patrones y la arquitectura J2EE para el diseño de la interfaz de usuario de la Herramienta Integral MoProSoft*. Págs. 100. Universidad Nacional Autónoma de México, 2005.

[Vázquez01] Marcos Oscar Vázquez Morales. *Tesis de maestría: Aplicación de patrones basados en J2EE para el diseño e implementación de la capa de control de la Herramienta Integral para MoProSoft (HIM)*. Págs. 79. Universidad Nacional Autónoma de México, 2005.

[Vicente01] Julián Vicente J., Vicente J. Botti. *Estudio de métodos de desarrollo de sistemas multiagente*. Págs. 1-15. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia.

[Wooldridge01] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd.

[Wooldridge02] [343] M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, October 1992. (Also available as Technical Report MMU–DOC–94–01, Department of Computing, Manchester Metropolitan University, Chester St., Manchester, UK). [Wooldridge02] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[Zunino01] Alejadro Zunino, Luis Berdún y Analía Amandi. *JavaLog: un Lenguaje para la Programación de Agentes*. Págs. 1-6. ISISTAN Research Institute, Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Pcia. de Buenos Aires, Argentina. amandi@exa.unicen.edu.ar.

[Zurita01] Hafiz Zurita Rendón. *Tesis de maestría: Arquitectura de la Herramienta Integral para MoProSoft*. Págs. 129. Universidad Nacional Autónoma de México, 2005.

Apéndice 1

NOTACIÓN DE MESSAGE

Esta sección describe la notación específica de la metodología MESSAGE, que toma como base el Lenguaje de Modelado Unificado (UML).

1. RESUMEN DE SÍMBOLOS

1.1 Meta-conceptos

Cada símbolo representa los conceptos fundamentales del MESSAGE.

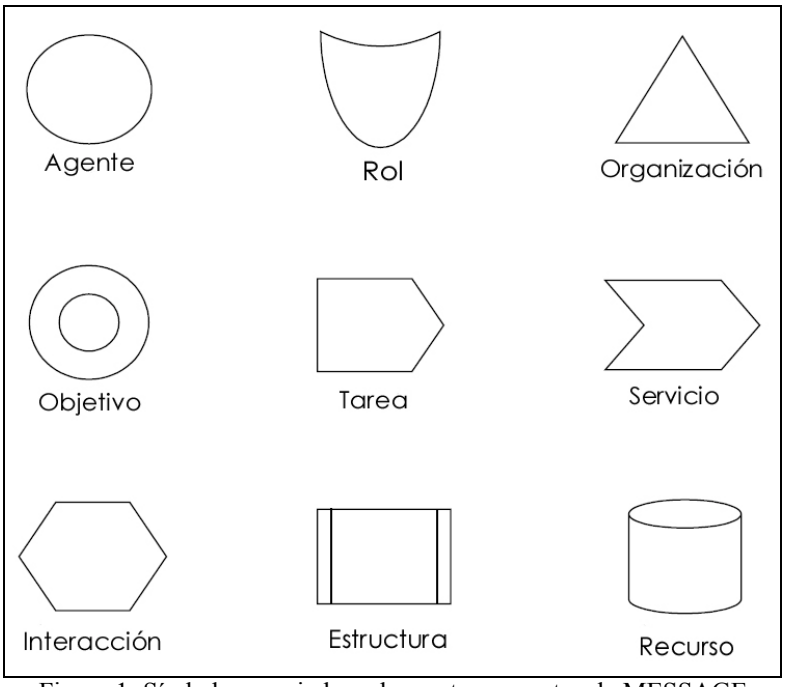

Figura 1: Símbolos asociados a los meta-conceptos de MESSAGE

1.2 Meta-relaciones

Cada símbolo representa las relaciones que se pueden modelar entre los conceptos de la figura anterior.

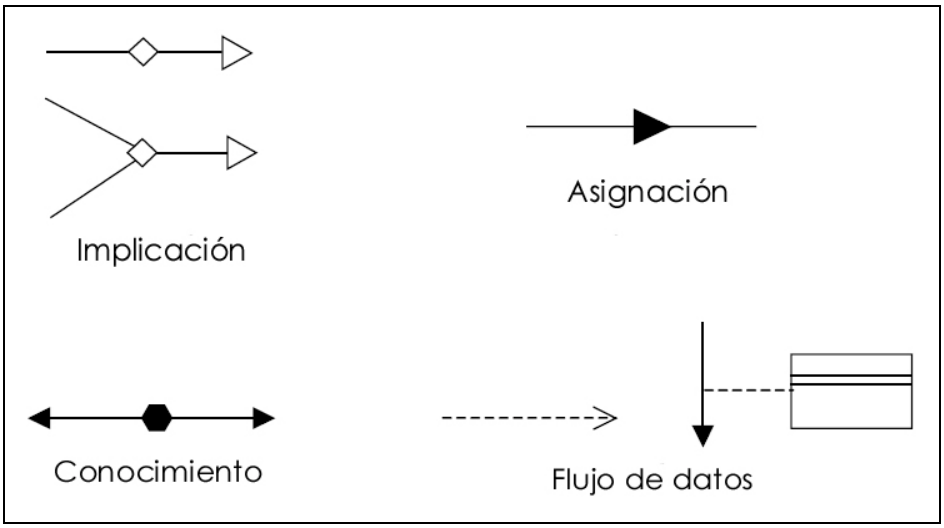

Figura 2: Símbolos asociados a las meta-relaciones de MESSAGE

La **implicación** liga uno o más elementos que tienen un atributo de tipo estado, a un solo elemento que tiene un atributo de tipo estado. La relación es direccional y su semántica es que el estado, que es un atributo del elemento apuntado por la relación, es implicado por el AND lógico de los estados, que son atributos de los elementos donde la relación comienza.

Para una mejor comprensión de esta relación se tiene el siguiente ejemplo:

Objetivo1 alcanzado => Tarea1 factible
Tarea1 completada exitosamente => Tarea2 factible
Tarea2 completada exitosamente => Objetivo2 alcanzado

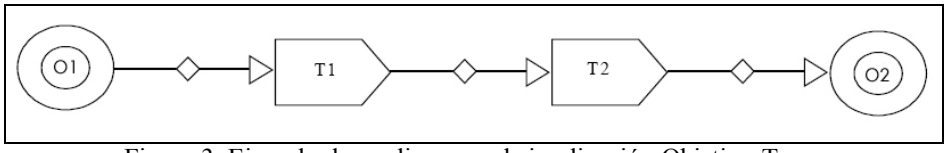

Figura 3: Ejemplo de un diagrama de implicación Objetivo-Tarea

La relación de **asignación** liga un elemento de tipo *autonomousEntity* (capítulo 3) a un elemento que tiene un atributo de tipo *autonomousEntity*. La semántica es tal que la asignación de una *autonomousEntity* a otra sigue la dirección de la flecha.

Existen diferentes tipos de asignación dependiendo del tipo específico de *autonomousEntity*. Las diferentes relaciones se indican con el mismo símbolo de flecha y los símbolos de los *meta-conceptos* que se muestran enseguida:

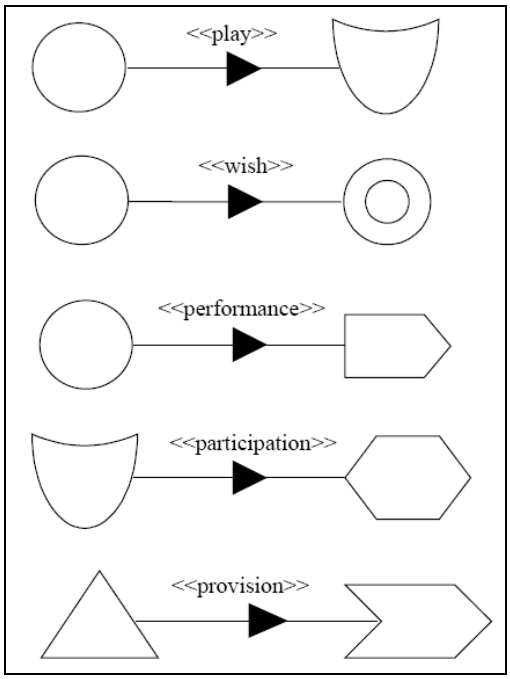

Figura 4: Ejemplos de las diferentes relaciones de asignación de la notación de MESSAGE

La relación de **conocimiento** (*acquaintance*), liga dos *autonomousEntity* y su semántica es tal que al menos una interacción es requerida entre estas dos entidades.

La relación de **flujo de datos** liga una entidad productora de datos (una tarea) a una entidad de información. Indica la misma relación de flujo de objetos (*ObjectFlow)* de UML y por ello es utilizado el mismo símbolo.

2. EJEMPLOS DE DIAGRAMAS

A continuación se muestran un conjunto de diagramas adaptados de [Evans01], que ejemplifican la notación descrita y facilitan la comprensión de los diagramas generados en el capítulo 3.

2.1 Diagrama de organización

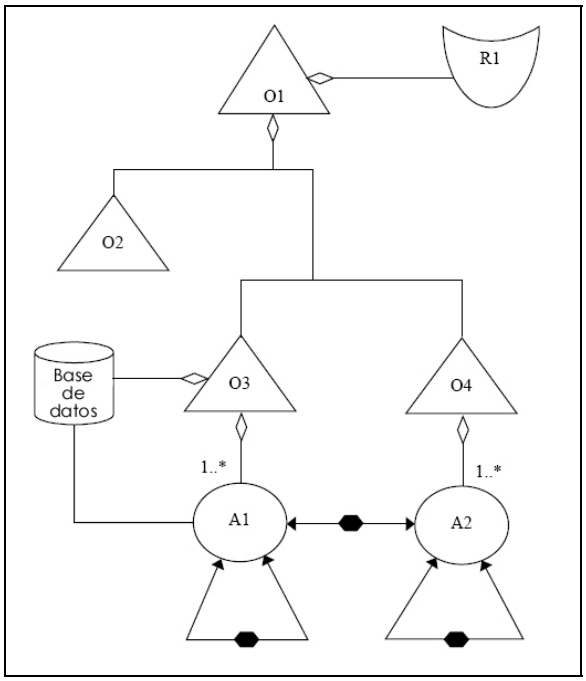

Figura 5: Ejemplo de diagrama de organización

2.2 Diagrama de Objetivo-Tarea

El diagrama objetivo-tarea se muestra en la figura 3.

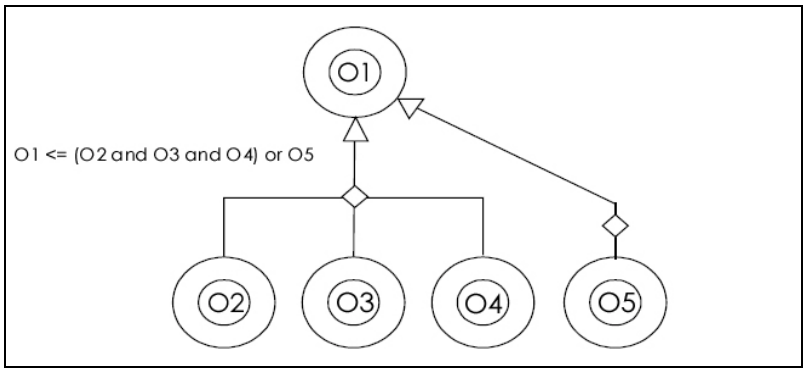

Figura 6: Ejemplo de un diagrama de descomposición de objetivos

2.3 Diagrama de flujo de trabajo

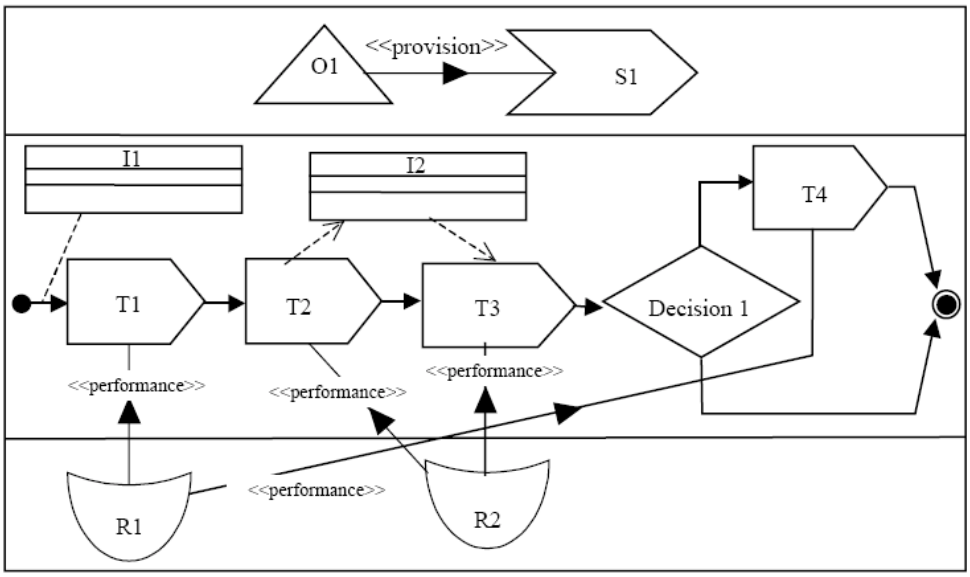

Figura 7: Ejemplo de diagrama de flujo de trabajo

2.4 Diagrama de agente

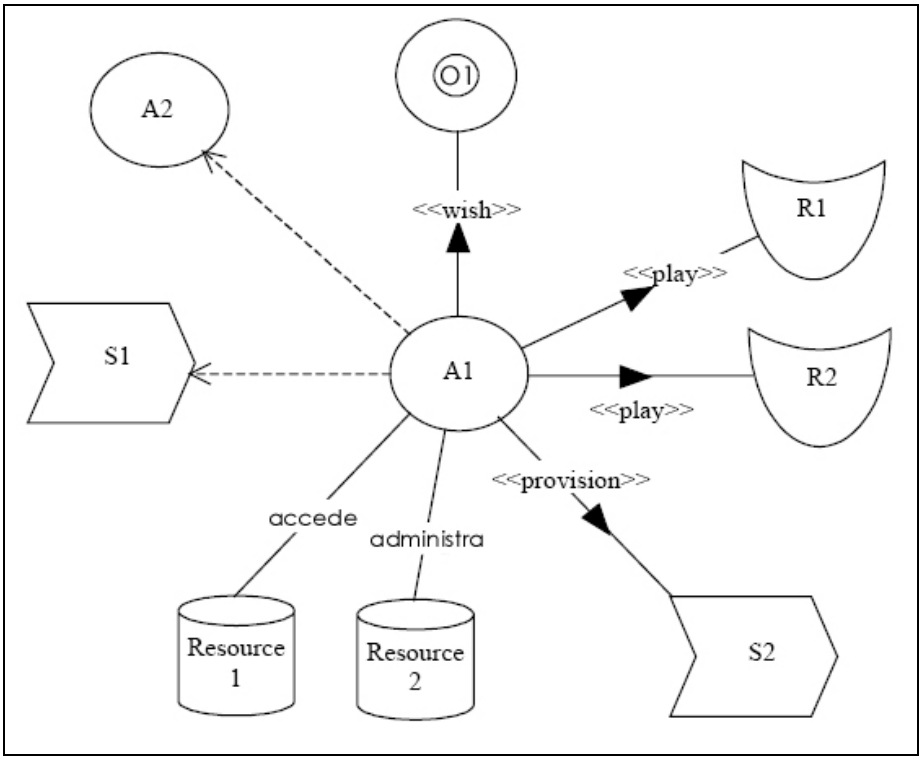

Figura 8: Ejemplo de diagrama de agente

2.5 Diagrama de interacción

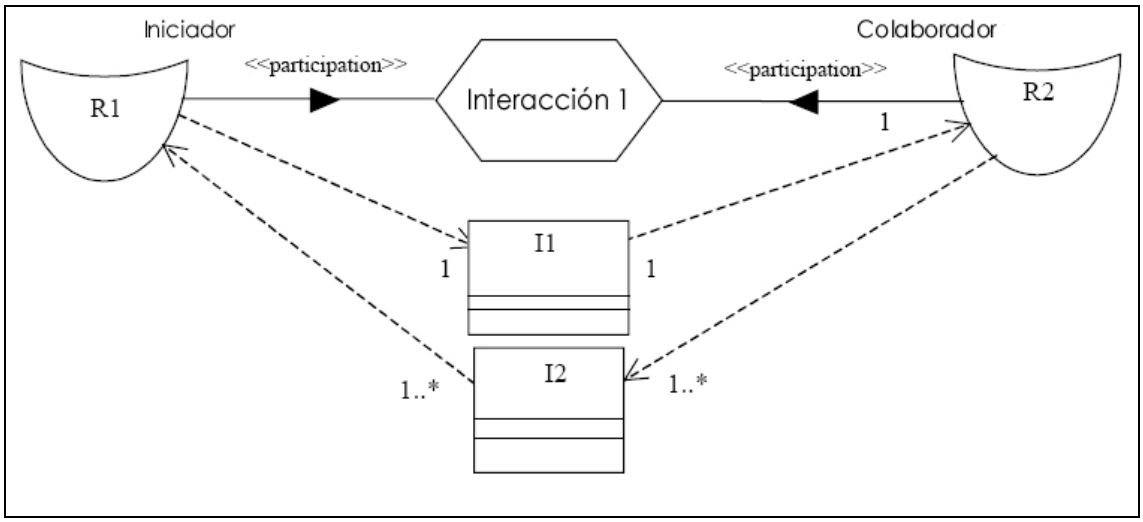

Figura 9: Ejemplo de diagrama de interacción

Apéndice 2

ESQUEMAS DEL ANÁLISIS y DISEÑO DEL AsistenteHIM

Esta sección contiene los esquemas complementarios de las distintas etapas de la fase de análisis del AsistenteHIM desarrollada en el capítulo 3. Así como los mensajes ACL generados en la etapa de diseño.

1. MODELO DE AGENTE, FASE 1: ESQUEMAS DE AGENTES

1.1 Esquema del *agenteSupervisor*

Agente: Supervisor
Identidad: El agente es identificado como el responsable de supervisar las actividades de los usuarios, controlando el flujo de trabajo, coordinándolos y contactándolos.
Servicios: Recordar tareas pendientes y Coordinar y contactar usuarios.
Objetivo: pasivo (significa que su objetivo es impuesto por otro agente: *agenteCoordinador*)
Interacciones con el entorno: *Entrada receptora:* Mensajes de petición de servicio por parte del *agenteCoordinador* y la información necesaria para brindar los servicios (rol, proceso y actividad que se realiza). Utilizando los servicios del *agenteBuscador* puede conocer los roles involucrados en la actividad (RDF)[1] y el estado de la misma (hecha o no). *Relaciones:* Se necesita tener contacto con el *agenteCoordinador* que es el que lo llamará y le proporcionará los datos básicos para el servicio, además a ese mismo se le manda el resultado. También se requiere contacto con el *agenteBuscador* que sirve para realizar todas las consultas necesarias a la base del conocimiento almacenada en RDF. *Propiedad de recurso y acceso:* 1. Acceso de lectura a información de estado de actividades en el marco de trabajo (RDF), por medio del *agenteBuscador*. 2. Acceso de lectura a información de roles participantes en un actividad (RDF), por medio del *agenteBuscador*. *Acciones:* 1. (comunicar) Obtener solicitud de servicio por parte del *coordinador*. 2. (comunicar) Obtener datos necesarios para llevar a cabo el servicio solicitado (usuario, rol y actividad). 3. (interna) Calcular la acción a realizar y la información necesaria. 4. (interna) Formular mensaje de petición de servicio. 5. (comunicar) Solicitar servicio de información al *buscador*. 6. (comunicar) Recibir información solicitada. 7. (interna) Evaluar la información obtenida y decidir la siguiente acción. 8. (interna) Estimar resultados de acuerdo al servicio, por ejemplo, si la actividad que se está realizando tiene dependencia con otra y quien es el responsable, si hay tareas no hechas, o simplemente si hay otro rol que pueda ayudar en la actividad. 9. (comunicar) Mandar respuesta al *agenteCoordinador*.
Estado mental y comportamiento: *Propósito:* Los objetivos de este agente son (1) recordar al usuario sus tareas pendientes por medio de la consulta de actividades no hechas, y (2) coordinar y contactar los usuarios de acuerdo a la actividad que se esté realizando, por ejemplo consultar las dependencias entre actividades e indicarle al

[1] Involucra una consulta a los archivos en RDF, o sea, un servicio del *agenteBuscador*.

usuario si hay algún problema (que se este realizando una actividad dependiente de otra que aún no está terminada), o consultar la actividad y los roles participantes para que se coordinen (ver diagrama de agente *agenteSupervisor* de la fase 0). *Comportamiento:* El agente espera a recibir un mensaje de solicitud de servicio para comenzar a trabajar y una vez recibido, se concentra en realizar las tareas indicadas en los diagramas de flujo de trabajo para este agente y mantener la relación con los demás agentes. Puede realizar las acciones internas, comunicativas o externas (de las listadas anteriormente) que sean necesarias. *Conocimiento y creencias:* 1. Tiene conocimiento de las tareas (y su orden) necesarias para brindar sus servicios. 2. Sabe cómo encontrar la información que necesita (mensajes al *agenteBuscador*). 3. Sabe cómo interpretar los mensajes o entidades de información que recibe de los demás agentes. 4. Sabe cómo transferir los resultados al *agenteCoordinador*.
Información adicional: El agente indica las actividades pendientes de acuerdo al proceso, orden en que están sugeridas en MoProSoft y el estado hecha=falso.

Tabla 1: Esquema del Agente *agenteSupervisor*

1.2 Esquema del *agenteRBC*

Agente: RBC
Identidad: El agente es identificado como el responsable de sugerir cómo llevar a cabo las tareas que realiza el usuario, sugerirle soluciones a posibles problemas mediante la provisión de ideas preventivas (igual que el objetivo de sugerir cómo llevar a cabo las tareas pero como la intención en prevenir, los resultados serán distintos), mediante la utilización del enfoque de RBC.
Servicios: Sugerir cómo llevar a cabo las tareas, sugerir soluciones y dar ideas preventivas.
Objetivo: pasivo
Interacciones con el entorno: *Entrada receptora:* Mensajes de petición de servicio por parte del *agenteCoordinador* y la información necesaria para formar los casos (rol del usuario, actividad que se realiza y tamaño y nivel de la empresa). Utilizando los servicios del *agenteBuscador* tiene acceso a información adicional. *Relaciones:* Se necesita tener contacto con el *agenteCoordinador* que es el que lo llamará y le proporcionará los datos básicos para el servicio, además a ese mismo se le manda el resultado. También se requiere contacto con el *agenteBuscador* que le servirá para realizar todas las consultas necesarias a la base del conocimiento almacenada en RDF. *Propiedad de recurso y acceso:* 1. Acceso de lectura/escritura a la biblioteca de casos del razonador. 2. Acceso de lectura a información en el marco de trabajo (RDF), por medio del *agenteBuscador*. *Acciones:* 1. (comunicar) Obtener solicitud de servicio por parte del *agenteCoordinador* o el usuario. 2. (comunicar) Obtener datos necesarios para llevar a cabo el servicio solicitado (formación de

caso). 3. (interna) Calcular la acción a realizar y la información necesaria. 4. (interna) Formular mensaje de petición de servicio. 5. (comunicar) Solicitar servicio de información al *buscador* (si es necesario). 6. (comunicar) Recibir información solicitada. 7. (interna) Evaluar la información obtenida y decidir la siguiente acción. 8. (externa) Consultar la biblioteca de casos en busca de resultados. 9. (interna) Procesos propios del razonador basado en casos (capítulo 4). 10. (comunicar) Mandar respuesta al *coordinador* o al usuario.
Estado mental y comportamiento: *Propósito:* Los objetivos de este agente son sugerir cómo llevar a cabo determinadas tareas, sugerir soluciones a posibles problemas, y dar ideas preventivas. Todo por medio de la utilización de las entidades de información proporcionadas por el *agenteCoordinador* y el usuario, utilizando el enfoque de razonamiento basado en casos. Un propósito adicional es realizar las actividades comunes de un razonador basado en casos (capítulo 4). *Comportamiento:* El agente espera a recibir un mensaje que indique la solicitud de un servicio para comenzar a trabajar. Una vez recibido se concentra en realizar las tareas indicadas en los diagramas de flujo de trabajo para este agente y mantener la relación con los demás agentes. Puede realizar las acciones internas, comunicativas o externas (de las listadas anteriormente) que sean necesarias. *Conocimiento y creencias:* 1. Tiene conocimiento de las tareas (y su orden) necesarias para brindar sus servicios. 2. Sabe cómo encontrar la información que necesita (mensajes al *buscador*). 3. Sabe cómo interpretar los mensajes o entidades de información que recibe de los agentes. 4. Sabe cómo transferir los resultados al *agenteCoordinador*. 5. Sabe cómo llevar a cabo las tareas involucradas con el enfoque RBC.

Tabla 2: Esquema del Agente *agenteRBC*

1.3 Esquema del *agenteBuscador*

Agente: Buscador
Identidad: El agente es identificado como el responsable de proporcionar de manera eficiente el servicio de consulta de información requerida por los demás agentes, haciendo uso del marco de trabajo en RDF. Además de brindar el servicio informativo a los usuarios cuando lo requiera el *agenteCoordinador*.
Servicios: Brindar la información solicitada.
Objetivo: pasivo
Interacciones con el entorno: *Entrada receptora:* Mensajes de petición de servicio por parte de cualquier agente y la información necesaria para brindar los servicios (rol, proceso o actividad que se realiza). Tiene acceso a toda la información que puede brindar el marco de trabajo desarrollado en RDF. *Relaciones:* Se necesita tener contacto con el *agenteCoordinador* que es el que lo llamará y le proporcionará los datos básicos para el servicio de proporcionarle información general al usuario (de su rol, producto o

actividad), además a ese mismo agente se le manda el resultado. También se requiere contacto con los agentes *agenteSupervisor y agenteRBC,* ya que les brinda el servicio de información solicitada. *Propiedad de recurso y acceso:* 1. Acceso de lectura a toda la información del marco de trabajo en RDF, por medio de consultas RDQL. *Acciones:* 1. (comunicar) Obtener solicitud de servicio por parte de cualquier agente. 2. (comunicar) Obtener datos necesarios para llevar a cabo el servicio solicitado. 3. (interna) Calcular la acción a realizar. 4. (externa) Formular consulta en RDQL. 5. (comunicar) Recibir información solicitada. 6. (interna) Evaluar la información obtenida y decidir la siguiente acción. 7. (comunicar) Mandar respuesta al agente que hizo la petición de servicio.
Estado mental y comportamiento: *Propósito:* Los objetivos de este agente son brindar la información solicitada a los agentes *agenteSupervisor, agenteCoordinador y agenteRBC;* y proveer de información útil al usuario. *Comportamiento:* El agente espera a recibir un mensaje de solicitud de servicio para comenzar a trabajar y una vez recibido, se concentra en realizar las tareas indicadas en los diagramas de flujo de trabajo para este agente y mantener la relación con los demás agentes. Puede realizar las acciones internas, comunicativas o externas (de las listadas anteriormente) que sean necesarias. *Conocimiento y creencias:* 1. Tiene conocimiento de las tareas (y su orden) necesarias para brindar sus servicios. 2. Sabe cómo encontrar la información que necesita (consultas RDQL o acceso a los objetos DAO, capítulo 5). 3. Sabe cómo interpretar los mensajes o entidades de información que recibe de los agentes. 4. Sabe cómo transferir los resultados a los agentes 5. Conoce algunas restricciones de los roles de usuarios, que condicionan que les de información.
Información adicional: Este agente es el único que tiene acceso a la base de conocimiento de la HIM, por lo que es parte fundamental del sistema. El resultado de sus consultas depende de la información (parámetros) proporcionada por los demás agentes.

Tabla 3: Esquema del Agente *agenteBuscador*

2. MODELO DE INTERACCIÓN, FASE 1: ESQUEMAS DE INTERACCIÓN

Interacción 2.2	Petición de información
Motivador	Coordinar y contactar usuarios
Iniciador	*agenteCoordinador*
Colaborador(es)	*agenteSupervisor*
Entradas	Clave que indica la información que se requiere, rol del usuario y actividad.

Salidas	Información de los usuarios participantes en la actividad (rol, nombre, teléfono y correo electrónico). Y lista de otras actividades (no hechas) con las que se tiene dependencia junto con sus roles relacionados
Procesamiento	El *agenteSupervisor* revisa si hay roles relacionados con la actividad que se está llevando a cabo, o con actividades que halla dependencia, para darle más información al usuario
Restricciones	Cuando no se encuentre la información requerida, el *agenteSupervisor* tendrá que notificar al coordinador de la situación
IP	FIPA-query Protocol

Tabla 4: Esquema de interacción con motivador *Coordinar y contactar usuarios*

Interacción 5	Petición de información
Motivador	Coordinar y contactar usuarios
Iniciador	*agenteSupervisor*
Colaborador(es)	Agente Buscador
Entradas	Clave que indica la información que se requiere y actividad
Salidas	Lista de actividades de las que depende la actividad que realiza el usuario, con su estado y sus roles relacionados
Procesamiento	Cuando el supervisor recibe la salida del *agenteBuscador*, revisa el estado de las actividades y elimina las que ya fueron hechas. El *agenteBuscador* hace la consulta en los archivos en RDF
Restricciones	Cuando no se encuentre la información requerida, el *agenteBuscador* tendrá que notificar al supervisor de la situación
IP	FIPA-query Protocol

Tabla 5: Esquema de interacción con motivador *Coordinar y contactar usuarios (2)*

Interacción 2.3	Petición de información
Motivador	Sugerir cómo llevar a cabo tareas, sugerir soluciones y dar ideas preventivas
Iniciador	*agenteCoordinador*
Colaborador(es)	*AgenteRBC*
Entradas	Información disponible: usuario, rol, actividad, producto y proceso.
Salidas	Oración que sugiera como llevar a cabo determinada tarea, una solución a un problema detectado o una idea preventiva. La oración es resultado del proceso de RBC que lleva a cabo el agente y el cual se explica ampliamente en el capítulo 4.
Procesamiento	El *agenteCoordinador* manda información al *agenteRBC* para ver si hay casos disponibles que se le puedan dar al usuario
Restricciones	Cuando no se encuentre la información requerida (casos que coincidan), el *agenteRBC* tendrá que notificar al coordinador de la situación
IP	FIPA-query Protocol

Tabla 6: Esquema de interacción con motivador *Sugerir cómo llevar a cabo tareas, sugerir soluciones y dar ideas preventivas*

Existen más interacciones necesarias para el *agenteRBC*, sin embargo, se optó por excluirlas ya que el desarrollo de este agente en específico, se abarca ampliamente en el capítulo 4 por ser el tema de Razonamiento Basado en Casos. Algunas observaciones son las siguientes:

- Si el usuario solicita el servicio del *agenteRBC* directamente desde la interfaz, la interacción es la misma que la anterior pero el iniciador es el usuario.

- También hace falta una interacción con el *agenteBuscador*, en el caso de que se requiera información de los archivos en RDF, pero tiene el mismo formato que las otras interacciones anteriores con el *agenteBuscador*, lo que cambia es la entrada, que serán los datos requeridos por el *agenteRBC* que dependen del razonador basado en casos.

- Hay otra interacción con la biblioteca de casos en la que se pretende recuperar un caso adecuado dependiendo de la situación y los parámetros, pero es igual a la interacción del agente buscador con la base de conocimiento, solo cambia el procesamiento interno de cada una de las fuentes de información.

Interacción 6	Petición de información
Motivador	Brindar la información solicitada
Iniciador	Agentes (todos)
Colaborador(es)	Agente Buscador
Entradas	Clave que indica la información requerida y los parámetros necesarios (cada interacción de solicitud de información al buscador, es una sub-interacción de este tipo e indica los parámetros requeridos en cada caso)
Salidas	Información requerida
Procesamiento	El *agenteBuscador* analiza la instrucción, decide la consulta que hay que hacer a los archivos RDF (base de conocimiento de configuración y repositorio), y realiza dicha consulta
Restricciones	Si las consultas no dan ningún resultado, el *agenteBuscador* tiene que notificar al agente iniciador de tal resultado
IP	FIPA-query Protocol

Tabla 7: Esquema de interacción con motivador *Brindar la información solicitada*

3. MENSAJES ACL INTERCAMBIADOS POR LOS AGENTES DEL AsistenteHIM

A continuación se presentan todos los mensajes intercambiados entre agentes, construidos con base en los componentes que especifica el estándar ACL, en la etapa de diseño del AsistenteHIM, capítulo 3.

Entidad que envía el mensaje	Mensaje
interfazHIM	:performative inform :receiver: agenteCoordinador :content: accion=1|2|3|4|5|6, datos :language fipa-sl0 NOTA: Este propiamente no es un mensaje que será traducido a ACL, solo ejemplifica que por medio de las acciones del usuario en la interfaz gráfica de HIM, se contactará al agenteCoordinador para que lleve a cabo una acción.
agenteCoordinador 0	:performative Inform :reveiver interfazHIM :content "información para el usuario, dependiendo de lo provisto por los demás agentes"

	:language fipa-sl0 NOTA: Tampoco es un mensaje ACL en sí, y solo termina de ejemplificar la relación de información que debe haber entre la interfaz y el agenteCoordinador. Objetivo: recordar al usuario sus tareas pendientes :performative request :reveiver agenteSupervisor :content: accion=1, datos (rol, proceso) :language fipa-sl0 :conversation-id 011 NOTA: La preformativa request(a) indica que el receptor debe realizar una acción a y además responder al emisor el resultado de dicha acción con un acto comunicativo inform [Mas01]. La mayoría de los mensajes de este sistema son de este tipo, y como no involucran conversaciones complejas. Objetivo: coordinar y contactar usuarios :performative request :reveiver agenteSupervisor :content accion=2, datos (rol, actividad, proceso=GN) :language fipa-sl0 :conversation-id 012 Objetivo: detectar dependencias :performative request :reveiver agenteSupervisor :content accion=3, datos (actividad, proceso-GN) :language fipa-sl0 :conversation-id 013 NOTA: Se trata de detectar dependencias con actividades aún no hechas Objetivo: brindar información solicitada-rol :performative request :reveiver agenteBuscador :content accion=4, datos (rol) :language fipa-sl0 :conversation-id 021 Objetivo: brindar información solicitada-actividad-producto :performative request :reveiver agenteBuscador :content accion=5, datos (actividad, proceso=GN, producto) :language fipa-sl0 :conversation-id 022 NOTA: El mensaje se envía al momento en que la interfaz notifica que el usuario ha seleccionado una actividad o producto a realizar. Objetivo: tareas RBC :performative request :reveiver agenteRBC :content accion=6, datos (rol, actividad, tamaño empresa, nivel empresa) :language fipa-sl0 :conversation-id 031 Objetivo: Confirmar que se recibió la información :performative inform :reveiver agentes :content accion=1|2|3|4|5|6, recibido=true :language fipa-sl0 :conversation-id 0#8, donde # es el número que identifica al agente receptor
agenteSupervisor 1	Objetivo: recordar al usuario sus tareas pendientes :performative inform :reveiver agenteCoordinador :content accion=1, datos (rol, proceso, actividadesPendientes) :language fipa-sl0 :conversation-id 011 Objetivo: coordinar y contactar usuarios :performative inform

	:reveiver agenteCoordinador :content accion=2, datos (lista de roles relacionados con la actividad tal y sus datos de usuario) :language fipa-sl0 :conversation-id 012 Objetivo: detectar dependencias :performative inform :reveiver agenteCoordinador :content accion=3, datos (lista de actividades con las que se tiene dependencia y no estén hechas) :language fipa-sl0 :conversation-id 013 Objetivo: brindar información solicitada-actividades pendientes :performative request :reveiver agenteBuscador :content accion=1, datos (rol, proceso=GN) :language fipa-sl0 :conversation-id 121 Objetivo: coordinar y contactar usuarios :performative request :reveiver agenteBuscador :content accion=2, datos (rol, actividad, proceso=GN) :language fipa-sl0 :conversation-id 122 Objetivo: detectar dependencias :performative request :reveiver agenteBuscador :content accion=3, datos (actividad, proceso-GN) :language fipa-sl0 :conversation-id 123
agenteBuscador 2	Objetivo: brindar información solicitada-rol :performative inform :reveiver agenteCoordinador :content accion=4, datos (rol, Información útil del rol) :language fipa-sl0 :conversation-id 021 Objetivo: brindar información solicitada-actividad-producto :performative inform :reveiver agenteCoordinador :content accion=5, datos (actividad, Información útil de la actividad y/o producto) :language fipa-sl0 :conversation-id 022 NOTA: La información puede ser cualquiera de la disponible en los archivos RDF. Objetivo: brindar información solicitada-actividades pendientes :performative inform :reveiver agenteSupervisor :content accion=1, datos (lista de actividades no hechas para determinado rol y proceso) :language fipa-sl0 :conversation-id 121 Objetivo: coordinar y contactar usuarios :performative inform :reveiver agenteSupervisor :content accion=2, datos (lista de roles relacionados con la actividad tal y sus datos de usuario) :language fipa-sl0 :conversation-id 122 Objetivo: detectar dependencias :performative inform :reveiver agenteSupervisor :content accion=3, datos (lista de actividades con las que se tenga dependencia y no estén hechas) :language fipa-sl0 :conversation-id 123

	Objetivo: solicitud de información :performative inform :reveiver agenteRBC :content acccion, datos :language fipa-sl0 :conversation-id 321 Objetivo: solicitud de información :performative request :reveiver Base de conocimiento (Configuración y Repositorio) :content query :language fipa-sl0 NOTA: Este no es un mensaje ACL en sí, solo ejemplifica la interacción que debe haber entre el agenteBuscador y la base de conocimiento en RDF, a través de consultas RDQL.
agenteRBC 3	Objetivo: tareas RBC :performative inform :reveiver agenteCoordinador :content accion=6, datos (Oración con la sugerencia, prevención o instrucción que surja del RBC). :language fipa-sl0 :conversation-id 031 Objetivo: solicitud de información :performative request :reveiver agenteBuscador :content accion, datos :language fipa-sl0 :conversation-id 321

Tabla 8: Mensajes de los agentes del AsistenteHIM

Apéndice 3

EL RAZONADOR BASADO EN CASOS

Esta sección contiene los resultados de las entrevistas realizadas para la recolección de casos (capítulo 4) y la lista completa de dichos casos, que constituyen la base para el razonador basado en casos implementado en el *agenteRBC* del AsistenteHIM.

1. RESULTADO DE LAS ENTREVISTAS

Las entrevistas para la recolección de casos se realizaron con grabadora de voz a tres representantes de las distintas empresas seleccionadas para las pruebas controladas de MoProSoft y a una de las consultoras que participó en el proceso. Se enfocó sólo en la categoría de Dirección que marca MoProSoft. A continuación a manera de oraciones, los aspectos más relevantes de cada entrevista.

1.1 Entrevista al Ing. Edgardo Herrera (ARTEC S.A. de C.V.)

- Sí implantaron todas las actividades de MoProSoft.
- Se encontraban en nivel 0 después de la primera avaluación y terminaron en nivel 1 en todos los procesos.
- Como es una empresa pequeña cada empleado tenía más de un rol asignado.
- Es importante contar con una herramienta de apoyo al MoProSoft, de hecho ellos están desarrollando una.

1.2 Entrevista a la Ing. Angélica Su (Itera S.A. de C.V.)

- Todas las empresas tuvieron problemas en la definición del *Plan Estratégico* y fue el único producto que generaron (exceptuando Magnabyte S.A. de C.V.).
- Tuvieron problemas para interpretar algunos términos de MoProSoft, como son proceso, actividad y rol.
- Se les dificultó seguir el flujo de trabajo del MoProSoft.
- Para la realización del *Plan Estratégico* se recomendó utilizar la herramienta analítica FODA (Fortalezas, Oportunidades, Debilidades y Amenazas).
- La empresa e-genium S.C. siguió un taller de planeación estratégica para poder cumplir con las actividades.
- Revisar documentos para ver partes del *Plan Estratégico*. De acuerdo a la parte (visión, misión, estrategias…) que se esté desarrollando se pueden sugerir algunas cosas.
- El modelo está enfocado a PyMEs dedicadas al desarrollo y mantenimiento de software.
- El escalar los problemas a un rol indicado en el modelo no se llevó a cabo porque las empresas al ser pequeñas hacían reuniones de evaluación. En los demás procesos eso si es posible.

1.3 Entrevista a la Ing. Ángeles Naranjo (SGA S.C.)

- Empresa pequeña, trabajadores multi-rol.
- Ya contaban con un documento con algunos de los elementos que conforman el *Plan Estratégico* así que lo adaptaron para desarrollar dicho documento.
- Llevaron a cabo un foro de discusión por e-mail y después una reunión para analizar y redefinir conceptos. El iniciador del foro y el que dio los conceptos principales fue el director general.
- Para el desarrollo del plan estratégico sugieren involucrar a las áreas directivas y además a algunos representantes técnicos porque son los que saben cómo se hacen las cosas.
- Contaban con expertos para articular la misión, visión, objetivos, etc., que sabían el significado de los términos y como desarrollarlos.

- Se asignaron los roles pero también se involucraba a más gente de las áreas técnicas.
- Sugieren no proporcionar modelos de documentos porque generalmente todas las empresas ya cuentan con sus formatos y documentos establecidos. Más bien sugerir coordinar el flujo de esos documentos.
- Para la parte de entendimiento de la situación actual del *Plan Estratégico* definitivamente sirve un FODA, una lluvia de ideas y las sugerencias de la gente de desarrollo que está día a día trabajando en los proyectos de la empresa.
- Para ambiente de trabajo, ayudan encuestas con los trabajadores.
- Recordar que el *Plan Estratégico* va cambiando cada cierto periodo, pero no todos sus elementos, algunos sólo se revisan y afinan.
- Son importantes las estadísticas hechas dentro de la empresa para el plan estratégico y las propuestas de mejora.
- Para el *Plan Estratégico* son buenos los estudios de mercado, evaluación del desempeño de la empresa en el periodo previo. Ayuda a definir el siguiente paso y la estrategia a seguir.
- Lección aprendida: Guardar un registro de lo que haces, como planeas, de lo que practicaste, y de lo que decidiste que es mejor entre una gama de opciones.
- Nadie registró lecciones aprendidas por lo pronto pero se dieron cuenta que es muy importante registrar.
- Conformación de la estructura organizacional: Ya tenían definida una estructura de empresa y se repartieron los roles tomando en cuenta algunas características tanto del modelo como del perfil de los empleados.
- Quieren mejorar el *Plan Estratégico* por medio del plan de mejora, con respecto a la estructura de la organización porque al asignar roles, conforme se van llevando a cabo las actividades del modelo, se observan las cargas de trabajo de cada uno, las responsabilidades y aptitudes adecuadas y es necesario hacer ajustes para que el trabajo sea equilibrado.
- Para conformar la estructura de la organización se debe tomar la estructura que ya se tenga implementada, identificar ciertas funciones, asignar y redistribuir las tareas y roles de cada persona. Aquí tuvieron problemas y la manera de evitarlo es estudiar un poco más las actividades y funciones de los roles y asignarlos mejor.
- El desarrollar el *Plan Estratégico*, ya sea desde abajo, o adaptando otros documentos, sirve para detectar fallas o nichos que necesitan mayor atención.
- En MoProSoft no se toma en cuenta el trabajo de las áreas de soporte técnico.
- Estaban en nivel 0 y subieron a nivel 1 en los nueve procesos.
- El apoyo del consultor es importante, sugerirlo siempre.
- Es importante tener ayuda para controlar el flujo de trabajo y llevar a cabo todas las actividades mencionadas en el modelo.
- Tuvieron problemas en la interpretación de los conceptos, el consultor ayudó.
- Problemas en adaptarse a una nueva forma de hacer las cosas: Registrar actividades, trabajar ordenadamente y secuencial, etc.
- La escalabilidad de los roles no se llevó a cabo en este proceso porque está muy pequeña la empresa y más bien se hacían reuniones generales.
- Importa el tamaño de la empresa.
- El personal pedía un diagrama único en el que se vieran todos los productos y sus dependencias.
- Problema con las dependencias entre actividades y productos. ¿Dónde dejar de hacer un proceso para comenzar otro? ¿Cuáles se pueden hacer en paralelo? ¿Qué hay de los niveles? Para eso ayuda mucho el MoProSoft coloreado.

- Faltaron sugerencias o guías de cómo comenzar a implantar el modelo y de las interacciones entre los procesos.

1.4 Entrevista al Ing. Oscar Flores (Magnabyte, S.A. de C.V)

- Trabajaron en la implantación del modelo de septiembre a febrero.
- Estaban en nivel 1 en los procesos de GN y DMS y en los demás en nivel 0. Objetivo de subir un nivel. Esfuerzo de subir dos niveles. Al final todo quedó en nivel 2 menos GP y GPYE por recomendación de las consultoras.
- Consultora Claudia Alquicira.
- A esta empresa le fue mejor, alcanzó objetivos y los mejoró.
- Ya contaban con una estructura de empresa.
- Se fue ubicando las áreas y se asignaron responsables a los roles de MoProSoft.
- Multi-rol.
- 14 personas implicadas en el modelo.
- El área de soporte fue la que estuvo viendo la parte de conocimiento de la organización.
- No tenían implementado ningún otro modelo de procesos.
- Sí se repartieron todos los roles.
- Se siguieron todas las actividades.
- En cuanto a los temas de GN ya se contaba con mucha documentación que analizándola se pudo ubicar o incluir dentro de las actividades y productos que sugería MoProSoft. Se fueron revisando la lista de actividades y productos complementando con lo que ya se tenía.
- Tenían conocimiento en la parte de ISO y un software para seguimiento de flujo de trabajo de las actividades del modelo, de las validaciones, verificaciones, responsables, asignación de roles, etc. Trabajaron con ese software. Ese es el mejor camino porque ayuda mucho al ser la parte documental y de seguimiento de proceso.
- Tenían definidos misión, visión, valores, objetivos y metas anuales, cartera de proyectos, estructura de la organización, presupuesto y sólo se aterrizaron al MoProSoft.
- El *Plan de Comunicación e implantación* ya se tenía en la práctica y hubo que documentarlo.
- Las validaciones y verificaciones formales son importantes.
- Es recomendable irse a los números para ver cómo va avanzando la empresa.
- Empezar con una hoja de presupuesto anual que contendrá cuáles son los costos fijos, las inversiones que se van a hacer durante el ejercicio en base en los objetivos que se estén planteando y otra parte de presupuesto necesario para ventas, que será generado por los responsables de esa área y no por los directivos, con el fin de tener planes realistas y contribuir a que la gente tenga un compromiso a cumplir lo que ellos mismos se plantearon. Que el director de ventas diga cuáles son sus metas durante el año y su presupuesto, después se revisan en dirección. Costos fijos, costo de inversión y plan de ventas para establecer un punto de equilibrio o de ganancia. Durante el año de forma trimestral, se van haciendo gráficas para ver el movimiento de la empresa y ver cómo van de acuerdo a lo especificado.
- Seguimiento de los estados financieros como métricas que sugiere MoProSoft, se incluyeron como parte de los productos. Se sumó al seguimiento del plan estratégico.
- Guía de ajuste al *Plan de comunicación e implantación*: tener cuidado con la información confidencial manejada en el *Plan Estratégico*.

- Como parte del seguimiento del *Plan de comunicación e implantación,* se utilizó la Internet e intranet. En la PC de cada empleado, la página de inicio sale la misión, visión, etc. Pero hacer énfasis en que se entienda.
- Guía de ajuste. En cuanto a escalar los problemas, no se hacía como lo especifica el modelo ya que para este proceso no es necesario, más bien se hacen reuniones mensuales para discutir el seguimiento de los procesos.
- Hacer énfasis en los ciclos, es decir, en que se tienen que estar revisando (anualmente por ejemplo) los documentos generados en este proceso.

2. CONTENIDO DE LA BASE DE CASOS

A continuación se muestra una tabla con la base de casos del sistema razonador basado en casos, resultado de las entrevistas anteriores y la consulta de información en el documento de MoProSoft.

Case Id	rol	actividad	Nivel Empresa	Tamaño Empresa	actividadObjetivo	actividadSugerencia	actividadPrevencion
0	ANY	GNA0.1	0	ANY	Entrar al sistema	Si se tienen problemas con la definición de algunos términos de MoProSoft, consultar el documento del modelo en la página 5.	NADA
1	ANY	GNA0.2	0	Pequeña	Seleccionar el proceso GN	Durante este proceso (GN), MoProSoft define una jerarquía de roles para escalar los problemas, sin embargo, al ser pequeña su empresa, los problemas se pueden tratar entre todo el grupo de GN en reuniones.	NADA
2	GD	GNA1.1	0	Pequeña	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.	Como GD Ud. puede sugerir la utilización del análisis FODA o lluvia de ideas entre sus colaboradores.	NADA
3	RGN	GNA1.2	0	Pequeña	Entender la situación actual.	Esta actividad involucra estar enterado de la situación interna y externa, por lo que se recomienda tener contacto con las diferentes áreas de la empresa y sus necesidades. Se pueden realizar encuestas entre los trabajadores.	No se deben descuidar las finanzas de la empresa para tener planes realistas basados en números y estadísticas. Por ejemplo es recomendable incluir al director de ventas para que participe en esta actividad.
4	RGN	GNA1.3	1	Pequeña	Desarrollar o actualizar Objetivos y Estrategias, considerando las Propuestas de Mejora, en caso de existir.	Tener a la mano los documentos de GN desarrollados durante el nivel 0, y sobre todo verificar que las propuestas de mejora sean realistas dentro de la empresa.	NADA
5	RGN	GNA1.4	0	Pequeña	Definir o actualizar los procesos y proyectos, considerando las Propuestas de Mejora en caso de existir.	Como RGN es conveniente tener buen conocimiento de las áreas de la empresa dedicadas a los procesos y proyectos, así como tener información financiera que sea coherente con lo que se pretende especificar en los documentos.	NADA
6	RGN	GNA1.5	0	Pequeña	Definir la Estructura de la Organización adecuada para la implantación del plan, para lo cual es necesario considerar las Propuestas de Mejora en caso de existir.	NADA	Analizar cuidadosamente el documento de MoProSoft y tomar en cuenta las capacidades y responsabilidades asignadas a cada rol, esto para no tener trabajadores con responsabilidades desiguales o con un perfil no adecuado.
7	RGN	GNA1.6	0	Pequeña	Definir o actualizar la Estrategia de Recursos, considerando las Propuestas de Mejora en	Tener comunicación con el RGR y si es posible pedirle un reporte de su área, y analizarlo antes de llevar a cabo esta actividad.	NADA

					caso de existir.		
8	RGN	GNA1.7	0	Pequeña	Calcular el presupuesto requerido (gastos e ingresos esperados) para lograr la implantación del Plan Estratégico, y determinar el periodo para el que aplicará.	NADA	Esta actividad se debe de realizar teniendo toda la información necesaria del área de ventas y finanzas de la empresa, así se tendrá información realista.
9	RGN	GNA1.8	0	Pequeña	Definir o actualizar la Periodicidad de Valoración del Plan Estratégico, considerando las Propuestas de Mejora, en caso de existir.	NADA	NADA
10	RGN	GNA1.9	0	Pequeña	Definir los mecanismos de comunicación con el cliente para su atención y documentarlos en el Plan de Comunicación con el Cliente.	Se puede sugerir la colaboración del área de mercadotecnia de la empresa para que ayude con ideas.	NADA
11	RGN	GNA1.10	0	Pequeña	Integrar y documentar el Plan Estratégico.	NADA	NADA
12	RGN	GNA1.11	0	Pequeña	Verificar el Plan Estratégico (Ver1).	NADA	NADA
13	RGN	GNA1.12	0	Pequeña	Corregir defectos encontrados en el Plan Estratégico con base al Reporte de Verificación y obtener la aprobación de las correcciones.	NADA	Al corregir o modificar el Plan Estratégico no olvidar tomar en cuenta el control de versiones.
14	GD	GNA1.13	0	Pequeña	Validar el Plan Estratégico (Val1).	Se deben de tomar en cuenta cada una de las partes que conforman el Plan Estratégico para evaluar si está correcto. No descuidar los reportes financieros y material alternativo que pueda ser de utilidad.	NADA
15	RGN	GNA1.14	0	Pequeña	Corregir defectos encontrados en el Plan Estratégico con base al Reporte de Validación y obtener la aprobación de las correcciones.	NADA	NADA
16	RGN	GNA1.15	0	Pequeña	Elaborar el Plan de Adquisiciones y Capacitación para el proceso de Gestión de Negocio.	NADA	NADA
17	RGN	GNA2.1	0	Pequeña	Preparar el ambiente adecuado para la implantación del Plan Estratégico.	NADA	NADA
18	RGN	GNA2.2	0	Pequeña	Definir y ejecutar el Plan de Comunicación e Implantación del Plan Estratégico.	Se puede sugerir publicar en cada computadora de la empresa, la visión, misión y valores de la misma, y al mismo tiempo promover que se comprenda por medio de talleres a los empleados.	NADA
19	GD	GNA2.3	0	Pequeña	Validar el Plan de Comunicación e Implantación (Val2).	NADA	NADA
20	RGN	GNA2.4	0	Pequeña	Corregir defectos encontrados en el Plan de Comunicación e Implantación con base al Reporte de Validación y obtener la aprobación de las correcciones.	NADA	NADA
21	RGN	GNA3.1	0	Pequeña	Análisis de la información y evaluación del desempeño.	NADA	NADA
22	RGN	GNA3.2	0	Pequeña	Generación de Reporte de Valoración en donde se registran los detalles de la tarea A3.1	NADA	NADA
23	RGN	GNA3.3	0	Pequeña	Generación de Propuesta de Mejoras al Plan Estratégico actual.	Mantener la comunicación con el grupo de gestión y organizar sus propuestas de manera que se puedan presentar al GD de una manera eficiente.	NADA
24	GD	GNA3.4	0	Pequeña	Validar la Propuesta de	NADA	NADA

					Mejoras (Val3).		
25	RGN	GNA3.5	0	Pequeña	Corregir defectos encontrados en la Propuesta de Mejoras con base al Reporte de Validación y obtener la aprobación de las correcciones.	NADA	NADA
26	RGN	GNA3.6	0	Pequeña	Generar el Reporte de Mediciones y Sugerencias de Mejora de este proceso, de acuerdo al Plan de Mediciones de Procesos.	NADA	NADA
27	RGN	GNA3.7	0	Pequeña	Identificar las Lecciones Aprendidas e integrarlas a la Base de Conocimiento.	NADA	NADA
28	GG	GNA1.4	0	Pequeña	Definir o actualizar los procesos y proyectos, considerando las Propuestas de Mejora en caso de existir.	Se pueden generar informes de acuerdo al área (procesos, proyectos o recursos) para informar al RGN y tomar decisiones más acertadas.	NADA
29	RGN	GNA1.4	1	Pequeña	Definir o actualizar los procesos y proyectos, considerando las Propuestas de Mejora en caso de existir.	NADA	Es recomendable tomar en cuenta las lecciones aprendidas para establecer la cartera de proyectos y evitar no cumplir con los compromisos que se vayan a adquirir.
30	RGN	GNA1.4	1	mediana	Definir o actualizar los procesos y proyectos, considerando las Propuestas de Mejora en caso de existir.	Requerir informes de cada área, con los resultados del periodo pasado, para ahorrar tiempo y tomar decisiones objetivas.	No se recomienda pedir la participación de todos los trabajadores ya que se perdería mucho tiempo por el tamaño de la empresa, en su lugar utilizar la jerarquía de roles que marca MoProSoft para resolver cualquier problema.
31	GD	GNA1.7	0	Pequeña	Calcular el presupuesto requerido (gastos e ingresos esperados) para lograr la implantación del Plan Estratégico, y determinar el periodo para el que aplicará.	Como GD se puede invitar al responsable de ventas y finanzas a la actividad y así comprometer su participación en el cumplimiento de lo que se establezca en esta actividad.	NADA
32	GD	GNA1.8	0	Pequeña	Definir o actualizar la Periodicidad de Valoración del Plan Estratégico, considerando las Propuestas de Mejora, en caso de existir.	NADA	NADA
33	GD	GNA1.9	0	Pequeña	Definir los mecanismos de comunicación con el cliente para su atención y documentarlos en el Plan de Comunicación con el Cliente.	NADA	Como GD se debe de asegurar que lo definido en esta actividad con la colaboración del RGN vaya de acuerdo a las expectativas de la empresa en todo sentido.
34	GG	GNA3.3	0	Pequeña	Generación de Propuesta de Mejoras al Plan Estratégico actual.	Como responsable de un área de la empresa, basar las sugerencias en resultados e informes para el RGN y el GD.	NADA
35	GD	GNA3.3	0	Pequeña	Generación de Propuesta de Mejoras al Plan Estratégico actual.	Tomar en cuenta las propuestas de mejora del RGN y decidir con base en ello.	NADA
36	RGN	GNA1.5	1	Pequeña	Actualizar la Estructura de la Organización adecuada para la implantación del plan, para lo cual es necesario considerar las Propuestas de Mejora en caso de existir.	Para actualizar la estructura de la organización tomar en cuenta los resultados de las actividades de MoProSoft realizadas cuando se estaba en fase 0 y de acuerdo a su análisis tomar las decisiones apropiadas.	NADA
37	RGN	GNA1.9	1	Pequeña	Definir los mecanismos de comunicación con el cliente para su atención y documentarlos en el Plan de Comunicación con el Cliente.	Evaluar los resultados o lecciones aprendidas en relación a este tema y modificar al Plan de Comunicación lo que se considere necesario.	NADA
38	GG	GNA3.3	1	mediana	Generación de Propuesta de Mejoras al Plan Estratégico actual.	Puede pedir reportes de cada área de la cual Ud. sea responsable y elaborar las	NADA

						sugerencias de modificación de acuerdo al contenido de dichos reportes.	
39	GD	GNA2.2	0	Pequeña	Definir y ejecutar el Plan de Comunicación e Implantación del Plan Estratégico.	NADA	Tener cuidado en no publicar partes con información confidencial de la empresa, en el Plan Estratégico.
40	GD	GNA2.1	0	Pequeña	Preparar el ambiente adecuado para la implantación del Plan Estratégico.	NADA	NADA
41	GD	GNA3.1	0	Pequeña	Análisis de la información y evaluación del desempeño.	NADA	NADA
42	GD	GNA1.1	1	Pequeña	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.	Tomar en cuenta los documentos anteriores y actualizarlos o adaptarlos al MoProSoft.	NADA
43	ANY	GNA0.1	0	ANY	Entrar al sistema	Se puede hacer uso de algún software que ayude a controlar el flujo de trabajo y control de versiones de los documentos que se generen.	NADA
44	GD	GNA1.1	0	Pequeña	Articular, documentar o actualizar la Misión, Visión y Valores, para definir la política de calidad.	Se puede organizar un taller de planeación estratégica para definir conceptos y utilizarlos, que ayude a aprender y llevar a cabo las actividades de MoProSoft.	NADA

Tabla 9: Base de Casos del RaBC del AsistenteHIM

Apéndice 4

DESARROLLO E INTEGRACIÓN

Esta sección complementa al capítulo 5 de este documento, ya que contiene parte del código importante de la aplicación, y la documentación necesaria para poder llevar a cabo la integración del sistema multi-agente AsistenteHIM con la Herramienta Integral para MoProSoft.

Se registraron los cambios que se hicieron en el código de la HIM y el JColibri, así como a los archivos y ontologías RDF. También se agregó información relacionada con las bibliotecas utilizadas.

Por último, se explica el contenido del disco compacto anexo a este documento de tesis.

1. CÓDIGO DE COMPONENTES IMPORTANTES DEL AsistenteHIM

En las siguientes secciones se encuentra el código de los componentes que fueron descritos en el capítulo 5. El código de todos los componentes no se encuentra en este apéndice porque ya ha sido descrito en el capítulo 5, pero puede ser consultado en el disco compacto anexo a este documento, al igual que la totalidad de la aplicación.

1.1 Código del servlet AgentServlet

A continuación se muestra el código del servlet AgentServlet que liga el sistema multi-agente con la HIM y es explicado en la sección 5.2.3.

```
// ----------------------------------------------------------------------------
package asistentehim;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import jade.core.Runtime;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.wrapper.*;
////////////
////////////
///      ///
///  1  ////
////////////
////////////
public class AgentServlet extends HttpServlet
{
  private static AgentController agenteCoordinador = null;
  private static AgentController agenteSupervisor = null;
  private static AgentController agenteBuscador = null;
  private static AgentController agenteRBC = null;
  /**
    Synchronized static method called at the initialization of the servlet. It ensures that a
    non-main container is created once and only once, with a proxy agent associated to this servlet.
  */
////////////
////////////
///      ///
///  2  ////
////////////
////////////
  synchronized private static void SyncInit()
  {
    // Have the proxy agent and its container already been initialized?
    if (agenteCoordinador==null)
    {
      // Get a hold on JADE runtime
      Runtime l_JADERunTime = Runtime.instance();

      // Create a default profile
      Profile l_JADEProfile = new ProfileImpl();

      // Create a new non-main container, connecting to the default main container (i.e. on this host, port
1099)
      AgentContainer l_NewContainer = l_JADERunTime.createAgentContainer(l_JADEProfile);
      try
      {
        // Create the object used to synchronize the starting of the Servlet and the Proxy agent
        Object l_Arg[] = new Object[1];
        Synchronizer l_Sync = new Synchronizer();
        l_Arg[0] = l_Sync;

        // Create the Proxy Agent and pass it the synchronizing object as argument
        agenteCoordinador                =                l_NewContainer.createNewAgent("agenteCoordinador",
agenteCoordinador.class.getName(),l_Arg);
        agenteSupervisor                 =                 l_NewContainer.createNewAgent("agenteSupervisor",
agenteSupervisor.class.getName(),null);
        agenteBuscador = l_NewContainer.createNewAgent("agenteBuscador", agenteBuscador.class.getName(),null);
        agenteRBC = l_NewContainer.createNewAgent("agenteRBC", agenteRBC.class.getName(),null);
        // Start the Agents
        agenteCoordinador.start();
        agenteSupervisor.start();
```

```
        agenteBuscador.start();
        agenteRBC.start();
        // Wait for synchronization signal
        l_Sync.waitOn();
       }
       catch(Exception l_Exception) { l_Exception.printStackTrace(); }
     }
  } // End of SyncInit

  /**
    Just call the synchronized initialization.
  */
  public void init() throws ServletException
  {
    SyncInit();
  }

  /**
    Respond to an HTTP Request by passing an Interaction object to the proxy agent using
    the object2agent channel and waiting for the signal that the Response of the Interaction was updated.
   * @param p_Request incoming HTTP request.
   * @param p_Response outgoing HTTP Response.
   */
/////////////
/////////////
///      ///
///  3  ////
////////////
////////////
  public void doGet (HttpServletRequest p_Request, HttpServletResponse p_Response) throws ServletException,
IOException
  {
   // Create an interaction object wrapping the HTTP request and response
   Interaction l_Interac = new Interaction(p_Request,p_Response);

   // Pass the interaction to the Proxy Agent and wait for the signal that the interaction was updated
   try
   {
     agenteCoordinador.putO2AObject(l_Interac, AgentController.ASYNC);
     l_Interac.waitChangedResponse();
   }
   catch(Exception l_Exception) { l_Exception.printStackTrace(); }
  } // End of doGet
} // End of Servlet
```

1.2 Código del razonador basado en casos (RaBC.java)

El siguiente es el código del razonador basado en casos que utiliza el agenteRBC.

```
package asistentehim;

import jcolibri.cbrcore.*;
import jcolibri.cbrcore.packagemanager.*;
import java.util.*;
import java.net.*;

public class RaBC {

        CBRCore core;
        ArrayList preCycleTaskList  = new ArrayList();
        ArrayList cycleTaskList     = new ArrayList();
        ArrayList postCycleTaskList = new ArrayList();


        /**
        * Init method of the application. Will invoke several methods
        * to leave ready the application.
        */
        public void init() {
                core = new CBRCore("RaBC");

        PackageManager.getInstance().getPackageByName("Core").setActive(true);
        PackageManager.getInstance().getPackageByName("User Components").setActive(true);

        core.init();

                setCBRApplicationConfiguration();
        }

        /**
         * Resets the CBR application
         */
```

```
        public void reset()
        {
            try
        {
            core.resetContext();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
        }

        /**
        * Describes the context (current state) of the CBR Application
        * @returns String describing current context.
        */
        public String getContext()
        {
            return core.getContext().toString();
        }


        /**
        * This method will be invoked to run some tasks of the CBR Application
        * @return CBRContext, result of the CBR application execution
        */
        protected CBRContext executeTasks(List taskList) {
                try {
                        core.executeTasks(
                                (CBRTask[]) taskList.toArray(new CBRTask[taskList.size()]));
                } catch (Exception ee) {
                        ee.printStackTrace();
                }
                return core.getContext();
        }

        /**
        * This method will be invoked to run the pre-cycle of the CBR Application
        * @return CBRContext, result of the pre-cycle execution
        */
        public CBRContext executePreCycle()
        {
                return this.executeTasks(preCycleTaskList);
        }

        /**
        * This method will be invoked to run the main cycle of the CBR Application
        * @return CBRContext, result of the main cycle execution
        */

        public CBRContext executeCycle()
        {
                return this.executeTasks(cycleTaskList);
        }

        /**
        * This method will be invoked to run the post-cycle of the CBR Application
        * @return CBRContext, result of the post-cycle execution
        */
        public CBRContext executePostCycle()
        {
                return this.executeTasks(postCycleTaskList);
        }


        /**
        * This method holds the automatic code generated for your application
        */
        public final void setCBRApplicationConfiguration() {
                CBRTask task;
                CBRMethod method;
                HashMap parametersMap;
                ArrayList taskList;

////////////////////////////////////////////////////////////////////////////////////////////////////
                // Código añadido: Declaración de variables
////////////////////////////////////////////////////////////////////////////////////////////////////
                String rol = "GD";
                String actividad = "11";
                String nivelEmpresa = "0";
                String tamañoEmpresa = "pequeña";
                String actividadObjetivo = "";
                String actividadSugerencia = "";
                String actividadPrevencion = "";

////////////////////////////////////////////////////////////////////////////////////////////////////
                // FIN de Código añadido: Declaración de variables
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
                try {
                        /*****************************/
                        /* PreCycle Tasks & Methods */
                        /*****************************/
                        taskList = preCycleTaskList;

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("PreCycle","PreCycle7200");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().
                          createInstance("jcolibri.method.PreCycleMethod","jcolibri.method.PreCycleMethod844");
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Obtain cases task","Obtain cases task6037");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.LoadCaseBaseMethod","jcolibri.method.LoadCaseBaseMethod8086");
                        core.resolveTaskWithMethod(task, method);

                        //!!!NOTE: Final version should maybe modified these values
                        //These are serialized version of the values
                        //assigned during application configuration.
                        parametersMap = new HashMap();
                        //parameter:Connector=C:jCOLIBRIexamplesRaBCconnector.xml

parametersMap.put("Connector",jcolibri.util.JColibriClassHelper.deserializeObject(URLDecoder.decode("%EF%BE%AC%
EF%BF%AD%00%05t%00%27C%3A%5CjCOLIBRI%5Cexamples%5CRaBC%5Cconnector.xml","UTF-8")));
                        method.setParameters(parametersMap);


                        /*****************************/
                        /* Cycle Tasks & Methods */
                        /*****************************/
                        taskList = cycleTaskList;

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("CBR Cycle","CBR Cycle3990");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().
                          createInstance("jcolibri.method.CBRMethod","jcolibri.method.CBRMethod9430");
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Obtain query task","Obtain query task8169");//                    ****
******** ****** Aquí van los cambios...para quitar la GUI
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.ConfigureQueryMethod","jcolibri.method.ConfigureQueryMethod5296");
                        core.resolveTaskWithMethod(task, method);

                        //!!!NOTE: Final version should maybe modified these values
                        //These are serialized version of the values
                        //assigned during application configuration.
                        parametersMap = new HashMap();                                                  // ------
---------- OJO aquí se agregan los demás parámetros ---------------------
                        //parameter:Case Structure=C:jCOLIBRIexamplesRaBCcaseStructure.xml
                        parametersMap.put("Case
Structure",jcolibri.util.JColibriClassHelper.deserializeObject(URLDecoder.decode("%EF%BE%AC%EF%BF%AD%00%05t%00%
2BC%3A%5CjCOLIBRI%5Cexamples%5CRaBC%5CcaseStructure.xml","UTF-8")));

////////////////////////////////////////////////////////////////////////////////////////////////////
                        // Código añadido: Agregar parámetros a la HashMap
////////////////////////////////////////////////////////////////////////////////////////////////////
                       parametersMap.put("rol", rol);
                        parametersMap.put("actividad", actividad);
                        parametersMap.put("nivelEmpresa", nivelEmpresa);
                        parametersMap.put("tamañoEmpresa", tamañoEmpresa);
                        parametersMap.put("actividadObjetivo", actividadObjetivo);
                        parametersMap.put("actividadSugerencia", actividadSugerencia);
                        parametersMap.put("actividadPrevencion", actividadPrevencion);

////////////////////////////////////////////////////////////////////////////////////////////////////
                 // FIN de Código añadido: Agregar parámetros a la HashMap
////////////////////////////////////////////////////////////////////////////////////////////////////
                        method.setParameters(parametersMap);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Retrieve Task","Retrieve Task2070");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.RetrieveComputationalMethod","jcolibri.method.RetrieveComputationalMethod8382")
;
                        core.resolveTaskWithMethod(task, method);
```

```
                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Select working cases task","Select working cases task7708");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.SelectAllCasesMethod","jcolibri.method.SelectAllCasesMethod4244");
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Compute similarity task","Compute similarity task4651");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.NumericSimComputationMethod","jcolibri.method.NumericSimComputationMethod2682")
;
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Select best task","Select best task5566");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.SelectBestCaseMethod","jcolibri.method.SelectBestCaseMethod1608");
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Reuse Task","Reuse Task3899");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.ReuseComputationalMethod","jcolibri.method.ReuseComputationalMethod8315");
                        core.resolveTaskWithMethod(task, method);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Prepare Cases for Adaptation Task","Prepare Cases for Adaptation
Task4529");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.CopyCasesforAdaptationMethod","jcolibri.method.CopyCasesforAdaptationMethod3265
");
                        core.resolveTaskWithMethod(task, method);

                        //!!!NOTE: Final version should maybe modified these values
                        //These are serialized version of the values
                        //assigned during application configuration.
                        parametersMap = new HashMap();
                        //parameter:Case Structure=C:jCOLIBRIexamplesRaBCcaseStructure.xml
                        parametersMap.put("Case
Structure",jcolibri.util.JColibriClassHelper.deserializeObject(URLDecoder.decode("%EF%BE%AC%EF%BF%AD%00%05t%00%
2BC%3A%5CjCOLIBRI%5Cexamples%5CRaBC%5CcaseStructure.xml","UTF-8")));
                        method.setParameters(parametersMap);

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("Atomic Reuse Task","Atomic Reuse Task787");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.DirectCopyMethod","jcolibri.method.DirectCopyMethod9861");
                        core.resolveTaskWithMethod(task, method);

                        //!!!NOTE: Final version should maybe modified these values
                        //These are serialized version of the values
                        //assigned during application configuration.
                        parametersMap = new HashMap();
                        //parameter:Description Attribute=Description.actividad
                        parametersMap.put("Description
Attribute",jcolibri.util.JColibriClassHelper.deserializeObject(URLDecoder.decode("%EF%BE%AC%EF%BF%AD%00%05t%00%
15Description.actividad","UTF-8")));                        //parameter:Solution Attribute=Description.actividad
                        parametersMap.put("Solution
Attribute",jcolibri.util.JColibriClassHelper.deserializeObject(URLDecoder.decode("%EF%BE%AC%EF%BF%AD%00%05t%00%
15Description.actividad","UTF-8")));
                        method.setParameters(parametersMap);


                        /***************************/
                        /* PostCycle Tasks & Methods */
                        /***************************/
                        taskList = postCycleTaskList;

                        task = TasksInstanceRegistry.getInstance().
                          createInstance("PostCycle","PostCycle2419");
                        taskList.add(task);
                        method = MethodsInstanceRegistry.getInstance().

createInstance("jcolibri.method.CloseConnectorMethod","jcolibri.method.CloseConnectorMethod4234");
                        core.resolveTaskWithMethod(task, method);
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

2. ADICIONES AL PROYECTO HIM

Como se describió en el capítulo 5, para completar el desarrollo del AsistenteHIM y su integración con la HIM, fue necesario añadir algunos archivos y modificar otros cuantos:

Archivos añadidos a la HIM:

1. agenteCoordinador.java
2. agenteSupervisor.java
3. agenteBuscador.java
4. agenteRBC.java
5. AgentServlet.java
6. Interaction.java
7. Synchronizer.java
8. DAORDFActividad_AH.java
9. IDAORDFActividad_AH.java
10. RaBC.java
11. menuContextual.js
12. menuContextual.css

Archivos modificados en la HIM:

1. PlantillaProcesos.jsp
2. actividad.js
3. actividad.owl
4. GestionNegocio.rdf

De los archivos añadidos ya se han descrito los agente*.java y los DAO en el capítulo 5, y en la sección anterior se incluyó el RaBC. El código restante se presenta a continuación.

2.1 Código del Script menuContextual.js

El siguiente es el código en lenguaje JavaScript que genera el menú contextual que se despliega al hacer clic derecho sobre el árbol de actividades de la interfaz de la HIM, y que le muestra al usuario cuatro de las opciones de servicios del AsistenteHIM.

```
function openMenu(event, id) {
  var el, x, y, layer1;
  el = document.getElementById(id);
  // Obtenemos la capa donde se abre el menu
  layer = document.getElementById('Layer1');
  if (window.event) {
    x = window.event.clientX + layer.scrollLeft - layer.offsetLeft;
    y = window.event.clientY + layer.scrollTop - layer.offsetTop;
  }
  else {
```

```
    x = event.clientX + window.scrollX;
    y = event.clientY + window.scrollY;
  }
  x -= 2; y -= 2;
  // Posicionamos el menu de acuerdo al espacio dado por la capa
  // Sobre el eje X
  if((x+el.clientWidth-layer.scrollLeft) > layer.clientWidth) {
          var sobrante = (x+el.clientWidth-layer.scrollLeft) - layer.clientWidth;
          x = x - sobrante;
  }
  // Sobre el eje Y
  if((y+el.clientHeight-layer.scrollTop) > layer.clientHeight) {
          var sobrante = (y+el.clientHeight-layer.scrollTop) - layer.clientHeight;
          y = y - sobrante;
  }

  el.style.left = x + "px";
  el.style.top  = y + "px";
  el.style.visibility = "visible";
}

function closeMenu(event) {
  var current, related;
  if (window.event) {
    current = this;
    related = window.event.toElement;
  }
  else {
    current = event.currentTarget;
    related = event.relatedTarget;
  }
  if (current != related && !contains(current, related))
    current.style.visibility = "hidden";
}

function contains(a, b) {
  // Return true if node a contains node b.
  while (b.parentNode)
    if ((b = b.parentNode) == a)
      return true;
  return false;
}
```

2.2 Código de la hoja de estilo menuContextual.css

El siguiente código es una hoja de estilo que le da formato al elemento menú contextual.

```
.menu {
  background-color: gray;
  border-color: #80e0ff #006080 #006080 #80e0ff;
  border-style: solid;
  border-width: 1px;
  position: absolute;
  left: 0px;
  top: 0px;
  visibility: hidden;
}

a.menuItem {
  border: white;
  color: white;
  cursor: default;
  display: block;
  font-family: MS Sans Serif, Arial, Tahoma,sans-serif;
  font-size: 7pt;
  font-weight: normal;
  padding: 2px 12px 2px 8px;
  text-decoration: none;
}

a.menuItem:hover {
  background-color: #666;
  color: white;
}
```

3. CAMBIOS AL PROYECTO HIM

3.1. Cambios hechos al JSP PlantillaProcesos.jsp

Los cambios al JSP PlantillaProcesos.jsp son muy leves. Consisten en incluir dentro del archivo una función que llame al *AgentServlet* dentro de una nueva ventana, cuando el usuario realice alguna acción que tenga relación con los servicios del AsistenteHIM. Además se incluye el *script* del menú contextual y su respectiva hoja de estilo.

A continuación se presenta en negritas el código de los cambios. Se ha dejado un poco del código original para ubicar dónde van los cambios dentro del JSP completo.

```
<app:listaProcesos> </app:listaProcesos>
<app:listaRolesProceso> </app:listaRolesProceso>    ←-Código original al inicio del JSP
<html>
<head>
<title>Herramienta Integral MoProSoft</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<link rel="StyleSheet" href="style_css/menuContextual.css" type="text/css" />
       <script type="text/javascript" src="javascripts/menuContextual.js"></script>

<script language="javascript" type="text/javascript">
<!--
function popitup(caso, actividad, producto)
{
       var parametros;
        var rol;
        var proceso = "GestionNegocio.rdf";//por lo pronto ya que la herramienta es sólo para GN
pero se puede extender a más procesos.
        rol                                                                                    =
       document.CambioProcesoActionForm.rolesProceso.options[document.CambioProcesoActionForm.rol
esProceso.selectedIndex].value;
        //proceso                                                                              =
       document.CambioProcesoActionForm.procesos.options[document.CambioProcesoActionForm.proceso
s.selectedIndex].value;
       parametros                                                                              =
"caso="+caso+"&actividad="+actividad+"&producto="+producto+"&rol="+rol+"&proceso="+proceso;
       url = 'DirectoryServlet?' + parametros;
       newwindow=window.open(url,'name','height=280,width=550');
       if (window.focus) {newwindow.focus()}
}

// -->
</script>


</head> ←-Código original
```

Además se agregaron las imágenes debajo de los controles de roles y procesos de la siguiente manera:

```
<a href="javascript:popitup(1);"><img src="/WebModuleHIM/Imagenes/informacion.gif" border="0"
alt="Información"></a>
```

y

```
<a href="javascript:popitup(4);"><img src="/WebModuleHIM/Imagenes/informacion.gif" border="0"
alt="Información"></a>
```

3.2. Cambios hechos al *Script* actividad.js

A continuación el código del archivo actividad.js con las agregaciones marcadas con negritas. Nota: Para no ocupar tanto espacio, se cortó el código dejando sólo las partes para que se pueda ubicar dónde va el cambio y el mismo cambio.

```
--- más código ---

// Creates the node icon, url and text

dTree.prototype.node = function(node, nodeId) {

        // ************************************************************************
        // @AHIM
        // # MODIFICACION
        // Descripcion: Se agregó el evento onContextMenu() para desplegar un menu
        // ************************************************************************
        if (this.root.id != node.pid) {
        var str = '<div class="dTreeNode" oncontextmenu="javascript:' + this.obj + '.miMenuContextual(' +
nodeId + '); return false;">' + this.indent(node, nodeId);
        }
        else {
        var str = '<div class="dTreeNode" oncontextmenu="return false;">' + this.indent(node, nodeId);
        }
        // ************************************************************************

        if (this.config.useIcons) {

                if (!node.icon) node.icon = (this.root.id == node.pid) ? this.icon.root : ((node._hc) ?
this.icon.folder : this.icon.node);

                if (!node.iconOpen) node.iconOpen = (node._hc) ? this.icon.folderOpen : this.icon.node;

                if (this.root.id == node.pid) {

                        node.icon = this.icon.root;

                        node.iconOpen = this.icon.root;

                }

                str += '<img id="i' + this.obj + nodeId + '" src="' + ((node._io) ? node.iconOpen : node.icon)
+ '" alt="" />';

        }

        if (node.url) {

                str += '<a id="s' + this.obj + nodeId + '" class="' + ((this.config.useSelection) ? ((node._is
? 'nodeSel' : 'node')) : 'node') + '" href="' + node.url + '"';

                if (node.title) str += ' title="' + node.title + '"';

                if (node.target) str += ' target="' + node.target + '"';

                if (this.config.useStatusText) str += ' onmouseover="window.status=\'' + node.name +
'\';return true;" onmouseout="window.status=\'\';return true;" ';

                if (this.config.useSelection && ((node._hc && this.config.folderLinks) || !node._hc))

                        str += ' onclick="javascript: ' + this.obj + '.s(' + nodeId + ');"';

                str += '>';

        }

        else if ((!this.config.folderLinks || !node.url) && node._hc && node.pid != this.root.id)

                str += '<a href="javascript: ' + this.obj + '.o(' + nodeId + ');" class="node">';

        str += node.name;

        if (node.url || ((!this.config.folderLinks || !node.url) && node._hc)) str += '</a>';


        // ************************************************************************
        // @AHIM
        // # MODIFICACION
        // Descripcion: Se agrega el menu contextual a todos los nodos excepto a la raiz
        // ************************************
        if (this.root.id != node.pid) {
```

```
                str +=  this.addMenuContextual(node, nodeId);
        }
        //*************************************************************************

--- más código ---

if (!Array.prototype.pop) {

        Array.prototype.pop = function array_pop() {

                lastElement = this[this.length-1];

                this.length = Math.max(this.length-1,0);

                return lastElement;

        }

};

// ************************************************************************
// @AHIM
// # NUEVAS FUNCIONES
// ************************************************************************


// Funcion que abre el menu contextual definido en menuContextual.js
dTree.prototype.miMenuContextual = function(id) {
        openMenu(event, 'myMenu'+id);
};


// Funcion para agregar el menu contextual de cada nodo del arbol
// Descripcion: Se agregan las opciones del menu y el scritp para llamar la rutina de cierre del menu cuando
//               sale el raton.
dTree.prototype.addMenuContextual = function(node, nodeId) {
        // Elementos de menu
        var txt_opciones = [' Roles relacionados',' Dependencias','      Información','RBC'];
        var img_opciones = ['roles.gif', 'dependencias.gif', 'informacion.gif', 'rbc.gif'];
        var acc_opciones = [2, 3, 5, 6];
        // Nombre de la actividad(padre del nodo) y el producto(nodo)
        var actividad = this.aNodes[node.pid].name;
        var producto = node.name;
        // Cadena donde se genera dinamicamente el menu contextual
        var str = '';

        str += '<div id="myMenu'+ nodeId +'" class="menu"> ';
        for (var n=0; n<txt_opciones.length; n++) {
                str += '<a class="menuItem"
href="javascript:popitup('+acc_opciones[n]+',\''+actividad+'\',\''+producto+'\');">'+
                '<img src="images/'+ img_opciones[n] + '"></img>' +
                txt_opciones[n]+
                '</a>';
        }
        str +=  '</div>';
        str +=  '<script> document.getElementById("myMenu'+ nodeId +'").onmouseout = closeMenu;</script>'

        return str;
}
```

3.3. Cambios hechos a la ontología actividad.owl

Se hizo un pequeño cambio a la ontología de actividad, agregando la propiedad *dependeActividad* para poder agregar las actividades de las que depende directamente cada actividad.

```
<rdf:Property rdf:ID="dependeActividad">
        <rdfs:domain rdf:resource="#Actividad"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
```

3.4. Cambios hechos al archivo GestionNegocio.rdf

Se utilizó la propiedad definida en el punto anterior (*dependeActividad*) en cada una de las actividades que aparecen en el archivo RDF que representa el proceso de Gestión de Negocio, con la siguiente sintaxis:

<actividad:dependeActividad>A1.15</actividad:dependeActividad>

Es importante mencionar que el valor de la propiedad para cada actividad, fue resultado de un análisis detallado al documento del MoProSoft. Dicho análisis no se incluyó como apéndice en la tesis ya que sería más extensa, pero puede ser consultado en el disco compacto anexo bajo el nombre de Diagrama_de_Actividades_GN_1.1.doc.

4. CAMBIOS AL MARCO DE TRABAJO JColibri

Como se mencionó en la sección 5.3.5 del capítulo 5, se modificó el archivo …/JColibri/src/jcolibri/method/*ConfigureQueryMethod.java* del marco de trabajo JColibri para lograr la integración con la HIM. A continuación una descripción de los cambios apoyada con el código del método, dónde se muestran en negritas las adiciones.

Como para las otras secciones, los archivos con el código siguiente se encuentran en el disco compacto anexo a este documento, dentro de la carpeta JColibri.

```
package jcolibri.method;
import java.util.ArrayList;
import java.util.HashMap;
//MÁS IMPORTS…
//SE AGREGÓ LA SIGUIENTE:
import java.util.Set;

public class ConfigureQueryMethod extends jcolibri.cbrcore.CBRMethod {
        private static final long serialVersionUID = 1L;
        private String datos[] = new String[7];           //Se agregó datos[] y contador;
        private int contador;

        /** Parameter name containing the case structure file path. */
        public static final String CASE_STRUCTURE_FILE = "Case Structure";
        public CBRContext execute(CBRContext context) throws ExecutionException {
                String configFile = (String) this
                                .getParameterValue(CASE_STRUCTURE_FILE);

////////////////////////////////////////////////////////////////////////////////////////////////////
// Código añadido: Recuperar parámetros de la HashMap que viene de RaBC para llenar la query en lugar de la GUI
////////////////////////////////////////////////////////////////////////////////////////////////////
                datos[0] = (String) this.getParameterValue("rol");
                datos[1] = (String) this.getParameterValue("actividad");
                datos[2] = (String) this.getParameterValue("nivelEmpresa");
                datos[3] = (String) this.getParameterValue("tamañoEmpresa");
                datos[4] = (String) this.getParameterValue("actividadObjetivo");
                datos[5] = (String) this.getParameterValue("actividadSugerencia");
                datos[6] = (String) this.getParameterValue("actividadPrevencion");

////////////////////////////////////////////////////////////////////////////////////////////////////
//FIN de Código añadido:
////////////////////////////////////////////////////////////////////////////////////////////////////

                CaseStructure caseStructure = new CaseStructure();
                caseStructure.readFromXMLFile(configFile);
                List requestedAttributes = extractSimpleAttributes(caseStructure.getDescription());
                HashMap queryValues = new HashMap();

////////////////////////////////////////////////////////////////////////////////////////////
              // Cambios hechos para evitar la GUI que se genera automáticamente y estorba ya que mi aplicación
              // es para la web. Se necesita llenar la HashMar queryValues con los valores apropiados para
               // crear una consulta. SE COMENTÓ (ELIMINÓ) EL SIGUIENTE CÓDIGO
```

```
//              GenericMethodParametersPane pane = new GenericMethodParametersPane(
//                              requestedAttributes, queryValues);
//              JDialog dialog = pane.createJDialog(CBRArmGUI.getReference(), true,
//                              "Query");
//              dialog.setVisible(true);
//              boolean _continue = (pane.getValue() == GenericMethodParametersPane.OK);
//
//              if (!_continue)
//                      System.exit(0);

            for (Iterator it = requestedAttributes.iterator(); it.hasNext();) {
              MethodParameter param = (MethodParameter) it.next();
              ParameterEditor editor = ParameterEditorFactory.getEditor(param.getType());
              queryValues.put(param.getName(), editor);
            }
          contador = 0;
          Set keys = queryValues.keySet();
          for (Iterator it2 = keys.iterator(); it2.hasNext();) {
            String name = (String) it2.next();
            queryValues.put(name,  datos[contador]);  //Siendo  objetoValor  un  objeto  de  tipo  string  que
contiene el
            //valor del atributo.
            contador ++;
          }
////////////////////////////////////////////////////////////////////////////////////////////////////
// FIN de        Cambios hechos por Elena Cárdenas Vargas
////////////////////////////////////////////////////////////////////////////////////////////////////

              CBRQuery query = createQuery(caseStructure, queryValues);
              context.setQuery(query);
              return context;
      }
--- MÁS CÓDIGO QUE SE DEJÓ IGUAL ---
```

5. BIBLIOTECAS AGREGADAS AL PROYECTO HIM

Para el correcto funcionamiento de la HIM dentro de JBuilder es necesario tener algunas bibliotecas dentro del proyecto: *Servlet, Struts1.0.2, Jena, Circeos, log4j-1.2.7,* etc. De la misma manera, para integrar el AsistenteHIM en la HIM, fue necesario agregar dos bibliotecas más: *Jade* y *JColibri*, que contienen los archivos *.jar* que necesita la aplicación.

La figura 10 muestra las propiedades del proyecto HIM en JBuilder y las bibliotecas que utiliza. En el disco compacto del proyecto de tesis se encuentran todos los archivos que incluyen.

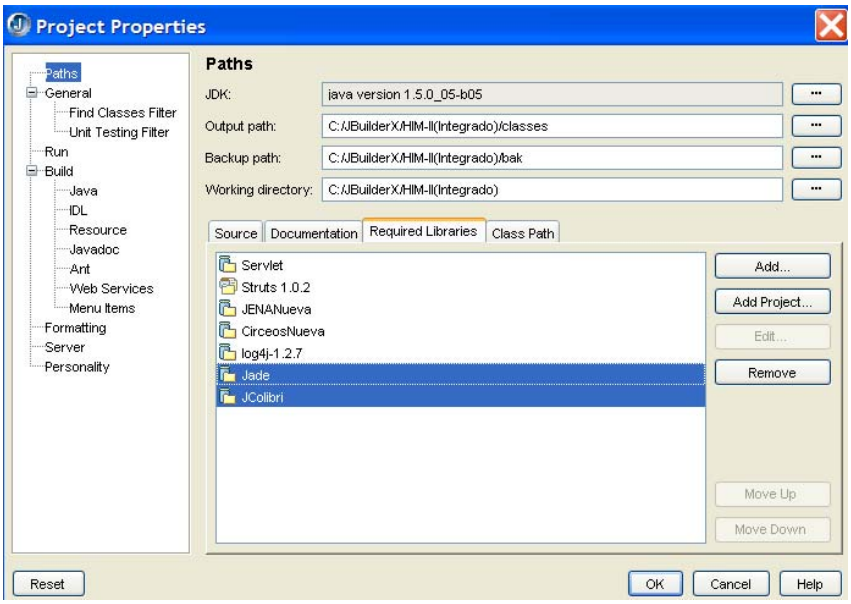

Figura 10: Propiedades del proyecto HIM en JBuilder integrado con AsistenteHIM

6. CONTENIDO DEL DISCO COMPACTO DEL PROYECTO

El documento de este proyecto de tesis se acompaña de un disco compacto que tiene la siguiente estructura:

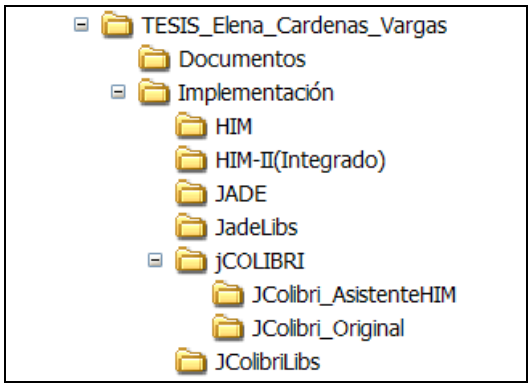

Figura 11: Estructura de archivos del disco compacto del proyecto de tesis AsistenteHIM

Dónde

- *TESIS_Elena_Cardenas_Vargas* es el directorio raíz.
- *Documentos* contiene un archivo *TESIS_AsistenteHIM.pdf* con el presente documento y *Diagrama_de_Actividades_GN_1.1.doc* con el análisis de actividades mencionado en este apéndice.
- *Implementación* contiene
  - *HIM:* Base de conocimiento con los archivos RDF.
  - *HIM-II(Integrado):* Proyecto completo con HIM y el AsistenteHIM.
  - *JADE:* Proyecto JADE.
  - *JadeLibs:* Librerías de JADE que hay que agregar al proyecto HIM.
  - *jCOLIBRI:* Proyecto JColibri modificado para que funcione con el AsistenteHIM y el proyecto JColibri original.
  - *JColibriLibs:* Librerías de jColibri que hay que agregar al proyecto HIM.

Acrónimos

ACC	*Agent communication Chanel.* Canal de Comunicación de Agentes.
ACL	*Agent Communication Language.* Lenguaje de Comunicación de Agentes.
AMCIS	Asociación Mexicana para la Calidad de Ingeniería de Software.
AMS	*Agent Management System.* Sistema de Administración de Agentes.
AOSE	*Agent Oriented Software Engineering.* Ingeniería de Software Orientada a Agentes.
APE	Proceso de Administración de Proyectos Específicos (MoProSoft).
API	*Application Programming Interface.* Interfaz de Programación de Aplicaciones.
AsistenteHIM	*Herramienta de guía y supervisión para el uso automatizado del modelo de procesos MoProSoft*
BD	Base de Datos.
BDI	*Belief-Desire-Intention*. Arquitectura de comportamiento de agentes basada en actitudes mentales como creencias, deseos e intenciones.
BSI	Sub-proceso de Bienes, Servicios e Infraestructura (MoProSoft).
CAL	*Comunicative Act Library.* Librería de Actos Comunicativos.
CASE	*Computer-Aided Software Engineering.* Ingeniería de Software Asistida por Computadora.
CDPS	*Cooperative Distributed Problem Solving.* Resolución Cooperativa de Problemas Distribuidos.
CLDC	*Connected Limited Device Configuration.* Configuración de Dispositivo Limitado Conectado.
CMM	*Capability Maturity Model.* Modelo de Madurez de Capacidades.
DAO	*Data Access Object.* Objeto de Acceso a Datos (patrón de diseño).

DIR	Categoría de Alta Dirección (MoProSoft).
DF	*Directory Facilitator.* Facilitador de Directorios. O Agente de Servicio de Directorio en JADE.
DMS	Proceso de Desarrollo y Mantenimiento de Software (MoProSoft).
EJB	*Enterprise JavaBeans.*
EURESCOM	*European Institute for Research and Strategic Studies in Telecommunications.* Instituto Europeo para la Investigación y Estudios Estratégicos en Telecomunicaciones
FIPA	*Foundation for Intelligent Physical Agents.* Fundación para Agentes Inteligentes Físicos.
FODA	Fortalezas, Oportunidades, Debilidades y Amenazas.
GAIA	*Group for Artificial Intelligence Applications.* Grupo para Aplicaciones de Inteligencia Artificial.
GD	Rol Grupo Directivo (MoProSoft).
GES	Categoría de Gestión (MoProSoft).
GN	Proceso de Gestión de Negocios (MoProSoft).
GNA	Gestión de Negocio-Actividad (MoProSoft).
GP	Proceso de Gestión de Procesos (MoProSoft).
GPY	Proceso de Gestión de Proyectos (MoProSoft).
GR	Proceso de Gestión de Recursos (MoProSoft).
GUI	*Graphical User Interface.* Interfaces Gráfica de Usuario.
GPRS	*General Packet Radio Service.*
HIM	Herramienta Integral para MoProSoft
IA	Inteligencia Artificial.
IAD	Inteligencia Artificial Distribuida.
IDE	*Integrated Development Environment.* Entorno Integrado de Desarrollo.

IEEE	*Institute of Electrical and Electronics Engineers.* Instituto de Ingenieros Eléctricos y Electrónicos.
IP	*Interaction Protocol.* Protocolo de Interacción (FIPA).
IS	Ingeniería de Software.
JADE	*Java Agent Development Framework.* Marco de trabajo en Java para el Desarrollo de Agentes.
JDBC	*Java Database Connectivity.* Conectividad Java para Bases de Datos.
JSP	*Java Server Page.* Página Java de Servidor
JVM	*Java Virtual Machine.* Máquina Virtual de Java.
J2EE	*Java 2 Platform, Enterprise Edition.* Plataforma Java 2, Edición de Empresa.
J2ME	*Java 2 Platform, Micro Edition.* Plataforma Java 2, Edición Pequeña.
J2SE	*Java 2 Platform, Standard Edition.* Plataforma Java 2, Edición Estándar.
KQML	*Knowledge Query and Manipulation Language.* Lenguaje ampliamente utilizado para comunicar agentes.
LGPL	*Lesser General Public License.*
MA	Modelo del Agente (MESSAGE).
MD	Modelo del Dominio (MESSAGE).
MESSAGE	*Methodology for Engineering Systems of Software AGent.* Metodología para la Ingeniería de Sistemas de Agentes de Software
MI	Modelo de Interacción (MESSAGE).
MIDP	*Mobile Information Device Profile*. Perfil de Dispositivio de Información Móvil.
MO	Modelo de la Organización (MESSAGE).
MO/T	Modelo de Objetivo/Tarea (MESSAGE).
MoProSoft	Modelo de Procesos para la Industria de Software
MVC	Modelo, Vista, Controlador (patrón de arquitectura)

OWL	*Ontology Web Language.* Lenguaje de Red para Ontologías.
OPE	Categoría de Operación (MoProSoft).
PIB	Producto Interno Bruto.
PIE	Productos de Información Especializada (HIM).
PMBOK	*Project Management Body of Knowledge*. Cuerpo de Conocimiento para la Administración de Proyectos.
POO	Programación Orientada a Objetos.
PROSOFT	Programa para el Desarrollo de la Industria de Software.
PSM	*Problem Solving Method.* Método Solucionador de Problemas.
PSP	*Personal Software Process.* Proceso Personal de Software
PU	Proceso Unificado de Desarrollo de Software.
RaBC	Razonador Basado en Casos
RBC	Razonamiento Basado en Casos.
RDF-S	*Resource Description Framework Schema.* Esquema para el Marco de Trabajo para la Descripción de Recursos.
RDQL	*RDF Data Query Language.* Lenguaje de Consulta de Datos RDF.
RGN	Rol Responsable de Gestión de Negocio (MoProSoft).
RGP	Rol Responsable de Gestión de Procesos (MoProSoft).
RGPY	Rol Responsable de Gestión de Proyectos (MoProSoft).
RGR	Rol Responsable de Gestión de Recursos (MoProSoft).
RHAT	Sub-proceso de Recursos Humanos y Ambiente de Trabajo (MoProSoft).
RMA	*Remote Monitoring Agent*. Agente de Monitoreo Remoto.
RMI	*Remote Method Invocation.* Invocación de Métodos Remotos.
SE	Secretaría de Economía.
SEI	*Software Engineering Institute.* Instituto de Ingeniería de Software.

SL	*Semantic Language*. Lenguaje semántico propuesto por la FIPA.
SL0	Semantic Language Subset 0. Subconjunto mínimo del lenguaje SL.
SMA	Sistema Multi-Agente.
SQL	*Structured Query Language.* Lenguaje Estructurado de Consultas.
SWEBOK	*Software Engineering Body of Knowledge.* Cuerpo de Conocimiento para la Ingeniería de Software.
TI	Tecnologías de Información.
TIC	Tecnologías de Información y Comunicación.
TILAB	Telecom Italia Lab
TSP	*Team Software Process.* Proceso de Software en Equipo.
UML	*Unified Modeling Language.* Lenguaje Unificado de Modelado.
URI	*Uniform Resource Identifier.* Identificador de Recurso Uniforme.
W3C	*World Wide Web Consortium.*
XML	*eXtensible Markup Language*. Lenguaje de Marcado Extensible.
4R's	Ciclo de un sistema RBC general descrito por los cuatro procesos siguientes: Recuperar, Reutilizar, Revisar y Retener según Agnar Aamodt y Enric Plaza [Aamodt01].