INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

DESARROLLO DE UNA HERRAMIENTA PARA LA CREACIÓN DE AGENTES SOBRE LA PLATAFORMA DE AGENTES COMPONENTES

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS
DE LA COMPUTACIÓN

PRESENTA
LIC. ERNESTO GERMÁN SOTO

DIRECTOR:
DR. LEONID B. SHEREMETOV

México D.F. Octubre de 2002

A la memoria de mi padre,
con la promesa irrevocable de seguir
por siempre y para siempre la guía de su ejemplo.

Agradecimientos

A José Luis, mi hermano el Mayor,
a quien la influencia inequívoca de
sus ideales y principios obedece
el concurso de mis modestos esfuerzos.

A Miguel Contreras padre,
que incondicionalmente me ha mostrado,
con calidad humana, su visión del mundo.
Y a Miguel Contreras hijo, por supuesto.

A Manuel y Claudia,
cuya amistad, apoyo y compañía estarán siempre
al alcance de cualquier pájaro de la memoria,
soportando el embate de los olvidos más tenaces.

A mi madre, a mis hermanos y a mis sobrinos,
fuente y fuerza de mi superación.

Al Dr. Leonid Sheremetov,
por su valiosa orientación en mi formación profesional
y en el desarrollo de esta tesis.

A todos los que contribuyeron en la realización de este trabajo.

ÍNDICE

ÍNDICE DE FIGURAS

Resumen

Las tendencias en el desarrollo de software indican que la próxima innovación tecnológica en este campo debe integrar organizaciones enlazadas y múltiples plataformas para las aplicaciones. Los desarrolladores deben construir sistemas unificados para el manejo de la información que utilizan las organizaciones a través de tecnologías de software avanzadas. Los agentes de software son una de las tecnologías para el desarrollo de aplicaciones computacionales que más llaman la atención actualmente porque pueden ser usados para construir rápida y fácilmente sistemas empresariales integrados. La idea de tener agentes de software es que pueden realizar tareas complejas en favor de los usuarios que los utilizan.

A pesar de un desarrollo impresionante en los últimos años, para que la tecnología de agentes se refleje del laboratorio a la práctica industrial, se requieren resolver varios problemas de ingeniería de software basado en agentes. Estos problemas radican en el hecho de que (i) las herramientas para facilitar el desarrollo de diferentes tipos de agentes prácticamente no existen, convirtiendo el proceso de diseño y desarrollo en un arte, (ii) existen problemas de interoperabilidad de agentes con otro software distribuido y (iii) de reutilización de agentes.

En este trabajo se propone el desarrollo de herramientas de software que faciliten el proceso de creación de agentes que puedan comunicarse utilizando los servicios que les brindan los objetos de la plataforma de agentes componentes CAP (*Component Agent Platform*). Para el diseño de tales herramientas de software es necesario y fundamental establecer un marco general de ideas que se enfoque al tipo de agentes colaborativos. Para resolver problemas de manera distribuida, esta idea se basa en un nuevo nivel de abstracción: el de la cooperación a nivel de conocimiento entre entidades autónomas y en entornos de sistemas flexibles, inciertos y distribuidos.

La idea central de la investigación es crear agentes a partir de controles ActiveX dado el contexto de la plataforma CAP, basada en la tecnología COM (*Component Object Model*) y DCOM (*Distributed COM*). Una motivación extra para utilizar la tecnología de componentes COM y ActiveX es desarrollar una infraestructura para facilitar la implementación de aplicaciones basadas en agentes en entornos de programación de la plataforma Windows de Microsoft. Para lograr los objetivos planteados, las herramientas desarrolladas están orientadas a facilitar la programación en aspectos clave como son el uso de un lenguaje de comunicación de agentes, un lenguaje de contenido, manipulación de conocimiento y mecanismos de control interno de los agentes.

La plataforma CAP fue desarrollada bajo las especificaciones de FIPA (*Foundation for Intelligent Physical Agents*) como parte de los proyectos de investigación del Laboratorio de Agentes del Centro de Investigación en Computación (CIC) del Instituto Politécnico Nacional (IPN). Por esta razón, los agentes que se registren y trabajen en la CAP deben seguir también las especificaciones de FIPA.

El documento de la tesis está dividido en 9 capítulos. El primer capítulo consiste de la parte introductoria y su finalidad es dar el marco protocolario de la tesis. El capítulo 2 contiene el

estado del arte que menciona los antecedentes y aspectos fundamentales de la teoría de agentes y describe las ideas básicas y algunos trabajos previos en el área de herramientas de desarrollo de agentes. El capítulo 3 está formado por las bases técnicas que dan soporte a la tesis. Se describen las características de la plataforma CAP, las especificaciones de FIPA en las que se apoya la implementación de agentes y el tema de la tecnología de componentes COM, DCOM y ActiveX.

El capítulo 4 comprende los aspectos concretos que conforman la parte de desarrollo de un conjunto de ideas y un modelo que sirve como punto de partida para la creación de agentes y Sistemas MultiAgente (SMA). En el capítulo 5 se explica la implementación de una plantilla de control de agente básico que se ha creado con el fin de implementar la funcionalidad básica que heredan todos los agentes que son creados bajo este esquema.

El capítulo 6 trata de la implementación de la herramienta CAP-AgentTool. Se explica detalladamente cada uno de los pasos que se deben seguir en el proceso para crear una clase de control de agente y algunos detalles técnicos importantes en la implementación de la herramienta. En el capítulo 7 se habla del tema de las clases utilitarias que fueron creadas para auxiliar y complementar la tarea de la programación de SMA que utilicen controles de agente creados con CAP-AgentTool.

El capítulo 8 se refiere al tema de las pruebas de la herramienta. Por medio de un prototipo se muestra el proceso que se debe seguir en la implementación de SMA. El último capítulo está compuesto por las conclusiones del proyecto, las contribuciones, resultados, las limitaciones identificadas de acuerdo con la implementación del software y el trabajo futuro. Esto es seguido por los apéndices que describen los detalles técnicos del diseño e implementación de CAP-AgentTool.

El presente documento de la tesis se acompaña por un CD ROM con el software de CAP-AgentTool desarrollado y los contenidos de los apéndices.

Abstract

The trends in software development indicate that the next technological innovation in the field should integrate networking organizations and multiple platforms for the applications. The developers should build systems unified for handling the information that they use through the web and advanced software technologies. Agent based computing is one of the technologies for the development of complex applications attracting attention at the moment because agents can be used to build managerial systems quickly and integrate them easily. The idea of having software agents is that they can carry out complex tasks amid the users that use them.

In spite of an impressive development in the last few years, it is required to solve several technological problems to transfer agent technology from the laboratory to the industrial practice. The main problems are the following (i) a lack of tools to facilitate the development of different types of agents, transforming the design process and development into an art; (ii) interoperability of agents with other (non-agent) distributed software; and (iii) the reusability of agents.

In this work, the development of software tools to facilitate the process of agents' creation that can communicate using the services offered by the objects of the Component Agent Platform (CAP) is described. For the design of such software tools, it is necessary and fundamental to develop a general framework that is focused on the collaborative type of agents. A new level of abstraction is defined: the cooperation at knowledge level among autonomous entities in flexible, uncertain and distributed environments.

The central idea of this research work is to create agents based on ActiveX controls within the context of the CAP platform, based on the COM and DCOM technology. An extra motivation to use the COM and ActiveX component technology is to develop an infrastructure to facilitate the implementation of applications based on agents in Microsoft Windows programming environments. To achieve the outlined objectives, the developed tools facilitate the programming in key aspects, such as the use of a language of agents' communication, a content language, the mechanisms for knowledge manipulation and of agents' internal control.

The CAP platform was developed under the specifications of FIPA (Foundation for Intelligent Physical Agents) like part of the research projects of the Agents Laboratory at the Computing Research Center of the National Polytechnic Institute (IPN). For this reason, the agents that register and work on the CAP, should also follow the FIPA specifications.

The document of the thesis is divided in nine chapters. The first chapter consists of the introductory part and its purpose is to give the protocolary framework of the thesis. Chapter two contains the state of the art, mentions the antecedents, the fundamental aspects of the theory of agents and describes the basic ideas and some previous works in the area of agent development tools. Chapter three is formed by the technical bases that support the thesis. The characteristics of the CAP platform are described, as well as some aspects of the FIPA

specifications related to the implementation of agents. Component technology is presented, making emphasis on COM, DCOM and ActiveX technologies.

In Chapter four, the conceptual model of a collaborative agent based on ActiveX technology is described. In Chapter five, the implementation of a template of a basic agent's control is explained. This template has been created with the purpose of implementing the basic functionality that all agents inherit.

Chapter six is about the implementation of the CAP-AgentTool. Each step of the process of creation of an agent control and some important technical implementation details of the tool are explained. In Chapter seven, implementation issues of the utilitarian classes that were created for auxiliary purposes to facilitate the task of the MAS programming using agent controls created with CAP-AgentTool are discussed.

The Chapter eight refers to the experiments carried out with the CAP-AgentTool. By means of a prototype, the SMA software development process is shown. The last Chapter is formed by the conclusions of the thesis, the contributions and results, the identified limitations of the software implementation, and future work. It is followed by the Appendixes describing technical details of the CAP-AgentTool design and implementation.

This thesis is accompanied by the CD ROM, which includes the software of CAP-AgentTool and the contents of the Appendixes.

Capítulo 1. Introducción

Resumen

Incluye la introducción de la tesis y está integrado de tal forma que permite conocer los objetivos del trabajo, tanto general como específicos. Además plantea la problemática que se está abordando y su justificación.

Objetivo del Capítulo

Plantear los objetivos de la tesis.

Establecer la problemática que se aborda en el contenido y justificar su estudio.

1.1 Objetivos

Objetivo General

Desarrollar las herramientas de software y mecanismos que faciliten la tarea de creación de agentes de software que corran sobre la plataforma CAP y que se ejecuten como controles ActiveX para utilizarlos en las herramientas de desarrollo de aplicaciones para Windows.

Objetivos Específicos:

1. Desarrollar un modelo conceptual de agentes colaborativos que utilicen CAP como soporte de comunicación.

2. Desarrollar una herramienta de software que permita diseñar y crear controles ActiveX para los agentes que trabajen sobre la plataforma CAP.

3. Extender la funcionalidad de la CAP para dar el soporte a la ejecución de agentes como controles ActiveX.

4. Implementar mecanismos de comunicación a través de un subconjunto del lenguaje de comunicación FIPA-ACL, el lenguaje de contenido FIPA-RDF y protocolos de interacción. Revisar las especificaciones de los *performatives* de FIPA-ACL para manejar las acciones colaborativas de agentes.

5. Diseñar e implementar una plantilla básica de agente ActiveX para que cada clase de control de agente nuevo herede el comportamiento y funcionalidad de agente mínimo necesario para funcionar en un ambiente de agentes.

6. Implementar clases de soporte para la comunicación de agentes codificando el contenido de los mensajes en XML.

1.2 Problemática

En la actualidad es ampliamente conocido que la interacción es probablemente la característica más importante del software complejo. Las arquitecturas de software que contienen muchos componentes interactivos dinámicamente, cada uno con su propio hilo de control y coordinándose bajo ciertos esquemas de interacción típicamente son más difíciles de construir que aquellas que simplemente se procesan en un hilo de ejecución. Los agentes ayudan a entender, diseñar e implementar estas aplicaciones.

La ingeniería de software orientado a agentes se está desarrollando actualmente como un nuevo paradigma para la creación de aplicaciones de agentes. Pero para que se vuelva un paradigma utilizado por la industria del software se tienen que desarrollar metodologías y herramientas robustas y fáciles de usar. La mayoría de las aplicaciones que se están construyendo con agentes son sistemas complejos, distribuidos e interactivos.

Una buena parte de la investigación en el área de agentes está enfocada a desarrollar modelos teóricos que permitan lograr que los agentes alcancen niveles operacionales en los cuales se plasme un comportamiento inteligente y autónomo combinados con modelos de comunicación sofisticados basados en conocimiento para lograr sus metas y objetivos. En el área de agentes de software se ha desarrollado una extensa y rica teoría, muchos conceptos nuevos, ideas y formalismos para representar el funcionamiento de software basado en agentes. Pero algo que está claro es que a la hora de llevarlos a la práctica por medio de implementaciones en entornos computacionales, lenguajes, herramientas de modelado y desarrollo de aplicaciones con agentes, se ha convertido en un problema difícil de resolver.

Aunque existen algunas herramientas de desarrollo de aplicaciones basadas en agentes que han significado avances en el sentido de reducir el abismo entre teoría y práctica de agentes, por lo general están orientadas hacia la plataforma de Java, por ejemplo Zeus, FIPA-OS y JADE. Existen además otros esfuerzos encaminados a la programación de software de agentes con lenguajes no Java, por ejemplo el lenguaje LALO (Gauvin, Marchal, Saldaña, 1997) y recientemente 3APL (Hindriks, de Boer, van der Hoek, Meyer, 1999), pero han tenido poco éxito. Cada una de estas herramientas y lenguajes están basadas en modelos de agentes definidos de forma particular y establecen ciertos conceptos bajo las cuales se pueden desarrollar SMA de diferentes tipos y con diferentes características. Queda claro que algunas herramientas son mejores que otras para ciertos tipos de aplicaciones y dominios, según las bases que las soportan.

Muchos investigadores de la teoría de agentes han planteado la necesidad de desarrollar herramientas y lenguajes para la creación de agentes y el desarrollo de SMA. Actualmente, esta labor es una de las necesidades más apremiantes en el área de agentes. Aunado a lo anterior, es también muy importante la necesidad de desarrollar las herramientas siguiendo especificaciones y estándares para buscar la interoperabilidad de SMA heterogéneos. En este sentido, FIPA es una organización que se ha preocupado por especificar y estandarizar los aspectos centrales de agentes y las plataformas que los soportan; hoy en día es el centro de las implementaciones de agentes y SMA.

De acuerdo con lo anterior, en el Laboratorio de Agentes del CIC se ha desarrollando la plataforma CAP, basada en las especificaciones de FIPA. Esta plataforma está desarrollada sobre el modelo de componentes COM y DCOM para proporcionar los servicios básicos. Este es un antecedente que se toma como base para esta tesis y los aspectos centrales que se abordan tienen que ver con la extensión de la infraestructura para el desarrollo de agentes y SMA y en particular con la necesidad de herramientas que faciliten la creación y programación de agentes como controles ActiveX, soporte a lenguajes de contenido, protocolos de interacción nativos en los agentes y nuevas plantillas de agentes a partir de controles ActiveX.

Para soportar los agentes, también es necesario definir un modelo de agentes colaborativos basados en la comunicación a través de un lenguaje de comunicación de agentes. Para fortalecer esta idea se establece que el dominio de aplicación natural de estos agentes está dado por aquellas aplicaciones y sistemas en donde se puedan establecer ambientes

colaborativos entre agentes que busquen en todo momento interactuar en su ambiente para llevar a cabo sus tareas a través de la compartición e intercambio de conocimiento, que es una característica fundamental para los agentes en FIPA. Los agentes ofrecen algunas de sus capacidades a los demás agentes publicándolas como servicios que son capaces de realizar, 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 componentes que sirve de base para la interacción entre agentes. Además, los agentes pueden compartir y utilizar los servicios de agentes de ontologías para asegurar el entendimiento correcto del contenido en la comunicación con los demás, en busca de lograr una autonomía cada vez mayor.

Por otra parte, otro punto de vista del problema que se trata de resolver nos plantea la necesidad de desarrollar aplicaciones basadas en Windows que utilicen CAP, en busca de lograr acercar el paradigma de agentes al mundo de la programación en los lenguajes de programación más populares como Visual Basic, Visual FoxPro, Visual C++, Delphi, y en general, a todos aquellos lenguajes que brinden soporte al uso de controles ActiveX.

En resumen, la finalidad que se persigue con la elaboración del presente trabajo de investigación es desarrollar herramientas de desarrollo de software para crear agentes que utilicen la tecnología de los controles ActiveX. Se pueden ver dos enfoques para este trabajo: desde el punto de vista del desarrollador de controles agente y desde el punto de vista del usuario de los controles agente. En el primero es necesario cuidar los detalles técnicos para el diseño e implementación de los controles agente. En el segundo, se considera que los controles agente que se creen podrán estar disponibles para los programadores que utilizan lenguajes y herramientas basadas en la plataforma computacional Windows y en general en cualquier ambiente de programación que acepte el uso de controles ActiveX; esto los habilitará para que sus aplicaciones usen la tecnología de agentes. Los agentes son controles fácilmente incrustables en las aplicaciones y tienen la característica de que su ciclo de vida se regula a través de la plataforma de agentes componentes CAP.

1.3 Justificación

La noción de agentes autónomos heterogéneos para resolver problemas es una metáfora poderosa para la ingeniería de sistemas de software interoperable y distribuido. Esta idea basada en agentes introduce un nuevo nivel de abstracción, el de la cooperación a nivel de conocimiento entre sistemas autónomos, que permite la interoperabilidad, escalabilidad y reconfigurabilidad entre los sistemas distribuidos. Sin embargo, hasta ahora, la promesa de la idea de agentes se ha desarrollado poco en la comunidad de ingeniería de software distribuido. Esto se debe a un buen número de factores, incluyendo la actual falta de estándares para la tecnología de agentes, pero primeramente debido a la complejidad inherente de la construcción de los sistemas de agentes colaborativos.

A pesar de un desarrollo impresionante en los últimos años, para que la tecnología de agentes se refleje del laboratorio a la práctica industrial, se requieren resolver varios problemas de ingeniería de software basado en agentes. Estos problemas radican en el

hecho de que (i) las herramientas para facilitar el desarrollo de diferentes tipos de agentes prácticamente no existen, convirtiendo el proceso de diseño y desarrollo en un arte; (ii) existen problemas de interoperabilidad de agentes con otro software distribuido y (iii) de reutilización de agentes.

Para facilitar la realización a gran escala de la idea de agentes colaborativos para la ingeniería de software distribuido es necesario desarrollar *frameworks*, metodologías y herramientas que soporten el rápido desarrollo de sistemas multiagente. Para este trabajo se tiene como antecedente inmediato el desarrollo de la plataforma CAP. Por ello, esta tesis se enfoca en el problema de desarrollar herramientas de programación de agentes y SMA que corran sobre la plataforma CAP. Se estudia el soporte que ofrece la tecnología de componentes COM y DCOM para construir los agentes como controles ActiveX y utilizar así los mecanismos de instanciación, comunicación y ejecución de esta tecnología, lo cual es necesario para permitir que los agentes interactúen con los componentes de la plataforma CAP.

Además, relacionado a lo anterior, tiene relevancia el hecho de que hoy en día las herramientas de programación de agentes más utilizadas están ligadas al lenguaje Java para programar la funcionalidad de los agentes y las aplicaciones que los utilizan. En parte, por esto se decidió desarrollar este trabajo, con la finalidad de establecer los medios necesarios para programar agentes en los lenguajes de programación para Windows, en base a los controles ActiveX.

Otro aspecto central de la tesis tiene que ver con el tipo de agentes y aplicaciones que se quiere desarrollar con ellos. Se trata de realizar agentes colaborativos para crear aplicaciones distribuidas y con agentes heterogéneos. La necesidad de colaboración entre agentes ocurre por un buen número de razones; sin embargo, la mayoría están ligadas al problema de la escasez de recursos (de cómputo, información, conocimiento, etc.). Los agentes individuales poseen diferentes recursos y capacidades; una solución a un problema dado puede estar más allá de las capacidades de cualquier agente, requiriendo que un buen número de agentes agrupen sus recursos y colaboren con otros para resolver el problema.

Dado que la colaboración entre los agentes se da en el nivel de conocimiento, esto hace que se requiera investigar mucho en varios aspectos sobre los agentes. No son pocas las necesidades de mecanismos para descubrir información a través del cual los agentes descubren la existencia, direcciones de red, capacidades y/o roles de otros agentes. También, son requeridos un lenguaje de comunicación independiente del agente, que los agentes usan para comunicarse con los otros. Y, una ontología que define conceptos del dominio de la aplicación que son comunicados entre los agentes.

Además, para la resolución efectiva y coherente, los agentes necesitan mecanismos de razonamiento acerca de ellos mismos y capacidades para la resolución de problemas de otros agentes y para coordinar sus actividades. En muchos ambientes dinámicos, los problemas son empeorados por los requerimientos adicionales para el comportamiento reactivo manejado por datos, que se integra con las actividades deliberativas manejadas por metas de los agentes.

1.4 Limitaciones previas

En esta sección se presenta una serie de comentarios y aclaraciones pertinentes con respecto a algunas limitaciones que se han identificado con respecto a la plataforma CAP. Esto es necesario mencionar debido a que repercute en el desarrollo de este trabajo. Además, permite entender mejor algunos de los resultados alcanzados y limitaciones propias.

Un primer punto a considerar se refiere al diseño de la plataforma CAP. En este sentido, la plataforma está formada por servidores COM para los servicios básicos especificados, y el acceso a estos, por parte de los agentes de las aplicaciones es centralizado. Este esquema cliente / servidor puede significar en un detrimento de la eficiencia en el manejo de la interacción con los agentes. Además, pone en duda la escalabilidad de los sistemas de agentes al centralizar la interacción de los agentes a través de la plataforma, sobre todo si consideramos que los objetos COM de ésta son objetos de instancia única dentro del espacio de proceso de la plataforma.

Además, también limita a las aplicaciones de agentes en tanto que la tolerancia a fallos no está implementada y en sistemas grandes, con un buen número de agentes, es algo que merece especial atención.

La plataforma CAP aun no es totalmente compatible con la especificación FIPA, ya que el desarrollo del *gateway* para IIOP (*Internet Inter-ORB Protocol*) no se ha terminado, por lo tanto, está limitada para soportar agentes y SMA en esta plataforma únicamente. Con relación a CAP también, aun no se ha probado su desempeño en plataformas distintas a Windows de Microsoft.

Es importante aclarar que los comentarios anteriores se refieren exclusivamente a la primera versión de la plataforma CAP, con la que se hicieron las pruebas del presente trabajo, y que actualmente se lleva acabo el análisis y diseño de su segunda versión que contempla la solución de las limitaciones mencionadas algunas de las cuales se detectaron en el proceso de desarrollo del presente trabajo.

CAPÍTULO 2. ESTADO DEL ARTE

Resumen

Este capítulo contiene el estado del arte relacionado con la tesis. Se menciona el concepto de agente y describe las ideas básicas de la computación basada en agentes y algunos trabajos previos en el área de herramientas de desarrollo de agentes.

Con respecto a las herramientas de desarrollo que aquí se mencionan, se trata de analizar los puntos centrales de la filosofía que se utiliza como fundamento para la realización de aplicaciones con agentes y algunos detalles de la implementación de los agentes. Las herramientas de desarrollo de agentes consideradas para esta parte son: Zeus, Fipa-OS, JADE, la arquitectura JAS, Paradigma de Jini y la arquitectura RETSINA.

Objetivo del capítulo

Establecer el estado del arte de la investigación con relación, sobre todo, a las herramientas más novedosas para el desarrollo de agentes que existen actualmente.

2.1 Definiciones básicas sobre agentes

2.1.1 Concepto de agente

Un agente es visto como otro tipo de abstracción en el software de más alto nivel que da una forma conveniente y poderosa para describir una entidad de software compleja. Un agente es definido en términos de su comportamiento.

Existe un mínimo de características comunes que tipifican a un agente de software. Un agente de software es autónomo; el agente es capaz de operar como un proceso independiente y realizar acciones sin intervención del usuario. También el agente es comunicativo; este se comunica con otros agentes de software o con otros procesos de software. El agente debe ser perceptivo, esto es, que debe ser capaz de percibir y responder a cambios en su ambiente.

Los agentes de software, como la gente, pueden poseer diferentes niveles de competencia al realizar una tarea particular; esto esta condicionado por sus capacidades internas, sus programas, es decir, sus especificaciones de comportamiento. Esto define el grado de inteligencia de los agentes.

Los agentes de software son apropiados para usarse en la implementación de ciertos tipos de aplicaciones; en otros dominios de problemas, tal vez otras tecnologías serán más apropiadas.

Una cosa que está muy clara actualmente en el campo de los agentes es que no se ha logrado un acuerdo acerca de la definición para la palabra agente. Esto es algo muy parecido a lo que pasa con los investigadores de Inteligencia Artificial y su definición de IA.

La respuesta de algunos investigadores de agentes a la falta de consenso en la definición ha sido inventar algunos sinónimos cuando apenas resuelve algunas cosas y sólo añade más confusión. De esta manera, se tienen sinónimos como *knowbots* (robots basados en el conocimiento), *softbots* (robots de software), *taskbots* (robots basados en tareas), *userbots*, robots, agentes personales, agentes autónomos y asistentes personales. Es muy importante usar la palabra más cuidadosamente y selectivamente.

El uso de agentes puede darse en aplicaciones que involucren computación distribuida o comunicación entre componentes. Otro tipo de aplicaciones para las que se puede utilizar esta tecnología son aquellas en las que se requiera razonar acerca de los mensajes u objetos recibidos a través de una red. Debido a que los agentes mantienen una descripción de su propio estado de procesamiento y de su entorno, ellos son ideales para aplicaciones de automatización y operación autónoma.

A continuación se define a un agente de software desde la perspectiva de un conjunto de características que puede tener (Ferber, 1999):

- Es un sistema computacional abierto (ensamble de aplicaciones, redes y sistemas heterogéneos)
- Puede comunicarse con otros agentes
- Es guiado por sus propios objetivos
- Posee recursos propios
- Tiene una representación parcial de otros agentes
- Posee servicios que puede ofrecer a otros agentes
- Su comportamiento atiende sus objetivos, tomando en cuenta sus recursos y habilidades disponibles y dependiendo de sus representaciones y las comunicaciones que recibe.

Los agentes son capaces de actuar, no sólo de razonar como en los sistemas de la IA clásica. El concepto de acción está basado en el hecho que los agentes llevan a cabo acciones que tienden a modificar el ambiente de los agentes, y por lo tanto, sus decisiones futuras. Ellos pueden también comunicarse con otros agentes para poder interactuar.

2.1.2 Tipología de agentes

Los agentes pueden ser clasificados de acuerdo a varios atributos que podrían exhibir. Una lista de tres atributos que se ha propuesto consiste en los siguientes: autonomía, aprendizaje y cooperación. Además se presenta una tipología de agentes definida en base a los atributos antes mencionados (Nwana & Ndumu, 1997).

La autonomía se refiere al principio de que los agentes puedan operar por sí mismos sin la necesidad de la intervención de un humano. La cooperación con otros agentes es muy importante: esta es la razón para tener múltiples agentes. Por último, los sistemas de agentes deben ser verdaderamente inteligentes, deberían poder aprender cuando ellos reaccionan y/o interactúan con su ambiente. Aunque es difícil tratar de definir el concepto de inteligencia, la habilidad para aprender es algo que se debe tener para decir que algo es inteligente. Sobre la base de las tres características mínimas anteriores, se pueden derivar cuatro tipos de agentes en la tipología: agentes colaborativos, agentes de aprendizaje colaborativos, agentes de interfaz y agentes inteligentes. Esto se puede ver en la Figura 1.

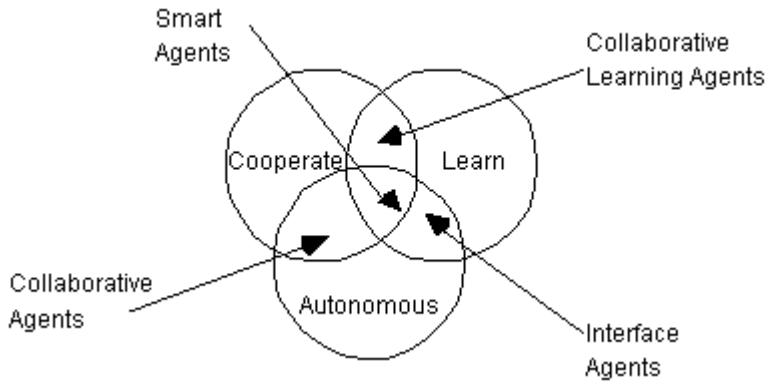

Figura 1 Una tipología de Agentes

Hay algunas aplicaciones que combinan de dos o más de estas categorías y se les conoce como sistemas de agentes heterogéneos.

2.1.3 Agentes Colaborativos

Los agentes colaborativos hacen énfasis en la autonomía y cooperación para realizar tareas. Ellos pueden aprender, pero este aspecto no es típicamente el mayor interés de su operación. Para tener una configuración de coordinación en agentes colaborativos, ellos tienen que negociar para alcanzar acuerdos mutuamente aceptables en algunos temas.

Las características clave generales de estos agentes incluyen autonomía, habilidad social, reactividad y pro actividad. Además, ellos deberían ser capaces de actuar racionalmente y autónomamente en ambientes multiagente restringidos en el tiempo y abiertos. Ellos tienden a ser estáticos y grandes. Normalmente, la mayoría de los agentes colaborativos implementados actualmente no tienen aprendizaje complejo, aunque pueden o no realizar aprendizaje paramétrico limitado o rutinario (Nwana, Wooldridge, 1996).

La idea de tener sistemas de agentes colaborativos es una especificación de la meta de la Inteligencia Artificial Distribuida (DAI, Distributed Artificial Intelligence). Esto es, crear sistemas que interconecten agentes colaborativos desarrollados separadamente, permitiendo el ensamble para funcionar más allá de las capacidades de cualquiera de sus miembros.

Existen varios temas que están abiertos a la investigación relacionados a este tipo de agentes. Entre otras cosas, es necesario investigar en ingeniería para la construcción de sistemas de agentes (Smith, 1996), coordinación entre agentes, estabilidad, escalabilidad y desempeño (Nwana, 1996).

2.1.4 Sistemas de agentes heterogéneos

Se refieren a un conjunto integrado de dos o más agentes que pertenecen a dos o más clases de agentes diferentes. Un sistema de agentes heterogéneos puede contener también agentes que combinen diferentes arquitecturas de comportamiento interno, por ejemplo pueden ser reactivos o deliberativos.

En (Genesereth & Ketchpel, 1994) se describe claramente la motivación para los sistemas de agentes heterogéneos. El argumento esencial es el de las limitaciones del mundo con una rica diversidad de productos de software que proveen un amplio rango de servicios para un igual amplio rango de dominios. Aunque estos programas trabajan aislados, hay una demanda en aumento para tenerlos a ellos interoperando, de una manera que den valor añadido de manera ensamblada, más que lo que ellos hacen de manera individual.

Una vez que los agentes están disponibles, hay dos posibles arquitecturas para escoger: una en la cual todos los agentes manejan su propia coordinación; otra en la que grupos de agentes pueden confiar en programas especiales para alcanzar la coordinación. La última idea, llamada también federada es preferida típicamente como se ve en la Figura 2.

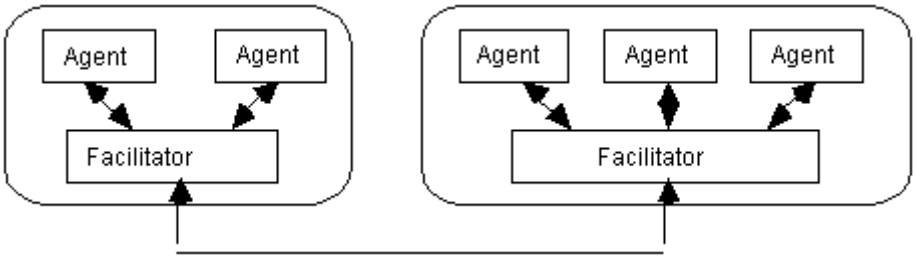

Figura 2 Un sistema federado de agentes

2.1.5 Sistemas Multi-agente

Un SMA está constituido por un conjunto de entidades autónomas llamadas agentes que coordinan sus habilidades para la resolución de problemas individuales o globales.

La investigación en SMA está relacionada al estudio, comportamiento y construcción de una colección de agentes autónomos preexistentes que interactúan con otros agentes y con el ambiente. Un SMA puede ser definido como una red débilmente acoplada de resolvedores de problemas que interactúan para resolver problemas que van más allá de las capacidades individuales o conocimiento de cada resolvedor. Estos resolvedores, llamados agentes, son autónomos y heterogéneos por naturaleza (Sycara, 1998).

Los agentes de software pueden ser más útiles cuando trabajan con otros agentes en la realización de tareas. Una colección de agentes que se comunican y cooperan entre sí es llamada una agencia. La idea basada en agentes permite al diseñador de sistemas implementar el sistema usando múltiples agentes, con cada agente especializado en un conjunto de tareas particulares o servicios. Por ejemplo, una aplicación de comercio electrónico puede tener agentes compradores, agentes vendedores, agentes de almacén, agentes de bases de datos, agentes de correo electrónico, etc. Todos estos agentes se necesitan comunicar con los demás y deben tener la capacidad de trabajar juntos para alcanzar un conjunto de metas comunes.

Una solución basada en agentes es útil y atractiva debido a que los diversos agentes usados en la solución inherentemente conocen cómo hacer muchas cosas. Por ejemplo, los agentes saben cómo comunicarse con los otros agentes. El desarrollador del sistema no tiene que diseñar protocolos de comunicación y formatos de mensajes. El agente da esta capacidad como parte del mecanismo básico de agente. Todo lo que los programadores deben hacer es especificar lo que los agentes van a hacer en determinadas situaciones.

2.2 Herramientas para programar agentes

Dado que una buena parte de este trabajo se enfoca hacia el desarrollo de una herramienta de software para facilitar la tarea de crear y ejecutar agentes, se presenta en este apartado un breve repaso de las principales herramientas que se utilizan actualmente para programar agentes.

Podemos ver que hay dos aspectos que se deben considerar: en primer lugar existe un conjunto de herramientas, que no son lenguajes propiamente, que están enfocadas en el desarrollo de agentes y SMA que corren sobre una plataforma compatible con la especificación de la arquitectura de FIPA, por ejemplo ZEUS, FIPA-OS, JADE y JAS. Pero por otro lado se tienen otras herramientas que no son compatibles con la especificación de FIPA, pero que es importante mencionar en este trabajo; por ejemplo, Paradigma de Jini y RETSINA.

El repaso se limita a mencionar y destacar las principales características que presenta cada una de las herramientas. No se trata de ninguna manera de establecer parámetros de comparación entre ellas ni tampoco compararlas con nuestra propuesta.

2.2.1 ZEUS

Se trata de un *toolkit* para la construcción de aplicaciones multiagente en donde los agentes se distinguen por ser colaborativos. Provee un ambiente integrado para el desarrollo rápido de sistemas multiagente. En ZEUS se definen ideas generales para el diseño de SMA y se soportan dichas ideas sobre un ambiente visual para la captura de la especificación de agentes por el usuario. A partir de la especificación se genera código fuente en el lenguaje Java para los agentes (Collis et al., 1998).

El *toolkit* de ZEUS consiste de un conjunto de componentes, escritos en el lenguaje Java, y puede ser dividido en tres grupos de componentes o bibliotecas: Una biblioteca de componentes de agentes, una herramienta de construcción de agentes y un grupo de agentes utilitarios que comprende al servidor de nombres, el facilitador y el visualizador de agentes. Esto se ve en la figura 3.

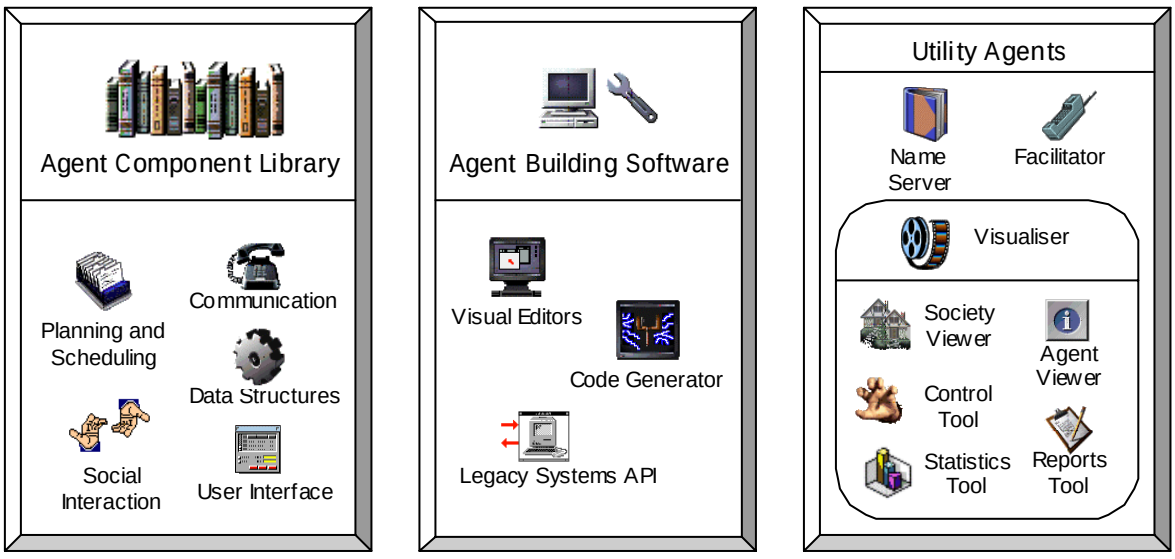

Figura 3 Componentes de ZEUS

La biblioteca de componentes de agentes es una colección de clases que forman los bloques de construcción de agentes individuales. Estas clases en conjunto implementan la funcionalidad independiente de la aplicación que se requiere para los agentes colaborativos.

Para fines de comunicación, esta biblioteca de componentes ofrece un lenguaje de comunicación entre agentes basado en *performatives*, un sistema de paso de mensajes asíncrono y basado en *sockets*, un editor para describir ontologías específicas del dominio y por último un lenguaje de representación de conocimiento basado en *frames* para representar conceptos del dominio (Collis, 1999).

Para razonamiento y coordinación entre agentes la biblioteca de componentes de agentes considera varios aspectos:

- Un sistema de planificación y calendarización de propósito general para dominios de aplicación orientados a tareas normalmente y para la resolución de problemas cooperativa, característica muy importante de estos dominios de aplicación.
- Un motor de coordinación que controla el comportamiento social de un agente.
- Una biblioteca de protocolos de coordinación como el protocolo de red de contratos y varios protocolos de subastas.
- Mecanismos de representación de conocimiento y bases de datos para almacenar los recursos y competencias de un agente.

El principio subyacente del *toolkit* de ZEUS es que los agentes específicos del dominio puedan ser construidos a partir de la configuración de un agente ZEUS genérico y poniéndole la funcionalidad necesaria de la aplicación (Fonseca, Griss & Letsinger, 2001). Para facilitar el desarrollo rápido se ofrece una idea de desarrollo de agentes de alto nivel que oculta la complejidad de la biblioteca de componentes para el desarrollador de agentes. Esta idea tiene dos aspectos claves:

- Una metodología para la creación que guía al desarrollador a través del análisis y diseño del sistema.
- Un ambiente de desarrollo visual que soporta la metodología de creación.

El generador de agentes es un grupo de editores integrado que han sido diseñados para permitir a los usuarios crear agentes interactivamente y especificar sus atributos visualmente; en la figura 4 se puede ver la interfaz del generador de agentes. Los editores que se incluyen son:

- Un editor de ontologías para definir los conceptos de una ontología del dominio.
- Un editor de hechos para describir instancias específicas de hechos y variables.
- El editor de definición de agentes para describir a los agentes lógicamente. Esto incluye especificar las tareas de los agentes, recursos iniciales y las dimensiones de su plan diario.
- Un editor de descripción de tareas para especificar los atributos de tareas.
- El editor de la organización tiene como finalidad definir las relaciones organizacionales entre agentes, y las creencias de los agentes acerca de las habilidades de otros agentes.

- El editor de coordinación sirve para seleccionar el conjunto de protocolos de coordinación con los que cada agente estará equipado.

Para generar el código para una aplicación específica, el Generador hereda código desde la biblioteca de componentes de agente, y la integra con los datos de los distintos editores visuales. El programa resultante puede ser compilado y ejecutado normalmente.

Figura 4 Generador de agentes ZEUS

Es una práctica estándar en la sociedad de agentes distribuidos el tener una infraestructura de servicios utilitarios. El grupo de agentes utilitarios de ZEUS consiste de un servidor de nombres y un facilitador que permite el descubrimiento de información, y un agente visualizador para observar o depurar sociedades de agentes ZEUS. Los tres agentes utilitarios fueron construidos usando los componentes básicos de la biblioteca de componentes de agentes, y son simplificaciones del agente genérico.

Desde el punto de vista de la estructura interna, las aplicaciones de agentes ZEUS están formadas por los siguientes conceptos: agentes, metas, tareas y hechos.

Concepto de Agente

A su vez, para definir lo que es un agente en ZEUS se han establecido algunos conceptos. La figura 5 resume el concepto de agente de acuerdo con las definiciones siguientes:

- Definición del agente: Es usada para describir las habilidades de los agentes (por ejemplo que puede ser multitarea, que puede llevar a cabo planes) y conocimiento.
- Organización: Un agente le puede solicitar a otro agente que realice alguna tarea o puede colaborar con otros para alcanzar metas comunes.

- Sensor: El agente detecta cambios en su ambiente a través de sus sensores, y actualiza su conocimiento de acuerdo a ello.
- Efector: Un agente puede cambiar su ambiente externo deliberadamente, típicamente como un efecto lateral de la ejecución de alguna tarea.
- Ciclo de vida: Define los estados válidos de un agente. Inicialmente un agente es creado. Cuando está vivo, un agente puede ser suspendido (hasta ser removido temporalmente de su ambiente), subsecuentemente reactivado. Finalmente, un agente puede ser destruido, para removerlo permanentemente.
- Protocolo de interacción: El agente puede poseer una variedad de protocolos de interacción, para facilitar el diálogo con otros agentes.

Concepto de meta

Una meta será creada por un agente con la intención de cumplir con un compromiso hecho con otro agente y se puede satisfacer realizando una acción en particular. Una meta se define a través de cuatro conceptos: restricciones, tipo, motivación y ciclo de vida.

Hechos

Un hecho representa cualquier cosa que el agente cree que es verdadero, acerca de sí mismo o de su ambiente externo. Un hecho tiene un grupo de características: ciclo de vida, visibilidad, restricciones, dueño, atributo y tipo.

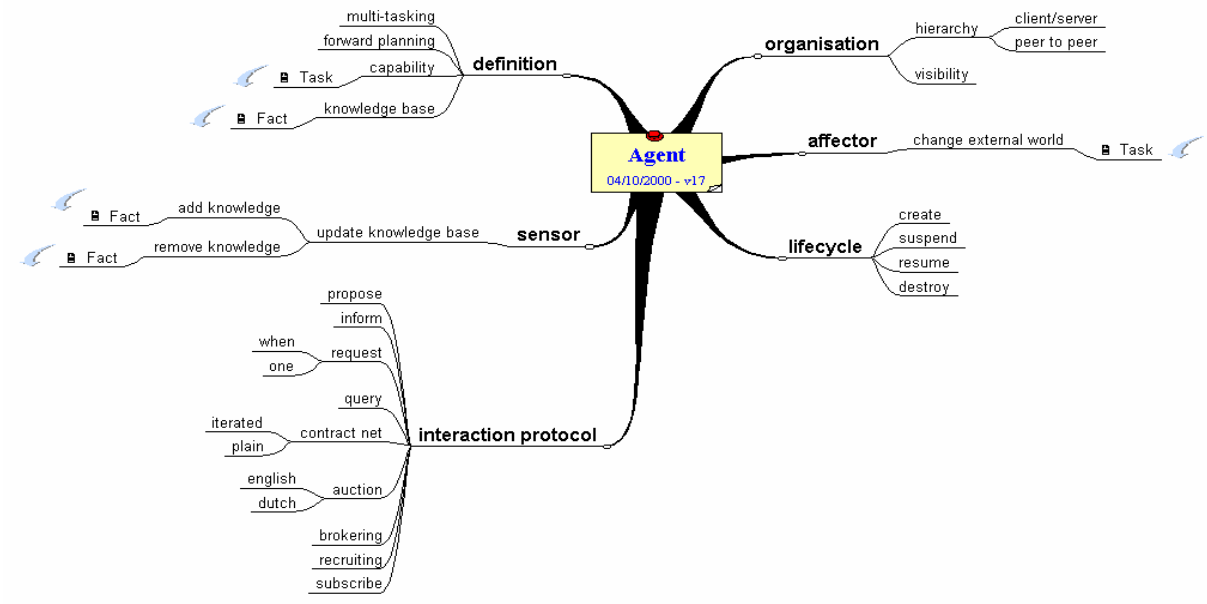

Figura 5 Mapa de conceptos de agente

Tareas

La tarea que un agente tiene que realizar para alcanzar una meta puede tener algunas precondiciones. Tales precondiciones pueden estar basadas en conocimiento en la forma de

hechos obtenidos de la base de datos. En algunos casos puede ser necesario actualizar la base de conocimiento antes de iniciar la tarea. Una tarea puede también producir hechos al ser terminada.

En algunos entornos puede ser necesario descomponer una tarea en subtareas autónomas. Cada tarea tiene atributos (como costo y duración) que le dan al agente la información necesaria para propósitos de planificar y tomar decisiones.

2.2.2 FIPA-OS

FIPA-OS es un *toolkit* basado en componentes que permite el desarrollo rápido de agentes compatibles con FIPA. Es una implementación de código fuente abierto de los elementos obligatorios contenidos en la especificación de FIPA para la interoperabilidad de agentes. FIPA-OS también provee una arquitectura basada en componentes para permitir el desarrollo de agentes específicos del dominio que pueden utilizar los servicios de los agentes de la plataforma FIPA. FIPA-OS es un marco general experimental para agentes, que es producto de la investigación en los Laboratorios Harlow de Nortel Networks en el Reino Unido y está implementado en Java (Poslad, Buckle & Hadingham, 2000).

El objetivo principal de FIPA-OS es reducir las barreras actuales en la adopción de la tecnología de FIPA, a través de complementar los documentos de la especificación técnica con código fuente abierto.

Características

FIPA-OS está diseñado para soportar los estándares de agentes FIPA. El modelo de referencia de FIPA define los componentes centrales del paquete de distribución de FIPA-OS: El Facilitador de Directorios (DF), El Servicio de Administración de Agentes (AMS), el Canal de Comunicación de Agentes (ACC) y el Transporte de Mensajes Interno de la Plataforma (IPMT) (Poslad, Buckle & Hadingham, 2001).

Además de los componentes obligatorios del modelo de referencia de FIPA, FIPA-OS incluye soporte para:

- Diferentes tipos de *Shells* de Agentes para producir agentes que se puedan comunicar con los demás, usando para ello las facilidades de FIPA-OS.
- Soporte para la comunicación de agentes multicapa.
- Administración de conversaciones y mensajes.
- Configuración dinámica de la plataforma para soportar múltiples IPMT, múltiples tipos de persistencia (permitiendo integración con software legado persistente) y múltiples esquemas de codificación.
- Patrones de diseño de software e interfaces abstractas.
- Herramientas de visualización y diagnóstico.

La Figura 6 muestra los componentes disponibles y su relación con los demás (El calendario de planes no está actualmente disponible).

Los componentes fábrica de base de datos, la fábrica de *parsers* y CCL (Choice Constraint Language) son opcionales y no tienen una relación explícita con los demás componentes en el *toolkit*. El calendario de planes generalmente no tiene la habilidad para interactuar con todos los componentes de un agente.

El *Shell* de agentes

La clase FIPAOSAgent provee un *shell* para la implementación de agentes simplemente extendiendo de esta clase.

Administrador de tareas

Ofrece la habilidad de dividir la funcionalidad de un agente en unidades disjuntas de trabajo más pequeñas conocidas como tareas. El objetivo es que esas tareas son piezas de código auto contenidas que llevan a cabo alguna tarea y (opcionalmente) retornan un resultado, teniendo la habilidad de enviar y recibir mensajes, y tienen poco o preferiblemente ninguna dependencia del agente en el que ellas se están ejecutando.

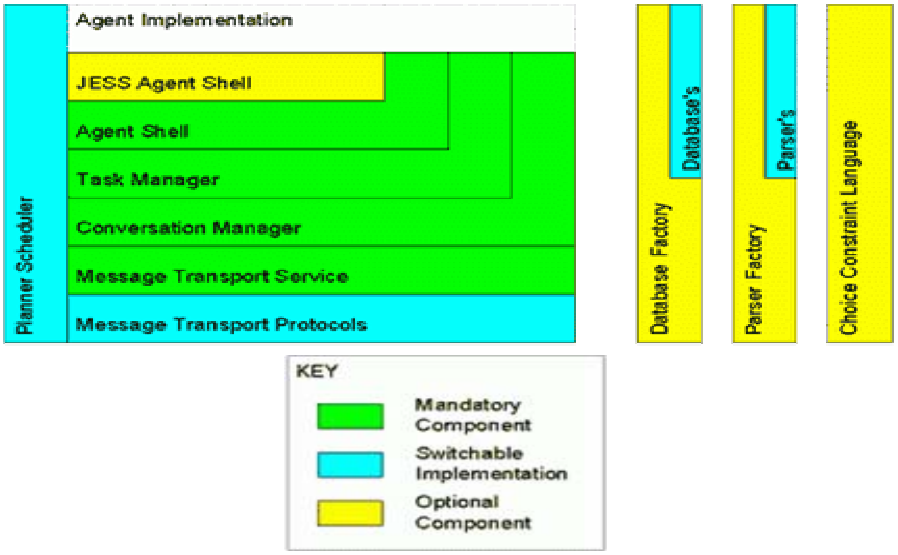

Figura 6 Arquitectura de FIPA-OS

Administrador de Conversaciones

Permite la habilidad de llevar un control del estado de la conversación en el nivel de *performative*, así como mecanismos para agrupación de mensajes de la misma conversación. Si una conversación es especificada para seguir en un protocolo particular, el administrador de conversaciones asegurará que el protocolo está siendo seguido por ambos participantes de la conversación.

MTS (*Message Transport Service*, Servicio de Transporte de Mensajes)

Permite la habilidad de enviar y recibir mensajes para y desde un agente implementado. Se ofrecen varios MTP (Protocolos de transporte de mensajes) dentro de FIPA-OS, para permitir al MTS que se comunique usando varios protocolos.

Shell de Agentes JESS (*Java Expert System Shell*)

Esta extensión del *Shell* de agentes estándar permite soporte para escribir agentes FIPA-OS que puedan aprovechar las ventajas del sistema JESS.

Comunicación ACL *multi-tier*

La comprensión de un mensaje ACL (*Agent Communication Language*) requiere procesar el mensaje con relación a su posición temporal dentro de una secuencia de interacción en particular. Esto involucra la comprensión del tipo de comunicación llamado acto comunicativo, comprensión de la estructura del contenido y finalmente, la comprensión de la semántica de la solicitud (FIPA-OS, 2001).

Como la comunicación ACL es tan rica, ésta es representada usando cuatro subcapas de componentes: conversación, mensaje ACL, contenido (sintaxis) y ontología (semántica del contenido). No todos estos componentes necesitan ser usados por cada agente. Esta flexibilidad es necesaria ya que en un mundo heterogéneo, diferentes agentes pueden codificar y transportar la información de manera diferente. FIPA-OS soporta cadenas ASCII y XML para los mensajes ACL usando el apropiado decodificador y *parser*. Para el contenido existen varios esquemas de codificación: FIPA-SL0, FIPA-SL1 y FIPA-RDF para codificar el contenido en XML.

La actual versión de la plataforma soporta IIOP como el protocolo de transporte base. Este puede ser usado como el protocolo IPMT y para soportar la comunicación por el agente ACC. Hay diferentes APIs de transporte para productos CORBA (*Common Object Request Broker Architecture*).

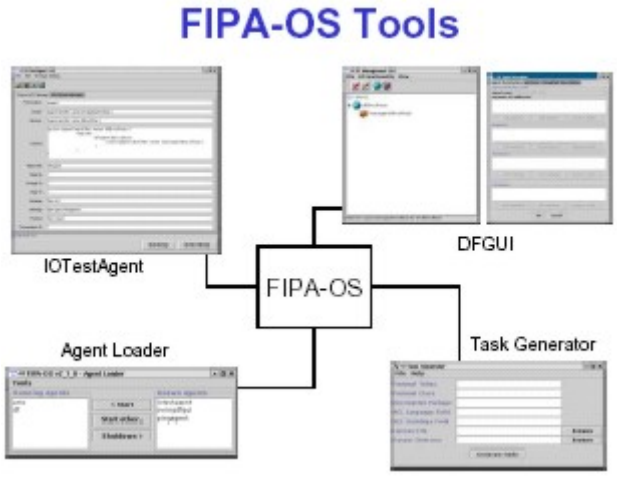

Figura 7 Herramientas de FIPA-OS

En la figura 7 se muestran las herramientas que FIPA-OS ofrece en su *toolkit* para programación, ejecución y depuración de agentes.

2.2.3 JADE

JADE (*Java Agent Development Framework*, Marco de Desarrollo de Agentes Java) es un marco general de software desarrollado completamente en Java para la construcción de sistemas multiagente y aplicaciones conforme los estándares de FIPA para agentes inteligentes. Este simplifica la implementación de sistemas multiagente a través de un *middle-ware* que afirma cumplir con las especificaciones de FIPA y que soporta la depuración y despliegue a través de un conjunto de herramientas (Bellifemine, Poggi & Rimaza, 1999).

La arquitectura de comunicación ofrece un sistema de mensajería eficiente y flexible, en donde JADE crea y administra una cola de mensajes ACL que llegan, privados a cada agente; los agentes pueden acceder a sus mensajes vía una combinación de modos: bloqueo, por elección, por tiempo y basado en igualación de patrones. El modelo de comunicación de FIPA completo ha sido implementado y sus componentes han sido distinguidos claramente e integrados completamente: protocolos de interacción, envolturas(*envelope*), ACL, lenguajes de contenido, esquemas de codificación, ontologías y protocolos de transporte.

El mecanismo de transporte, en particular, es como un camaleón ya que se adapta a cada situación, escogiendo transparentemente el mejor protocolo disponible. RMI de Java, notificación de eventos e IIOP son usados actualmente.

La mayoría de los protocolos de interacción definidos por FIPA están disponibles ya y pueden ser instanciados después de definir el comportamiento dependiente de la aplicación de cada estado del protocolo. El lenguaje de contenido SL (*Semantic Language*) y la ontología para la administración de agentes ya han sido implementados, así como también el soporte a lenguajes de contenido definidos por el usuario y ontologías que pueden ser implementados, registrados con agentes, y usados automáticamente por JADE.

JADE también ha sido integrado con JESS, un *shell* de Java, con la finalidad de explotar las capacidades de razonamiento.

JADE incluye dos productos: la plataforma de agentes y un paquete para desarrollar agentes JADE. Esta herramienta le ofrece a los programadores de aplicaciones piezas de funcionalidad ya listas para usarse e interfaces abstractas para personalizar tareas dependientes de la aplicación (Bellifemine et al., 2000).

Características

La siguiente es la lista de las características que JADE ofrece a los programadores de agentes:

- Plataforma de agentes distribuida. La plataforma de agentes puede ser dividida entre varias computadoras (siempre que puedan ser conectadas con RMI). Los agentes son implementados como hilos de java y viven dentro de contenedores de agentes que dan el soporte de ejecución a los agentes.
- Interfaz gráfica de usuario para administrar varios agentes y contenedores de agentes desde un *host* remoto.
- Herramientas de depuración para ayudar en el desarrollo de aplicaciones multiagente basadas en JADE.
- Movilidad de agentes inter-plataforma, incluyendo el estado y código del agente.
- Soporte para la ejecución de actividades de agentes múltiples, paralelas y concurrentes a través del modelo de comportamiento. JADE calendariza el comportamiento de los agentes de una manera no preferente.
- La plataforma cumple con las especificaciones de FIPA, incluye AMS, DF, y el ACC. Los tres componentes son activados automáticamente en el arranque de la plataforma.
- Muchos DFs compatibles con FIPA pueden ser iniciados en tiempo de ejecución con la finalidad de implementar aplicaciones de múltiples dominios, en donde un dominio es un conjunto lógico de agentes, cuyos servicios son informados a través de un facilitador común.
- Un mecanismo eficiente para el transporte de mensajes ACL dentro de la misma plataforma de agentes. De hecho, los mensajes son transferidos codificados como objetos de Java, como *strings*, con la finalidad de evitar los procedimientos de ordenamiento y desordenamiento (*marshalling/unmarshalling*).
- Una biblioteca de protocolos de interacción FIPA.
- Registro y desregistro automático de agentes con el AMS.
- Servicio de nombres compatible con FIPA.
- Soporte para lenguajes de contenido y ontologías definidos por el usuario.
- Interfaz dentro de proceso para permitir a las aplicaciones externas alimentar agentes autónomos.

Herramientas

JADE viene con algunas herramientas que simplifican el desarrollo de aplicaciones y la administración de la plataforma, y son las siguientes:

- Agente de Administración Remota, (RMA, *Remote Management Agent*). Actúa como una consola gráfica para administrar y controlar la plataforma.
- El Agente *dummy*, es una herramienta de monitoreo y depuración, hecho de una interfaz gráfica de usuario y un agente JADE subyacente en la interfaz. Usándolo es posible componer mensajes ACL y enviarlos a otros agentes y también desplegar la lista de todos los mensajes ACL enviados o recibidos, completados con información de hora para permitir grabar la conversación entre agentes y ensayar.
- El *Sniffer* es un agente que puede interceptar mensajes ACL mientras están en camino, y los despliega gráficamente usando una notación similar a los diagramas de secuencia de UML. Es útil para depurar las sociedades de agentes observando la manera en que ellos intercambian mensajes.
- El agente *Introspector* es una herramienta muy útil que permite monitorear el ciclo de vida de un agente y sus mensajes ACL intercambiados.

- El agente *SocketProxyAgent* es un agente simple, actúa como una compuerta bidireccional entre una plataforma JADE y una conexión TCP/IP ordinaria. Los mensajes ACL, viajando sobre un servicio de transporte propietario de JADE son convertidos a cadenas ASCII simples y enviados sobre una conexión por *socket* y viceversa.
- Hay una interfaz gráfica de usuario que es usada por el Facilitador de Directorios *default* de JADE y que puede también ser usado por cualquier otro DF que el usuario necesite. Esta interfaz permite, en una forma simple e intuitiva, controlar la base de conocimiento de un DF para federar un DF con otros DFs.

Creación de agentes en JADE

La clase *Agent* representa la clase base común para los agentes definidos por el usuario. Desde el punto de vista del programador, un agente JADE es simplemente una instancia de una clase de Java definida por el usuario que extiende de la clase *Agent*. Esto implica la herencia de características para cumplir interacciones básicas con la plataforma de agentes (registro, configuración, conexión remota, etc.) y un conjunto básico de métodos que pueden ser llamados para implementar el comportamiento particular del agente (por ejemplo enviar / recibir mensajes, usar protocolos de interacción estándar, registrarse con varios dominios, etc.).

El modelo computacional de un agente es multitarea, donde las tareas (o comportamientos) son ejecutadas concurrentemente. Cada funcionalidad o servicio dado por un agente debería ser implementado como uno o más comportamientos. Un calendario interno de la clase *Agent* y oculto al programador, automáticamente maneja los comportamientos.

JADE controla el nacimiento de un nuevo agente de acuerdo con varios pasos: el constructor del agente es ejecutado, este se registra con el AMS, pone su estado en activo y finalmente ejecuta el método *setup*(); este método es el punto por donde cualquier agente definido en la aplicación inicia su actividad. El programador tiene que implementar este método para inicializar el agente. Por lo general este método es usado para varias cosas: si es necesario, modificar el registro del agente en el AMS, registrarse con uno o más dominios, y añadir tareas a la cola de tareas listas usando el método *addBehaviour*().

2.2.4 JAS

2.2.4.1 Introducción

Se ha determinado que es útil y deseable tener un API de Java que soporte el despliegue e interoperabilidad de agentes comunicativos en un contexto de negocios.

El proyecto de Servicios de Agentes Java (JAS, *Java Agent Services*) es una iniciativa para definir una especificación estándar de la industria y un API para desarrollo de arquitecturas de redes de agentes y servicios (JAS,2002).

No hay duda que la tecnología más penetrante en uso actualmente para crear sistemas de agentes FIPA es Java. Sin embargo, hasta la fecha, no existe un API de Java estándar para

crearlos, una omisión que debe ser corregida si es que los agentes están por penetrar el mundo de las aplicaciones de negocios. La iniciativa JAS intenta responder a este requerimiento desarrollando un API, en el espacio de nombres javax.agent, que instancia las características de la arquitectura abstracta de FIPA.

Por lo tanto, en Septiembre de 2000 fue sacada una propuesta por la Comunidad de Procesos de Java (JCP, *Java Community Process*), que describe el API de JAS. El JCP fue creado por Sun Microsystems como un medio para involucrar a la comunidad de usuarios de Java en el diseño y creación de nuevas extensiones para el núcleo del lenguaje.

Consecuentemente, el objetivo del JAS es definir un conjunto de objetos y servicios que soporten el despliegue y operación de agentes a través de tres entidades primarias: clases de Java para describir los distintos componentes del mensaje, clases de Java para definir nombres y descripciones de agentes y, en tercer lugar, interfaces Java que corresponden a los agentes de servicios para mensajería, directorio y nombres. La intención es que las interfaces de servicios pueden ser implementadas en términos de un buen número de diferentes tecnologías.

Las implementaciones construidas usando el API de JAS son fuertemente ligadas a espacios de aplicaciones *business to business*, permitiendo actividades como comercio electrónico, administración de procesos de negocios e integración de socios comerciales.

El API de JAS le da al desarrollador de sistemas de agentes una herramienta flexible y poderosa para crear ambientes que soporten agentes, manejando la interoperabilidad y dando una arquitectura abierta y, de alguna forma, dando soporte a la semántica del significado de la comunicación necesaria para hacer la mayoría de los servicios en Internet.

2.2.4.2 Ámbito de JAS

El enfoque de este proyecto es crear el API de Java que facilite el intercambio de mensajes entre redes de agentes y servicios. Debido a que los agentes tienen una naturaleza intrínsecamente flexible, esto puede ser cumplido a través de muchas formas de transporte de mensajes diferentes, servicios de directorios, lenguajes de comunicación y servicios adicionales de terceros. Los siguientes temas definen mejor el ámbito de JAS:

- Soporte para representar lenguajes de comunicación entre agentes.
- Soporte para representar lenguajes de contenido.
- Soporte para codificar traducción entre lenguajes.
- Soporte para interoperabilidad en el transporte de mensajes.
- Soporte para varios servicios de directorios de agentes.
- Soporte para varios servicios de nombres de agentes.

Usando JAS es posible crear sistemas de agentes que implementen todas estas áreas, o algún subconjunto suficientemente rico para asegurar un nivel adecuado de interoperabilidad. La primer versión del API de JAS también incluye un conjunto de interfaces para construir mensajes basados en una Representación de Contenido Abstracta (ACR, *Abstract Content Representation*), pero no es un elemento obligatorio.

JAS no soporta explícitamente la formación o generación de agentes o servicios. El enfoque particular es proveer a los agentes con una infraestructura sobre la cual se comuniquen. El diseño de agentes es responsabilidad del dueño y desarrollador.

Requerimientos para la implementación del API de JAS

Esta lista de requerimientos establece el conjunto de elementos que JAS le da al implementador de esta especificación.

- Liberar una infraestructura independiente de la tecnología para soportar servicios requeridos para la interoperabilidad de *end-to-end* entre agentes.
- Soportar el intercambio de datos semánticamente ricos entre agentes y servicios.
- Proveer el proceso de arranque de los servicios de la plataforma para los agentes.
- Dar interfaces para un conjunto de fábricas que enumerarán y liberarán puntos finales hacia el conjunto mínimo definido de servicios de la plataforma.
- Dar interfaces para los componentes de primera clase de transporte de mensajes.
- Proporcionar una interfaz de acceso común a los servicios de nombres que liberan identificadores únicos globales para los nombres de agentes.
- Una interfaz de acceso común a los servicios de directorios de agentes que permita a los agentes y servicios su registro, desregistro, modificación y búsqueda.
- Dar una interfaz de acceso común a los servicios de transporte arbitrarios que permiten el envío de mensajes y capacidades de recepción de mensajes. La recepción de mensajes puede estar basada en mecanismos de elección, de respuesta a petición, o ambos.

2.2.5 Paradigma: *Framework* de Jini para sistemas basados en agentes

Aquí se presenta Paradigma, un *framework* basado en Jini para el desarrollo de agentes. Este Paradigma sirve como punto de referencia para implementar un conjunto de ideas sobre agentes y con el objetivo de explicar cómo un sistema de agentes puede ser integrado con Jini. Este ha sido desarrollado como un sistema de “prueba de conceptos” basado en un *framework* teórico muy fuerte esperando proveer una descripción genérica de los agentes implementables directamente.

Una idea clave es que todos los agentes usan su conocimiento especializado acerca del mundo para realizar algunas tareas sobre su medio y ellos tienen un grado de libertad al realizar sus tareas.

Dos características principales, actuar por otros y autonomía, son usadas como la base para definir a los agentes computacionalmente y ayudan a derivar otros atributos que éstos agrupan.

2.2.5.1 Jini como base de Paradigma

Jini proporciona un conjunto de elementos que pueden ser utilizados como base para una infraestructura o plataforma de desarrollo y despliegue de sistemas basados en agentes. A continuación se mencionan.

- Descubrimiento de agentes. Cuando alguien está construyendo sistemas multiagente la primer preocupación es cómo los agentes pueden descubrir a otros. En este sentido, Jini es la plataforma ideal al contar con el mecanismo de *lookup* para descubrir dinámicamente entidades en la red y para manipularlas posteriormente.
- Administración de agentes. Los mecanismos de *leasing* de Jini dan una solución a este problema, permitiendo registrar entidades para ser descargadas desde los servicios de *lookup*. Además, las utilidades administrativas de Jini permiten monitorear algunos servicios Jini.
- Identificación y descripción de agentes. Provee la especificación del esquema de atributos Jini, que permite a los agentes o a algún servicio, informar sus atributos y capacidades a quien esté interesado y sea capaz de acceder al servicio de *lookup*.
- Soporte de transacciones. Una de las potencias de los sistemas multiagente es la habilidad de que un número de resolvedores de problemas juntos pueden resolver problemas que van más allá de sus capacidades individuales. Para ello se necesita asegurar la ejecución confiable en la presencia de concurrencia y presencia de fallos. Jini ofrece la Especificación de Transacciones Jini, incluso Sun implementa un administrador de transacciones en *mahalo*.
- Comunicación de agentes. Java es sin lugar a dudas el lenguaje más relacionado con la red disponible. Como resultado de eso, existen varios métodos disponibles para la comunicación de agentes, por ejemplo *sockets* y RMI.
- Compartir información de agentes. Aun que los *JavaSpaces* no son parte de Jini, este es un servicio que tiene mucho que ofrecer a los sistemas multiagente al proveer un espacio distribuido en el que se puede compartir información de manera distribuida, y aun intercambiar código y capacidades.

El ambiente para la implementación de agentes Paradigma da una forma de describir agentes a través de sus atributos y capacidades así como mecanismos para registrarlos con el servicio de *lookup*. Este permite la interacción de agentes autónomos con entidades más simples a través del uso de la noción de agente servidor, concebido como una entidad simple actuando con la finalidad de alcanzar una meta para otro agente.

2.2.5.2 Características de Paradigma

El marco general tiene como finalidad capturar el ámbito completo de entidades posibles en un dominio. Se propone un mundo jerárquico de cuatro capas que se construye desde simples objetos hasta agentes autónomos, haciendo incrementalmente más complejas a las entidades (Sing, 2000).

Cada cosa en el mundo es una entidad con atributos asociados. Los atributos son características descriptivas del mundo como puede ser el tamaño o el color de un carro. Los objetos son entidades con ciertas capacidades o acciones que tienen un efecto en el estado del ambiente.

Los agentes son objetos con un conjunto de metas, donde las metas se definen como un conjunto de valores de atributos deseables. Finalmente, los agentes autónomos son agentes que son capaces de generar sus propias metas a través de un conjunto de motivaciones que los conducen a ellas. Las motivaciones pueden ser pensadas como preferencias o deseos de

un agente autónomo que lo guían a producir metas para satisfacer sus deseos. En la figura 8 se resumen las características de las entidades en Paradigma de Jini.

Característica	Entidades	Objetos	Agentes	Agentes autónomos
Atributos				
Capacidades				
Metas				
Motivaciones				

Figura 8 Características de los agentes en Paradigma de Jini

Atributos

Los atributos son características descriptivas del ambiente. El conjunto de todos los atributos del ambiente podría dar una descripción de ese ambiente. Los tipos de atributos que existen en el mundo dependen de las entidades de ese mundo.

En Paradigma, los atributos son definidos por su tipo, su valor y si ellos son estáticos o variables. Los atributos son descritos en un archivo XML, el cual es leído y procesado produciendo un objeto DOM (Modelo de Objeto Documento, *Document Object Model*). Un DTD (Definición de Tipo de Documento, *Document Type Definition*) describe cómo estos archivos XML deberían ser estructurados. En Paradigma se usa el paquete Sun JAXP para manejar archivos XML.

Capacidades

A través de las capacidades las entidades actúan en el ambiente, cambiando o sensando atributos del ambiente. Igual que los atributos, estas son descritas en archivos XML y las fábricas de capacidades se encargan de instanciar los objetos capacidad.

El método más importante que los objetos capacidad necesitan implementar es el método *perform*(). Esta es una idea simple pero que ofrece un comportamiento interesante.

Los atributos de una capacidad son: una ruta para el archivo, su nombre de archivo totalmente calificado, el nombre de la capacidad, descripción y tipo. Los primeros dos atributos permiten la carga de la capacidad en la máquina virtual de Java y dan algo así como el *codebase*. Los últimos tres son usados para describir la capacidad.

2.2.5.3 Los agentes en Paradigma

Hay muchas formas para describir agentes. Van en el rango de las estructuras de control más básicas hasta complicadas entidades de razonamiento que usan técnicas de Inteligencia Artificial. Pero para definir un marco general para agentes autónomos es necesario encontrar lo que permitiría una exacta implementación del comportamiento tan simple o complicado como se espera.

En ese sentido, los agentes tienen metas adscritas, o en el caso de los agentes autónomos, generan metas por sí solos. Las metas son alcanzadas a través de la implementación de planes que se llaman sobre la ejecución de las capacidades del agente. Metas y Planes son dados a los agentes a través de archivos XML.

Una meta tiene un nombre y un atributo. Un ejemplo de meta es dado a continuación:

```
<Definition>
        <Name>Ajustar temperatura</Name>
        <Atribute>
                <Type>Temperatura</Type>
                <Value>6</Value>
        </Atribute>
</Definition>
```

Los planes tienen cuatro partes: nombre, bajo qué condiciones son invocados, qué necesita ser cierto para que el plan sea aplicable, y las acciones que se necesitan realizar.

Un ejemplo de plan se da a continuación:

```
<Definition>
        <Name>Decrementar Temperatura</Name>
        <InvocationCons>
                <Goal>Ajustar temperatura</Goal>
        </InvocationCons>
        <Precons>
                <Atribute>
                        <Type>Temperatura</Type>
                        <Value>7</Value>
                </Atribute>
        </Precons>
        <Perform>
                <Action>Decrementar temperatura</Action>
        </Perform>
</Definition>
```

Un agente conocerá si una meta ha sido exitosa, siguiendo la ejecución de un plan, creando una vista del ambiente y viendo si el atributo en cuestión ha sido establecido al valor deseado.

Agentes autónomos

Los agentes autónomos dan significado al sistema basado en agentes a través de su habilidad para generar sus propias metas, manejadas por sus motivaciones. La selección de metas a través de sus motivaciones puede ser hecha en cualquier número de formas desde algunas muy simples hasta otras muy complicadas. En Paradigma se opta por una estructura llamada una m-tripleta (m, v, b); las motivaciones tienen un nombre <m>, una fuerza <v> y

una variable booleana <b> que indica si la motivación es estática o variable. Las motivaciones, junto con los planes y metas definen el comportamiento del agente. Cuando se instancia un agente autónomo, archivos XML son leídos para crear todos los atributos, capacidades, metas, planes y motivaciones.

Basado en sus motivaciones, el agente escogerá la meta que ofrezca una utilidad motivacional más grande. Esto lo guiará para intentar seleccionar e implementar un plan que, a su vez, lo guiará hacia la activación de un cierto número de capacidades, o el encargo a otros agentes para alcanzar sus metas. Una vez que la meta ha sido obtenida, el agente autónomo reajustará sus motivaciones e intentará la siguiente meta mejor desde su base de metas.

Enseguida se muestra un ejemplo de una motivación.

```
<Definition>
        <Name>conectarse</Name>
        <Value>1.0</Value>
        <Variable>si</Variable>
</Definition>

<Definition>
        <Name>obedecer comandos</Name>
        <Value>1.0</Value>
        <Variable>no</Variable>
</Definition>

 <Definition>
        <Name>controlar utilidades</Name>
        <Value>0.0</Value>
        <Variable>si</Variable>
</Definition>
```

Un agente autónomo que escoge estar disponible para cooperar con otros agentes necesita reunirse en una comunidad Jini para poder ser descubierto. El agente podría facilitar tal comunicación en diferentes formas. Algunas de las principales las constituyen la comunicación basada en *sockets*, comunicación basada en RMI y otra es el ruteador de mensajes.

2.2.6 RETSINA

RETSINA (*Reusable Environment for Task-Structured Intelligent Networked Agents*), es un sistema multiagente abierto que soporta comunidades de agentes heterogéneos. Esta infraestructura parte de la premisa de que los agentes deberían formar comunidades interactivas permitiendo así estructuras de coordinación emergentes, más que como resultado de restricciones impuestas por la infraestructura misma. Para ello, RETSINA implementa servicios distribuidos que facilitan las interacciones entre agentes (Sycara et al., 1996).

Como parte de la infraestructura proporciona cuatro tipos básicos de agentes: agentes de interfaz, agentes de tareas, agentes intermediarios y agentes de información. La investigación en la que se basa este proyecto facilita el problema de la comunicación entre agentes heterogéneos a través de agentes intermediarios que sirven de enlace entre agentes que solicitan servicios y agentes que ofrecen servicios. Para lograr la comunicación entre agentes se ha desarrollado un lenguaje de descripción de capacidades de agentes que permite que agentes heterogéneos interoperen.

Cada agente RETSINA tiene cuatro módulos reusables para comunicación, planificación, calendarización y monitoreo de la ejecución de tareas y solicitudes de otros agentes.

Desde el punto de vista de la implementación de nuevos tipos de agentes, RETSINA ofrece a los programadores de agentes la Fundación de Clases de Agentes (AFC, *Agent Foundations Classes*). AFC está basada en bibliotecas de clases de los lenguajes C y C++. C++ es usado para programar agentes de mayor complejidad ya que C está limitado para construir agentes pequeños (RETSINA,2002). Por medio del AFC es posible construir agentes que tengan las siguientes características:

1.- Interoperar entre sí y con otros tipos y sistemas de agentes heterogéneos.
2.- Registrar sus servicios y capacidades y encontrar agentes cuyas capacidades sean necesarias, a través del *matchmaker* de RETSINA.
3.- Encontrar y comunicarse con otros agentes a través de sistemas distribuidos.
4.- Enlazarse a un componente de planificación o razonamiento que controla las actividades del agente.

2.2.6.1 Diseño de agentes con el AFC de RETSINA

Cada agente está basado en el agente básico de RETSINA. En términos de AFC, cada agente hereda de la clase BasicAgent. Cualquier clase derivada del agente básico forma parte de la capa de abstracción de agente. Los demás componentes de más bajo nivel forman la capa de la abstracción de la comunicación. Estos componentes son usados por el agente básico y están disponibles para todos los agentes. Aunque es posible construir un agente a partir de la clase BasicAgent, se recomienda que los programadores de agentes basen sus nuevos agentes a partir de las subclases existentes que derivan del agente básico como se ve en la figura 9.

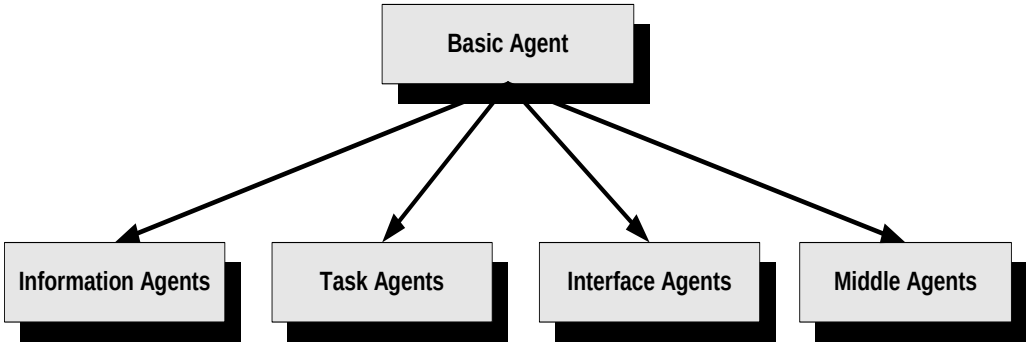

Figura 9 Anatomía de un agente

La definición de agente que sirve de base para esta infraestructura considera que un agente es una pieza de código individual con comportamiento comunicativo e inteligente.

El agente básico corre y administra un conjunto de módulos cliente diseñados para manejar datos y diálogos con entidades externas (ver figura 10). Sus tareas pueden variar desde seguimiento de actividades en archivos hasta visualización de la interacción. El AFC ofrece un conjunto de herramientas y clases base para desarrollar clientes personalizados y estas deben ser usadas para desarrollar agentes que se comuniquen entre sí.

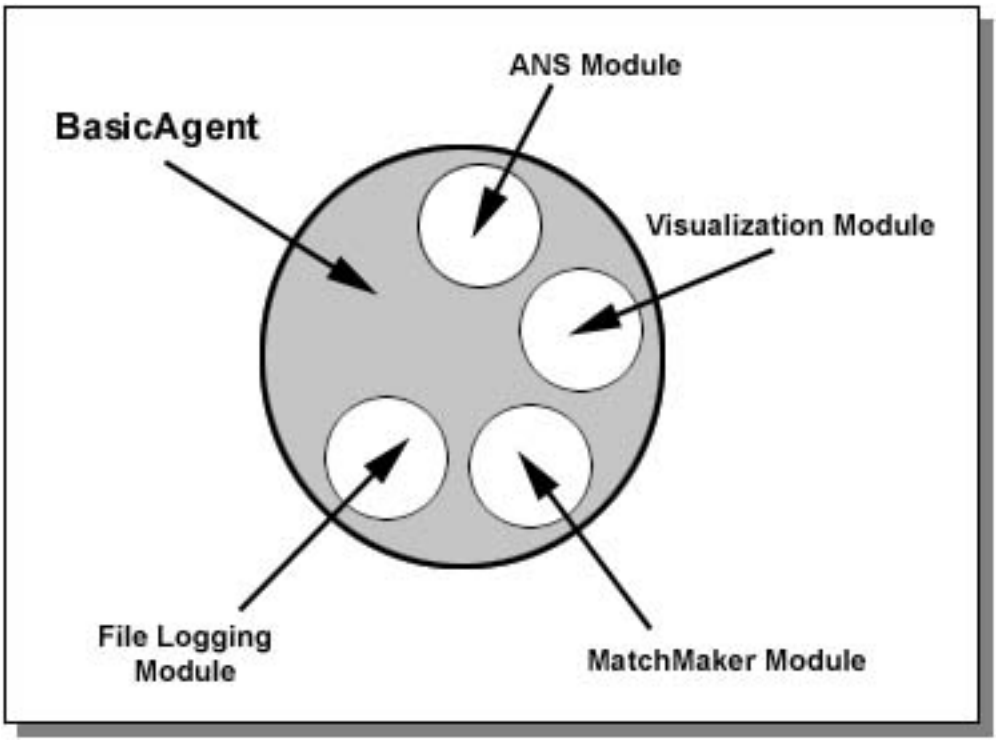

Figura 10 Agente básico en RETSINA

2.2.6.2 Construcción de agentes RETSINA

Para construir un agente utilizando el AFC de RETSINA es necesario tener instalado el Microsoft Visual Studio 6.0. Obviamente, también se debe tener instalado AFC. En el Visual C++ se debe seleccionar la opción de crear un nuevo proyecto; el tipo de proyecto debe ser *MFC Retsina Agent AppWizard*. Es muy importante crear los agentes a partir de la guía de Visual C++. Enseguida se crea un *workspace* (figura 11) de Visual C++ para implementar la funcionalidad del nuevo tipo de agente. En la figura 11 se muestra un nuevo proyecto de agente y una ventana de diálogo vacía que puede ser usada por el agente. Una barra de estado ha sido incluida, la cual mostrará todos los mensajes generados por los componentes y módulos a través de AFC. Con el proyecto de agente se crean varios archivos, la mayoría de ellos son particulares a Visual C++. El archivo c_AgentA.cpp, en el ejemplo, contiene la implementación del agente derivado de la clase CBasicAgent; esta es la clase base que implementa todo el núcleo del comportamiento y funcionalidad.

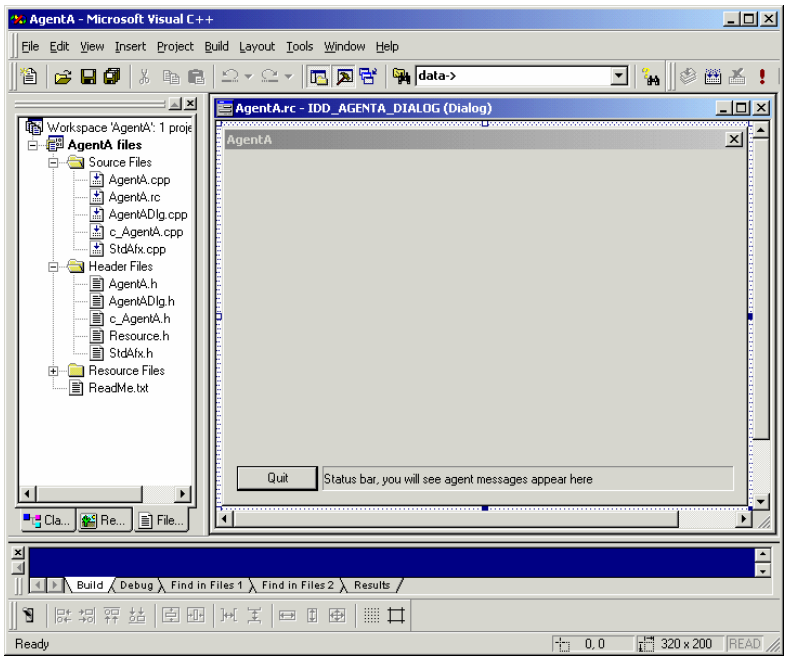

Figura 11 *WorkSpace* del agente AgentA

2.2.6.3 Programación de agentes RETSINA

El agente básico en el AFC por *default* tiene un evento *timer* en cada segundo implementado a través del método *process_timer*. Se puede usar este evento de tiempo para actualizar el estado interno del agente. Cuando se crea el agente este método está vacío. A continuación se muestra un ejemplo de su implementación.

```
void CAgentA::process_timer (void)
 {
  char message [512];

  counter++;

  if (counter>step)
  {
   counter=0;
   step++;

   sprintf (message,":number %d",step);

   char *reply=Communicator->comm_sendmessage ("tell",
                                               "AgentB",
                                               "default-language",
                                              "default-ontology",
                                               NULL,
                                               NULL,
                                               NULL,
                                               message,
                                               NULL);
   if (reply!=NULL)
    debug (reply);
```

```
   else
    debug ("Message sent to Agent");
  }
 }
```

Figura 12 Envío de mensajes en RETSINA

Algo que es muy importante resaltar en el código fuente de la figura 12 es lo que se refiere a la forma en que estos agentes pueden enviar y recibir mensajes. El agente crea un mensaje dirigido al agente AgentB, usando para ello el método *comm_sendmessage* del componente *Communicator* del agente. El lenguaje de comunicación de agentes utilizado es KQML. El contenido del mensaje es simplemente una cadena de caracteres para el campo *:number*. Ahora que se ha enviado un mensaje es necesario escribir el código para recibir el mensaje en la implementación del agente AgentB.

El agente básico llama el método *process_message* cuando un mensaje llega. Este método está vacío cuando el agente se crea con el AFC. El ejemplo siguiente implementa la recepción del mensaje enviado anteriormente.

```
BOOL CAgentB::process_message (char *data)
 {
  CParser *c_parser=new CParser;
  if (c_parser->parse_message (data)==FALSE)
  {
   delete c_parser;
   debug ("<CAgentB> Unable to parse incoming message");
   return (FALSE);
  }
  char *sender =c_parser->find_sender ();
  char *content=c_parser->find_content ();
  if ((sender==NULL) || (content==NULL))
  {
   delete c_parser;
   debug ("<CAgentB> Either sender or content field is NULL, unable to
proceed");
   return (FALSE);
  }
  CParser *r_parser=new CParser;
  if (r_parser->parse_message (content)==FALSE)
  {
   delete r_parser;
   delete c_parser;
   debug ("<CAgentB> Unable to parse content field");
   return (FALSE);
  }

  char *number=r_parser->find_token ("number");
  if (number==NULL)
  {
   delete r_parser;
   delete c_parser;
   debug ("<CAgentB> Number not found in content field");
   return (FALSE);
  }
```

```
  step=atoi (number); // change the step
  delete r_parser;
  delete c_parser;

  return (TRUE); // we processed the message so we have to indicate this
back
}
```

Figura 13 Recepción de mensajes en RETSINA

El agente puede descomponer el mensaje a través de un objeto *parser* y llamando el método adecuado.

c_parser->parse_message (data)

En este ejemplo se puede ver que el campo *content* puede contener, a su vez, varios campos, creados al enviar el mensaje. Por lo tanto se debe crear otro *parser* y se invoca el método *parse_message* con el campo *content* como parámetro para obtener cada uno de los campos del contenido del mensaje.

r_parser->parse_message (content)

Siguiendo con el ejemplo, si el *parser* trabajó bien, ahora es posible obtener el campo *:number* que es el campo que se envió al formar el mensaje.

*char *number=r_parser->find_token ("number");*

CAPÍTULO 3. ESPECIFICACIONES Y TECNOLOGÍA SUBYACENTE DE LA PROPUESTA

Resumen

En este capítulo se tiene como propósito fundamental establecer las bases tecnológicas que son importantes para soportar el desarrollo de las ideas centrales de esta tesis. Se trata de conocer los aspectos básicos de la tecnología que se utiliza durante el desarrollo de los agentes, la herramienta para su creación, así como los aspectos de su implementación y operación computacional. Los temas que se mencionan son: las características de la plataforma CAP, algunos aspectos relacionados con las especificaciones de FIPA para la interoperabilidad entre agentes y la tecnología de componentes COM.

Objetivo del capítulo

Conocer los principales elementos base que forman la tecnología subyacente sobre la que se soportan los agentes para entender mejor el ambiente de ejecución de los SMA que se logren desarrollar con estos.

3.1 Introducción

En este capítulo se tiene como propósito fundamental establecer las bases tecnológicas que son importantes para soportar el desarrollo de las ideas centrales de esta tesis. Se trata de conocer a los aspectos básicos de la tecnología que se utiliza durante el desarrollo teórico y conceptual de los agentes, la herramienta para su creación, así como los aspectos de su implementación y operación computacional. Con estos elementos será posible entender mejor el ambiente de ejecución de los agentes y los SMA que se logren desarrollar con estos.

Se comenzará por un repaso de las características que presenta la plataforma CAP; en este punto también se hacen algunos comentarios acerca de las especificaciones de FIPA con respecto a los agentes y servicios que debe brindar obligatoriamente toda plataforma de agentes compatible con FIPA, en la idea de entender mejor la forma en que funciona dicha plataforma.

En la sección 3.3 se abarcan aspectos relacionados también con las especificaciones de FIPA para la implementación e interoperabilidad de sistemas de agentes que se ejecutan sobre una plataforma FIPA compatible. En concreto, se consideran temas como el lenguaje de comunicación de agentes, los lenguajes de contenido y la inicialización de agentes.

Por último, se hace un análisis de la tecnología COM sobre todo visto desde la perspectiva de la posible utilidad que nos puede brindar con respecto a la tecnología de agentes, lo cual se traduce en un punto central, tal vez el más importante, para esta investigación.

3.2 La Plataforma de Agentes Componentes CAP

La plataforma de Agentes Componentes (*Component Agent Platform*, CAP) está basada en el modelo de referencia de FIPA y fue desarrollada en el Laboratorio de Agentes del Centro de Investigación en Computación del Instituto Politécnico Nacional (Contreras, 2001).

Consiste en una herramienta novedosa en el área de programación orientada a agentes que permite un acercamiento del paradigma de agentes hacia un mayor número de desarrolladores de aplicaciones, en gran medida, de aplicaciones basadas en Windows.

Los servicios básicos y obligatorios de la plataforma de agentes CAP tienen la característica de que se han implementado sobre servidores DCOM (Sheremetov & Contreras, 2001). Todo ello se agrupa en los componentes AMS, DF y ACC.

La ventaja de utilizar servidores DCOM para soportar las capacidades básicas de la plataforma permite que esté disponible en todos los entornos Windows del mercado. Cada uno de los componentes de CAP es un objeto DCOM de instancia única de tal forma que se asegura que todos los clientes (agentes) trabajen con la misma instancia de los objetos dentro de CAP.

Los componentes de la plataforma residen en la misma máquina, mientras que los agentes de la aplicación multiagente pueden estar ubicados físicamente en distintas máquinas o en la misma.

La comunicación dentro de la plataforma, y entre plataformas CAP, se da a través del protocolo nativo de COM que consiste en llamadas a procedimientos remotos de objetos (ORPC, *Object Remote Procedure Call*). Para lograr la comunicación con plataformas de agentes que no son CAP, pero sí compatibles con FIPA, se debe usar el *gateway* de la plataforma. En la figura 14 se presenta la arquitectura de la plataforma.

El lenguaje de comunicación entre agentes de CAP es FIPA-ACL (ACL propuesto por FIPA). Esto es necesario debido a que FIPA impone que la comunicación entre agentes debe ser de alto nivel refiriéndose a la manipulación e intercambio de conocimiento entre los agentes. En un ambiente distribuido un agente suele cumplir con sus objetivos a través de la interacción con otros agentes y ejecutando acciones, lográndolo precisamente a través del uso de su lenguaje de comunicación entre agentes. FIPA-ACL está basado en la teoría de actos del habla.

Una característica que es importante mencionar es que CAP codifica los mensajes ACL por medio del lenguaje de marcado extensible XML (*eXtended Markup Language*), utilizando para ello el *parser* XML DOM de Microsoft y verifica la sintaxis de los mensajes a través de un DTD (*Document Type Definition*) que FIPA propone para este fin.

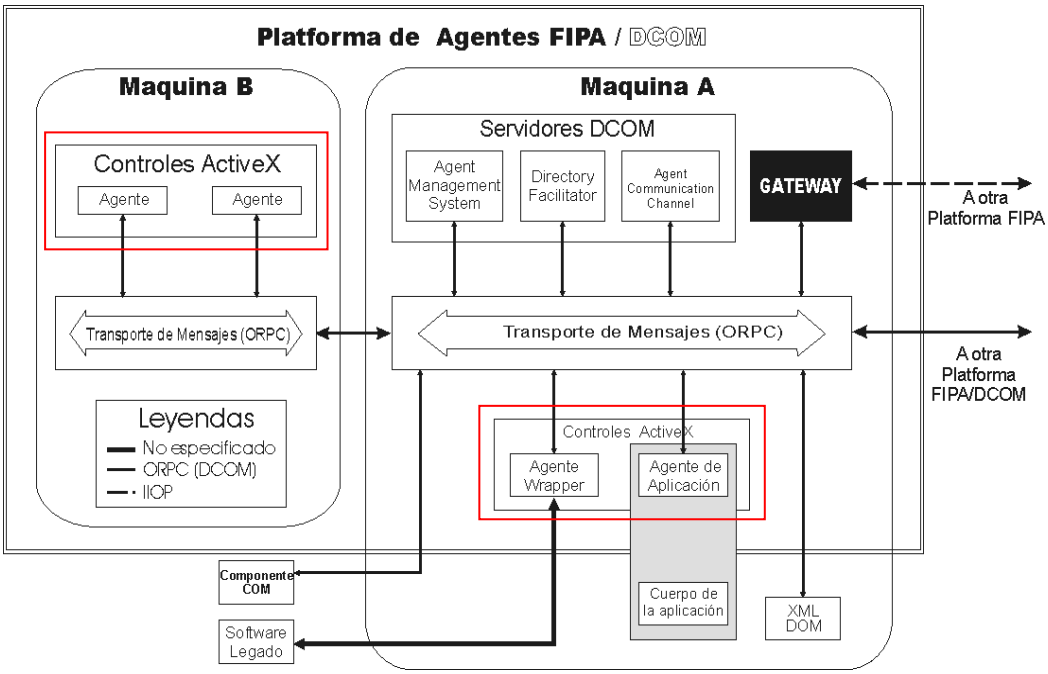

Figura 14 Arquitectura de CAP

CAP viene acompañada, además de los componentes básicos, con algunas herramientas que permiten administrar, monitorear y depurar el funcionamiento de la plataforma y los agentes que se han registrado en ella (Sheremetov & Contreras,2001).

Una de estas herramientas permite la Administración Remota, con la cual es posible establecer comunicación remota con los componentes AMS, DF, y ACC de la plataforma para monitorear su estado, enviar comandos administrativos como *start*, *stop*, *reset*, etc. así como ver a los agentes que están registrados en el AMS y DF, desregistrarlos, cambiar su descripción, y enviarles mensajes administrativos.

Otra herramienta permite configurar los parámetros de operación de la plataforma por ejemplo, los perfiles de comunicación disponibles y las características de soporte a agentes móviles y registro dinámico de agentes. Esta herramienta se llama Administrador de la plataforma.

La Herramienta para el monitoreo y filtrado de mensajes permite ver todos los mensajes enviados y recibidos en la plataforma, y filtrar los mensajes capturados sobre la base de distintos parámetros como tipo de *performative*, emisor del mensaje, receptor, etc.

El visualizador de sociedad es una interfaz gráfica que muestra a todos los agentes registrados en la plataforma y su finalidad principal es mostrar la comunicación que se da en el tiempo de la ejecución de los agentes. Para ello ofrece dos modalidades de visualización: una permite desplegar el paso de mensajes entre agentes a manera de diagrama de interacción. La segunda forma de visualizar los mensajes es donde cada mensaje se representa como iconos entre los agentes codificados por color de acuerdo con el tipo de *performative*.

Por último, CAP tiene un visualizador simple de mensajes que se utiliza en el visualizador de sociedades de agentes para analizar y probar el contenido de los mensajes ACL de forma fácil y rápida, apoyando de esta forma el proceso de depuración y desarrollo de aplicaciones de agentes.

3.3 Especificaciones de FIPA

El objetivo de esta parte es mencionar algunas de las especificaciones más importantes de FIPA (FIPA-1, 2000) que sirven de base para la implementación de una plataforma de agentes y los sistemas de agentes que corren en ésta; obviamente se mencionan los aspectos que son de interés para este trabajo.

Dentro de los temas que más importan se encuentran las líneas para la especificación de sistemas de agentes en términos de software particular y tecnologías de comunicación o también llamadas guías para la instanciación. Además, es importante mencionar las guías para la interoperabilidad que consisten en especificaciones que regulan la interoperabilidad y conformación de agentes y sistemas de agentes.

La meta de FIPA es crear estándares de agentes para promover las aplicaciones de agentes interoperables y los sistemas de agentes. En el centro del modelo de FIPA para los sistemas de agentes está la comunicación entre agentes, en donde estos pueden pasarse mensajes con un significado semántico con la finalidad de cumplir las tareas requeridas por la aplicación (FIPA-2, 2001).

FIPA deja claro que para construir sistemas de agentes comerciales es importante y necesario comprender y usar los ambientes de software existentes, como son:

- Lenguajes de programación y plataformas de computación distribuida
- Servicios de paso de mensajes
- Servicios de seguridad
- Servicios de directorios
- Tecnologías de conectividad

La especificación FIPA 2000 de una arquitectura de agentes es utilizada para el desarrollo de las principales implementaciones de este trabajo por ser la especificación más reciente. Su meta principal es permitir la creación de sistemas de agentes que se puedan integrar dentro de su ambiente computacional específico mientras interoperan con aplicaciones de agentes que residen en ambientes separados.

El enfoque principal de FIPA es crear la infraestructura necesaria para el intercambio de mensajes con un significado semántico entre agentes, para lo cual pueden ser usados diferentes mecanismos de transporte de mensajes, lenguajes de comunicación entre agentes o diferentes lenguajes de contenido. Esto requiere numerosos puntos de interoperabilidad potenciales:

- Interoperabilidad en el transporte de mensajes
- Soporte de varias formas de representación de ACL
- Soporte de varias formas de lenguajes de contenido
- Soporte de múltiples representaciones de servicios de directorios

La arquitectura de FIPA define un nivel abstracto de cómo los agentes se pueden localizar y comunicar con los demás registrándose ellos mismos e intercambiando mensajes. Aquí se mencionan algunas nociones fundamentales para la implementación de una infraestructura concreta para el desarrollo de agentes.

Un agente primero se trata de registrar como un proveedor de servicios. Este se conecta a uno o varios mecanismos de transporte. En algunas implementaciones, delegará esta tarea al servicio de transporte de mensajes; en otras, este manejará los detalles de, por ejemplo, contactar un ORB, registrarse con un registro de RMI o configurarse como un escuchador sobre una cola de mensajes. Una vez hecho esto, el agente debe registrarse ante el servicio de directorios para lograr la interacción con otros agentes.

Los agentes se comunican intercambiando mensajes los cuales representan actos del habla, y son codificados en un lenguaje de comunicación de agentes. Los dos aspectos fundamentales en la comunicación de mensajes entre agentes son la estructura del mensaje y el transporte. La estructura del mensaje está formada por tuplas llave-valor y está escrita en un lenguaje de comunicación entre agentes como FIPA-ACL. El contenido de un mensaje es expresado en un lenguaje de contenido, como KIF (*Knowledge Interchange Format*), SL (*Semantic Language*) o RDF (*Resource Description Framework*). El lenguaje de contenido puede referenciar una ontología para establecer el significado de los conceptos

que están siendo discutidos en el contenido. Los mensajes también contienen el emisor y receptor expresados como nombres de agentes.

3.3.1 El lenguaje de comunicación de agentes

Un lenguaje de comunicación de agentes es un lenguaje en el cual se pueden expresar actos comunicativos. La arquitectura de FIPA es definida en términos de un ACL abstracto. Una sintaxis abstracta es una sintaxis en la cual los operadores y objetos de un lenguaje son expuestos, junto con una semántica precisa para estas entidades. Su principal objetivo es aclarar el significado semántico de las construcciones del lenguaje de manera legible y con conveniencia de expresión.

FIPA-ACL

Un mensaje FIPA-ACL contiene un conjunto de uno o más elementos del mensaje. Los elementos que son necesarios para lograr una comunicación efectiva variarán de acuerdo a la situación; el único elemento que es obligatorio en todos los mensajes ACL es el *performative*, aunque también se espera que la mayoría de los mensajes contengan emisor, receptor y elementos de contenido (FIPA-3, 2001).

Si un agente no reconoce o es incapaz de procesar uno o más de los elementos o valor de elemento, este puede responder con el apropiado mensaje *not-understood*.

Algunos elementos del mensaje pueden ser omitidos cuando su valor puede ser deducido por el contexto de la conversación. Sin embargo, FIPA no especifica algún mecanismo para manejar tales condiciones, de tal forma que aquellas implementaciones que omitan algún elemento del mensaje no garantizan la interoperarabilidad con otras.

El conjunto completo de elementos de un mensaje FIPA-ACL se muestra en la tabla de la figura 15, sin importar el esquema de codificación específico en una implementación.

Elemento	Categoría del elemento	Descripción
performative	Tipo de acto comunicativo	Denota el tipo de acto comunicativo del mensaje ACL. Es un elemento obligatorio de todos los mensajes ACL.
sender	Participante en la comunicación	Denota la identidad del emisor del mensaje. Es posible omitir el emisor si, por ejemplo, el agente que envía el mensaje quiere permanecer anónimo.
receiver	Participante en la comunicación	Denota la identidad de los receptores del mensaje.
reply-to	Participante en la comunicación	Indica que los mensajes subsecuentes en este hilo de conversación serán dirigidos al

		agente cuyo nombre aparezca en el campo *reply-to*, en lugar del agente emisor.
content	Contenido del mensaje	Denota el contenido del mensaje; equivalentemente denota el objeto de la acción. La mayoría de los mensajes ACL requieren una expresión de contenido.
language	Descripción del contenido	Denota el lenguaje en el que el elemento contenido es expresado. El elemento contenido es expresado en un lenguaje formal; este campo puede ser omitido si los agentes que reciben el mensaje pueden asumir que conocen el lenguaje de la expresión de contenido
encoding	Descripción del contenido	Denota el esquema de codificación específico en el que está expresado el contenido.
ontology	Descripción del contenido	Denota la ontología usada para dar significado a los símbolos en la expresión de contenido.
protocol	Control de conversación	Es el protocolo de interacción que se está usando con este mensaje ACL.
conversation-id	Control de conversación	Introduce una expresión (identificador de conversación) que es usado para identificar la secuencia de actos comunicativos que juntos forman una conversación.
reply-with	Control de conversación	Introduce una expresión que será usada por el agente que responda para identificar este mensaje.
in-reply-to	Control de conversación	Denota una expresión que referencia una acción anterior para la cual este mensaje es una respuesta.
reply-by	Control de conversación	Denota una expresión de hora / fecha que indica el último tiempo en el que el agente emisor le gustaría recibir una respuesta.

Figura 15 Elementos del mensaje en FIPA-ACL

Cada especificación para la representación de mensajes ACL contiene descripciones sintácticas precisas para codificación de mensajes ACL basadas en XML, cadenas de texto y algunos otros esquemas.

Además de proporcionar la estructura de los mensajes ACL, FIPA mantiene una biblioteca de actos comunicativos que pueden ser utilizados en la comunicación de agentes en el campo *performative* (FIPA-4, 2001). Los detalles de algunos de los actos comunicativos más importantes se verán en la parte de las clases utilitarias creadas para implementar un subconjunto de ACL, en el capítulo 7.

3.3.2 Lenguajes de contenido de FIPA

FIPA mantiene una lista de lenguajes de contenido y sus especificaciones correspondientes en la biblioteca de lenguajes de contenido (CLL, *Content Language Library*) (FIPA-5, 2001). Los objetivos para estandarizar y definir esta biblioteca de especificaciones de los lenguajes de contenido son, por una parte, permitir mayor flexibilidad y eficiencia para las aplicaciones. Por otro lado, también se busca incrementar la interoperabilidad entre agentes desarrollados en diferentes contextos, facilitando el reuso de diferentes lenguajes de contenido. Esto debería permitir instanciar el uso de esquemas RDF emergentes en la web, o la capacidad de expresar restricciones en un lenguaje de contenido.

Si un agente informa que acepta alguno de los lenguajes de contenido entonces debe implementar un *parser* o intérprete para el lenguaje conforme éste es definido en la biblioteca de lenguajes de contenido de FIPA. Sin embargo, los agentes compatibles con FIPA no están obligados a implementar alguno de estos lenguajes. FIPA CLL facilita el uso de lenguajes de contenido estandarizados para agentes desarrollados en diferentes contextos. Esto también da un incentivo mayor a los desarrolladores para hacer sus lenguajes de contenido aplicables generalmente.

Cada lenguaje de contenido compatible con FIPA debe especificar al menos un esquema de codificación canónico que puede ser usado para este lenguaje en un mensaje ACL. El esquema más común es el de *string*, pero se permiten otros formatos de codificación, de tal forma que los mensajes ACL puedan representar más eficientemente el contenido de los formatos de datos específicos de la aplicación. Los esquemas de codificación que un lenguaje de contenido FIPA puede soportar deben ser dados como parte de la especificación del lenguaje, junto con un nombre que identifique a cada codificación. En la figura 16 se presentan los lenguajes de contenido registrados actualmente en la biblioteca de lenguajes de contenido de FIPA.

Lenguaje de Contenido	Especificación FIPA	Fecha	Estatus
FIPA-SL-00008	[FIPA00008]	2000/07/21	Experimental
FIPA-CCL-00009	[FIPA00009]	2000/07/21	Experimental
FIPA-KIF-00010	[FIPA00010]	2000/07/21	Experimental
FIPA-RDF-00011	[FIPA00011]	2000/07/21	Experimental

Figura 16 Biblioteca de lenguajes de contenido de FIPA

3.3.2.1 FIPA-RDF

Esta especificación describe la forma en que RDF puede ser usado como lenguaje de contenido en un mensaje de FIPA-ACL (FIPA-6, 2001). Aunque FIPA no requiere que un lenguaje de contenido sea capaz de representar acciones, una parte de los actos comunicativos requiere especificar acciones en su contenido. Para ello, RDF es extendido para expresar:

Objetos: se construyen para representar una entidad identificable (abstracta o concreta) en el dominio de discurso.

Proposiciones: son enunciados que expresan que alguna sentencia en el lenguaje es verdadera o falsa.

Acciones: Tratan de expresar una actividad que puede ser llevada a cabo por un objeto.

Por medio de los mecanismos existentes en RDF (RDF, 1999), FIPA propone extensiones modulares formando el lenguaje de contenido fipa-rdf0. La motivación principal es definir un estándar común para incrementar el nivel de interoperabilidad; de ahí la importancia de utilizar las fortalezas del lenguaje RDF como son la extensibilidad, reusabilidad y simplicidad. Otra ventaja de RDF es que los datos y esquemas son intercambiados de una forma similar. El modelo RDF propone el XML (XML, 1999) como su sintaxis de codificación pero no prohíbe que alguien use esquemas de codificación alternativos.

RDF es un *framework* que está basado en un modelo entidad-relación. El modelo de datos de RDF está descrito por medio de recursos, propiedades y sus valores. Un recurso específico junto con una o más propiedades más los valores de estas propiedades es una descripción de RDF (una colección de enunciados). Junto con el modelo de datos de RDF, la especificación de los esquemas provee un sistema de tipos para los recursos y propiedades usados en los datos RDF. Define conceptos como clases, subclases, propiedades o subpropiedades y permite expresar restricciones. Tanto el modelo de datos como los esquemas RDF proponen XML como una sintaxis de serialización (FIPA-6, 2001).

RDF es un fundamento para el procesamiento de meta-datos y su finalidad es proveer interoperabilidad entre aplicaciones que intercambian información incomprensible por la máquina. Esto sugiere que RDF puede ser útil para facilitar, compartir e intercambiar conocimiento entre agentes.

Para ser capaz de usar RDF como un lenguaje para mensajes FIPA-ACL, se tiene que explorar la manera de expresar objetos, proposiciones y acciones, sin poner en peligro la extensibilidad clave inherente al lenguaje. Por otro lado, se debe tratar de preservar la simplicidad de RDF. Se propone por FIPA el nombre de fipa-rdf0 para el uso combinado de RDF con los esquemas básicos de extensión necesarios y definidos por FIPA.

Objetos: Tomando en cuenta lo anterior, es obvio ver una analogía entre un objeto ACL y un recurso RDF, debido a que ambos están definidos como descripciones de cierta entidad identificable. Esto permite usar identificadores de recursos RDF y referencias como referencias e identificadores de objetos ACL.

Proposiciones: En el mismo contexto parece lógico modelar proposiciones ACL usando enunciados RDF. Un enunciado RDF está compuesto por tres partes: sujeto (recurso), predicado (propiedad) y el objeto (literal / valor). Como un ejemplo, considere la sentencia "W. Richard Stevens es el Autor de TCP/IP ilustrado". Los componentes RDF de esta proposición son: el sujeto (TCP/IP ilustrado), el predicado (Autor) y el objeto (W. Richard Stevens). Esta sentencia o enunciado puede entonces ser descrito en RDF como se muestra en el documento XML de la figura 17.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://description.org/schema/">
<rdf:Description ID="TCP/IP Ilustrado">
<s:autor>W. Richard Stevens</s:autor>
</rdf:Description>
</rdf:RDF>
```

Figura 17 Ejemplo de proposición en RDF

La figura 18 representa este enunciado como un grafo RDF. De esta forma se tiene un punto de inicio para expresiones lógicas sobre el contenido. Se puede decir que al expresar el enunciado de esta manera se tiene la creencia sobre este enunciado. Así, siempre se asume que una sentencia RDF expresa una creencia. Esta idea podría ser suficiente en cualquier contexto donde el nivel de lógica involucrado es limitado.

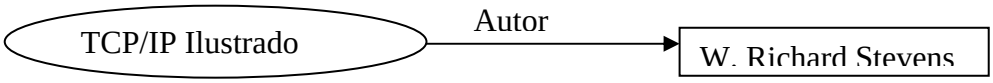

Figura 18 Diagrama de la proposición como un enunciado RDF

Retomando lo anterior, se explica enseguida la manera en que la creencia lógica (verdadero o falso) de un cierto enunciado podría ser expresada explícitamente usando RDF como un recurso con cuatro propiedades definidas:

La propiedad sujeto define el recurso que está siendo descrito por medio del enunciado modelado; esto es, el valor de esta propiedad es el recurso acerca del cual el enunciado original fue hecho.

El predicado identifica la propiedad del enunciado original; esto es, el valor es la propiedad específica en el enunciado original.

El objeto identifica el valor de la propiedad en el enunciado original; esto es, el valor es el objeto en el enunciado original.

El valor de la propiedad tipo describe el tipo del nuevo recurso. Todos los enunciados creados son instancias de rdf:*statement*.
Un nuevo recurso con las cuatro propiedades anteriores representa un enunciado original y puede ser usado como el objeto de otro enunciado y tener enunciados adicionales hechos a

partir de éste. El recurso con estas cuatro propiedades no es un reemplazo para el enunciado original, pero este es un modelo del enunciado. El esquema propuesto para esto se presenta en la figura 19.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<rdfs:Class rdf:ID="http://www.fipa.org/schemas#Proposition">
<rdfs:label xml:lang="en">proposition</rdfs:label>
<rdfs:label xml:lang="fr">proposition</rdfs:label>
<rdfs:comment>This describes the set of propositions</rdfs:comment>
<rdfs:subClassOfrdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement"/>
</rdfs:Class>
<rdfs:ConstraintProperty rdf:ID="http://www.fipa.org/schemas#belief">
<rdfs:label xml:lang="en">belief</rdfs:label>
<rdfs:label xml:lang="fr">acte</rdfs:label>
<rdfs:domain rdf:resource="#Proposition"/>
<rdfs:range rdf:resource="http://www.w3c.org/TR/1999/PR-rdf-schema-
19990303#Literal"/>
</rdfs:ConstraintProperty>
</rdf:RDF>
```

Figura 19 Esquema de RDF para las proposiciones

Usando este método se puede describir fácilmente una proposición ACL en RDF. Como ejemplo, se modela la siguiente proposición: “W. Richard Stevens es el autor de TCP/IP ilustrado es verdadera”. Una forma de representar esta proposición en FIPA-RDF es como lo muestra la figura 20.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fipa=http://www.fipa.org/schemas/fipa-rdf0#">
<fipa:Proposition>
<rdf:subject>TCP/IP Ilustrado</rdf:subject>
<rdf:predicate rdf:resource="http://description.org/ schema#author"/>
<rdf:object>W. Richard Stevens</rdf:object/>
<fipa:belief>true</fipa:belief>
</fipa:Proposition>
</rdf:RDF>
```

Figura 20 Una proposición en FIPA-RDF

La expresión de que el mismo enunciado es falso, es igualmente fácil; nada más hay que reemplazar el valor *true* por *false*. El grafo RDF que representa el enunciado falso se despliega en la figura 21.

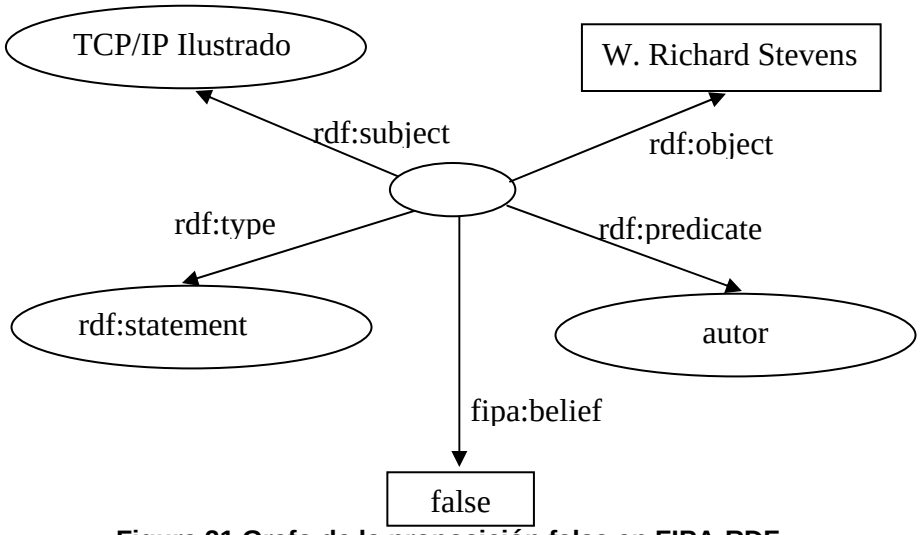

Figura 21 Grafo de la proposición falsa en FIPA-RDF

Representación de Acciones en RDF: Una acción expresa una actividad llevada a cabo por un objeto. Hay tres diferentes propiedades relacionadas a una acción:

Un acto identifica la parte operativa de la acción; esto puede servir para identificar el tipo de acción o meramente para describir la acción. En el último caso especifica los tipos de clases de acción que pueden ser derivadas de la clase *Action*. Un actor identifica la entidad responsable para la ejecución de la acción, esto es, el valor es la entidad específica que podrá/puede/podría realizar la acción (frecuentemente el receptor, pero posiblemente otro agente o entidad bajo control del receptor). Un argumento (opcional) identifica una entidad que puede ser usada para la ejecución de la acción; esto es, el valor es la entidad que es usada por el actor para realizar la acción. Una acción puede tener múltiples argumentos.

Cuando se mira a una acción de esta forma, hay una analogía estructural con una sentencia RDF. Para modelar una acción, el modelo de la sintaxis de RDF puede ser extendido con un nuevo tipo RDF: fipa:*Action* la cual tiene estas propiedades. Como ejemplo, la siguiente acción será modelada: “Juan abre la puerta1 y la puerta2”. En este pequeño ejemplo, las propiedades son: Acto (abrir), el actor (Juan) y los argumentos (puerta1 y puerta2). Esta acción puede ser descrita como se aprecia en la figura 22.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fipa="http://www.fipa.org/schemas/fipa-rdf0#">
<fipa:Action rdf:ID="JuanAccion1">
<fipa:actor>Juan</fipa:actor>
<fipa:act>abrir</fipa:act>
<fipa:argument>
<rdf:bag>
<rdf:li>puerta1</rdf:li>
<rdf:li>puerta2</rdf:li>
</rdf:bag>
```

```
</fipa:argument>
</fipa:Action>
</rdf:RDF>
```

Figura 22 Representación de una acción en FIPA-RDF

Al modelo de acción aun le falta la habilidad para establecer cuándo alguna acción ha terminado y cuál es el resultado de la ejecución de la acción. Esto se puede resolver añadiendo propiedades extra a la definición de la acción.

Como ejemplo, supóngase que Mary solicita a Juan abrir la puerta1 y la puerta2 y después quiere que Juan le informe si realizó la acción y el resultado que se obtuvo. Este pequeño escenario consiste de dos mensajes: *request* de Mary a Juan conteniendo la descripción de la acción; y también un mensaje *inform* de Juan a Mary, refiriéndose a la acción y el status de la ejecución de la acción. Usando FIPA-ACL combinado con el contenido en RDF, el primer mensaje podría ser expresado como en la figura 23.

```
(request :sender Mary
:receiver Juan
:content ( <?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fipa="http://www.fipa.org/schemas#">
<fipa:Action rdf:ID="JuanAccion1">
<fipa:actor>Juan</rdf:actor>
<fipa:act>abrir</rdf:act>
<fipa:argument>
<rdf:bag>
<rdf:li>puerta1</rdf:li>
<rdf:li>puerta2</rdf:li>
</rdf:bag>
</fipa:argument>
</fipa:Action>
</rdf:RDF>)
:language fipa-rdf0)
```

Figura 23 Mensaje request con una acción en FIPA-RDF

Y el subsecuente mensaje de respuesta podría ser como lo muestra la figura 24.

```
(inform :sender Juan
:receiver Mary
:content ( <?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:fipa="http://www.fipa.org/schemas#">
<rdf:Description about="#JuanAction1">
<fipa:done>true</fipa:done>
<fipa:result>puertas cerradas</fipa:result>
</rdf:Description>
```

```
</rdf:RDF> )
:language fipa-rdf0)
```

Figura 24 Mensaje *Inform* de respuesta a la acción

3.4 COM, DCOM y ActiveX

3.4.1 Resumen de tecnologías COM

COM es un modelo de componentes de software independiente del lenguaje diseñado por Microsoft para permitir la interacción entre componentes de software y las aplicaciones. COM es una especificación que define un estándar de interfaces binarias entre módulos de objetos. Esta interfaz define una metodología de llamado de funciones, una técnica estándar de paso de datos basado en estructuras y además llamadas a funciones estándar. Usar COM significa que no importa cuál lenguaje de programación se utiliza para desarrollar módulos de la aplicación; la interfaz para este módulo es la misma en el nivel binario. Microsoft extiende esta tecnología con ActiveX, el cual inicialmente es usado para el desarrollo en Intranets (Delphi, 1999).

El aspecto clave de COM es que permite la comunicación entre componentes, entre aplicaciones, y entre clientes y servidores a través de interfaces definidas claramente. Las interfaces dan una forma para que los clientes pregunten a un componente COM qué características soporta en tiempo de ejecución. Para proveer con características adicionales a los componentes simplemente se añade una interfaz adicional para dichas características.

COM es una especificación y una implementación. La especificación COM define la creación de objetos y la comunicación entre objetos. De acuerdo con esta especificación, los objetos COM pueden ser escritos en diferentes lenguajes, correr en diferentes espacios de procesos y en diferentes plataformas. Conforme los objetos se adhieren a la especificación, ellos se pueden comunicar. Esto permite integrar código legado como un componente con nuevos componentes implementados en lenguajes orientados a objetos.

La implementación COM es la biblioteca COM (incluyendo OLE32.dll y OLEAut32.dll) la cual ofrece un número de servicios centrales que soportan la especificación escrita. La biblioteca COM contiene un conjunto de interfaces estándar que definen la funcionalidad central de un objeto COM, y un pequeño conjunto de API de funciones diseñadas para el propósito de crear y manejar objetos COM.

Conforme COM se ha desarrollado, ha sido extendido más allá de los servicios básicos. Sirve como la base para otras tecnologías como Automatización, controles ActiveX, Páginas de Servidor Activo (ASP, *Active Server Pages*) y Documentos Activos.

En suma, se pueden crear objetos COM que pueden trabajar dentro del ambiente *Microsoft Transaction Server* (MTS). MTS es un sistema de procesamiento de transacciones basadas en componentes para construir, desplegar y administrar grandes aplicaciones de servidores de Internet e intranet. Aunque MTS no es parte de la arquitectura de COM, está diseñado para extender sus capacidades en ambientes distribuidos grandes.

Cuando se implementa una aplicación COM se tienen los siguientes elementos: una interfaz COM, un servidor COM y un cliente COM.

Una interfaz COM es la forma en la que un objeto expone sus servicios externamente a los clientes. Un objeto COM ofrece una interfaz para cada conjunto de métodos relacionados (funciones miembro) y propiedades (datos miembro y/o contenido). Los clientes COM se comunican con los objetos a través de interfaces COM. Las interfaces son grupos de rutinas relacionadas lógica y semánticamente las cuales proveen comunicación entre un proveedor de un servicio (objeto servidor) y sus clientes. Los objetos pueden tener múltiples interfaces, donde cada interfaz implementa una característica. Una interfaz da una forma conveniente para que el cliente conozca los servicios que un objeto ofrece, sin dar detalles de la implementación de cómo o dónde el objeto ofrece este servicio. Las interfaces pueden ser redirigidas por COM a través de *proxies* para permitir la invocación de métodos a través de hilos, procesos y máquinas en la red.

Un servidor COM es un módulo, ya sea una dll (*dynamic link library*), exe u ocx que contiene el código para un objeto COM. Las implementaciones de los objetos residen en los servidores COM. Un objeto COM implementa una o más interfaces. Un servidor COM es una aplicación o biblioteca que provee servicios a una aplicación cliente o biblioteca. Un servidor COM consiste de uno o más objetos COM, donde un objeto COM es un conjunto de propiedades (datos miembro o contenido) y métodos (o funciones miembro). Los clientes no conocen la manera en que un objeto COM realiza sus servicios, la implementación del objeto permanece encapsulada. Un objeto hace disponibles sus servicios a través de sus interfaces.

Un cliente COM es el código que llama las interfaces para obtener los servicios requeridos desde el servidor. Los clientes saben lo que ellos quieren obtener desde un servidor (a través de la interfaz). Es importante diseñar una aplicación COM donde los clientes puedan consultar las interfaces de un objeto para determinar lo que un objeto es capaz de ofrecer. Un cliente típico es un controlador de automatización. Este es la parte de una aplicación que tiene una amplia perspectiva acerca del uso probable de la aplicación. Este conoce los tipos de información que necesita desde diferentes objetos en el servidor, y solicita los servicios cuando son necesarios.

3.4.2 DCOM

Las aplicaciones pueden acceder a los componentes COM y sus interfaces que existen en la misma computadora que la aplicación, o que existen en otra computadora sobre la red usando un mecanismo llamado COM Distribuido o DCOM.

DCOM (*Distributed Component Object Model*) de Microsoft extiende COM para soportar la comunicación entre objetos sobre diferentes computadoras (ya sea en una red de área local (LAN), una red de área amplia (WAN) o el Internet). Con DCOM una aplicación puede ser distribuida en las localidades en donde sea necesario para los clientes y para la aplicación (DCOM, 1998).

En el centro de COM se encuentran los mecanismos para establecer conexiones a componentes y para crear nuevas instancias de componentes. Estos mecanismos son comúnmente conocidos como mecanismos de activación.

La biblioteca COM busca el código binario apropiado (dll o exe) en el registro del sistema, crea el objeto y retorna un apuntador de interfaz a quien lo solicitó. Para DCOM, el mecanismo de creación de objetos de la biblioteca COM es extendido para permitir la creación de objetos en otras máquinas. Para poder crear objetos remotos, la biblioteca COM necesita conocer el nombre del servidor en la red. Una vez que el nombre del servidor y el identificador de la clase son conocidos, una parte de la biblioteca COM llama al administrador de control de servicios (SCM, *Service Control Manager*), sobre la máquina cliente; luego, se conecta al SCM del servidor y solicita la creación del objeto.

DCOM ofrece dos mecanismos fundamentales para permitir a los clientes indicar el nombre del servidor remoto cuando un objeto es creado.

- Como una configuración fija en el registro del sistema o en la clase de almacenamiento de DCOM.
- Como un parámetro explícito al solicitar la creación del objeto.

DCOM utiliza el protocolo conocido como RPC de objetos u ORPC. Este es un conjunto de definiciones que extienden el protocolo estándar DCE (Distributed Computing Environment) RPC. Ha sido diseñado específicamente para el ambiente orientado a objetos DCOM, y especifica cómo son construidos a través de la red y cómo son representadas y mantenidas las referencias a objetos.

Los programadores, en la mayoría de las veces, no tienen que trabajar en el nivel de ORPC. El compilador del lenguaje de definición de interfaces de Microsoft (MIDL, *Microsoft Interface Definition Language*) puede ser utilizado para generar automáticamente el código que se necesita para transferir datos a través de la red, basado simplemente en un archivo de definición de interfaces. Estrictamente hablando, MIDL no pertenece a DCOM y cualquier herramienta puede ser usada para generar el código de empaquetado de datos a través de la red.

3.4.3 ActiveX

3.4.3.1 Reseña histórica y evolución de ActiveX

La historia de ActiveX realmente comienza con DDE (*Dynamic Data Exchange*). Microsoft estaba buscando un método de mover datos de un lugar a otro dentro de Windows y éste fue su primer intento. DDE permite a las aplicaciones de Windows intercambiar datos a través del *clipboard*. DDE realmente tiene dos partes: intercambio de datos y lenguaje de macros. La parte de intercambio de datos de la especificación ha sido grandemente reemplazado por OLE (*Object Linking and Embedding*). Sin embargo, la porción de macros es usada todavía (Mueller, 1997).

El siguiente paso en el proceso de crear la conexión e intercambio de datos desde la perspectiva de Microsoft fue OLE; se necesitaba algo mejor a DDE. El proceso de copiar una pieza de datos al *clipboard* y después pegarlo en un documento dejó a los usuarios con una buena cantidad de documentos desactualizados. OLE es construido sobre los fundamentos creados para DDE; este usa el *clipboard* como una localidad de datos intermedia. Esto trae ventajas sobre DDE; una de las más importantes es que se puede crear un enlace dinámico para que los cambios en el documento fuente sean reflejados en el documento destino. Esto es debido a que en realidad se copia un objeto y no datos al *clipboard*.

Cada aplicación que soporta OLE es un cliente, un servidor o una combinación de los dos. Un cliente siempre actúa como un contenedor de objetos que éste recibe desde un servidor. El cliente no necesita saber qué hacer con estos objetos, esto es responsabilidad del servidor. La responsabilidad del cliente es almacenar los objetos en una forma tal que éste los pueda obtener intactos después. El cliente establece contacto con el servidor y solicita los datos que necesita. El servidor normalmente coloca los datos en el *clipboard.*

OLE también ofrece dos métodos para el intercambio de información: *linking* (ligado) y *embedding* (incrustado). *Linking* crea un apuntador a los datos contenidos en el documento fuente sin que el documento destino tenga una copia de los datos. *Embedding* coloca una copia de los datos, en realidad del objeto de datos, en el documento destino.

Existen dos versiones de OLE ahora. Microsoft introdujo OLE 1 como parte de Windows 3.x. Este ofrecía un conjunto básico de características para los procesos de *linking* y *embedding*. Uno de los más grandes problemas con esto era la gran cantidad de memoria que OLE requería. OLE 2 aparece supuestamente para corregir algunos de los problemas asociados con OLE 1 y provee mucho más funcionalidad.

En resumen, en OLE si se quería intercambiar información entre dos aplicaciones se tenía que encontrar una forma de comunicación entre ambas. Esto significaba cargar ambas aplicaciones. Ahora, ¿qué pasa si simplemente se cargan las piezas de la aplicación servidora requeridas para desplegar y editar un objeto de datos? Esta es parte del ímpetu junto al diseño de OCX (*OLE Control eXtension*). OCX es una dll con algo de OLE. Es una pieza de una aplicación que permite la transferencia de datos.

OCX son objetos y dll’s combinados. Un OCX provee un conjunto de rutinas para el manejo de objetos que un programador puede usar sin preocuparse acerca de la especificación de OLE. Dibujando un control OCX en una forma permite reusar código e incrementar la velocidad de programación; asignando valores a sus propiedades y añadiendo código a sus eventos, es decir, dan una gran facilidad de uso para los desarrolladores de aplicaciones.

3.4.3.2 Controles ActiveX

ActiveX es una forma avanzada de OCX. Sin embargo, OLE es la visión del usuario de ActiveX. Para un programador, ActiveX es también un conjunto de tecnologías que lo habilitan para el Internet. Para realmente apreciar ActiveX como programador se debe

mirar también a OLE desde la perspectiva del programador; esto significa fijarse en OCX. OCX es más que sólo intercambio de datos.

Un control ActiveX es una pieza de funcionalidad empacada en una forma que puede ser usada como un componente sobre una forma típica en los lenguajes de programación visuales. Una aplicación de Visual Basic puede ser desarrollada y probada en el IDE de Visual Basic. Un control ActiveX puede también ser desarrollado y probado dentro del mismo ambiente, pero existe una diferencia importante. Los controles no se pueden ejecutar directamente. En vez de ello, se debe colocar el control en una forma y después ejecutar el proyecto que contiene dicha forma (Petroutsos, 1999).

Cuando se desarrollan controles ActiveX, el programador debe pensar desde dos puntos de vista. Como el desarrollador del control y como desarrollador de aplicaciones que usan el control. Como desarrollador del control es posible ver los mecanismos internos del control. Como desarrollador de aplicaciones que usan el control se pueden ver los miembros del control (propiedades, métodos y eventos) sin preocupación acerca del código del control ni la forma en que fue construido.

Para simplificar el desarrollo de controles ActiveX, Microsoft usa la metáfora de la forma. Un control ActiveX es básicamente una forma, en la cual se pueden colocar controles, añadir código, y en general añadir funcionalidad de manera similar a lo que se hace en una aplicación normal de Visual Basic. Debido a que los controles ActiveX deben ser usados por los desarrolladores, ellos no tienen una interfaz de usuario. En vez de eso, proveen propiedades que el programador puede manipular para cambiar la apariencia del control, métodos que el programador puede usar para invocar las funciones del control, y eventos que el programador puede usar para programar la reacción del control en varias condiciones.

Los programadores se benefician del uso de COM. No se necesita saber acerca del trabajo interno del control. Los únicos factores importantes son los servicios que el control ofrece y cómo es posible interactuar con este.

Los controles ActiveX son los antes llamados controles OLE. Estos son componentes que aparecen en la barra de herramientas y que pueden ser colocados en cualquier forma para añadir alguna funcionalidad específica a alguna aplicación.

Los controles ActiveX son controles visuales que corren únicamente en servidores *in-process* y pueden ser conectados en una aplicación contenedora de controles ActiveX. No hay aplicaciones completas en ellos mismos, pero pueden ser pensados como controles OLE prefabricados que son reusables en varias aplicaciones. Los Controles ActiveX hacen uso de la automatización para exponer sus propiedades, métodos y eventos. Las características de los controles ActiveX incluyen la habilidad de disparar eventos, conectarse a fuentes de datos y soporte de licencia. Un uso común incremental de los controles ActiveX es sobre los sitios web como objetos interactivos en una página web. Como tal, ActiveX se ha vuelto un estándar que ha sido destinado especialmente al contenido interactivo para el *world wide web*, incluyendo el uso de documentos ActiveX usados para ver documentos no-html a través de un *browser* de web.

Un control ActiveX incluye muchos elementos cada uno de los cuales realiza una función específica. Los elementos incluyen propiedades, eventos, métodos y una o más bibliotecas de tipos asociadas.

Una biblioteca de tipos contiene la definición de los tipos para el control. Esta información de tipo, la cual provee más detalles acerca de la interfaz, da una forma para que los controles promuevan sus servicios para las aplicaciones cliente. Cuando se construye un control ActiveX la información de la biblioteca de tipos es compilada automáticamente en una dll del control como un recurso.

Una propiedad es un atributo. Un método es una solicitud a un control para realizar una acción. Un evento es una notificación desde un control para su contenedor de que algo ha pasado.

CAPÍTULO 4. MODELO CONCEPTUAL DE AGENTES

Resumen

Comprende el desarrollo de un conjunto de ideas y un modelo que sirve como punto de partida para la creación de agentes y SMA. Aquí se describe el modelo de agentes y SMA que servirá para la implementación de las herramientas. Se hacen las definiciones generales y se descirben las características de comportamiento de los agentes. A su vez, se mencionan los elementos internos que conforman a un agente, estos son: hechos, acciones, elementos de contenido, recepción y envío de mensajes ACL.

Además, se desarrolla un esquema general de la arquitectura de las aplicaciones que se pueden implementar considerando de manera más acoplada la plataforma CAP, los agentes que participan en la aplicación y los agentes de ontologías, esto a través de la comunicación de los agentes usando un lenguaje de comunicación de agentes.

Objetivos del capítulo

Establecer las ideas fundamentales que los agentes utilizan a través de las herramientas de programación propuestas.

Definir el ambiente y tipo de aplicaciones hacia los cuales se enfoca este trabajo con los agentes.

4.1 Introducción

Antes de utilizar la tecnología de controles ActiveX como base para la creación de agentes que corran sobre la plataforma CAP, es necesario definir los términos y conceptos que se van a manejar como parte de los agentes de software. Se debe establecer el marco conceptual desde la perspectiva de lo que se ha llamado paradigma de agentes para, por un lado, justificar su estudio en este trabajo y por otro, circunscribir en este ámbito las ideas que se pretenden desarrollar.

Para empezar con este tema, es importante mencionar que el objetivo de este trabajo no es proponer un marco general de agentes, que considere explícitamente alguno o varios de los diferentes enfoques que se han desarrollado en la teoría de agentes y SMA. Más bien, prevalece la idea de dar seguimiento a las especificaciones de FIPA para lograr una infraestructura completa para el desarrollo de SMA. Aunque como parte del modelo de agentes, como se verá más adelante, se considera una capa deliberativa.

¿Porqué definir agentes de esta forma? Siguiendo la especificación de FIPA es una buena forma de empezar por dos razones principalmente:

- La terminología de agentes permanece abierta a la implementación.
- Permite centrarse en utilizar la tecnología de controles y componentes ActiveX como la base para las aplicaciones de agentes.

En el primer caso, ayuda hacerlo de esta manera ya que FIPA no especifica los detalles de la definición e implementación del término agente, llegando a lo más a definir teóricamente su concepto abstracto. Pero sí especifica la manera en que el software de agente interactúa con una plataforma FIPA. Esta perspectiva es muy útil por que no ata a las implementaciones de agentes con aspectos como el cúmulo de teorías, formalismos, tipologías de agentes y arquitecturas que se han desarrollado para tratar de implementar agentes y SMA. Además, esto es de mucha ayuda si consideramos que la mayor parte de los trabajos que soportan dichos aspectos, no considera los estándares de FIPA en ninguna forma explícita, así es que por lo general su implementación representa un punto de vista muy particular del grupo que lo ha desarrollado. En ese sentido, carece de importancia para esta investigación. Con esto de ninguna manera se trata de quitar mérito a los logros alcanzados en la investigación sobre teoría de agentes, ya que hay que reconocer que muchas de las ideas que FIPA adoptó para sus especificaciones se basan en estos trabajos y que por medio de este esfuerzo se ha logrado, de alguna manera, integrar diferentes temas de agentes en un cuerpo de especificaciones y estándares estructurado.

Por otra parte, esto permite centrarse en la tecnología de ActiveX como base para la implementación de agentes, más que preocuparse por armar un marco de agentes general complejo y difícil de implementar y de utilizar, tal y como ha sucedido con otros trabajos en esta área. Por ello se propone un concepto de agente sencillo, pero suficiente en el cumplimiento de la especificación de FIPA que permite el desarrollo de los temas principales que se considera parte de todo agente FIPA, y a su vez, permite utilizar la tecnología de ActiveX para su implementación y experimentación; esto trae la ventaja de

que al final se puede tener una infraestructura más fácil de utilizar por los programadores de aplicaciones de agentes.

4.2 Definición de agente

Un agente es el actor fundamental de una plataforma de agentes que combina una o más capacidades de servicios en un modelo de ejecución integrado y unificado que puede incluir acceso a software externo, usuarios humanos y facilidades de comunicación. Un agente puede tener también ciertas capacidades de extracción de recursos para acceder al software.

Un agente debe tener al menos un dueño, por ejemplo, basado en una afiliación organizacional o una propiedad del usuario humano, y puede soportar varias nociones de identidad. Un identificador de agente lo etiqueta para que pueda ser distinguido sin ambigüedad en el universo de agentes.

Definido por FIPA, un agente es todo aquel proceso computacional que implementa la funcionalidad autónoma y de comunicación de una aplicación (FIPA-1, 2000). En este contexto, representa una entidad de software encapsulada con su estado propio, comportamiento, hilo de control, y la habilidad para interactuar con otras entidades. El paradigma de interacción de agentes les permite mediar, colaborar y cooperar para alcanzar sus metas. Un agente puede ser realizado de varias formas, por ejemplo, como un componente de Java, un objeto COM o un programa autocontenido de Lisp. Este se puede ejecutar como un proceso nativo en alguna computadora física bajo un sistema operativo, o ser soportado por un intérprete como una máquina virtual de Java. La relación entre el agente y su contexto computacional es especificada por el ciclo de vida del agente.

Para este trabajo en particular, llamaremos agente a *aquella entidad de software autónoma que tenga la capacidad de interactuar con otros agentes enviando mensajes comunicativos utilizando un lenguaje de comunicación de agentes a través de CAP y que estén soportados sobre el modelo COM en su modalidad de controles ActiveX*. En un sentido más amplio, se les puede ver como proveedores y clientes de servicios. Así mismo, la autonomía la podemos ver como la capacidad de los agentes para llevar a cabo sus acciones dinámicamente y sin tener predefinido previamente con qué agentes ni en qué momento va a interactuar. Esto es el resultado de la interacción con otros agentes y facilita en gran medida la colaboración distribuida y flexible entre los agentes.

Un hecho importante para establecer la definición anterior es que un agente está basado en el modelo de componentes COM, en particular con las interfaces de funcionalidad mínima de un control ActiveX. Esto es necesario si se quiere lograr uno de los objetivos más importantes de este trabajo que es proporcionar una herramienta que facilite la forma de implementar aplicaciones basadas en Windows (y sus lenguajes de programación) junto con el paradigma de agentes. Además, la definición se refiere a agentes que se comunican a través de la plataforma de agentes componentes CAP que está basada en las especificaciones de FIPA.

Como parte de la definición de agente, este tiene un soporte tecnológico. Conceptualmente los agentes pueden estar formados por diferentes capas, cada una definiendo una arquitectura de comportamiento. En el caso particular del modelo que se presenta, el comportamiento de los agentes está dividido en dos grandes arquitecturas: comportamiento básico y comportamiento deliberativo (Figura 25).

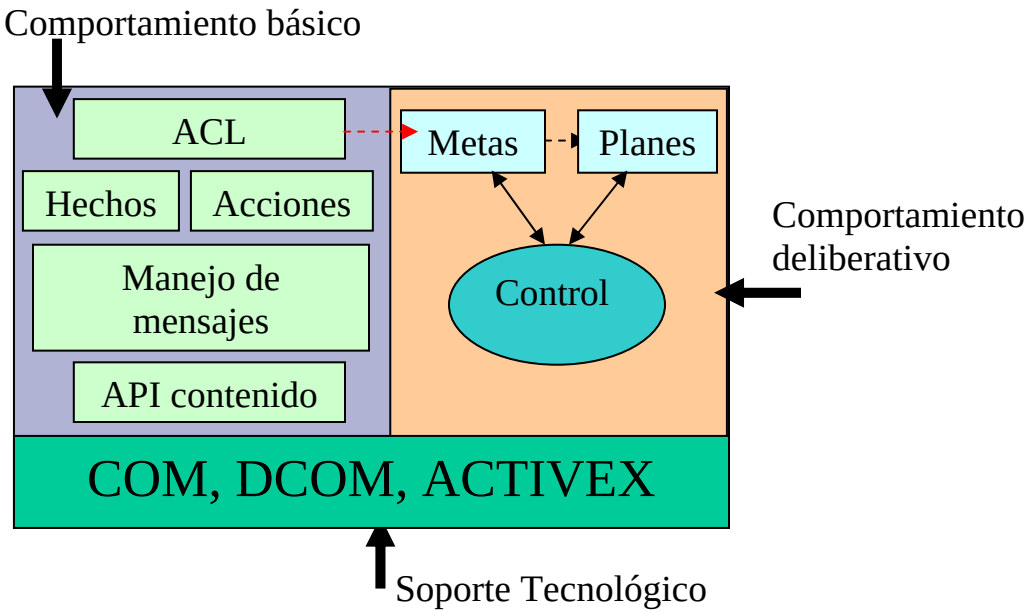

Figura 25 Capas de un Agente

En la capa de comportamiento básico un agente mantiene una base de conocimientos (formada por hechos y acciones), acceso a los mecanismos de comunicación por medio de un ACL (envía y maneja mensajes entrantes), y además contiene un API de clases para manejar el conocimiento del agente y para crear y manipular el contenido en los mensajes ACL. En la sección 4.3 se describe el comportamiento básico de un agente.

Adicionalmente, el esquema conceptual de agente presentado anteriormente incluye una capa de comportamiento deliberativo. Existen muchos enfoques y modelos que definen este tipo de comportamiento en agentes de software basados en definiciones fuertes de la IA como los de creencias, deseos, intenciones o compromisos (Cohen & Levesque, 1990). En esta ocasión se propone al uso de agentes basados en metas y planes para implementar ésta funcionalidad basada en modelos deliberativos para agentes inteligentes. En este sentido, al estar dirigidos por metas, debe existir un mecanismo de control que permita al agente decidir la selección de las metas y las acciones que forman parte del plan para lograr alcanzar la meta seleccionada, en (Shoham, 1990) se presenta más información en este tema. También podría ser necesario contemplar mecanismos de revisión de creencias y el establecimiento de esquemas lógicos para el razonamiento acerca del conocimiento y las actitudes mentales del agente.

Para la implementación de esta parte del modelo es necesario establecer las bases a través de alguna arquitectura de agentes deliberativos; esta no es una tarea sencilla y para ello se requiere un análisis más profundo acerca de esa arquitectura. Por lo tanto, no está

considerada para ser implementada durante el desarrollo de esta tesis y parte del trabajo futuro debe considerar la posibilidad de desarrollar esta capa para los agentes.
Los agentes están considerados para operar en ambientes abiertos y flexibles. La característica fundamental del ambiente es que por medio de este es posible encontrar a los demás agentes que dinámicamente conformarán el SMA. Los agentes pueden modelar y manipular su ambiente explícitamente a través de su base de conocimientos.

Los mecanismos de percepción de los agentes están dados por sus capacidades de comunicación e interacción con los demás, en particular, con su lenguaje de comunicación. Además, junto a las capacidades de percepción, los agentes las combinan con sus habilidades para colaborar con otros, ofreciendo y solicitando servicios y manipulando el conocimiento del ambiente que poseee. La base de la colaboración es la interacción a través del intercambio de mensajes comunicativos con los demás agentes que se encuentran en su entorno.

4.3 Descripción del comportamiento básico

Los agentes de software pueden ser construidos sobre un conjunto de estructuras de datos y paquetes de software. El agente puede manejar sus estructuras de datos internas a través de un conjunto de funciones bien definidas. El estado de un 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, un agente puede tener asociado un conjunto de acciones, con sus precondiciones y efectos respectivos. Un programa de agente especifica bajo qué condiciones una acción es realizada por el agente (Subrahmanian et al., 2000).

Como se aprecia en el diagrama de la figura 26, un agente puede tener una base de hechos. Un agente detecta el momento en que recibe mensajes ACL. Por medio del mecanismo de percepción puede determinar el tipo de mensaje que le llega y en base en ello, internamente dispara sus eventos programados para atender cada tipo de mensaje ACL. Al realizar esto, cada agente puede llevar a cabo varias cosas: primero, puede ir a su base de hechos para actualizarla si es que así conviene a la aplicación en la que está participando. En segundo lugar, puede solicitar la invocación de una acción para lograr algún propósito; y por último, puede requerir comunicarse con otro agente para lo cual necesita enviar mensajes comunicativos usando ACL.

El agente puede generar y enviar mensajes ACL en cualquier momento a través de una plataforma de agentes CAP, incluso fuera de sus propios eventos y acciones. En cualquier caso, puede utilizar el API para crear el contenido de los mensajes (APICont en la figura 26). Eso sí, todos los mensajes que llegan al agente deben ser procesados por el mecanismo interno del agente; este los analiza y determina el evento que se dispara de acuerdo al tipo de mensaje que se trata.

Del conjunto de acciones que tiene el agente, existen de diferentes naturalezas: hay acciones que le permiten al agente interactuar con la plataforma y solicitarle servicios a través de estas.

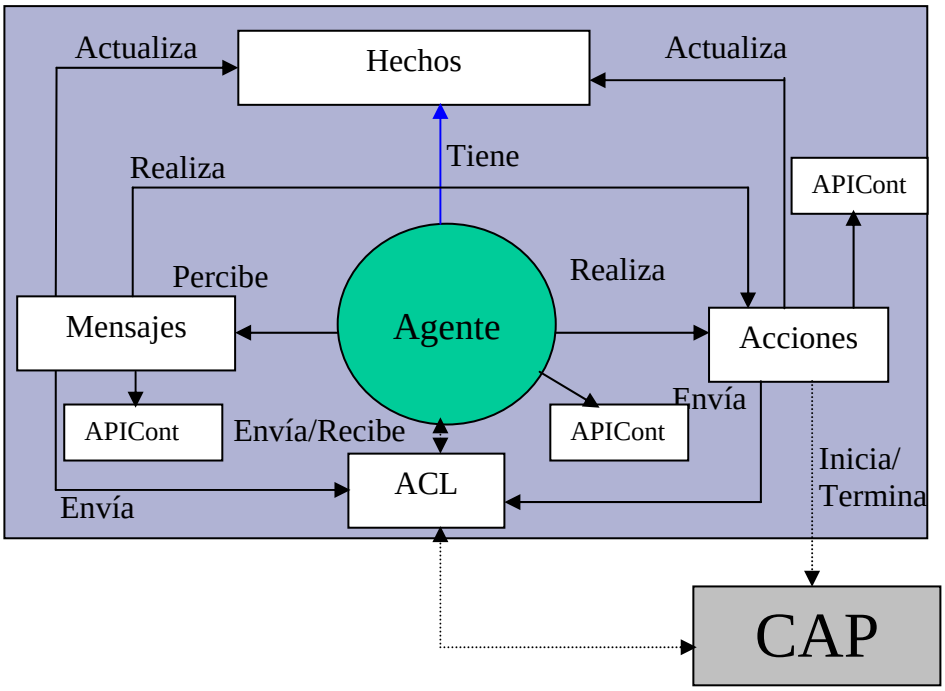

Figura 26 Modelo de agente colaborativo

Por lo general estas acciones no son publicadas como servicios en la plataforma, forman parte de la funcionalidad básica del agente y su invocación no requiere del uso de mensajes ACL para realizarse y se encargan de aspectos como la inicialización y finalización de los agentes en CAP. Otro grupo de acciones son públicas en el sentido que pueden ser solicitadas por otros agentes a través de la interacción en un SMA en el que participa el agente que las posee. Además, cada agente mantiene una lista de acciones comunicativas que puede utilizar para interactuar con los demás.

4.4 Arquitectura General de los SMA

En SMA, los agentes de software representan servicios distribuidos heterogéneos que interactúan usando un lenguaje de comunicación de agentes. Los SMA son un poderoso paradigma para acceso e integración de servicios distribuidos heterogéneos en ambientes abiertos. Aquí, proveedores de servicios, usuarios de servicios e intermediarios (posiblemente) son representados como agentes de software, los cuales interactúan usando un lenguaje de comunicación de agentes, basado en *speech acts* (Searle, 1969) y de preferencia compartiendo vocabularios llamados ontologías. El uso de un ACL le permite a los agentes interactuar usando información semánticamente rica.

En un mundo heterogéneo, se puede diseñar, desarrollar y realizar esta versión de agentes de diferentes formas, al menos en las fases de investigación iniciales. El desarrollo distribuido concurrente lleva a muchos tipos de SMA que representan conjuntos de funcionalidad que se pueden desarrollar con infraestructuras que se implementan siguiendo

estándares. En ese sentido FIPA y OMG están definiendo estándares para desarrollar aplicaciones industriales con agentes (Poslad, Buckle & Hadingham, 2001).

Un sistema distribuido es considerado abierto si es extensible. Existe un rango de apertura así como de diseños y modelos. La apertura está ligada a la reconfigurabilidad. Si las interfaces al sistema están explícitamente definidas y partes del sistema están débilmente acopladas entonces las partes pueden ser cambiadas y mejoradas. La reconfigurabilidad dinámica es un aspecto clave de los sistemas escalables así como sucede en una plataforma de agentes donde los facilitadores de comunicación acoplados con protocolos de paso de mensajes guían a un soporte natural para una arquitectura de servicios abierta. Esto permite que los servicios sean manejados dinámicamente. Dependiendo de los tipos de interacción, los servicios enlazados entre diferentes partes podrían ser modificados dinámicamente durante el despliegue de un sistema. Este soporte requiere de un medio común de representar, comprender e intercambiar servicios de información. Una plataforma compatible con FIPA toma en cuenta estos detalles para la implementación de sistemas de agentes.

Esto lleva a considerar como parte fundamental en este tipo de infraestructuras y de sistemas el tema de la colaboración multiagente a nivel de conocimiento. Por lo general, un agente necesita realizar una tarea que requiere que este colabore con otros agentes. Para hacer esto, usa el facilitador para descubrir los agentes con las habilidades requeridas, y el servidor de nombres de agentes para determinar la dirección de estos agentes. El lenguaje de comunicación interagente es usado para comunicarse con el servidor de nombres de agentes, el facilitador y otros agentes. La comunicación requiere una representación compartida y la comprensión de conceptos de dominio común, por ejemplo una ontología común (Genesereth & Ketchpel, 1994). Este esquema se ilustra en la figura 27.

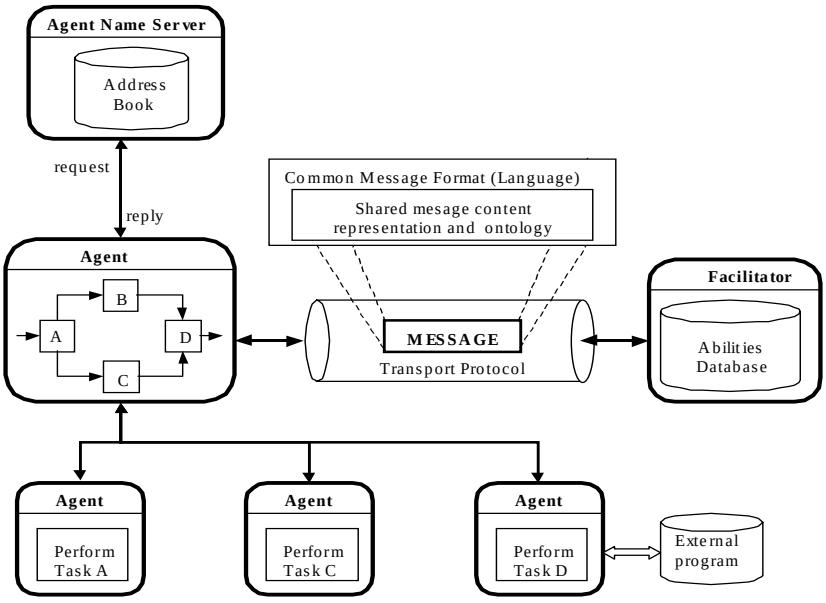

Figura 27 Colaboración multiagente a nivel de conocimiento

A continuación se presenta una definición informal de la arquitectura general de una aplicación que utilice el esquema de agentes componentes que en este trabajo se utiliza. Se dice que es un esquema informal, ya que su finalidad principal es poner más en claro el panorama de las aplicaciones que se desean desarrollar utilizando el paradigma de agentes y las herramientas que se están proponiendo aquí, sin tratar de entrar en detalles de su implementación (figura 28).

Por medio de esta arquitectura es posible comunicar agentes que se encuentren ejecutándose en diferentes aplicaciones, incluso en diferentes máquinas o en Internet. Distintas aplicaciones pueden contener un número indeterminado de agentes. Cada agente se registra en una plataforma de agentes (que puede ser local o remota) y por medio de ésta, los se pueden comunicar mensajes utilizando FIPA-ACL. Cada agente declara ante un facilitador de directorios o DF, los servicios que es capaz de ofrecer a los demás que lo soliciten, para fines de facilitar la comunicación entre agentes en el contexto de una misma aplicación o en aplicaciones heterogéneas entre sí.

En un sistema abierto, como el *world wide web*, pueden existir un buen número de agentes y de diferentes tipos sin un conocimiento previo de futuros colaboradores con los que se va a comunicar para llevar a cabo sus tareas. Este escenario trae consigo varios problemas que es importante considerar: La concurrencia es el primero. Un sistema de agentes distribuidos es concurrente por definición. Las estructuras de control secuencial simples no son suficientes para controlar la interacción entre agentes. Por esto, se deben considerar estructuras de coordinación para manejar problemas de sincronización, tolerancia a fallos y asincronía de mensajes.

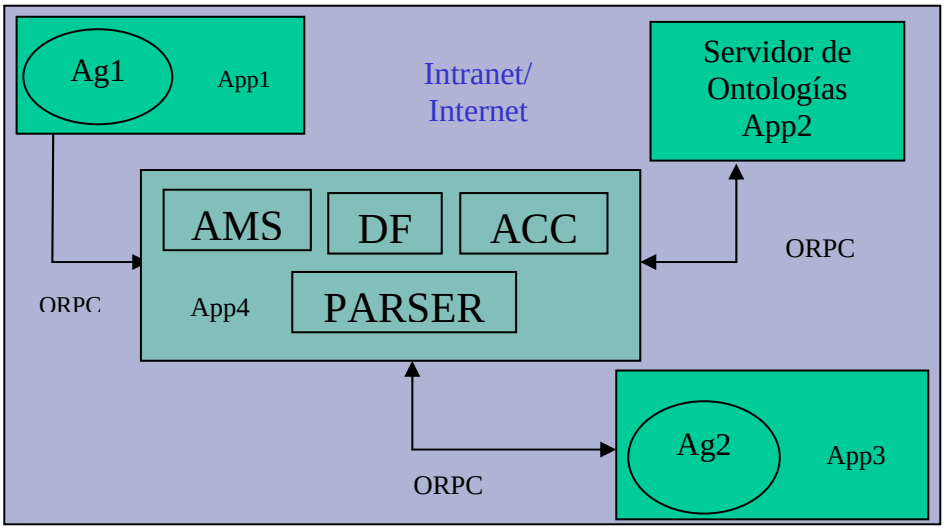

Figura 28 Arquitectura general de aplicaciones

Otra consideración importante que aparece con esta arquitectura general es la aparición de grupos de agentes interactuando. La coordinación de estos conjuntos de agentes en ambientes abiertos requiere de una forma especial de programación del sistema que los utiliza (Tambe, 1997) (Alvarado et al., 2002). En (Jamison & Lea, 1999) se establece que

se deben manejar protocolos que describan las acciones necesarias en conjuntos de agentes para llevar a cabo exitosamente las tareas de los agentes. Esta es la noción de grupos de agentes que se describe en (Jamison, 1998).

Un escenario puede ilustrar mejor el panorama particular de una aplicación que utilice los agentes componentes. Supongamos que tenemos una aplicación A que está implementada tal vez en un lenguaje como Visual Basic; esta aplicación contiene un agente llamado Ag1, el cual, está registrado debidamente a la plataforma de agentes y tiene acceso a los servicios de la plataforma de agentes AMS, DF, y ACC que le permiten la comunicación. Además, tal vez en otro programa y en otra computadora, tenemos una aplicación B, implementada quizá en Delphi o Visual C++, la cual contiene como uno de sus elementos un agente componente llamado Ag2, también registrado debidamente ante la misma plataforma de agentes que el agente Ag1. Ambos agentes pueden establecer una comunicación utilizando el ACL, conectándose a través del ACC de la plataforma para el transporte de éstos. De esta forma es posible tener aplicaciones distribuidas utilizando sistemas multiagente.

En la figura 28 se puede apreciar mejor el escenario antes descrito. Ahí aparece una aplicación especial llamada servidor de ontologías que está considerada como un programa independiente de las aplicaciones y de la plataforma CAP, que contiene un agente de ontologías que implementa la funcionalidad que FIPA establece para la utilización de ontologías de dominio entre los agentes y entre las aplicaciones de agentes. Por lo menos se requiere un agente de ontologías de servicios para propiciar el correcto entendimiento de la semántica de servicios solicitados y proporcionados entre los agentes. Implementar un agente de ontologías, siguiendo las especificaciones de FIPA, no es una tarea trivial, así que no se considera como una parte a desarrollar en esta investigación.

Los agentes pueden interoperar con software legado, es decir, software que no es de agentes pero que se requiere utilizar por el SMA para obtener sus servicios. Con relación a este punto, los agentes pueden utilizar los servicios de agentes *wrappers*, cuya función principal es permitir al software de agentes interactuar con el software legado y viceversa. Esta función puede darse de varias formas.

La primera, consiste en volver a escribir el software externo, para incorporar las capacidades de comunicación de agentes, que es la opción más costosa. La segunda considera la posibilidad de no tener el código fuente del software legado, y construir agentes que permitan la comunicación entre los dos programas que incluya el entendimiento de los protocolos de comunicación en ambos sentidos. La tercera opción para interactuar con software legado considera que se tiene el código fuente del software legado, de tal manera que es posible escribir piezas de código en éste para llevar a cabo la tarea de comunicación con los agentes. FIPA considera estas posibilidades dentro de la arquitectura abstracta de una plataforma de agentes. De igual manera, en la plataforma CAP se modela el uso de estos agentes especiales como parte de un SMA.

Por otra parte, los agentes (vistos como componentes de software) pueden ser accedidos por otros componentes que no son agentes, por medio de los mecanismos propios de la tecnología de componentes que se utilice. En el caso particular del modelo aquí considerado, los agentes controles ActiveX pueden ser accedidos por medio de otros

componentes COM desarrollados por otros programadores a través de las interfaces que cada control de agente ofrece públicamente. Esta es otra forma de interoperabilidad posible de los agentes.

CAPÍTULO 5. PLANTILLA DE AGENTE BÁSICO

Resumen

Se explica la plantilla de control de agente básico que se ha creado con el fin de implementar la funcionalidad básica que heredan todos los agentes que son creados bajo este esquema. Se mencionan las funciones y propiedades generales que permiten a los agentes controlar su operación interna y externa. Así mismo, se describen los mecanismos de control que le permiten a los agentes comunicarse, reaccionar a los mensajes recibidos y construir nuevos mensajes, entre otras características.

Objetivo del capítulo

Mostrar la implementación básica que es común a todos los agentes para controlar su estado interno y los mecanismos para interactuar en el entorno de la plataforma CAP.

5.1 Introducción

Como parte de las herramientas para la creación de controles de agente se ha desarrollado una plantilla de control de agente básico que a continuación se describe.

Esta plantilla consiste en un proyecto para control ActiveX de Visual Basic 6.0. Contiene los elementos básicos que se requieren para asegurar que todo agente que se cree con esta base pueda correr adecuadamente sobre la plataforma CAP.

El objetivo principal de tener este control básico es proporcionar la funcionalidad mínima que todos los controles de agente adoptan por medio del proceso de creación de agentes llevado a cabo a través de la herramienta CAP-AgentTool, descrita en el capítulo 6.

Dentro de las características más importantes que forman esta plantilla están las siguientes:

1.- Un conjunto de propiedades básicas para inicialización del agente.
2.- Eventos por medio de los cuales el agente puede percibir los mensajes ACL que están destinados para él. Uno para cada tipo de acto comunicativo soportado por FIPA-ACL.
3.- Métodos para su interacción tanto con la plataforma CAP, así como para el mantenimiento y consulta de sus propiedades y conocimiento.
4.- Una página de propiedades para la configuración inicial de las propiedades más importantes del agente en tiempo de diseño de la aplicación que lo contiene.
5.- Un formulario acerca de... para identificar cada clase de control de agente.
6.- Módulos de clase para el manejo del contenido de los mensajes ACL que están implementados.
7.- Controles constituyentes para la interfaz del control de agente.

Enseguida se va a explicar con mayor detalle los aspectos más importantes de cada una de las características antes mencionadas.

5.2 Propiedades para inicialización del agente

Algunas de las propiedades de los agentes tienen como finalidad mantener el estado interno con respecto a la plataforma CAP: por ejemplo las propiedades *registeredwithAms* y *status*. La propiedad *registeredWithAms* indica si el agente está actualmente registrado ante el AMS de la plataforma CAP.

En el caso de la propiedad *status*, sirve para indicar el status que guarda el agente en el AMS de la plataforma y puede ser alguno de los status especificados en el ciclo de vida de los agentes de FIPA.

Otras propiedades sirven para configurar la operación adecuada de cada agente. Entre estas se encuentran: *ListenIntervalo, plataforma, owner, controlOCX* y *rutaAgente*.

ListenIntervalo: Por medio de esta propiedad se le indica al agente que busque con cierta frecuencia el siguiente mensaje que hay para el agente en el ACC de la plataforma. Hay que

recordar que los agentes deben encargarse de ir a buscar los mensajes ACL a través de la invocación del método *getmessage* del componente ACC de la plataforma.

Plataforma: Indica el servidor CAP al que se debe conectar cada agente. Si el agente está corriendo en la misma máquina de la plataforma entonces esta propiedad no debe tener valor. Si el agente va a conectarse a una plataforma CAP remota entonces en esta propiedad se debe indicar la dirección o nombre del equipo servidor de la plataforma. Esta propiedad se utiliza como parámetro cada vez que se quiere registrar el agente en el AMS de la plataforma.

Owner: Es el propietario del agente, por lo regular sirve para hacer grupos de agentes que corresponden a una misma organización o sistema.

ControlOCX: En esta propiedad se guarda el nombre del control con extensión ocx del que se instanció el control de agente originalmente. Esto es importante para fines administrativos internos de cada agente.

RutaAgente: Indica la ruta en el sistema local donde se encuentra el control de agente desde el cual se instanció un agente. Igualmente, este dato permite administrar mejor el espacio requerido para cada instancia de agente que corre en el sistema. Es importante mencionar que los agentes requieren inicializarse a partir de su especificación en una base de datos que se encuentra en el lugar en donde fue creado o instalado el control de agente; y además pueden guardar alguna información propia que es importante durante su ejecución.

Cada agente tiene un conjunto de colecciones que le permiten guardar diferentes tipos de datos. A este tipo de propiedades pertenecen los objetos *acl*, *acciones*, *nicknames*, *ontologías*, *Lcontenido* y *KBHechos*.

Acl: Colección que tiene los nombres de los lenguajes de comunicación entre agentes que el agente puede manejar. Hasta este momento los agentes solo pueden procesar mensajes FIPA-ACL. Esto está muy relacionado con las capacidades de la plataforma CAP. Por *default* se entiende que los agentes que estamos desarrollando con estas herramientas utilizan el lenguaje FIPA-ACL. Pero la idea de este objeto es almacenar el nombre de otros lenguajes que pudieran ser añadidos a los agentes de tal manera que esto les ayude, en determinados momentos, a decidir la manera de procesar e interpretar los mensajes que les llegan, de acuerdo con el tipo de lenguaje especificado en los mensajes.

acciones: Colección para guardar internamente en el agente el conjunto de acciones públicas que puede realizar el agente. Este es el grupo de acciones que se especificaron durante la etapa de creación del agente a través de la herramienta CAP-AgentTool que más adelante se explica y las acciones comunicativas del agente.

Nicknames: Identificadores con los que se puede conocer a esta clase de agentes en su entorno de ejecución. Estos nombres se registran como parte de la descripción de los agentes en el DF.

ontologías: Grupo de ontologías especificadas para un agente. Con esta información se trata de proporcionar al agente un mecanismo para que conozca las diferentes ontologías que son parte del dominio de aplicación en los que puede operar cada clase de agente y de esta manera que sea más eficiente el proceso de búsqueda de los agentes de ontologías necesarios.

Lcontenido: Lenguajes de contenido que el agente puede utilizar para interpretar el campo *content* de los mensajes que recibe y envía. Hasta el momento solamente se ha implementado un lenguaje de contenido basado en FIPA-RDF0 (ver los detalles en la sección 7.3).

KBHechos: Base de conocimiento de hechos o proposiciones que el agente mantiene internamente. Esta es la suma de los hechos iniciales del agente y los hechos que han sido añadidos durante la ejecución de la aplicación.

Como parte de la arquitectura interna de los controles de agente se han agregado varios objetos con la finalidad de permitir a los controles de agente interactuar y utilizar los servicios de los diferentes componentes COM que conforman la plataforma de agentes CAP. En este punto se habla de los siguientes objetos que son miembros de cada control de agente: *ams*, *acc*, *df* y *parser*.

ams: Es un objeto que es obtenido a partir del servidor COM de CAP y sirve para instanciar el componente AMS de la plataforma al inicializar cada instancia de un control de agente. Permite utilizar los servicios que ofrece el AMS de CAP.

acc: Objeto que es obtenido a partir del servidor COM de CAP y sirve para instanciar el componente ACC de la plataforma que permite el envío y recepción de los mensajes de los agentes.

df: Permite a los controles de agente acceder a los servicios que ofrece el DF de CAP. Dentro de las principales actividades que se pueden realizar con el DF están las tareas de registrar servicios en el DF, buscar servicios que otros agentes ofrecen, y modificar la descripción de los servicios.

parser: La plataforma CAP ofrece este objeto COM que permite la creación y verificación de los mensajes ACL para su representación correcta. XML es el lenguaje de codificación de los mensajes y cada agente puede utilizarlo para generar y manipular mensajes.

5.3 Eventos para percibir los mensajes ACL.

Cada instancia de agente tiene un grupo de eventos que se disparan cuando se recibe cualquier tipo de mensaje ACL. Esto le sirve al agente para “escuchar” los mensajes que le llegan desde otros agentes. En la figura 29 se listan los diferentes eventos de los controles de agente y el tipo de mensaje que reciben.

Evento	**Tipo de mensaje ACL recibido**
HandleSucces	*Success*
HandleOther	Performative no conocido
HandlePropagate	*Propagate*
HandleProxy	*Proxy*
HandleInformRef	*Inform-ref*
HanldeInformIf	*Inform-if*
HandleSubscribe	*Subscribe*
HandleRequestWhenever	*Request-whenever*
HandleRequestWhen	*Request-when*
HandleRejectProposal	*Reject-proposal*
HandleRefuse	*Refuse*
HandleQueryRef	*Query-ref*
HandleQueryIf	*Query-if*
HandlePropose	*Propose*
HandleNotUnderstood	*Not-understood*
HandleInform	*Inform*
HandleFailure	*Failure*
HandleDisconfirm	*Disconfirm*
HandleConfirm	*Confirm*
HandleCfp	*Cfp*
HandleCancel	*Cancel*
HandleAgree	*Agree*
HandleAcceptProposal	*Accept-proposal*
HandleRequest	*Request*

Figura 29 Eventos para recibir mensajes ACL

Esta implementación para el manejo de los mensajes ACL es consistente con las especificaciones de FIPA en el sentido de que cada agente tiene la capacidad de percibir los mensajes ACL que le llegan de otros agentes, pero es responsabilidad de cada agente atenderlo de la forma que le sea más conveniente. De esta manera, cada implementación que se haga del agente (es decir, cada instancia del agente) puede tomar sus mensajes y procesarlos o no, incluso responder de la forma que le sea más conveniente a menos que esté participando en una conversación explícita o utilizando algún protocolo explícito de interacción; FIPA indica que cada agente es libre de responder a los mensajes que recibe.

5.4 Métodos internos del agente

La plantilla de agente básico proporciona un conjunto de métodos que son comunes a todos los controles de agente que se crean a partir de ésta. El objetivo central de tener estos métodos es dar la funcionalidad y comportamiento a los agentes para que puedan operar en el paradigma de agentes de software, utilizando para ello la base que le brinda la plataforma de agentes CAP (a través de sus componentes). Otro aspecto que queda incluido en esta plantilla es que se establecen los mecanismos básicos para el funcionamiento de los agentes

tanto en el entorno de controles ActiveX, y sobre todo, en su entorno de agentes de software como entidades comunicativas y colaborativas.

En esta parte se explican los métodos implementados para cubrir las expectativas arriba mencionadas. La meta final de este grupo de funciones y métodos es establecer la base funcional y operativa de los controles desde el punto de vista del paradigma de agentes que estamos investigando, desarrollando y probando actualmente.

El proceso de inicialización de los agentes por ejemplo, se lleva a cabo con su método AgInicializar, y se describe en el diagrama de actividades de la figura 30.

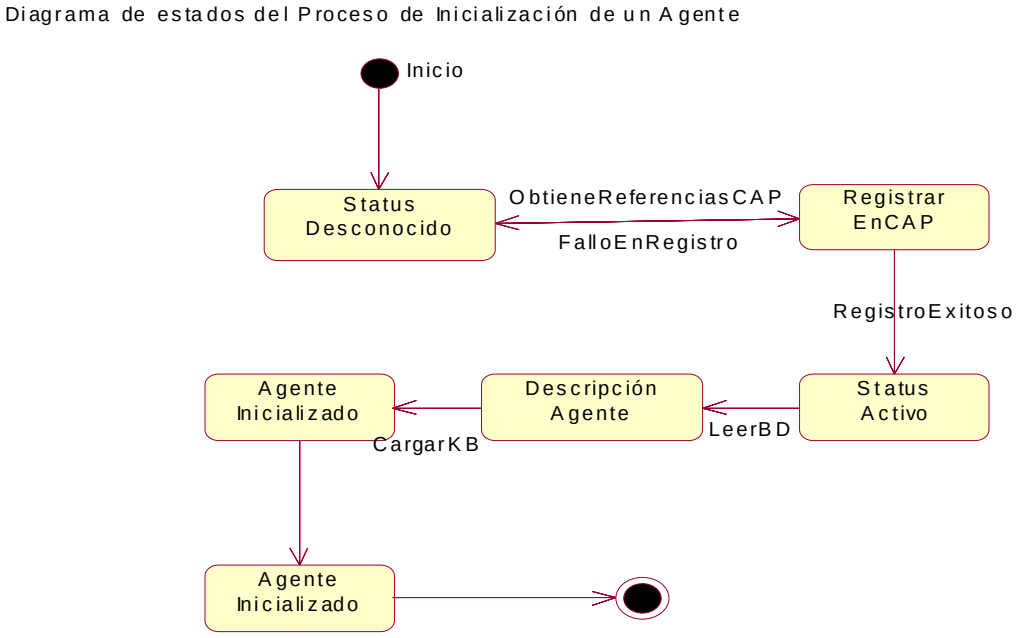

Figura 30 Diagrama de inicialización de un agente

En la siguiente tabla, figura 31, se explican todos los métodos que conforman la plantilla del control de agente básico.

Nombre del método	Descripción
RegisterwithDf	Permite registrar a un agente y sus servicios con el facilitador de directorios de la plataforma de agentes CAP.
RegisterwithAms	Registra la información del agente en el sistema administrador de agentes de CAP.
AgInicializar	Hace la tarea de inicializar el agente. Lleva a cabo una serie de tareas como registrar el agente en la plataforma y cargar la información inicial desde su especificación.
Forward	Envía un mensaje ACL
CargarKB	Lee la base de hechos del agente y los carga

	internamente en el agente.
CargarOntologias	Lee las ontologías que tiene especificadas el agente y las carga internamente en el agente.
CargarACL	Carga los lenguajes de comunicación de agentes que tiene especificado.
CargarLContent	Lee la especificación del agente para cargar los lenguajes de contenido que puede manejar.
Quit	Termina la ejecución del agente; esto se logra desregistrándolo con la plataforma CAP.
CargarNickNames	Lee los otros nombres con los que se puede conocer esta clase de agentes.
ExecNotUnderstood	En el contexto de un mensaje ACL recibido, permite contestar al agente emisor que el tipo de mensaje que ha solicitado no es entendido por el agente.
CargarAcciones	Lee de la especificación del agente las acciones que soporta y las incorpora internamente al agente.
ExecRefuse	En el contexto de mensajes ACL *request*, permite contestar rápidamente al agente emisor del *request* que la acción solicitada no va a ser realizada por el agente.
BuscarAccion	Busca en el agente las acciones que puede ejecutar.
InvocarAccion	En el contexto de mensajes ACL *request*, permite invocar la acción que se solicita en el campo *content* del mensaje. Si detecta que la acción fue ejecutada con éxito entonces envía un mensaje ACL *success*, si no tuvo éxito la acción, contesta con un *failure*. Si la acción solicitada no se encuentra en la base de acciones del agente entonces contesta con un mensaje *not-understood.*
EsActoComunicativo	Sirve para detectar si el campo acto de una acción solicitada al agente es un acto comunicativo de ACL.
BuscarHecho	Busca en la base de hechos de acuerdo a un cierto patrón de búsqueda especificado como un argumento del método.
AddHecho	Agrega un hecho a la base de hechos del agente.
BorraHecho	Elimina un hecho de la base de hechos del

	agente.
InicializarKB	Pone la base de hechos del agente de acuerdo con su especificación inicial.
ActualizaKB	Graba la base de hechos del agente en su archivo de base de datos que contiene la especificación individual de cada instancia de agente.

Figura 31 Métodos internos de un agente

5.5 Página de propiedades para configuración

La página de propiedades de la plantilla de control de agente básico está diseñada para permitir que el desarrollador interactúe con el control de agente utilizando cualquier interfaz de usuario que se defina, es decir, independientemente del contenedor de controles. Su razón es muy sencilla: no existe ninguna garantía de que un contenedor que utilice el control de agente tendrá una ventana propiedades para editar las propiedades del control. Para este tipo de contenedores, esta página de propiedades constituye la única forma de modificar las propiedades del control. En la figura 32 se puede ver la interfaz de la página de propiedades para los controles de agente que se proporciona en la plantilla; está en modo de diseño del control de agente. Contiene algunas propiedades que pueden servir para configurar el comportamiento del agente.

La página de propiedades contiene botones Aceptar, Cancelar y Aplicar, botones todos ellos utilizados por todas las páginas de propiedades. Esto sólo es posible observarlo en tiempo de ejecución del control de agente que contiene a la página de propiedades (figura 33) (Appleman, 2000).

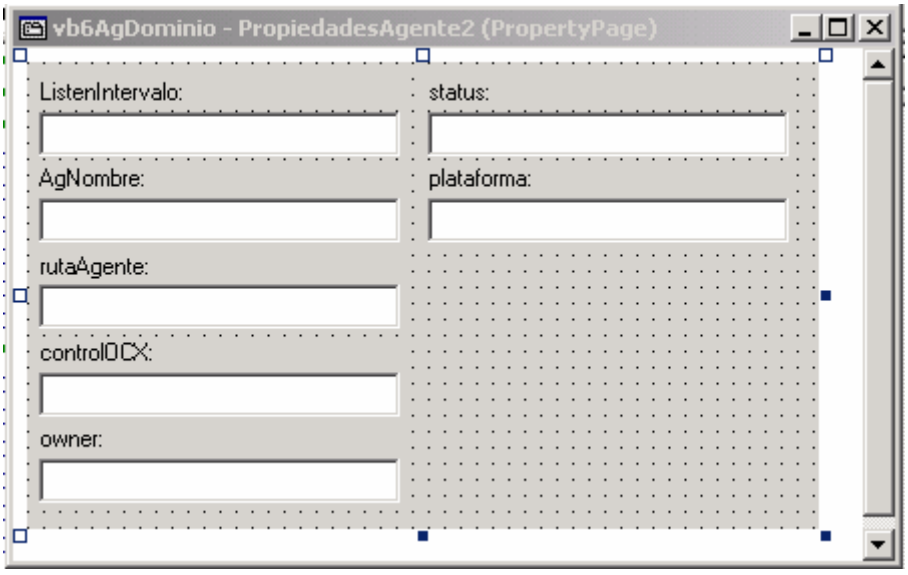

Figura 32 Página de propiedades en modo de diseño

Las páginas de propiedades sirven para modificar los valores de las propiedades de objetos complejos. Algunas consideraciones acerca de la página de propiedades son las siguientes:

a) La página de propiedades carga los valores de las propiedades del control.
b) El desarrollador tiene la alternativa de modificar los valores de las propiedades de la página.
c) El desarrollador puede aplicar o cancelar la asignación de valores de propiedades cambiadas del control.

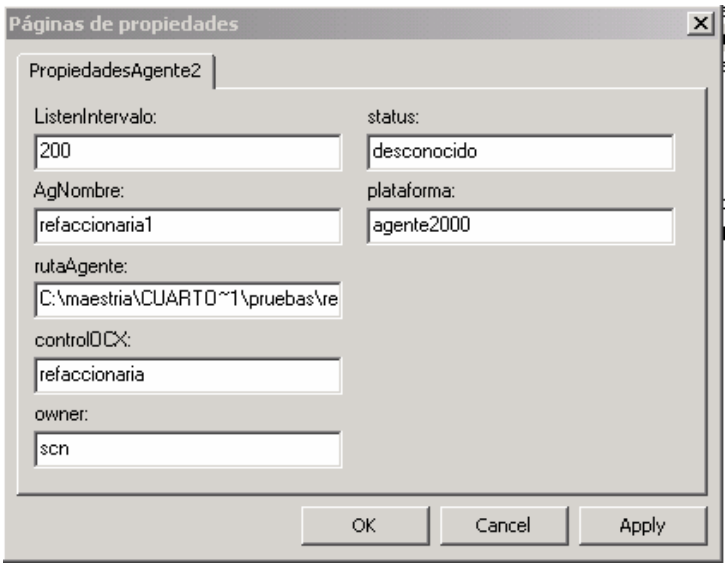

Figura 33 Página de propiedades en modo de ejecución

5.6 Identificación de los controles de agente.

Como parte de la plantilla de control de agente básico se incluye un formulario del tipo acerca de... para identificar algunos datos que el desarrollador del agente quiera poner para documentar a su control. Esto puede ayudar a mostrar la identidad de los controles de agente que se pueden utilizar en las aplicaciones. La interfaz del formulario se ve en la figura 34.

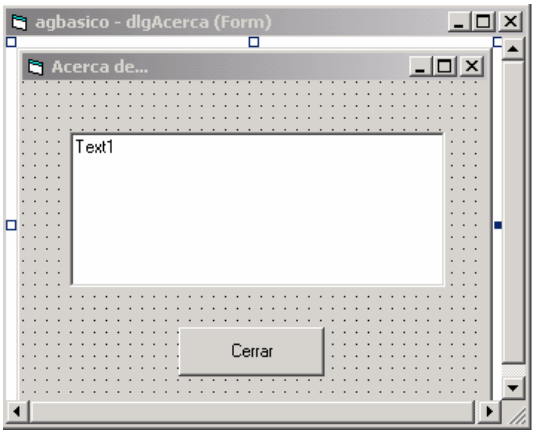

Figura 34 Formulario para identificar controles de agente

La figura 35 muestra un ejemplo a partir de un agente que está siendo utilizado en una aplicación. En las propiedades del control se puede ver su cuadro acerca de... y entonces se despliega la información del control.

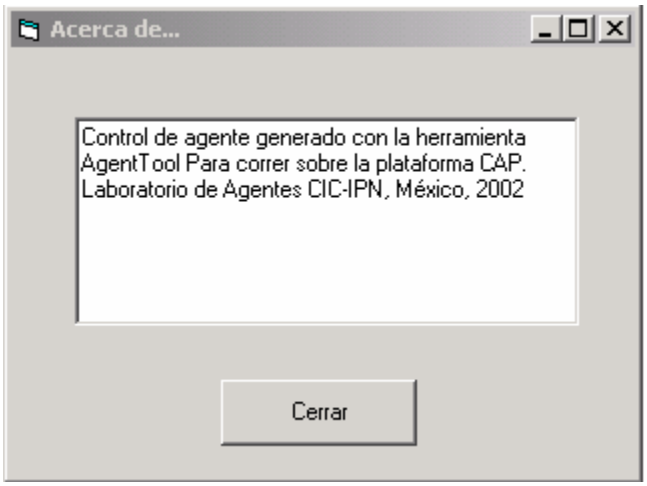

Figura 35 Identificación de un control de agente

5.7 Clases para el manejo del contenido de mensajes ACL

Cada agente tiene los módulos de clase con los cuales puede manejar el contenido de los *performatives* ACL que han sido implementados para ayudar al programador a desarrollar aplicaciones de agentes con los controles. También se le agregan a la plantilla de agente básico las clases para manipular hechos y acciones de los agentes. Para mayor detalle de estas clases ver la sección de implementación de las clases utilitarias para programación de agentes en las secciones 7.3 y 7.4.

5.8 Controles para la interfaz del control de agente.

El control de agente básico, a partir del cual se forman todos los demás controles de agente, es muy sencillo. Además de los elementos antes mencionados en esta sección, la plantilla de control de agente básico está formada por dos controles constituyentes; un control *Label* y un control *Timer* (Figura 36). El objeto *Label* sirve para visualizar el nombre del agente. El objeto *Timer* se utiliza como un sensor para buscar periódicamente mensajes nuevos para cada agente. Cuando se detecta un nuevo mensaje, entonces se verifica el tipo de éste y se dispara el evento correspondiente. En realidad la propiedad *ListenIntervalo* de los agentes se refiere a la propiedad *interval* de este control y sirve para especificar la periodicidad con que el agente va a buscar nuevos mensajes en el canal de comunicación de la plataforma CAP.

Figura 36 Interfaz de los controles de agente

CAPÍTULO 6. IMPLEMENTACIÓN DE LA HERRAMIENTA

Resumen

Trata de la implementación de la herramienta CAP-AgentTool. Se explica detalladamente cada uno de los pasos que se deben seguir en el proceso para crear una clase de control de agente y algunos detalles técnicos importantes en su implementación.

Objetivo del capítulo

El objetivo de este capítulo es explicar los detalles de la herramienta CAP-AgentTool como parte del proceso de creación de nuevos controles de agente.

6.1 Introducción

CAP-AgentTool es una herramienta que tiene por finalidad principal ayudar a los programadores en el proceso de creación de controles ActiveX de agente que corran sobre la plataforma de agentes CAP.

A través de ésta es posible ir paso a paso configurando algunos aspectos importantes que permiten la creación de clases de controles. Los agentes que corren sobre la plataforma CAP están conceptualizados en el paradigma de objetos componentes COM y DCOM. Por esto CAP-AgentTool es necesaria para permitir la creación de nuevos tipos de controles de agente.

Con esta herramienta se trata de facilitar la tarea de diseñar e implementar nuevos controles de agente. Se cubren dos aspectos principalmente:

1.- Por un lado se permite configurar parámetros que son intrínsecos a la tecnología de los controles ActiveX, y que se puede resumir en aspectos de la creación del proyecto fuente sobre el cual se va a crear el nuevo control de agente, su ubicación en el sistema y el tipo de control de agente. Técnicamente, CAP-AgentTool permite crear controles ActiveX con su código fuente en Visual Basic 6.0, a partir del proyecto de control definido en la plantilla de agente básico analizado en el capítulo anterior, el cual contiene las especificaciones mínimas con las que deben contar todos y cada uno de estos agentes.

2.- Por otro lado, es posible configurar las cuestiones propias de los conceptos de agente que se han definido en el paradigma que estamos desarrollando en el Laboratorio de Agentes del CIC-IPN. Es decir, aspectos tales como las ontologías que puede manejar el agente, lenguajes de comunicación de agentes, lenguajes de contenido, *nicknames* por los que puede ser reconocido cada tipo de agentes que se diseña, y tal vez lo más importante desde el punto de vista del paradigma de agentes es que se le puede especificar (desde la etapa de diseño) su conocimiento inicial por medio de hechos y sus capacidades (por medio de acciones del agente), ambos conceptos basados en la especificación de FIPA-RDF0 para el manejo e intercambio de contenido entre agentes.

La presentación de este capítulo está enfocada a explicar la funcionalidad asociada al proceso de creación de agentes. Los detalles de análisis y diseño del software que se implementó se encuentran en el apéndice A, en su sección 1. Así mismo, el código fuente de la herramienta que aquí se describe se puede consultar a través del apéndice D de la tesis.

La pantalla inicial de CAP-AgentTool se muestra en la figura 37. En ella se indica algunos aspectos informativos acerca de la herramienta. Para acceder al inicio de la herramienta el usuario debe hacer clic en cualquier parte de esta pantalla inicial. A continuación se describe cada uno de los pasos que la componen.

Figura 37 Acceso a CAP-AgentTool

6.2 Modo de Creación

El primer paso para crear un nuevo control de agente utilizando la herramienta CAP-AgentTool consiste en determinar el modo de creación del control.

Antes que nada, es importante señalar que la especificación de los atributos de los controles de agente, en esta versión de la herramienta, se encuentra en una base de datos de *Access* de Microsoft. Esto se ha hecho de esta forma por lo pronto, debido a que una característica que se ha agregado a los agentes es que puedan guardar su conocimiento durante varias ejecuciones del mismo control en la misma aplicación, dependiendo de las necesidades de cada problema. De igual forma, es importante conocer este detalle debido a que como parte del proceso de inicialización de las instancias de cada control de agente se hace la carga de los atributos del agente desde su especificación en la base de datos.

Para crear un control nuevo existen dos modalidades:

1.- Crear un control de agente completamente nuevo
2.- Crear un control de agente a partir de otro ya existente

La diferencia entre ambos modos de creación radica en que al crear un control a partir de uno ya existente permite al diseñador del control hacer uso de la configuración y todos los datos del agente que le sirve de base. Esto puede ser muy útil en momentos en que se quiere crear un nuevo control que tiene características muy parecidas a otro que ya existe, o simplemente que comparten dominios de conocimiento grandes y similares.

Al crear un control nuevo completamente se tiene la seguridad de que esta nueva clase de agente no posee de inicio ningún valor en su configuración y, además, no tiene ningún hecho ni acción asociada, por lo que será obligación del usuario de la herramienta proporcionar toda la información de este nuevo control de agente; pero eso será en las siguientes etapas de la creación del control. Por lo pronto debe continuar dando clic en el botón siguiente.

Como puede apreciarse en el gráfico de la figura 38, si se selecciona la opción “abrir existente”, entonces se requiere que indicar la ruta donde se encuentra el archivo con extensión .mdb de la base de datos que contiene la especificación del control que se quiere tomar como base para el nuevo control.

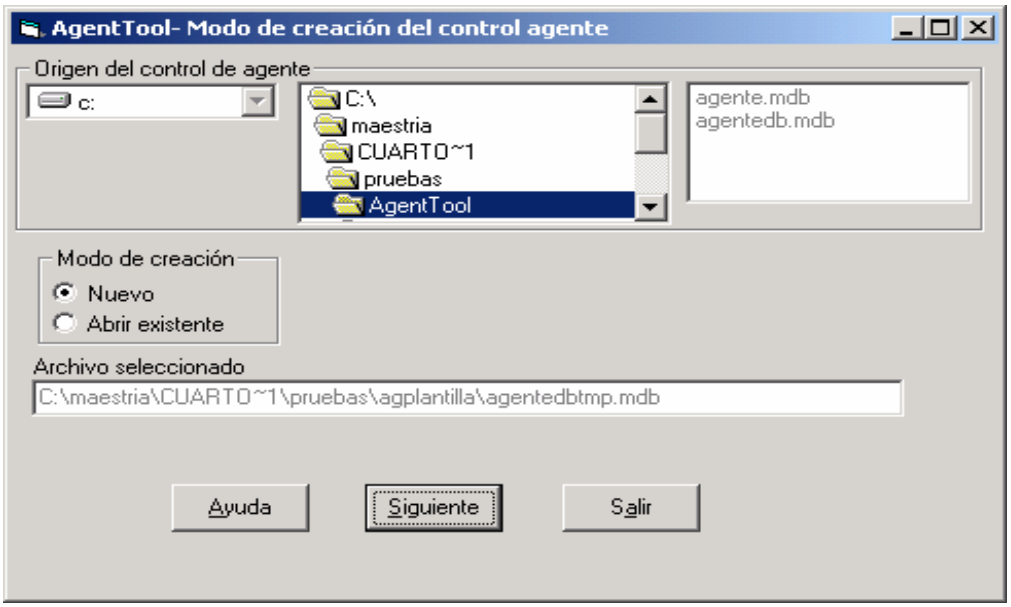

Figura 38 Modo de creación del control de agente

6.3 Datos Generales

Esta etapa se refiere a datos generales que van a ser necesarios para distinguir a una nueva clase de control de agente.

En primer lugar se tiene que especificar la ruta de destino del nuevo control de agente. Esto quiere decir que se debe indicar la ubicación donde se van a guardar todos los archivos que se crean para cada nuevo control. Los controles de agente que se crean con la herramienta tienen su propio directorio. En este directorio se guardan los archivos que se copian desde la plantilla del control de agente básico (que se distribuye con la herramienta) y además de eso, sirve para guardar los datos de cada instancia de control de agente, cuando se añaden controles en algún contenedor de controles ActiveX de alguna aplicación. Una vez que se ha seleccionado el directorio y la unidad de destino, se forma la ruta donde se va a colocar toda la información requerida para la creación del control de agente; esta ruta es muy importante para las siguientes fases del diseño y creación del agente.

Otros datos que conforman esta interfaz para el control de agente nuevo lo conforman sus *nicknames* o nombres con los que también es posible identificar a los agentes de este tipo. Tal y como lo especifica la FIPA, los agentes pueden ser de un sólo tipo y tener un sólo nombre, pero en determinados contextos pueden ser identificados con nombres diferentes; para ello sirve especificar en el diseño de los controles sus *nicknames*. Estos nombres se vuelven importantes desde el punto de vista de la colaboración entre agentes y entre aplicaciones de agentes heterogéneos, debido a que su conceptualización indica que se trata

de implementaciones desarrolladas, por lo general, por diferentes programadores, en lugares remotos y por grupos de desarrollo independientes entre sí, pero que en algún momento dado, necesitan cooperar a través de los agentes que tiene cada uno en sus aplicaciones. Así es que en ambientes dinámicos e inciertos, los *nicknames* pueden jugar papeles muy importantes para lograr interacción entre agentes. Como se ve en la figura 39, se puede agregar una cantidad de *nicknames* a la lista o borrar alguno existente.

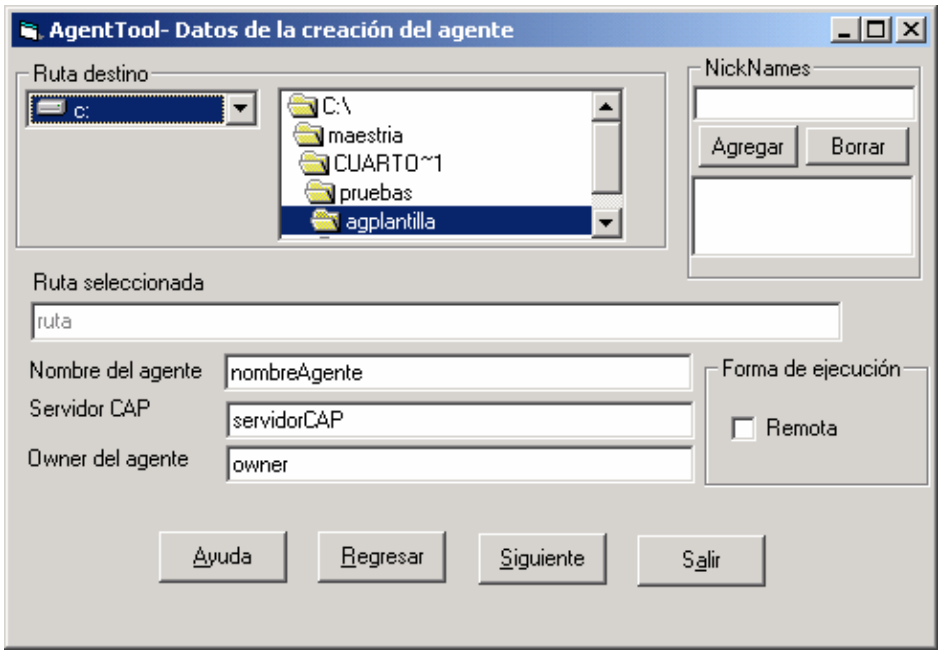

Figura 39 Datos generales

El nombre del control de agente es muy importante para la herramienta ya que en base a este dato se construye un nuevo proyecto de control ActiveX de Visual Basic. Este es el nombre con el que se llamará a la nueva clase de control que se está creando.

El campo etiquetado como “Servidor CAP” sirve para indicar el equipo servidor al que se va a conectar el control de agente para comunicarse con el servidor COM de la plataforma de agentes CAP. Si la opción “forma de ejecución” es seleccionada a “remota” entonces se considera que este control de agente estará ubicado en otra computadora distinta a donde se encuentra instalada la plataforma CAP.

Por último se debe especificar el campo *owner* del agente como un mecanismo para agrupar instancias de agentes que pertenecen al mismo sistema u organización.

Una vez que se ha seleccionado la opción de crear un agente nuevo, aparecen los campos inicializados tal y como se puede apreciar en la pantalla de ejemplo que aparece en la figura 39. Obviamente que es responsabilidad del usuario de la herramienta modificar cada uno de los campos que son de interés para el nuevo control que se quiere crear; de alguna manera esta información inicial le sirve al usuario para saber el tipo de contenido que se espera que ponga en cada campo.

6.4 Configuración de comunicación

En esta fase de la creación del control nuevo se busca proporcionar a los agentes aspectos concretos para su funcionamiento como entidades de software comunicativas. Para entender mejor el significado de estos conceptos que se manejan en esta etapa es importante recordar que con esta herramienta se pretende crear controles ActiveX para utilizarlos como agentes de software que sigan las especificaciones de FIPA que define las estructuras y términos de operación de agentes.

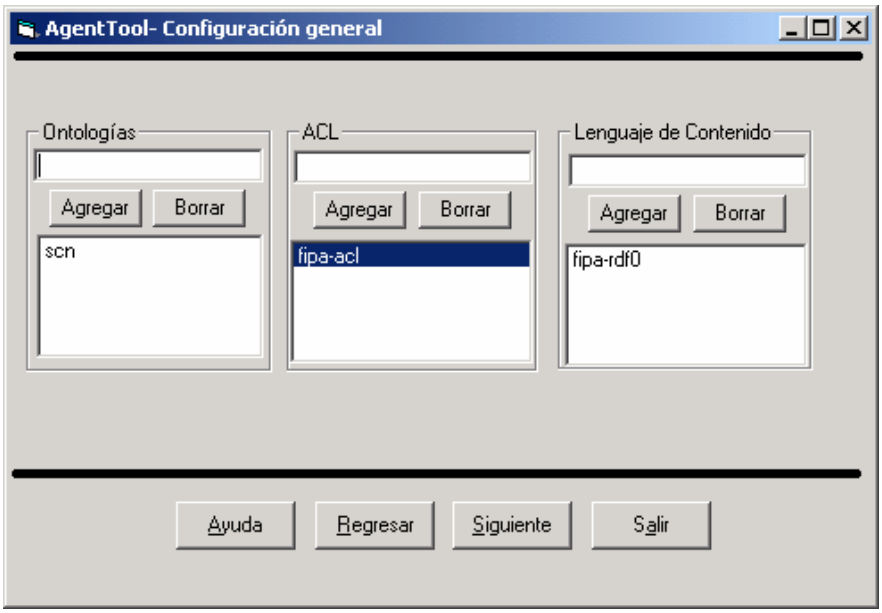

Figura 40 Configuración para comunicación

Pues bien, algunos de los conceptos clave que distinguen a los agentes de FIPA son los que aparecen en esta etapa de creación de los controles de agente (Figura 40).

En primer lugar aparece el concepto de ontología. Una ontología define un conjunto de términos, conceptos, objetos, propiedades y relaciones entre estos, que sirven para permitir a los agentes tener un entendimiento acerca del contenido de su comunicación en sus dominios de aplicación. El objetivo principal de una ontología de agentes es facilitar la tarea de dar significado al conocimiento y poder razonar acerca de los conceptos que se manejan en las conversaciones entre agentes y representa un mecanismo clave para lograr la interoperabilidad de los sistemas multiagente heterogéneos. Para tal efecto, se puede especificar un conjunto de ontologías que el agente puede requerir para funcionar apropiadamente; en este sentido, es muy común y recomendable que existan agentes especializados en la tarea de facilitar el uso de ontologías del dominio de las aplicaciones.

FIPA define y especifica la funcionalidad de los agentes de ontologías con el propósito de que existan estos agentes independientemente de las aplicaciones que los vayan a usar, de tal manera que ofrezcan sus servicios especializados y permitan a los SMA consultarlos cuando sea necesario. Con el afán de lograr aplicaciones de SMA más flexibles y abiertos, es una buena idea separar la funcionalidad de los agentes de ontologías de la funcionalidad

de los agentes de las aplicaciones específicas. De esta manera, es responsabilidad del programador de agentes asegurar que el conocimiento que comparte o intercambia o solicita a otros agentes tiene el significado que espera o necesita, consultando explícitamente a un agente de ontologías o haciéndolo implícito en su funcionalidad.

Otro tema que se presenta en esta etapa es el de los lenguajes de comunicación entre agentes o ACL. En la lista que se tiene en la interfaz de la herramienta se puede agregar o quitar ACL que se espera que el agente pueda manejar. Hasta este momento los agentes que se manejan con la herramienta solamente están preparados para manejar el lenguaje ACL de FIPA; esto es consecuencia directa de que por lo pronto los componentes de la plataforma CAP basan su comunicación a través de mensajes comunicativos en este lenguaje. Esto obliga a que los agentes que corran sobre la plataforma CAP, lo cual es el caso de los agentes que se crean con esta herramienta, deben saber procesar y enviar mensajes FIPA-ACL.

Esta opción simplemente se deja abierta para implementaciones futuras de otros lenguajes de comunicación entre agentes como KQML (*Knowledge Query and Manipulation Language*) (Finin, Labrou & Mayfield, 1997) o incluso nuevos lenguajes que se desarrollen en el futuro.

Por último se debe especificar los lenguajes de contenido que el agente sabe manejar como parte de los mensajes que recibe a través del lenguaje de comunicación. En particular a ACL, en el campo *content* de cada mensaje debe ir el contenido del mensaje, que al final de cuentas, se trata del conocimiento que se quiere comunicar con el mensaje. Para efectos de experimentación y pruebas del paradigma general de agentes componentes que se está desarrollando con esta y otras herramientas aquí presentadas, se ha optado por implementar un pequeño lenguaje de contenido basado en la especificación de FIPA-RDF0. Por *default* aparece este lenguaje en la lista de lenguajes de contenido. Esto no se debe quitar ni modificar a menos que en trabajos futuros se permita a los agentes procesar el contenido de los mensajes utilizando otros lenguajes como por ejemplo SL0, CCP, KIF, etc.

6.5 Protocolos de interacción

La mayoría de las conversaciones que sostienen los agentes siguen ciertos patrones. Inician de modo similar y en determinado momento se puede inferir el tipo de mensajes que siguen. Por ejemplo, después de un *request* es común que siga un *not-understood*, un *agree* o un *refuse*, pero no es usual que siga un *subscribe*. A estos patrones de conversación se les llama protocolos de interacción.

Un diseñador de aplicaciones de agentes puede decidir hacer a los agentes lo suficientemente conscientes del significado de los mensajes, sus objetivos, creencias y otros estados mentales que el agente posea, de tal forma que el proceso de planeación cause que los protocolos de interacción emerjan espontáneamente de las decisiones del agente. Esto sin embargo, implica una gran carga de capacidades y complejidad en la implementación del agente. Una forma alternativa y muy pragmática es pre-definir los protocolos de interacción para que agentes implementados en forma sencilla puedan involucrarse en

conversaciones que tengan sentido con otros agentes, simplemente apegándose al protocolo conocido.

El apegarse al uso de protocolos trae ventajas a los desarrolladores de agentes, pues pueden enfocarse a implementar sólo aquellos actos comunicativos involucrados en los protocolos en lugar de considerar todo el espectro de actos.

Actualmente FIPA cuenta con varios protocolos predefinidos. Los desarrolladores pueden crear sus propios protocolos y no se obliga a los agentes a implementar ninguno de los protocolos predefinidos, tampoco se les prohíbe que usen otros protocolos. Sin embargo, si un agente usa un protocolo predefinido debe hacerlo de manera consistente a como está especificado. La forma de especificar el protocolo en una conversación es poniendo su nombre en el parámetro *protocol* del mensaje. FIPA ha establecido un conjunto de protocolos estándar (FIPA *Brokering*, FIPA *Contract Net*, FIPA *Dutch Auction*, FIPA *English Auction*, FIPA *Iterated ContractNet*, FIPA *Propose*, FIPA *Query*, FIPA *Recruiting*, FIPA *Request*, FIPA *Request When* y FIPA *Subscribe*) (FIPA-7, 2001).

Actualmente, la herramienta permite que el diseñador del control de agente especifique los protocolos de interacción que va a contener cada tipo de agente. Se pueden seleccionar a través de esta etapa.

Existen tres protocolos de interacción que están disponibles para ser incluidos en el código fuente del agente de manera genérica: *Fipa-Request-Protocol*, *Fipa-Query-Protocol* y *Fipa-ContractNet-Protocol*. La implementación de cada protocolo es genérica en el sentido que ciertas etapas de la comunicación están validadas para asegurar que la secuencia de mensajes enviados y recibidos por los agentes sean los correctos. Pero existen ciertos puntos en los que cada tipo de agente debe implementar la funcionalidad y comportamiento específico de la aplicación o del dominio en el que va a funcionar el protocolo. En estas partes, el protocolo debe ser programado por el programador del agente, de tal forma que ciertas decisiones importantes sean implementadas específicamente.

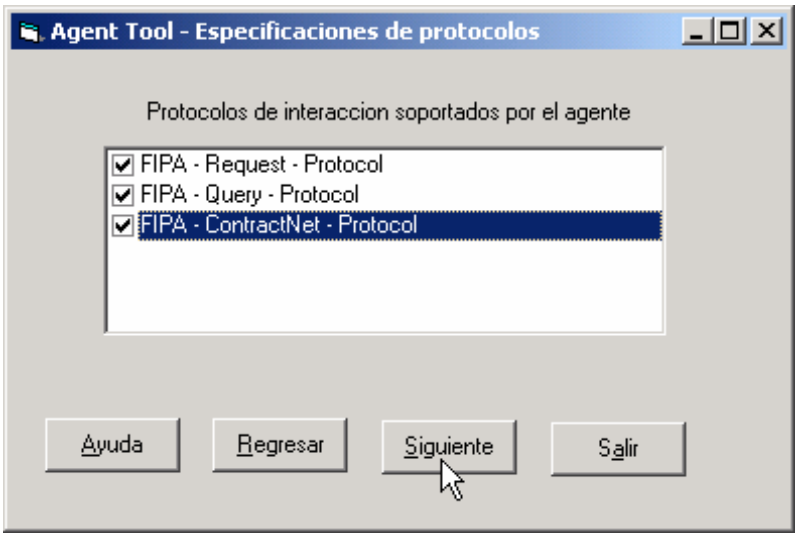

Figura 41 Selección de protocolos de interacción

En la figura 41 se puede ver la interfaz de la herramienta para especificar los protocolos de interacción que los agentes pueden utilizar.

El Protocolo de Interacción *Fipa-Request* permite a un agente que solicite a otro la realización de una acción y el agente receptor realiza la acción o responde, de alguna manera, que no puede realizarla (FIPA-8, 2001).

En el protocolo de interacción *Fipa-Query*, el agente receptor es solicitado a realizar algún tipo de acto *inform.* El acto *inform* es un *Query,* y hay dos tipos de actos *Query: querif y query-ref* y cualquier acto puede ser usado para iniciar este protocolo. En cualquier caso, un *inform* es usado en respuesta. En cuanto al contenido del *inform* dado en respuesta a un *query-ref* debería ser una expresión referenciada (FIPA-9, 2001).

En este protocolo al agente destinatario se le pide que efectúe una acción de carácter informativo. El protocolo puede ser iniciado con un acto *query-if* o un *acto query-ref*. El remitente puede contestar con un *not-understood*, un *failure*, un *refuse* o un *inform*.

En el protocolo de interacción *Fipa-Contract-Net* (Smith, 1980), se permite que un agente iniciador solicite a uno o varios agentes participantes propuestas de la prestación de un servicio, por medio de mensajes ACL, y así empiezan a comunicarse (FIPA-10, 2001). La descripción de la implementación de estos protocolos es explicada en el apéndice B. Además, para mayores detalles, en el apéndice C se incluye el código fuente de éstos agentes y programas que utilizan los protocolos de interacción.

6.6 Hechos Iniciales

Una característica fundamental de los agentes componentes bajo la perspectiva de controles ActiveX que se está desarrollando con este modelo es que basan su colaboración a través de su lenguaje de comunicación de agentes, que como ya se ha mencionado antes, es FIPA-ACL.

FIPA-ACL es un lenguaje de comunicación entre agentes que está definido a través de la teoría del habla o *speech acts*. Por consiguiente, FIPA ofrece una semántica para diferentes tipos de actos comunicativos o también llamados *performatives*. La idea central de la semántica está dada para facilitar la comunicación de los agentes a través del manejo, compartición e intercambio de conocimiento a través de ACL.

Además de lo anterior, otro elemento importante para la comunicación de agentes es el lenguaje de contenido. FIPA deja abierta la especificación para que los agentes utilicen el o los lenguajes de contenido apropiados. Debido a que se ha implementado por lo pronto solamente un lenguaje de contenido para los agentes, basado en la especificación FIPA-RDF0.

FIPA-RDF0 especifica que el conocimiento puede ser de tres tipos principalmente. Objetos del dominio, proposiciones o enunciados y acciones. Basado en ello, se ha optado por implementar el lenguaje de contenido basado en dos aspectos: proposiciones y acciones. Existe también FIPA-RDF1 que define conocimiento a manera de reglas, FIPA-RDF2 para

álgebra lógica, pero para esta etapa del trabajo no han sido considerados como parte del lenguaje de contenido.

¿Por qué proposiciones y acciones únicamente? La justificación principal de haber restringido nuestro lenguaje de contenido a estos dos elementos que plantea la especificación de FIPA-RDF0 es que la mayor parte de los actos comunicativos de FIPA-ACL están basados en el intercambio de proposiciones o tienen que ver con la ejecución y razonamiento acerca de acciones y desde nuestro punto de vista, son los más importantes para efectos de comunicación entre agentes.

Por lo anterior, se ha creado un tipo de objeto llamado hecho para denotar a las proposiciones de FIPA-RDF0. Para efectos de la captura de hechos iniciales de esta clase de agentes se proporciona esta interfaz en la herramienta como se ve en la figura 42.

Los hechos están formados por los siguientes elementos: predicado, sujeto, objeto, ontología, creencia y *owner* o dueño.

Además de los elementos básicos de una proposición de RDF (predicado, sujeto y objeto), se han añadido tres más: owner para denotar el agente al que originalmente pertenece el enunciado, que para fines de intercambiar y compartir conocimiento puede ser importante para las aplicaciones de agentes. Se añade también la propiedad ontología, de tal forma que los agentes puedan distinguir el significado de los hechos que se comunican durante la comunicación con los demás. También se añade la propiedad creencia para establecer el valor de verdad del hecho (por lo pronto este valor puede ser verdadero o falso basándose en una lógica de proposiciones).

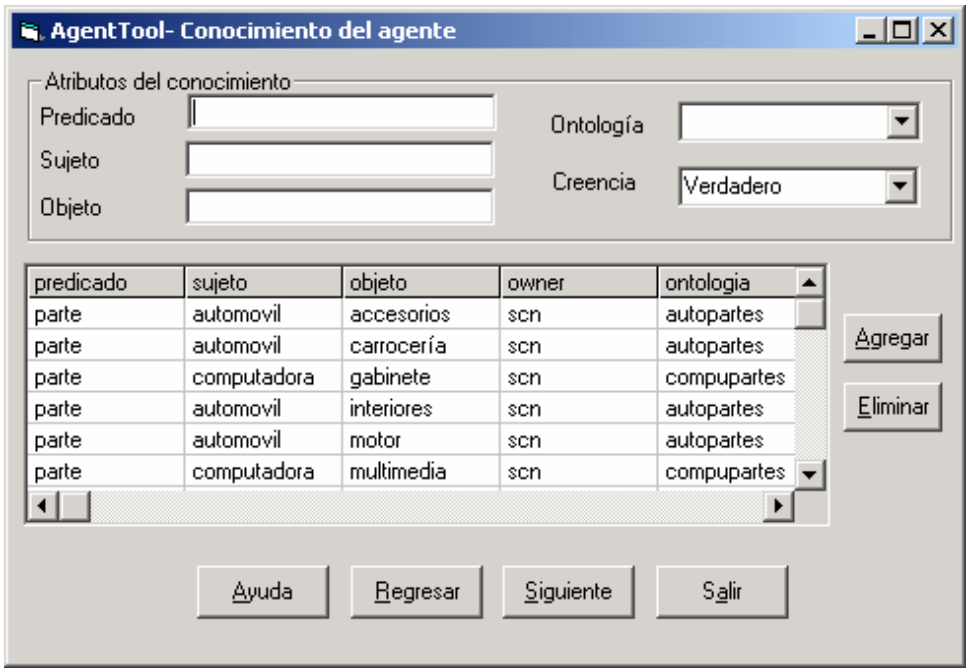

Figura 42 Especificación de Hechos iniciales

Los hechos iniciales se guardan como parte de la especificación de los agentes y se cargan automáticamente cuando el agente se inicializa o cuando explícitamente el programador de agentes necesita su inicialización. Los hechos iniciales de los agentes deben ser resultado de una fase previa a la implementación, más bien en el proceso de análisis del agente y sirve para indicarle la información del dominio y de su ambiente que es importante que conozca durante su ejecución.

6.7 Acciones

Una acción expresa una actividad llevada a cabo por un objeto. FIPA-RDF0 especifica la estructura que debe tener una acción cuando se considera desde la perspectiva de la comunicación a través de ACL.

Se definen tres propiedades relacionadas a una acción:

El acto identifica la parte operativa de la acción: este puede servir para identificar el tipo de acto o meramente para describirlo.

Un actor identifica la entidad responsable para la ejecución de la acción, esto es, el valor es la entidad específica que puede realizar la acción (por lo general, el agente receptor).

Los argumentos identifican entidades opcionales que pueden ser usadas para la ejecución de la acción; esto es, el valor es la entidad que es usada por el actor para realizar el acto.

Complementariamente a la definición de acción dada antes, también se debe tener la habilidad de conocer cuándo una acción ha terminado o qué resultado tuvo su ejecución. Esto es resuelto añadiendo algunas propiedades extra a la descripción de acción. Para fines prácticos es necesario tener una propiedad "*done*" que indique el status de la ejecución de la acción (éxito o fracaso); además la propiedad "resultados" para indicar las proposiciones que se consiguieron con la ejecución de la acción. Un ejemplo se puede ver en la figura 43.

En esta parte de la herramienta se permite capturar la semántica de las acciones que los controles de agente pueden tener. Para cada acción declarada para la clase de agente se puede especificar sobre la base de tres elementos: el nombre o descripción de la acción, los argumentos que va a recibir y los resultados que se deben obtener al ejecutar la acción.

En el caso de los argumentos y los resultados establecidos en esta herramienta se considera que son obligatorios y forman parte de la semántica de la acción. Vienen siendo las precondiciones y poscondiciones para la acción. De esta forma se trata de asegurar el comportamiento abierto de la acción para ser utilizada por otros agentes. No obstante es posible, y algunas veces necesario, especificar otros argumentos y resultados para la acción, dependiendo de recursos que se requieran y dependiendo de la naturaleza de la propia acción.

Los argumentos y resultados manejados en las acciones se modelan y representan como hechos del dominio de la aplicación.

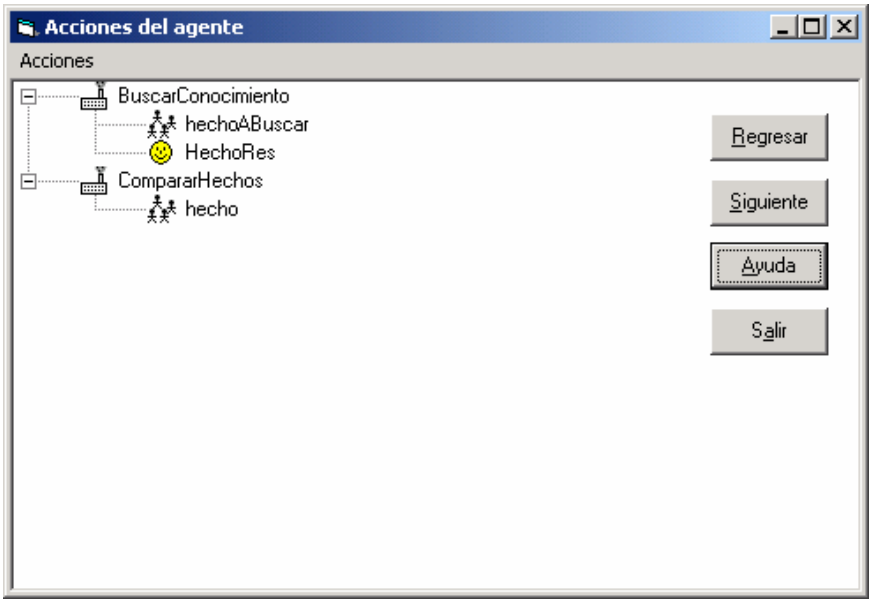

Figura 43 Añadiendo acciones

6.8 Creación

La idea principal de esta fase es resumir y generar el nuevo control de agente o en su caso, la modificación de alguno existente. Se muestra una serie de datos que han sido especificados durante las etapas anteriores y que tienen que ver sobre todo con aspectos generales del control. No incluye conocimiento inicial ni acciones del agente.

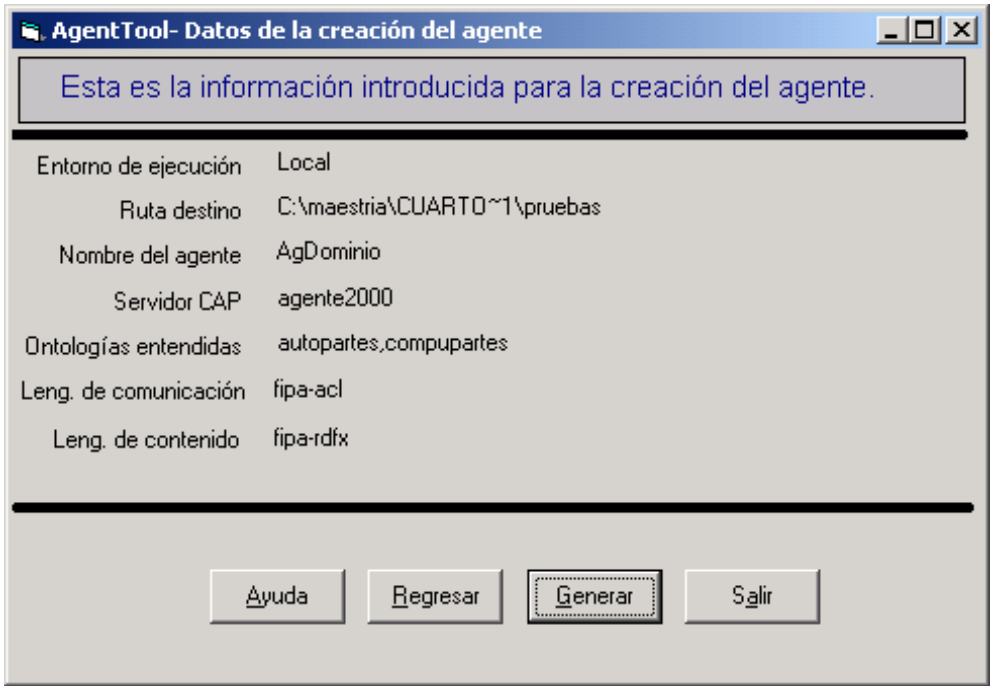

Figura 44 Resumen y generación del control

La parte más importante se produce cuando el usuario da clic en el botón “Generar”. A continuación se describe el proceso de generación del nuevo control de agente (Figura 44).

La generación del nuevo control de agente significa crear un proyecto de control ActiveX nuevo, con código fuente de Visual Basic. Este proyecto combina la sintaxis de los controles ActiveX y además añade algunas propiedades y métodos para su comportamiento de agentes. Se puede resumir en los siguientes puntos:

1. Crear la estructura de directorios y archivos con las características del agente especificado.
2. Copiar todo lo necesario de la plantilla del control de agente básico a este directorio.
3. Ajustar la plantilla de agente básico para crear un nuevo proyecto con el nombre de este control de agente.
4. Poner en el nuevo proyecto la declaración de las acciones que se especificaron en la herramienta. Estas acciones tienen código fuente necesario para la validación de los argumentos de entrada y de los resultados de salida, de acuerdo con lo especificado para cada acción.
5. Incluir el código fuente de los protocolos de interacción seleccionados.
6. Generar el archivo de documentación de la clase de control de agente.

Para cada nuevo tipo de control de agente que se crea con la herramienta se le forma un directorio base (especificado en la herramienta, a partir de la ruta de destino del control). Si el control de agente se toma de otro ya existente y se deja la ruta de destino sin modificar, se considera simplemente que se está modificando el control de agente y entonces se sobre escribe a la especificación que se está tomando.

Luego, se procede a copiar el proyecto de control de agente básico que se distribuye con la herramienta al directorio destino del nuevo control. A partir de este se edita el proyecto de acuerdo con los datos del control nuevo.

De esta forma, al final del proceso se tiene una carpeta con el nombre del agente especificado; la carpeta se encuentra ubicada a partir del directorio destino. En esa carpeta se pueden encontrar entre otras cosas el proyecto del control recién creado, los archivos de páginas de propiedades y el diálogo que describe al agente (acerca de...), la base de datos con su especificación y el archivo de texto que documenta al agente (el nombre de la clase de agente con extensión .txt, ver el ejemplo del apéndice G).

La siguiente fase de la creación del control de agente es abrir el proyecto de control nuevo e implementar mínimamente las acciones especificadas; luego se compila el proyecto para generar el control compilado en un archivo con extensión ocx. A partir de este momento el control de agente puede ser utilizado en las aplicaciones que soporten contenedores de ActiveX, tales como Visual Basic, Visual FoxPro, Visual C++, incluso, Microsoft Word y Excel, entre otros. Esta parte se explica más ampliamente durante el capítulo 8.

CAPÍTULO 7. CLASES UTILITARIAS PARA PROGRAMAR AGENTES

Resumen

Aquí se menciona el tema de las clases utilitarias que fueron creadas para auxiliar y complementar la tarea de la programación de SMA que utilicen controles de agente creados con CAP-AgentTool.

Por una parte se describen las clases que están enfocadas a ayudar a la tarea de envío y recepción de mensajes ACL entre los agentes, con una semántica y sintaxis del contenido de acuerdo con lo especificado en el lenguaje FIPA-ACL. Por otro lado, se presentan las clases utilitarias para la manipulación del conocimiento interno del agente, para hechos y acciones principalmente.

Objetivo del capítulo

Exponer las ideas fundamentales que se tomaron para el desarrollo de un conjunto de clases utilitarias que ayudan a programar agentes en el entorno de los controles de agente creados con CAP-AgentTool.

7.1 Introducción

Hasta este punto de la investigación se han presentado tres aspectos fundamentales para los objetivos del trabajo: se ha presentado un modelo de agentes basados en controles ActiveX, se ha implementado una plantilla de control de agente básico para formar los controles de agente y se ha desarrollado una herramienta de software que permite crear controles de agente.

Con estas tres cosas se tiene una buena base para empezar a desarrollar aplicaciones que utilicen controles de agente. Pero aun hace falta algo más. Basándonos en las especificaciones de FIPA para agentes, y en su propia especificación del lenguaje ACL, es necesario implementar el lenguaje de comunicación entre agentes siguiendo la semántica definida para los *performatives* que se establecen.

Ahora que se toca este tema es importante resaltar que la semántica de ACL establece que la base de la comunicación entre los agentes es el manejo e intercambio de conocimiento. La mayor parte de los actos comunicativos de ACL tiene que ver con proposiciones y la manipulación de acciones. Además, la implementación de ACL debe estar ligada a un lenguaje de contenido para buscar el entendimiento semántico y sintáctico de las comunicaciones de los agentes. Como ya se ha mencionado antes, se ha implementado una variante del lenguaje de contenido FIPA-RDF0 que está basado en los conceptos de hechos y acciones para expresar objetos de contenido en los mensajes de ACL que se intercambian entre los agentes.

La implementación que se propone aquí constituye un esfuerzo dirigido a fortalecer las técnicas para el desarrollo de SMA de tal forma que se concreta la tarea de implementar agentes de software colaborativo a través de ACL. Se le ofrece al programador de agentes varias clases utilitarias necesarias para asegurar, hasta cierto punto, que el contenido de los mensajes podrá ser recibido, procesado y comprendido correctamente por su contraparte.

En esta sección se describe la implementación del lenguaje ACL de acuerdo con la semántica de FIPA-ACL y utilizando los conceptos del lenguaje de contenido FIPA-RDF0.

7.2 Implementación de FIPA-ACL

La FIPA controla todo lo referente a los actos comunicativos que forman parte de ACL a través de su biblioteca de actos comunicativos. Por medio de dicha biblioteca se trata de definir la estructura y los requerimientos para los actos comunicativos propuestos para ser incluidos en la biblioteca. Esta especificación contiene una base formal para la semántica de cada acto comunicativo de FIPA-ACL.

La definición de un acto comunicativo que pertenece a la biblioteca de actos comunicativos de FIPA es normativa. Esto es, si un agente implementa un acto comunicativo entonces debe implementar ese acto de acuerdo con la definición semántica de la FIPA. Sin embargo, los agentes que son compatibles con FIPA no están obligados a implementar algunos de los actos comunicativos del lenguaje, con excepción del acto *not-understood*. La

biblioteca de actos comunicativos de FIPA facilita el uso de actos comunicativos estandarizados por agentes desarrollados en diferentes contextos. Además de la caracterización semántica e información descriptiva que es necesaria, cada acto comunicativo puede especificar información adicional como por ejemplo información de estabilidad, versiones, etc.

Como parte central de la implementación de ACL que se propone en este trabajo, se establece que cada acto comunicativo soportado por los agentes tiene una clase que se debe utilizar para manipular el campo *content* del mensaje ACL. Dichas clases tienen como objetivo principal obtener los objetos que se requieren para llevar a cabo el acto (de acuerdo con su semántica y sintaxis especificada en FIPA-ACL) y representar el contenido de los objetos utilizando el esquema de codificación en XML, de tal forma que se facilite la tarea de comunicación del contenido. De esta manera, las clases para la especificación del contenido de los actos comunicativos implementados tienen un método *parse* para convertir a XML el contenido que se pretende comunicar en el mensaje. También tienen un método *deparse* para obtener el contenido recibido en un mensaje ACL. El tipo de objetos que se envían y reciben con cada clase utilitaria varía de acuerdo al tipo de acto y a la semántica de su contenido.

7.3 Clases para implementar Contenido

Los agentes pueden manejar una base de hechos y de acciones. Además, como se ha mencionado antes, pueden utilizar un lenguaje de contenido que utiliza estas dos clases de objetos para expresar el contenido que se comunica en los mensajes. Se implementaron tres clases: la clase hecho, la clase accion, y la clase accionRespuesta.

La función de las clases sirve en dos aspectos de los agentes: en primer lugar, permiten a los programadores manipular hechos y acciones de la base de conocimiento del agente. En segundo lugar, ayudan a la manipulación de los objetos de contenido utilizados en los mensajes ACL, para los actos que fueron implementados. En el apéndice A, sección 2 se muestran algunos diagramas de clases que fueron creados como parte del diseño de este grupo de clases, así como las clases de las secciones 7.4 y 7.5 que aparecen más adelante en este capítulo.

La clase *hecho* sirve para representar las proposiciones. La clase *accion* es usada para representar acciones por realizar y la clase *accionRespuesta* tiene por finalidad integrar la respuesta para una acción realizada. Esto está basado en la especificación del lenguaje de contenido de FIPA-RDF0 (ver sección 3.3.2.1). Estas clases contienen dos métodos que son importantes: *parse* y *deparse*. Con el método *parse* se puede convertir a los objetos de contenido a una representación en XML. De forma inversa, con el método *deparse* se tiene la descripción del contenido en XML y se puede obtener y colocar en objetos de la aplicación.

Para utilizar los métodos *parse* y *deparse* de estas clases utilitarias se requiere un objeto para manipular documentos XML en Visual Basic. Por medio del objeto

MSXML.DOMDocument [MSDN, 1999] se crea y manipula la especificación del contenido de cualquiera de estos tipos.

7.3.1 La clase Hecho

Esta clase sirve para manejar las proposiciones o hechos que se pueden utilizar en el contexto de los dominios de las aplicaciones de SMA por los agentes. A continuación se va a entrar en los detalles de su implementación con la finalidad de explicar mejor su funcionalidad.

La clase hecho tiene los siguientes atributos, para denotar un hecho del dominio:

- Predicado, es la propiedad a la que se refiere el hecho
- Sujeto, se refiere a la entidad, objeto o recurso del dominio de la aplicación al que se refiere la propiedad o predicado.
- Objeto, consiste en el valor de la propiedad .
- *Belief*, denota la creencia del hecho, esto es, si es verdadero o falso.
- *Owner*, consiste en el propietario del hecho originalmente .
- Ontología, indica si el hecho es parte de alguna ontología .

Contiene los métodos

- *Parse(ObjHecho)*
- *deparse(string)*

Tiene algunos campos para el manejo de errores al hacer la acción *deparse*. Por último, tiene una propiedad *content* para guardar su representación en XML.

El objetivo del método *parse* es convertir el objeto hecho y sus atributos a una representación en XML para poder ser utilizado en el contenido de los mensajes ACL. El método *parse* funciona de la siguiente manera:

- Recibe como parámetro un objeto de clase hecho para tomar de este los atributos del *hecho* que se quiere procesar a XML.
- Crea una instancia del objeto *MSXML.DOMDocument* para formar un documento de XML.
- Crea los nodos necesarios para cada uno de los elementos que forman parte del contenido que se quiere formar. En este caso, crea un nodo para cada atributo del hecho.
- Llena el valor de cada nodo con el valor de cada atributo dado por el objeto hecho que es el argumento pasado al método.
- Se agregan los nodos creados al documento de XML.
- Se asigna la expresión de XML formada, a la propiedad *content* para que sea utilizada en la aplicación como la representación en XML del objeto hecho.

De manera contraria al método *parse*, está el método *deparse*; este sirve para obtener el objeto de contenido que va expresado en un formato de XML y funciona como se explica a continuación:

- Recibe como parámetro una cadena de caracteres que debe ser un hecho representado en XML.
- Debido a que se está utilizando la clase *hecho*, el programador implícitamente está esperando obtener un objeto de clase *hecho* a partir del parámetro del método.
- Se procede a validar el documento XML que se recibe en el parámetro. La validación se realiza por medio de un DTD creado para construir hechos en XML. En el directorio de instalación de la herramienta CAP-AgentTool debe existir el directorio DTD, lugar en donde se encuentran los distintos DTD para validación de las clases utilitarias que manejan algún tipo de objeto de contenido. Los hechos se validan con un DTD que se llama hecho.dtd (figura 45).
- Si por medio de la validación se determina que el *string* es un hecho que cumple con la estructura correcta, entonces se procede a extraer cada uno de los campos que forman el hecho.
- Enseguida se inicializa cada atributo miembro de la clase con los valores de los campos extraídos.
- Si hubo algún error en la creación del documento, entonces se inicializan los atributos para el manejo de errores que tiene el objeto hecho.

Para las otras clases el funcionamiento y estructura es muy similar; cambia únicamente el contenido que se involucra dependiendo de la naturaleza y necesidades de cada clase. Por ejemplo la clase *accion* requiere los atributos con los que se manejan las acciones: acto, actor y una lista de argumentos. De igual manera, la clase *accionRespuesta*, cuya finalidad es estructurar los elementos necesarios para representar los resultados obtenidos cuando una acción fue realizada, requiere el acto al que se refiere la acción, el status de la ejecución y una lista de proposiciones obtenidas como resultado de haber ejecutado la acción. Los métodos *parse* y *deparse* de cada una de estas clases funciona igual, lo único que se modifica es el DTD que se utiliza para validar su estructura cuando se utiliza el método *deparse*.

```
<?xml version="1.0"?>
<!-- Document Type: XML DTD
Propósito del documento: Verificar la construcción de hechos
-->
<!ELEMENT hecho (predicado, sujeto, objeto, belief, owner, hontologia)>
<!ELEMENT predicado (#PCDATA)>
<!ELEMENT sujeto (#PCDATA)>
<!ELEMENT objeto (#PCDATA)>
<!ELEMENT belief (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT hontologia (#PCDATA)>
```

Figura 45 DTD para la validación de hechos

7.3.2 La clase *accion*

Sirve para representar internamente en el agente cada una de las acciones que puede ejecutar. A su vez, esta clase es utilizada para expresar objetos de contenido de tipo *accion* que son necesarios en algunos actos comunicativos en el campo de contenido de los mensajes ACL.

Se define básicamente como lo especifica el esquema FIPA-RDF0 para representar acciones: los atributos que la forman son:

- Acto, para denotar el nombre de la acción representada.
- Actor, es el nombre del agente que se encargará de realizar la acción denotada por el campo Acto.
- Argumentos, es una lista de parámetros de tipo Hecho que se necesitan por la acción para poder ejecutarse.
- Ontología, denota la ontología del dominio a la que pertenece la acción.

La forma en que se implementaron los métodos *parse* y *deparse* de esta clase es similar a la clase *hecho*. La validación de acciones con la estructura requerida se hace con el DTD que se llama accion.dtd. Una lista de las clases para manejo de contenido internamente en el agente se da en la figura 46.

Clase	Métodos	Atributos	DTD
Hecho	*Parse* *Deparse*	Predicado, Sujeto, Objeto, Belief, Owner, Ontología	Hecho.dtd
Accion	*Parse* *Deparse*	Acto, Actor Argumentos, Ontología	Accion.dtd
accionRespuesta	*Parse* *Deparse*	Acto, Status, resultados	AccionResp.dtd

Figura 46 Clases para manejo de contenido

7.4 Actos comunicativos de FIPA-ACL implementados

Con base en la implementación del lenguaje de contenido y las clases que ahora se tienen para expresar el contenido de los mensajes, a continuación se listan los actos comunicativos que actualmente pertenecen a la biblioteca de actos comunicativos de FIPA, y que han sido implementados; además se da una descripción de la clase que se ha desarrollado para implementar la semántica de su contenido.

La estructura y funcionalidad de estas clases es muy similar a la de las clases utilitarias para expresar los tipos de contenido (ver sección 7.3). El problema es el mismo: manipular el contenido de los mensajes a través de las clases, tanto para enviar contenido en los mensajes ACL como para poder manejarlo cuando los agentes reciben. Por lo tanto, existen también los métodos *parse* y *deparse* para cada una de las clases y su correspondiente DTD para validación XML. Los agentes utilizan estas clases en tiempo de ejecución para manejar el contenido de los mensajes que les envían otros agentes. Para hacer esto requieren de una variable de entorno en la configuración del sistema. Al respecto, el apéndice E que acompaña a la tesis indica los aspectos de la instalación de CAP-AgentTool y la configuración requerida.

Para estas clases lo más importante de su implementación es el contenido del mensaje que se quiere comunicar en cada tipo de acto comunicativo. Por este motivo, la implementación de la semántica de FIPA-ACL que se ofrece en este trabajo se concentra en ese detalle, aunque quedan abiertos otros aspectos como las precondiciones de posibilidad y el efecto racional de cada acción comunicativa.

Para describir la manera en que funcionan los objetos de contenido se va a tomar como ejemplo el acto comunicativo *rejectProposal*. Su descripción dice que lleva a cabo la acción de rechazar una propuesta de realizar una acción. El contenido de este *performative* consiste en una expresión de acción que denota la acción a ser rechazada, las condiciones que se deben cumplir para realizarla y una proposición que denota las razones del rechazo.

Este acto comunicativo está implementado en la clase *RejectProposalContent* y se verifica su sintaxis con su DTD respectivo (figura 47) y enseguida se describe. La clase *RejectProposalContent* la forman tres miembros principales: Un objeto de clase *accion* y dos objetos de clase *collection*. El objeto de clase *accion* sirve para expresar la acción que se rechaza con este mensaje; el primer objeto *collection* se utiliza para expresar las condiciones que se marcaron para llevar a cabo la acción. El otro objeto contiene las razones por las cuales se está rechazando la propuesta a la que se está refiriendo el mensaje.

```
<?xml version="1.0"?>
<!-- Document Type: XML DTD
        Propósito del documento: Verificar la construcción del contenido de un reject-
proposal -->
<!ELEMENT rejectProposal (accion, condiciones, razones)>
<!ELEMENT accion (#PCDATA)>
<!ELEMENT condiciones (hecho*)>
<!ELEMENT hecho (predicado, sujeto, objeto, belief, owner, hontologia)>
<!ELEMENT predicado (#PCDATA)>
<!ELEMENT sujeto (#PCDATA)>
<!ELEMENT objeto (#PCDATA)>
<!ELEMENT belief (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT hontologia (#PCDATA)>
<!ELEMENT razones (hecho*)>
<!ELEMENT hecho (predicado, sujeto, objeto, belief, owner, hontologia)>
```

```
<!ELEMENT predicado (#PCDATA)>
<!ELEMENT sujeto (#PCDATA)>
<!ELEMENT objeto (#PCDATA)>
<!ELEMENT belief (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT hontologia (#PCDATA)>
```

Figura 47 DTD para el contenido de un mensaje *reject-proposal*

En la figura 48 se presenta un diagrama que representa la estructura del DTD usado por el acto comunicativo reject-Proposal y su clase de contenido rejectProposalContent.

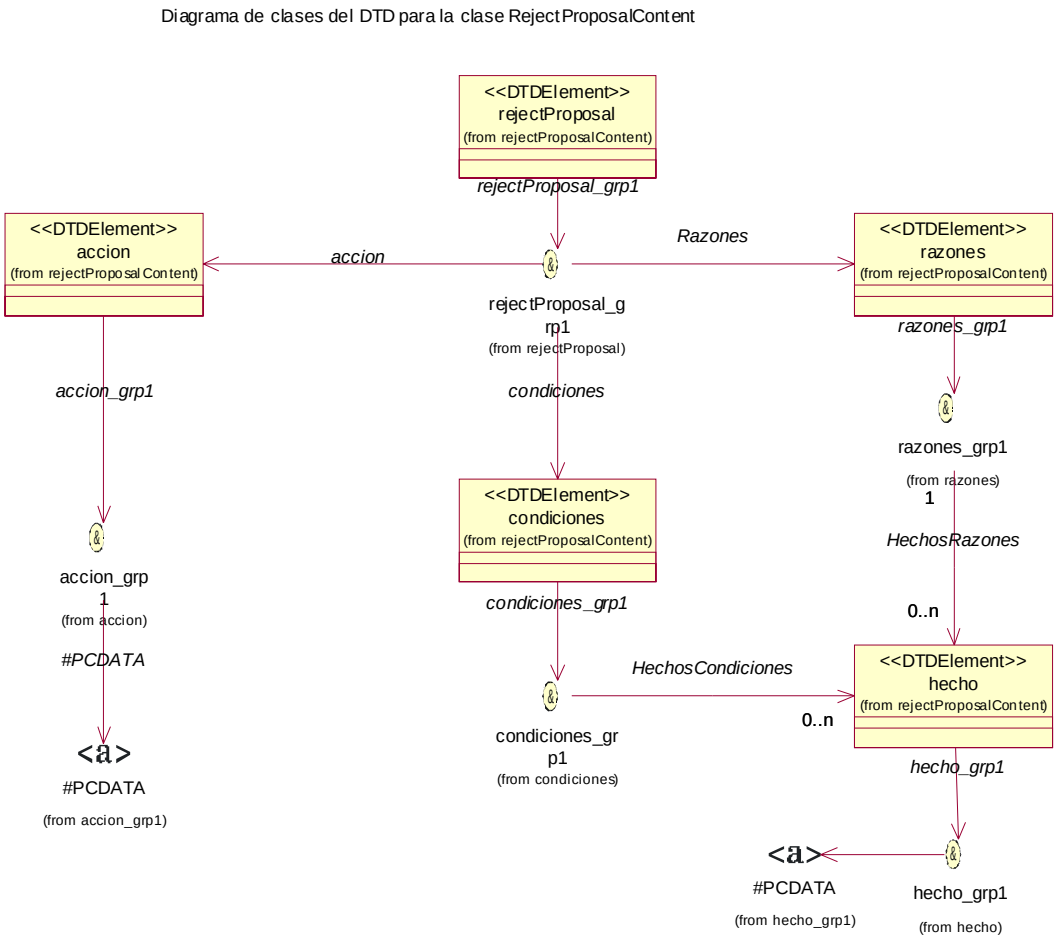

Figura 48 Diagrama del DTD del acto reject-Proposal

La tabla de la figura 49 muestra la lista completa de actos comunicativos implementados.

Acto	Resumen	Contenido del mensaje	Clase implementada
Accept-proposal	La acción de aceptar una propuesta, enviada previamente, para realizar una acción.	Una tupla que consiste de una expresión de acción que denota la acción a ser realizada,	AcceptProposalContent

		y una proposición que da las condiciones del acuerdo.	
Agree	La acción de aceptar la realización de una acción, posiblemente en el futuro.	Una tupla que consiste de una expresión de acción que denota la acción a ser realizada y una proposición que da las condiciones del acuerdo.	AgreeContent
Cancel	La acción en que un agente le informa a otro que ya no tiene más la intención de que el segundo agente realice alguna acción.	Una expresión de acción que denota la acción que ya no se debe intentar.	CancelContent
Cfp (Call for proposa)l	La acción de llamar por propuestas para realizar una acción dada.	Una tupla que contiene una expresión de acción que denota la acción a ser realizada, y una expresión que define una proposición que da las precondiciones de la acción.	CfpContent
Failure	La acción de decirle a otro agente que una acción fue intentada pero el intento falló.	Una tupla, que consiste en una expresión de acción y una proposición que da las razones del fallo.	FailureContent
Inform	El emisor le informa al receptor que una proposición es verdadera.	Una proposición	InformContent
Not-understood	El emisor del acto informa al receptor que percibe que el receptor ha realizado una acción pero que no comprende qué es lo que hizo. Un caso común particular es que no se entiende el mensaje que se ha recibido de otro	Una tupla que consiste de una expresión de acción, por ejemplo, un acto comunicativo, y una razón de explicación.	NotUnderstoodContent

	agente.		
propose	La acción de enviar una propuesta para realizar una determinada acción, dadas ciertas precondiciones.	Una tupla que contiene una descripción de acción, representando la acción que el emisor está proponiendo realizar, y una proposición que representa las precondiciones sobre la realización de la acción.	ProposeContent
Query-if	La acción de preguntar a otro agente cuándo una proposición dada es verdadera o no.	Una proposición	QueryIfContent
Refuse	La acción de rehusarse a realizar una acción dada y explica la razón de la negativa.	Una tupla, consiste de una expresión de acción y una proposición que da la razón del rehúso.	RefuseContent
Reject-proposal	La acción de rechazar una propuesta para realizar una acción durante una negociación.	Una tupla que consiste de descripción de acción y una proposición que forman la propuesta original que está siendo rechazada, y una proposición que denota la razón del rechazo.	RejectProposalContent
Request	El emisor solicita al receptor que realice alguna acción.	Una expresión de acción	RequestContent

Figura 49 Lista de actos comunicativos implementados con objetos de contenido

7.5 Propuesta del Acto *success*

Además de los actos comunicativos que se han presentado en la tabla anterior, se propone la inclusión de un acto más a la biblioteca de actos comunicativos de FIPA: el acto *success* para indicar que una acción ha sido realizada con éxito.

Resulta necesario tener un acto como *success* ya que esto facilita las cosas que tienen que ver con la comunicación de resultados al haber ejecutado una acción por un agente, acción que previamente fue solicitada por otro agente. Esto se hace aun más necesario si

consideramos que nuestro modelo de agentes que se está utilizando basa mucho de su funcionalidad en la colaboración entre agentes y el lenguaje de contenido está fuertemente ligado a conocimiento y acciones. En pocas palabras, con estas ideas se propicia el desarrollo de agentes en dominios colaborativos con agentes exponiendo sus capacidades a otros a través de la plataforma CAP y por tanto, requiriendo mejores mecanismos para la comunicación de los resultados de las acciones solicitadas.

Aprovechamos este espacio para definir el acto comunicativo siguiendo los requerimientos que FIPA establece para ello (FIPA-4, 2001).

En la idea de ir mejorando y extendiendo la lista de actos comunicativos, FIPA establece algunas guías fundamentales para la selección de actos comunicativos específicos.

Los criterios mínimos que deben ser satisfechos para que un acto comunicativo sea parte de la biblioteca de actos comunicativos de FIPA son los siguientes:

- Un resumen de la fuerza semántica del acto propuesto y el tipo de contenido.
- Una descripción detallada en lenguaje natural del acto comunicativo y sus consecuencias.
- Un modelo formal, escrito en el lenguaje semántico, de la semántica del acto, sus precondiciones formales, y su efecto racional.
- Ejemplos del uso del nuevo acto comunicativo.
- Documentación clara y suficiente.
- Aclarar la utilidad del nuevo acto comunicativo. En particular debe ser claro que la necesidad que se pretende resolver es razonablemente general.

En la tabla de la figura 50 se presenta la especificación del acto comunicativo *success* atendiendo los requerimientos de FIPA y el formato de presentación de cada acto.

resumen	La acción de decirle a otro agente que una acción fue intentada y fue exitosamente realizada.
Contenido del mensaje	Una tupla que consiste en una expresión de acción y una proposición para indicar los resultados de la acción.
Descripción	El acto *success* es una abreviación para informar que una acción se consideró posible por el emisor y fue completada con éxito, logrando un conjunto de resultados que desea informar al receptor. El agente receptor del *success* cree que: - La acción fue realizada - La acción es posible (o por lo menos era posible durante el tiempo en que el agente realizó la acción) La proposición que es el segundo elemento en la tupla del contenido, representa los resultados de ejecutar la acción.

Modelo formal	<i, success(j,a,ϕ)> ≡ <i, inform (j, (∃e)single(e) ∧ Done(e, Feasible(a) ∧ I_i Done(a)) ∧ ϕ ∧ Done(a))> FP: Bi ∝ ∧ ¬Bi (Bifj∝ ∨ Uifj∝) RE: Bj∝ Donde: ∝=(∃e)single(e) ∧ Done(e, Feasible(a) ∧ I_i Done(a)) ∧ ϕ ∧ Done(a)) El agente i le informa al agente j que, en el pasado, i tuvo la intención de realizar la acción *a* y *a* fue posible. i realizó la acción de intentar hacer *a* (esto es la acción / evento *e* es el intento de hacer *a*) y *a* fue realizada con éxito. ϕ es el conjunto de resultados obtenidos con la acción.
Ejemplo	El agente i le informa al agente j que ha realizado la acción open con éxito. (success :sender (agent-identifier: i) :receiver (set (agent-identifier j)) :content ((action agent-identifier :name j) (open “foo.txt”)) (success-message “ok”)) :language FIPA-SL

Figura 50 Acto *success* propuesto

CAPÍTULO 8. PROTOTIPO

Resumen

Se refiere al tema de las pruebas de las herramientas. Por medio de un prototipo se muestra el proceso que se debe seguir en la implementación de SMA. Con este caso de estudio se demuestra la funcionalidad de las herramientas y se permite observar el proceso de creación de agentes. La idea principal de esto es mostrar la forma de utilizar las diferentes características de los agentes y que pueden servir como recomendaciones a seguir para los desarrolladores de agentes con estas herramientas en futuras aplicaciones. Se hace esta demostración de manera distribuida y remota para especificar los detalles de la configuración de DCOM necesaria para la comunicación entre los agentes y la plataforma CAP.

Objetivo del capítulo

Mostrar el uso de las herramientas desarrolladas durante la tesis por medio del desarrollo de un prototipo en donde existen varios agentes comunicándose entre sí.

8.1 Introducción

Para mostrar la forma de uso de las ideas propuestas y el funcionamiento de las herramientas que se han desarrollado se presenta este ejemplo. Con esto se pretende facilitar la tarea de desarrollar aplicaciones de agentes en donde se combinen los elementos que conforman partes centrales de este trabajo de investigación.

La idea fundamental es ayudar a comprender el paradigma de agentes que se está desarrollando, así como también ilustrar el uso de las herramientas a través del desarrollo de una aplicación que utiliza varias clases de agentes.

El prototipo se enfoca en ir desarrollando la aplicación paso a paso buscando en todo momento explicar con especial interés las partes en donde se utilizan las herramientas de programación de agentes y SMA como son la herramienta CAP-AgentTool para la creación de agentes, la implementación de la funcionalidad de los agentes, el uso de las clases utilitarias para el apoyo en la programación de los agentes, y en general guiar en el proceso normal de una aplicación de agentes en el entorno de los controles ActiveX, hasta llegar a la manera de utilizar los controles de agente en aplicaciones de software.

El ejemplo se basa en un problema que está siendo estudiado actualmente en el Laboratorio de Agentes del CIC y que tiene que ver con la formación de redes de cadenas de suministro o SCN (*Supply Chain Network*). Se trata de la implementación de un sistema multiagente para formación de SCN. Este ejemplo ya ha sido explicado con más detalles en (Romero, Sheremetov, 2001) pero se ha hecho un rediseño de la aplicación para fines de demostrar la funcionalidad de las herramientas aquí descritas.

8.2 Planteamiento del problema.

El problema consiste en armar una red de cadena de suministros virtual por medio de la cual se satisfaga una demanda en el dominio de una empresa ensambladora de coches. Consiste en armar un producto determinado, haciendo uso de agentes de software a través de varias etapas de comunicación entre ellos, en un ambiente computacional distribuido, dinámico y flexible.

El proceso de creación de la cadena de suministros se desglosa en los siguientes pasos:

1. Existe un tipo de producto que se requiere armar, la etapa de la especificación de la demanda.
2. Un agente de software, aquí llamado agente integrador, recibe la petición de formar una cadena de suministros de acuerdo con el tipo de producto demandado.
3. El agente integrador busca los servicios de un agente del dominio de las SCN que le indique las partes requeridas para armar el producto solicitado.
4. El agente integrador se comunica con el agente de dominio, si es que encontró alguno, para solicitarle que le informe las partes del producto que necesita formar.
5. Cuando el integrador recibe, desde el agente de dominio, las partes del producto entonces crea agentes agrupadores, aquí llamados agentes de coalición, uno para

cada parte del producto. Cada uno de estos agentes de coalición se va a encargar de buscar y negociar las subpartes que forman a la parte que le corresponde.

6. Una vez creados los agentes de coalición necesarios, estos agentes se comunican con el agente de dominio de SCN para solicitarle las subpartes que forman la parte correspondiente y los servicios por medio de los cuales es posible obtener cada subparte.
7. Ya que conoce los servicios para obtener cada subparte, busca en el facilitador de directorios de la plataforma de agentes CAP los agentes proveedores de cada servicio.
8. Inicia un proceso de negociación con cada agente proveedor encontrado. Esta parte del algoritmo es simplificada para calcular el costo máximo que este agente está dispuesto a pagar por el servicio en cuestión.
9. El agente proveedor es simplificado de tal manera que produce respuestas a las solicitudes de propuestas recibidas en base a un costo que pide como pago al servicio que le están solicitando. Esto se lo comunica al agente de coalición que se lo solicitó.
10. Cuando todos los proveedores han dado su propuesta, incluida en ésta el costo del servicio, cada agente de coalición se comunica con el agente integrador para informarle que su coalición ha sido formada satisfactoriamente.
11. El agente integrador forma la cadena de suministro para el producto cuando todos los agentes de coalición le han informado que se han formado.
12. El agente inicializa su base de hechos y está listo para recibir más peticiones.

8.3 Desarrollo del sistema

A continuación se muestra el proceso de desarrollo de un SMA que se realizó siguiendo los pasos mencionados en el planteamiento del problema de formación de cadenas de suministros. Los siguientes son los pasos a seguir durante el desarrollo del sistema, estos se identifican para propósito de aclarar el panorama del lector en este ejemplo. De ninguna manera se establece que sea una metodología a seguir en el desarrollo de otras aplicaciones de SMA.

1. Identificar las clases de agentes requeridos.
2. Diseño de la aplicación enfocada a encontrar las relaciones que se establecen entre los diferentes agentes involucrados y otros componentes.
3. Proceso de creación de los agentes utilizando la herramienta CAP-AgentTool y los aspectos fundamentales de ésta para fines de demostración de su funcionalidad.
4. Implementación de la funcionalidad de un control de agente.
5. Compilación del control de agente.
6. Inserción de una instancia de agente a partir de un control de agente.
7. Implementación de la funcionalidad del SMA, comunicando a los agentes de la aplicación.
8. Ejecución del SMA

En la sección 3 del apéndice A se encuentran diagramas de diseño que detallan con más precisión la especificación de este sistema.

8.3.1 Identificar las clases de agentes requeridos

Iniciamos por identificar los diferentes tipos de agentes que intervienen en el problema. En el diagrama de clases de la figura 51 se ilustran mejor las clases de agente descritos a continuación:

Agente Integrador: Es el agente que se va a encargar de coordinar el proceso de formación de cadenas de suministros. Cuando se inicia el proceso este agente invoca su acción BuscarPartes para buscar las partes del producto especificado. Se comunica con el agente de dominio de la aplicación y crea a los agentes de coalición necesarios en el sistema.

Agente de Coalición: Es el agente que se va a encargar de determinar las subpartes que se requieren para armar una parte del producto que se le ha encomendado. A su vez, cuando conoce las subpartes, debe buscar los servicios que están registrados en algún agente de dominio. Posteriormente, encuentra agentes proveedores para que le ofrezcan los servicios que necesita.

Agente de Dominio: Un agente especial que se ha introducido para que brinde el conocimiento que identifica al dominio de aplicación de las SCN, y la funcionalidad estándar para buscar determinada información del dominio que es de interés para los demás agentes que participan en la aplicación. Actúa como un sustituto de agente de ontología de SCN muy limitado.

Agente Proveedor: Tiene como finalidad principal estar disponible a las solicitudes de servicios que ofrece a los agentes de la aplicación. Se supone que estos agentes son independientes de la aplicación de SCN y que en determinados momentos pueden establecer comunicación con otros agentes a fin de ponerse de acuerdo en la venta de un servicio que este ofrece. En el ejemplo, los agentes proveedores van a servir para dar servicios de vender determinadas piezas relacionadas con las autopartes.

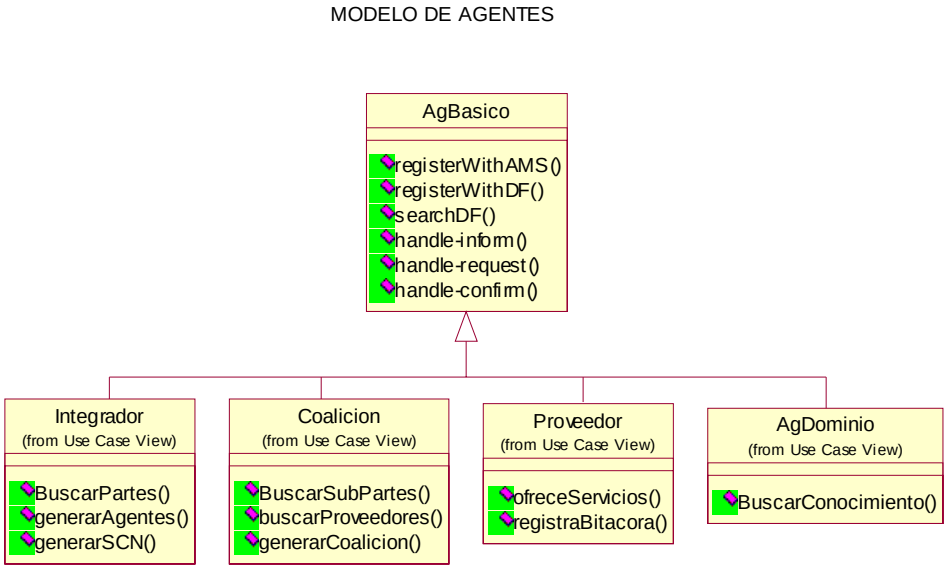

Figura 51 Diagrama de clases de los agentes

8.3.2 Modelo de Coordinación en el sistema

Todo comienza por el SMA, esta es la aplicación de software en donde se encuentra incrustado el control de agente integrador. A través de un servicio que este agente ofrece es que se puede invocar la iniciación del proceso de crear una SCN. Se puede ver aquí que el agente integrador y los agentes de coalición requieren comunicación directa con el agente del dominio de SCN. A su vez, los agentes de coalición están comunicados con el el agente proveedor para contratar servicios. Los agentes proveedores se comunican con los agentes de coalición que les solicitaron alguna propuesta de servicio, enviándoles sus propuestas. Al final, el agente integrador recibe las propuestas de cada agente de coalición que se formó dinámicamente y despliega la SCN en la interfaz del usuario. En el apéndice A, parte 3 se encuentran los diagramas de secuencia y de colaboración que modelan la interacción entre los agentes del sistema.

A continuación, en la figura 52 se muestra un diagrama en donde se puede ver la relación entre los diferentes agentes que participan en la aplicación.

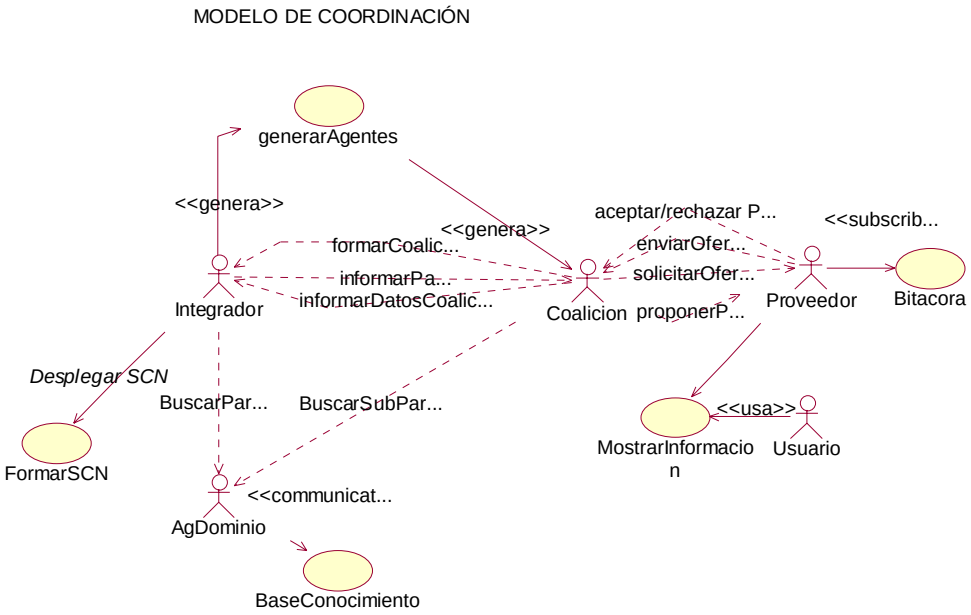

Figura 52 Modelo de coordinación de la aplicación

8.3.3 Proceso de creación de los controles de agente

Siguiendo los pasos que se indican en la herramienta para la creación de controles de agente CAP-AgentTool se inicia con la especificación del modo de creación. En esta pantalla debe seleccionar la opción de crear un agente nuevo. Por default está seleccionada esta opción así es que puede dar clic en el botón siguiente para continuar como aparece en la figura 53.

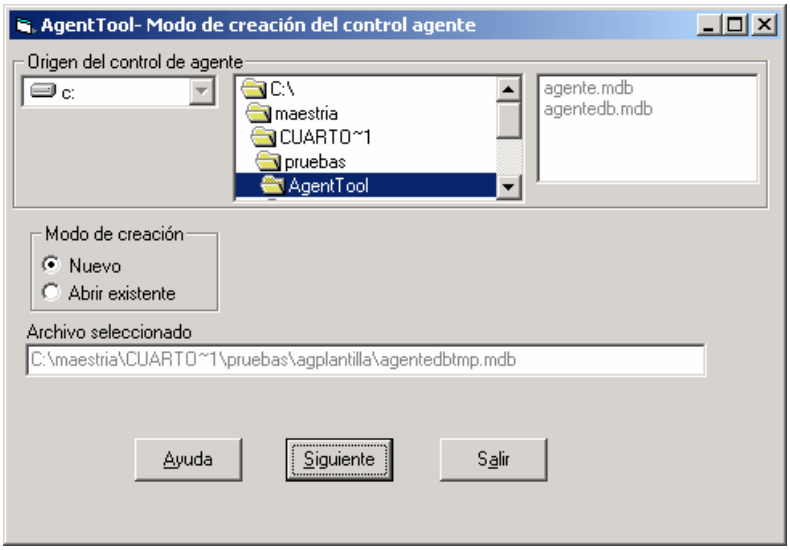

Figura 53 Seleccionando el modo de creación de los agentes

En el siguiente paso de la herramienta se le pide al usuario que especifique los datos que van a ayudar a la creación del agente (figura 54). Por ejemplo su ruta de destino en el equipo donde se va a guardar el código fuente y archivos adicionales para el trabajo del control, nombre del control, servidor CAP al que se va a conectar si es que va a trabajar de manera remota con respecto a la plataforma CAP. En este ejemplo se muestra la pantalla de creación del control del agente integrador. En la sección de *nicknames* aparece dado de alta el nombre coordinador; esto quiere decir que a los agentes de la clase integrador también se les puede buscar con el nombre de coordinador.

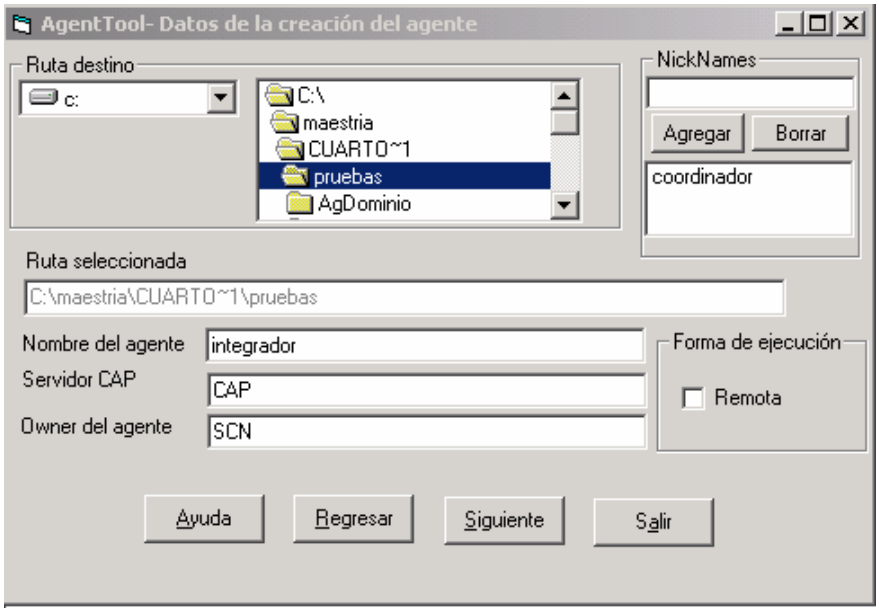

Figura 54 Datos del agente integrador

Luego sigue la especificación de aspectos fundamentales para la configuración de la comunicación del control de agente. En esta parte se debe especificar las ontologías que el

agente puede manejar, los lenguajes de comunicación con otros agentes y los lenguajes de contenido que puede utilizar para expresar el contenido de los mensajes ACL. En realidad no hay mucho que añadir aquí, únicamente las ontologías del agente ya que en lo que se refiere a ACL y lenguajes de contenido por el momento nada más está implementado fipa-acl como lenguaje de comunicación de agentes y fipa-rdf0 como lenguaje de contenido. Como se ve en la figura 55, al agente de dominio se le especifica que requiere entender la ontología "scn" para el contenido que va a comunicar en mensajes ACL.

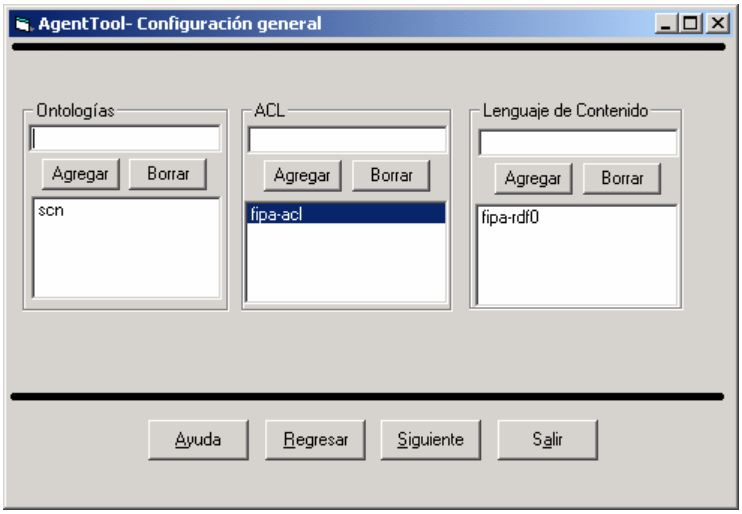

Figura 55 Etapa de configuración

A continuación se debe especificar el conjunto de hechos iniciales para cada agente. En la figura 56 se muestra la base de hechos que se le ha agregado al agente de dominio. En éste se puede apreciar hechos acerca de los productos manejados, sus partes y subpartes y servicios con los cuales se pueden obtener. De alguna manera se especifica una ontología para este dominio de aplicación. El agente de dominio ha declarado que puede manejar conocimiento de las ontologías de scn, autopartes y compupartes.

El siguiente paso es la especificación de las acciones que el agente puede realizar. Una acción se especifica sobre la base de tres cosas: su nombre, sus argumentos recibidos y sus resultados obtenidos. La lista de argumentos consiste en objetos de tipo hecho que se requieren para poder utilizar adecuadamente la acción. Además de esta lista de argumentos, el programador puede utilizar los argumentos que desee al implementar la acción, pero que no se consideran como obligatorios. La lista de resultados está compuesta por hechos que se deben obtener de manera obligatoria al final de la ejecución de la acción. Estos dos aspectos definen la semántica de una acción.

En la figura 57 se puede ver la pantalla de especificación de las acciones del control de agente de coalición. Esta clase de agentes declara las acciones para *BuscarProveedores*, *BuscarSubPartes* e *IniciarNegociación*. A su vez, los demás agentes declaran sus acciones públicas durante esta etapa de la herramienta.

Figura 56 hechos definidos en el agente de dominio

El proceso de creación de los agente muestra un breve resumen de los datos más importantes que fueron dados a través del proceso de creación seguido. Hasta aquí se llega con el proceso de creación de los controles de agente usando la herramienta CAP-AgentTool. Como resultado del proceso se tienen los proyectos de control ActiveX, para cada clase de agente, en Visual Basic que se debe completar como se indica en las secciones siguientes.

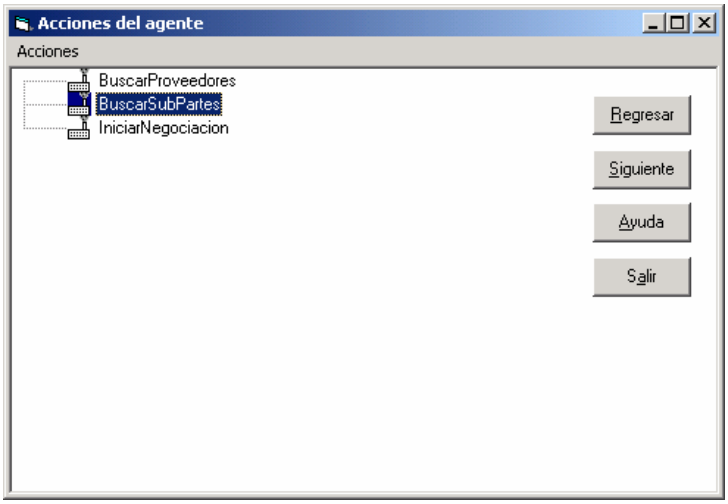

Figura 57 Acciones del agente de coalición

8.3.4 Implementación de la funcionalidad de un control de agente.

Continuando con nuestro ejemplo de SMA para la creación de SCN, se debe implementar la funcionalidad de los nuevos controles de agente que se necesitan. Hay que recordar que la herramienta ha creado los nuevos proyectos de control de agente en el lenguaje de programación Visual Basic y es necesario abrirlos desde la ruta que se especificó en la herramienta. Esto es para cada clase de agente de la aplicación.

Lo único que hace falta a estas alturas de la creación de un nuevo control de agente es implementar la funcionalidad de las acciones que le fueron especificadas con la herramienta CAP-AgentTool.

Para demostrar esta parte se tomará como ejemplo la tarea *IniciarCadena* del agente integrador. El código fuente de la acción al que se refieren los comentarios respectivos está puesto en el tipo de letra *itálica*.

Este es el encabezado de la acción. Como ya se ha mencionado antes, recibe como argumentos dos objetos de clase *collection*. Una para los argumentos y otra para los resultados obtenidos con la acción; su valor de retorno siempre es *boolean* para indicar si se ha realizado.

```
Public Function IniciarCadena(args As Collection, res As Collection) As Boolean
 'Todo método público del agente puede ser registrado como parte de un servicio
 'expuesto a otros agentes. Recibe un conjunto de argumentos de tipo hecho
 'Regresa un conjunto de hechos para comunicárselos al agente cliente
 Dim done As Boolean
 'Indica si la acción terminó de ejecutarse correctamente
 done = True
```

En esta sección de la acción se verifica que los argumentos recibidos sean los esperados por este método, de acuerdo a su especificación en la herramienta. Esto se hace a través del campo predicado del objeto hecho recibido como argumento. Si algún argumento no coincide con lo esperado entonces la acción regresa falso para indicar, a quien la haya solicitado, que no se realizó. En este ejemplo se puede ver que la acción *IniciarCadena* espera como argumentos dos objetos de clase *hecho*, uno para especificar el producto y otro hecho para denotar la ontología que se está utilizando. Nada de aquí se debe modificar ya que es colocado por la herramienta CAP-AgentTool de forma automática.

```
'Se checa el campo predicado del objeto hecho
 Dim hechoArg As Object
 For j = 1 To args.Count
   Set hechoArg = args.Item(CStr(j))
   If j = 1 Then
     If (Not hechoArg.predicado = "producto") Then
       MsgBox ("Error en argumentos recibidos: " + hechoArg.predicado + " Se esperaba:
producto")
       IniciarCadena = False
     End If
   End If
   If j = 2 Then
     If (Not hechoArg.predicado = "ontologia") Then
       MsgBox ("Error en argumentos recibidos: " + hechoArg.predicado + " Se esperaba:
ontologia")
       IniciarCadena = False
     End If
   End If
```

```
Next j
'******* Se debe implementar este método para dar la funcionalidad especificada aquí
```

La funcionalidad implementada debe seguir a este aviso. En el caso particular de la acción que se está revisando, se llevan a cabo las siguientes tareas:
1.- Recibe como parámetro el nombre del producto que se quiere integrar
2.- Verifica con el agente del dominio para ver si el producto solicitado está en el dominio

```
parser.Reset
parser.act = "request"
parser.SENDER_NAME = agnombre
parser.RECEIVER_NAME = "AgDominio1"
Dim objReqCont As requestContent
Set objReqCont = New requestContent
Dim acc As accion
Set acc = New accion
acc.Acto = "BuscarConocimiento"
acc.Actor = "AgDominio1"
Dim arg1 As Object
Dim arg2 As Object
Dim arg3 As Object

'El primer argumento de la función es un hecho que denota el producto a integrar
Set arg1 = args.Item(CStr(1))
'El tercer argumento es el tipo de petición que se hizo
'Este hecho se debe añadir a la base de hechos
Set arg3 = args.Item(CStr(3))
Me.AddHecho arg3

Dim producto As String
producto = arg1.objeto
'El segundo argumento se especifica así, denota el tipo de búsqueda
Set arg2 = New hecho
arg2.predicado = "tipoBusqueda"
arg2.sujeto = "hecho"
'El tipo 1 solicita verificar si el hecho existe en la kb del agente
arg2.objeto = "1"
acc.argumentos.Add Item:=arg1, Key:=CStr(1)
acc.argumentos.Add Item:=arg2, Key:=CStr(2)
x = acc.parse(acc)
'MsgBox ("accion a solicitar=" + acc.content)
x = objReqCont.parse(acc)
'MsgBox ("contenido del mensaje request")
'MsgBox (objReqCont.content)
parser.content = objReqCont.content
'El agente de dominio debe responder con in-reply-to="1"
parser.reply_with = "1"
```

```
 parser.parse

 Forward parser.Message
```

En la última sección se verifica que el resultado de la acción sea el esperado. Es decir, si una acción especifica que regresa un hecho de algún tipo, es obligación del programador introducir este hecho con sus valores respectivos, en la colección de resultados que se recibió como parámetro (res), para que este objeto *collection* pueda ser manejado como parte de la respuesta que el agente va a enviar como resultado de la acción. En este ejemplo se debe enviar el hecho *posibilidadJuego* como resultado.

```
'*******
 'Antes de salir de este método, checar el resultado de la ejecución
 If (done = True) Then
   'Verificar si la colección de resultados tiene los hechos especificados
   Dim hechoRes As Object
   For k = 1 To res.Count
     Set hechoRes = res.Item(CStr(k))
     If k = 1 Then
       If (Not hechoRes.predicado = "posibilidadJuego") Then
         MsgBox ("Error en resultados devueltos: " + hechoRes.predicado + " Se esperaba:
posibilidadJuego")
         IniciarCadena = False
         Exit Function
       End If
     End If
   Next k
   IniciarCadena = True
 Else
   IniciarCadena = False
 End If
End Function
```

8.3.5 Compilación del control de agente

Si el desarrollador del control de agente ha determinado que ya se ha implementado la funcionalidad de todas las acciones, entonces ya puede compilar el proyecto para crear el control de agente. No hay que olvidar que el control de agente se crea como un control ActiveX.

Para hacer esto, en el menú archivo de Visual Basic debe seleccionar la opción Generar nombreProyecto... . Esto hará que se compile y se genere el control ActiveX para el tipo de agente que se está trabajando. En la figura 58 se puede ver que se va a compilar el proyecto para el control del agente integrador. Durante el proceso de generación se lleva a cabo el registro del control en el *registry* de Windows, de tal manera que queda listo para ser insertado en las aplicaciones que lo soporten sobre la máquina en que se compiló el control.

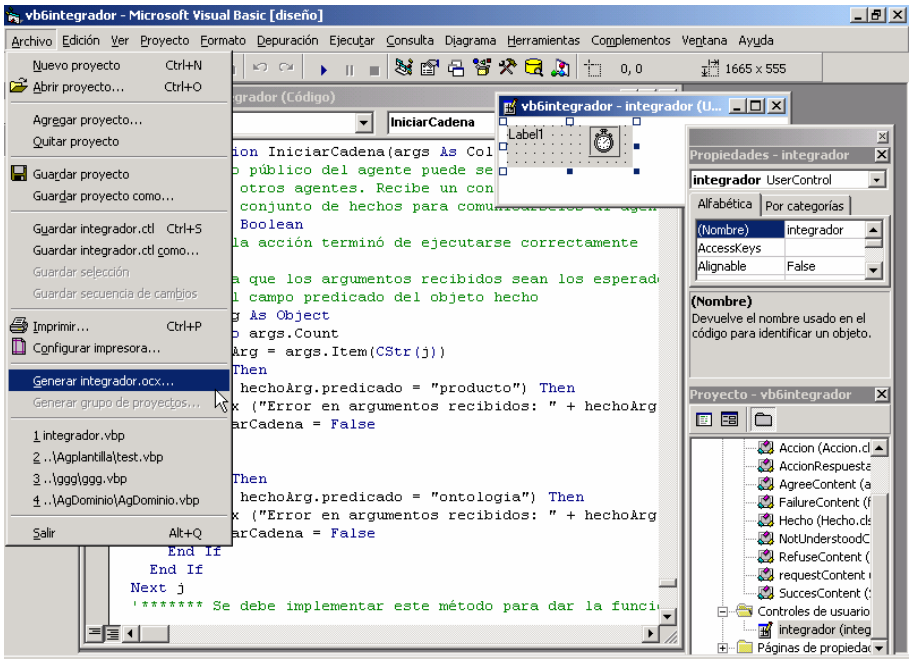

Figura 58 Compilación y generación del control de agente integrador

8.3.6 Inserción de un agente a partir de un control de agente

Si durante la compilación del nuevo control de agente no se han encontrado errores sintácticos, entonces ya se tiene un control ActiveX para la clase de agente. Este se encuentra en un archivo con extensión .ocx lo que quiere decir que ya se puede utilizar el control de agente en las aplicaciones de software en todo aquel lenguaje y entorno que brinde contenedores para controles ActiveX.

Es posible personalizar, para cada proyecto o contenedor de ActiveX, el conjunto de controles disponibles en el cuadro de herramientas de la aplicación. Cualquier control proporcionado debe estar en el cuadro de herramientas para que se pueda agregar a un formulario del proyecto o contenedor en general.

Se puede agregar controles ActiveX a un proyecto en Visual Basic si los agrega desde el cuadro de herramientas. Para agregar un control al cuadro de herramientas de un proyecto siga las siguientes instrucciones:

En el menú Proyecto, elija Componentes.

Aparecerá el cuadro de diálogo Componentes, tal como se muestra en la figura 59. Los elementos mostrados en este cuadro de diálogo incluyen todos los controles ActiveX registrados, objetos insertables y diseñadores ActiveX.

a) Para agregar un control (extensión de nombre de archivo .ocx) active la casilla de verificación que hay a la izquierda del nombre del control deseado.

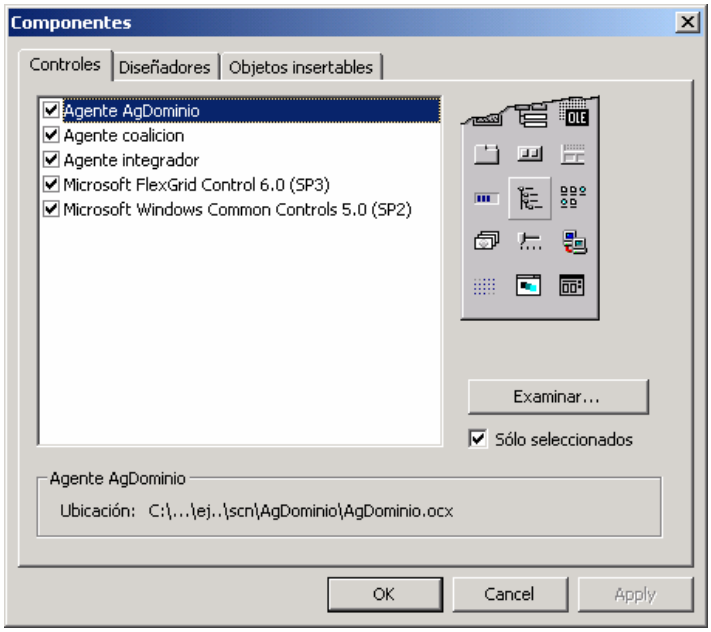

Figura 59 Añadiendo controles de agentes

Para ver los controles con extensiones de archivo .ocx, seleccione la ficha “Controles”.

Elija “Aceptar” para cerrar el cuadro de diálogo “Componentes”. Todos los controles ActiveX que haya seleccionado aparecerán ahora en el cuadro de herramientas.

Para agregar controles ActiveX al cuadro de diálogo “Componentes”, elija el botón “Examinar” y busque en otros directorios archivos con la extensión .ocx. Cuando agregue un control ActiveX a la lista de controles disponibles, Visual Basic activará automáticamente su casilla de verificación.

Cada control ActiveX viene acompañado de un archivo con extensión .oca. Este archivo almacena información de biblioteca de tipos y otros datos específicos del control. Los archivos .oca se almacenan habitualmente en el mismo directorio que los controles ActiveX y se vuelven a crear cuando sea necesario (los tamaños y las fechas de los archivos pueden cambiar).

Para quitar un control de un proyecto

a) En el menú “Proyecto”, elija “Componentes”.

Aparecerá el cuadro de diálogo “Componentes”.

b) Desactive la casilla de verificación que hay junto al control que desea quitar.

El icono del control desaparecerá del cuadro de herramientas.

No puede quitar ningún control del cuadro de herramientas si una instancia de ese control se está usando en cualquier formulario del proyecto.

8.3.7 Implementación de la funcionalidad del SMA, comunicando a los agentes de la aplicación.

En un breve resumen del proceso seguido hasta aquí se tiene que ya se ha planteado un problema de SMA de ejemplo, se han identificado los agentes que se requieren en la aplicación, se han creado los controles de agente para cada clase de agente identificado, se ha implementado la funcionalidad a través de la programación de las acciones especificadas para cada control, se ha explicado cómo se crean los controles de agente ActiveX a través del proceso de compilación en Visual Basic y se ha analizado la forma de configurar las aplicaciones que soportan controles ActiveX para poder insertar controles de agente.

Lo que sigue consiste en desarrollar la aplicación que utilice los controles de agente creados a través de su inserción en una aplicación de software para mostrar la funcionalidad de los controles de agente en un entorno de desarrollo de SMA. Para ello se va a desarrollar el SMA en Visual Basic 6.0. Además con esta aplicación se persigue la finalidad de mostrar la forma en que se deben utilizar las clases utilitarias que se han creado para manipular el contenido de los mensajes ACL, para el manejo del conocimiento de los agentes y la interacción con los componentes COM de la plataforma.

Se puede ver el entorno de desarrollo de Visual Basic 6.0 para el proyecto de SMA de una SCN en la figura 60. A continuación se explica algunos aspectos interesantes para lograr comprender el entorno con relación a los controles de agente y la implementación del SMA. El código fuente completo de este prototipo se incluye en el apéndice F.

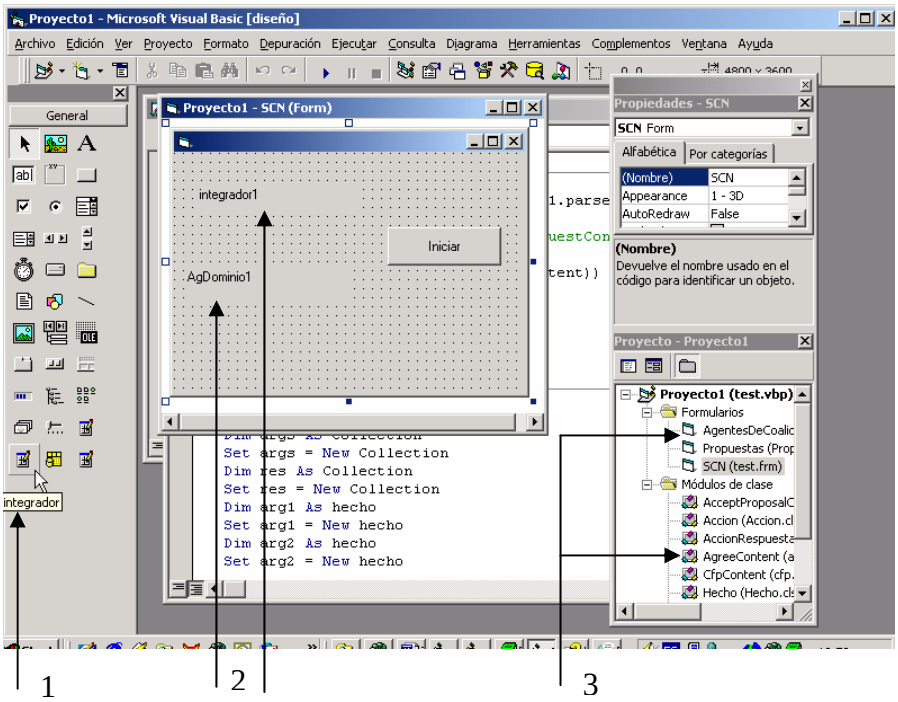

Figura 60 Ambiente de desarrollo del SMA

1.- Controles de agente que han sido agregados a la barra de herramientas. En este proyecto se han agregado los controles del agente integrador, el agente de dominio y el agente de coalición.

2.- Se han insertado dos instancias de controles de agente: una para el agente integrador y otra para el agente de dominio. Para el caso del agente de dominio, forma parte de la aplicación, pero bien puede estar en otra aplicación en otro programa, incluso en otra computadora.

3.- Componentes del proyecto. El proyecto está formado por dos tipos de componentes: formularios para la interfaz del SMA y clases utilitarias que han sido agregadas para soportar comunicación entre los agentes y para manejo de contenido.

El formulario que se está viendo en la figura 60 contiene las instancias de los controles de agente para el integrador y para el agente de dominio. Además tiene un botón de comando llamado Iniciar para iniciar el procesamiento del programa. Por medio de este botón se invoca una acción del agente integrador que permite iniciar el proceso de creación de la SCN de manera explícita; en este caso se solicita la formación de una SCN para armar un auto de acuerdo con el algoritmo establecido en el planteamiento del problema de esta sección. Adicionalmente, el agente integrador puede recibir la misma solicitud de iniciar la SCN desde algún mensaje ACL *request* que provenga de otro agente en otro sistema, tal vez en otro equipo en la red, si es que se decide que este agente integrador publique este servicio en el facilitador de directorios de la plataforma CAP.

El formulario que se llama Propuestas sirve para desplegar el contenido de las propuestas que han sido aceptadas con los agentes proveedores para formar parte de la SCN, como resultado final del procesamiento del programa.

Siguiendo el algoritmo de la aplicación se tiene que el agente integrador requiere crear agentes de coalición para que se encarguen de buscar los proveedores de cada una de las partes del producto solicitado. Por esta razón se tiene el formulario *AgentesDeCoalicion*, por medio del cual se cargan las instancias de los agentes de esta clase dinámicamente.

Por otra parte, el proyecto del SMA de SCN contiene un conjunto de módulos de clase que permiten al programador manipular el contenido que se maneja por el agente (hechos, acciones y respuesta de acciones) y el contenido que se requiere comunicar en cada uno de los actos comunicativos de ACL que han sido implementados. Para ver más detalles de las clases utilitarias implementadas en este trabajo ver las secciones 7.3, 7.4 y 7.5.

Para mostrar mejor la forma de utilizar las clases, enseguida se va a explicar a grandes rasgos la forma en que se ha implementado el SMA. Se utilizarán partes del código fuente de la implementación para mostrar aspectos clave, desde el punto de vista del uso de las clases. No es objetivo de este trabajo enseñar el paradigma de programación de agentes basados en FIPA, por lo cual se omite la parte de explicar a detalle todos y cada uno de los pasos de la implementación de la comunicación entre los agentes de la aplicación. Se considera por ello que la persona que vaya a utilizar estas herramientas conoce, con cierto

nivel de detalle, la labor de la programación de agentes y el paradigma basado en agentes FIPA y ACL.

Envío, Recepción y Manejo de los mensajes ACL

El SMA de SCN funciona de la siguiente manera:

Cuando se carga el formulario de la aplicación se inicializan los agentes. Esto significa que al invocarse el método *AgInicializar* los agentes se registran en la plataforma CAP.

```
Private Sub Form_Load()
  rr = integrador1.AgInicializar
  rr2 = AgDominio1.AgInicializar
  If (rr <> 1) Then
    MsgBox ("Error al inicializar el agente" + integrador1.agnombre)
    End
  End If
  If (rr2 <> 1) Then
    MsgBox ("Error al inicializar el agente" + AgDominio1.agnombre)
    End
  End If
  Load Propuestas
End Sub
```

Al hacer clic en el botón Iniciar se realiza lo siguiente:

```
  Dim args As Collection
  Set args = New Collection
  Dim res As Collection
  Set res = New Collection
  Dim arg1 As hecho
  Set arg1 = New hecho
  Dim arg2 As hecho
  Set arg2 = New hecho
  Se crean dos objetos de clase hecho para especificar el tipo de producto que se quiere
armar con la SCN y la ontología a utilizar
  arg1.predicado = "producto"
  arg1.sujeto = "scn"
  arg1.objeto = "auto"
  arg2.predicado = "ontologia"
  arg2.sujeto = "scn"
  arg2.objeto = "autopartes"
  Dim arg3 As hecho
  Set arg3 = New hecho
```

Se especifica el hecho para denotar el tipo de petición que se está atendiendo, en este caso es local porque se va a invocar el método de *iniciarCadena* desde la aplicación. Si se

invocara el método atendiendo a una petición de otro agente entonces el tipo de petición tendría que ser remoto para el valor de este hecho.

```
arg3.predicado = "TipoPeticion"
arg3.sujeto = "scn"
arg3.objeto = "local"
```

Se añaden los hechos a la lista de argumentos que se va a enviar

```
args.Add Item:=arg1, Key:=CStr(1)
args.Add Item:=arg2, Key:=CStr(2)
args.Add Item:=arg3, Key:=CStr(3)
```

Se añade este hecho en la base de hechos del agente, para indicar que actualmente está procesando esta acción en el agente integrador.

```
Dim h As hecho
Set h = New hecho
h.predicado = "procesando"
h.sujeto = agnombre
h.objeto = "scn"
integrador1.AddHecho h
```

Se invoca la acción que da inicio la comunicación entre agentes que implementan el algoritmo de formación de SCN.

```
Dim r As Boolean
  r = integrador1.IniciarCadena(args, res)
End Sub
```

A continuación se muestra parte de la implementación del método *IniciarCadena* del agente integrador. Una vez que se invoca este método, se verifican sus argumentos y entonces empieza enviando un mensaje ACL-*request* al agente de dominio, llamado AgDominio1.

Cada agente tiene un objeto *parser* para la manipulación de los mensajes ACL. En este caso se utiliza para el envío de un mensaje.

```
parser.Reset
parser.act = "request"
parser.SENDER_NAME = agnombre
parser.RECEIVER_NAME = "AgDominio1"
```

Se crea y utiliza una clase utilitaria para manejar el contenido de un mensaje *request*. Esta clase está implementada para construir el contenido de acuerdo con la semántica y sintaxis que FIPA especifica para el contenido de *request.*

```
Dim objReqCont As requestContent
Set objReqCont = New requestContent
```

El contenido de un *request* necesita un objeto de clase acción. Por medio de la clase acción se puede manejar una expresión de acción, basados en la especificación del lenguaje de contenido FIPA-RDF0. Una acción está compuesta por el acto (acción que se solicita),

actor (agente al que se solicita realice la acción) y una lista de argumentos requeridos por la acción.

```
Dim acc As accion
Set acc = New accion
acc.Acto = "BuscarConocimiento"
acc.Actor = "AgDominio1"
Dim arg1 As Object
Dim arg2 As Object
Dim arg3 As Object

'El primer argumento de la función es un hecho que denota el producto a integrar
Set arg1 = args.Item(CStr(1))
'El tercer argumento es el tipo de petición que se hizo
'Este hecho se debe añadir a la base de hechos
Set arg3 = args.Item(CStr(3))
Me.AddHecho arg3

Dim producto As String
producto = arg1.objeto
'El segundo argumento se especifica así, denota el tipo de búsqueda
Set arg2 = New hecho
arg2.predicado = "tipoBusqueda"
arg2.sujeto = "hecho"
'El tipo 1 solicita verificar si el hecho existe en la kb del agente
arg2.objeto = "1"
acc.argumentos.Add Item:=arg1, Key:=CStr(1)
acc.argumentos.Add Item:=arg2, Key:=CStr(2)
```

La clase *accion* tiene el método *parse* para construir la representación en XML del contenido de la acción para poder utilizarlo como elemento *content* de un mensaje como *request*.

```
x = acc.parse(acc)
```

Se construye la representación en XML de la acción que va a ser el contenido del mensaje *request* que se quiere enviar.

```
x = objReqCont.parse(acc)
```

Se asigna la propiedad *content* de la clase *requestContent* al campo *content* del objeto *parser*. Este es el contenido del mensaje ACL.

```
parser.content = objReqCont.content
'El agente de dominio debe responder con in-reply-to="1"
parser.reply_with = "1"
```

La clase *parser* del agente convierte todo el mensaje ACL a un formato en XML para que sea enviado el mensaje a la plataforma CAP.

```
parser.parse
```

El agente integrador utiliza su método *forward* para enviar el mensaje ACL. Para ello utiliza el campo *message* del objeto *parser*. Este campo contiene el mensaje ACL completo junto con su contenido (en este caso una acción), todo en XML.
Forward parser.Message

De igual manera que se utilizó la clase *requestContent* para formar el contenido de un mensaje *request* (cumpliendo con la sintaxis de *request*) ahora se debe realizar el proceso inverso consistente en recibir el mensaje ACL por parte del agente del dominio a quien se dirigió el mensaje anterior. Obviamente, para ello se va a utilizar también un objeto de la clase *requestContent*, ya que esta es la que sabe cuál es la sintaxis y semántica del contenido que viene en el mensaje. De esta forma el agente que recibe el *request* sabe cómo manejar el contenido y continuar el procesamiento. El siguiente segmento de código muestra la recepción del mensaje *request* que se ha enviado.

El agente de dominio se da cuenta que recibe un mensaje *request* y entonces dispara su evento *handleRequest* para permitir al programador de la aplicación manipular los mensajes.

El agente de dominio utiliza el método *deparse* de su objeto *parser* para extraer el mensaje ACL que le ha llegado y obtener cada uno de los campos que forman el mensaje.
AgDominio1.parser.deparse (AgDominio1.parser.Message)

```
Dim a As Object
'******Prueba de uso de la clase requestContent
Dim ar As New requestContent
```

Entre los campos que llegan está el contenido del mensaje. Para saber cómo manipular el contenido de *request* está la clase *requestContent*, la misma con la que el agente emisor del mensaje lo formó.

```
If (ar.deparse(AgDominio1.parser.content)) Then
```

Por medio del método *deparse* de la clase *requestContent* es posible obtener la descripción del contenido (que se encuentra codificado en un formato de XML) y forma los objetos que la sintaxis del contenido especifica. En este caso, la sintaxis de *request* dice que se debe comunicar una expresión de acción, así que la función de *deparse* es obtener un objeto *accion*. Este objeto se llama *accion* en la clase *requestContent.*

```
  Set a = New accion
  Set a = ar.accion
End If
Dim res As New Collection
```

Por último el agente de dominio decide realizar esta acción inmediatamente que se lo solicitan. Por eso llama su método *InvocarAccion* con la acción recibida (a) como argumento.
AgDominio1.InvocarAccion a

Esta es la forma en que se debe utilizar cualquier clase para expresar el contenido de un mensaje ACL. Si se quiere enviar otro mensaje ACL con otro acto comunicativo, entonces el programador debe utilizar la clase utilitaria que implemente la sintaxis de ese acto. Este

mecanismo es el mismo para enviar y recibir mensajes ACL, tal como se explicó en este ejemplo.

Para completar el proyecto de ejemplo falta mencionar que se tiene corriendo un agente proveedor al que se ha llamado refaccionaria1. Este agente forma parte de otra aplicación independiente del proyecto donde se encuentran funcionando los agentes integrador, de dominio y de coaliciones. Su funcionalidad se restringe a recibir solicitudes de propuestas (mensajes ACL *cfp*) por otros agentes y contestar con mensajes ACL *propose*.

8.3.8 Ejecución del SMA

Si ya se tiene implementada la comunicación requerida entre los agentes entonces ya se puede correr el SMA y ver su comportamiento.

Para ver cómo funciona el SMA de SCN se siguen los siguientes pasos:

a) Correr el proyecto de SCN.
b) Correr el proyecto del proveedor
c) Resultados. Automáticamente los agentes se comunican y como resultado se obtiene una lista con los agentes proveedores, los servicios y costos del servicio.

a) Correr el proyecto de SCN. El prototipo está compuesto por dos programas separados. En uno de ellos se corren los agentes integrador, de dominio y de coalición (test.exe, figura 61). En el otro programa se ejecutan los agentes proveedores de la aplicación (testProveedor.exe). En este paso de la ejecución del prototipo se puede ver la ejecución del programa test.exe. La plataforma CAP y este programa están ejecutándose en la máquina Agente2000 de la red del Laboratorio de Agentes del CIC.

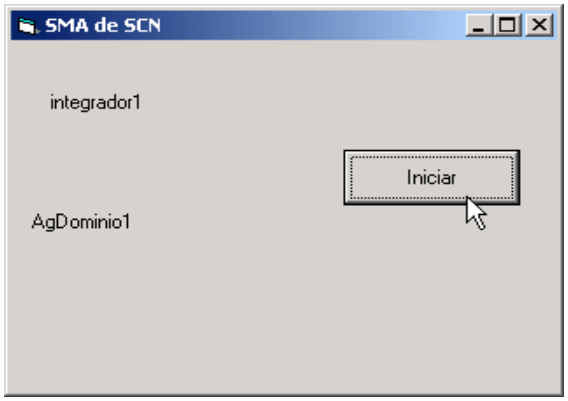

Figura 61 Ejecución del programa test.exe

b) Correr el proyecto del proveedor.

Para este ejemplo se va a correr esta aplicación de manera remota para mostrar los detalles que se deben considerar al hacer aplicaciones con agentes distribuidos en una red. El programa testProveedor.exe se va a ejecutar en una máquina distinta a donde se

encuentran los agentes del SMA y para ello se debe configurar algunos aspectos. Los detalles de esta configuración corresponden a una máquina con Windows 2000.

1.- Instalar el servidor COM de la plataforma CAP en la máquina cliente. El servidor COM se encuentra en el programa AP.exe que se distribuye con la herramienta CAP-AgentTool. Instalar, en esta ocasión, significa dar de alta el servidor COM en el *registry* de Windows. Para hacerlo debe correr el servidor COM desde su ubicación en la máquina cliente con el modificador */Regserver*, por ejemplo:

c:\maestria\cuartosemestre\plataforma\Ap.exe /*Regserver*

2.- Se debe configurar DCOM en la máquina cliente para que las aplicaciones que requieran el servidor COM de CAP se comuniquen con este. Para esto se debe correr el programa dcomcnfg.exe (figura 62). La aplicación que interesa configurar se registra con el nombre de "ACC de la Plataforma FIPA-DCOM".

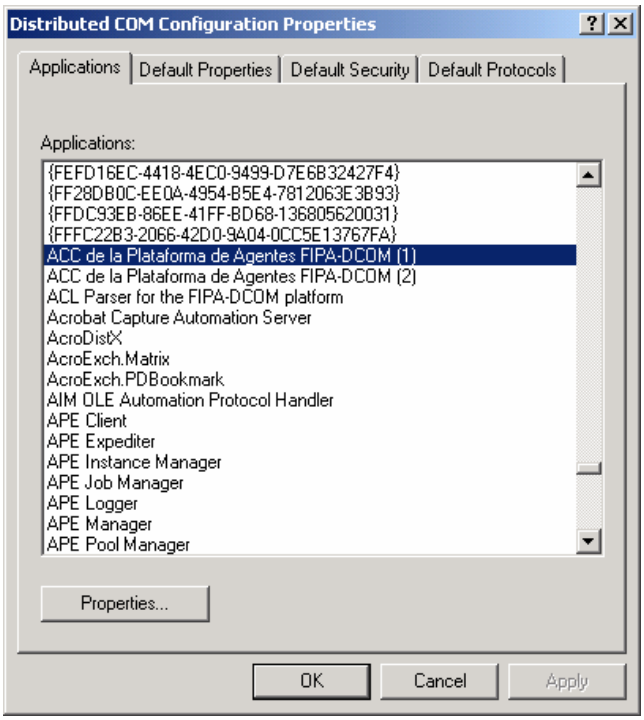

Figura 62 Configuración de DCOM con dcomcnfg.exe en el cliente

3.- En la pestaña de propiedades por default de esta ventana se debe configurar la opción para permitir DCOM en esta computadora.

4.- En el botón propiedades de *dcomcnfg.exe* se debe hacer clic para desplegar los detalles de configuración del servidor COM de la plataforma CAP. Seleccione la pestaña de ubicación para seleccionar la opción de "*Run application on the following computer:*" y seleccione el equipo donde se encuentra la plataforma instalada. En el ejemplo se ve que esa máquina es Agente2000 (figura 63).

5.- El acceso al servidor COM puede ser restringido a un usuario administrador en particular que se encargue de validar y dar permiso a los clientes que requieran trabajar con la plataforma CAP.

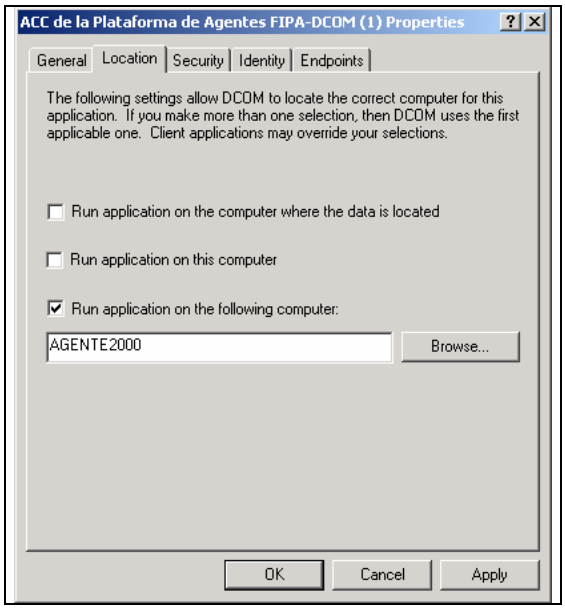

Figura 63 Especificando el servidor de la plataforma CAP

6.- Por último, se puede configurar los protocolos de comunicación utilizados por el cliente para acceder a los servicios del servidor. En el caso de este ejemplo se utilizan el protocolo TCP/IP en primer lugar y de no responder se intenta con NetBEUI.

7.- La pestaña General despliega los datos de la configuración. El nombre de la aplicación es el nombre del servidor COM de la plataforma CAP, el tipo de aplicación es servidor local o remoto lo cual indica que este servidor ya está configurado para hacer solicitudes a otras máquinas. La ruta del servidor se refiere a la ubicación física donde se instaló el servidor en el cliente (ver paso 1). También aparece el nombre de la computadora remota que fue seleccionada donde se encuentra la plataforma CAP instalada.

8.- Ahora ya se puede configurar el agente de la aplicación testProveedor.exe para que se comunique con la plataforma CAP que se encuentra instalada en la máquina Agente2000. Este programa se encuentra en una computadora distinta a la que tiene corriendo el programa test.exe. En la página de propiedades de los controles de agente está el campo Plataforma (figura 64). Entre otros datos, se puede capturar el intervalo de búsqueda de los mensajes dirigidos al agente, su identificador de agente, entre otros. Ahí también, se debe introducir el nombre de la máquina que tiene la plataforma CAP que se va a utilizar por el agente de la aplicación, en el ejemplo se trata de Agente2000. Si los los agentes van a conectarse con una plataforma CAP que está instalada en la misma máquina donde se encuentran estos, entonces este campo debe quedar vacío.

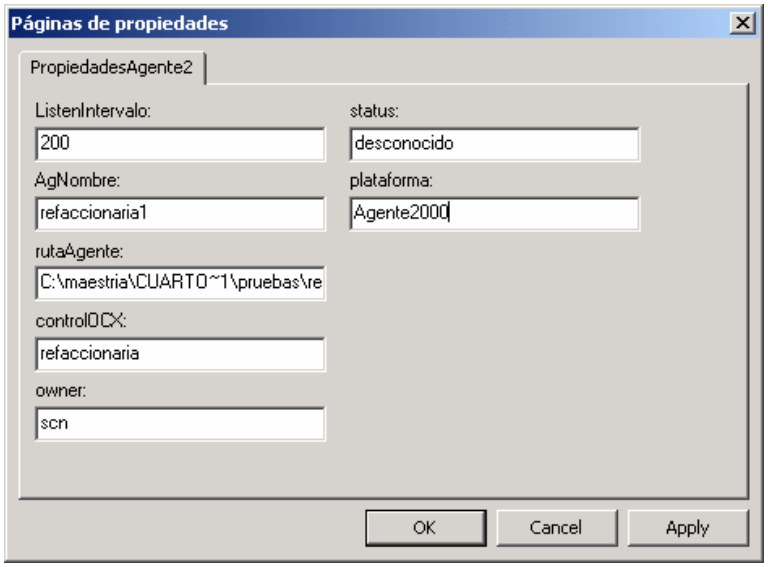

Figura 64 Especificación del servidor CAP en el agente proveedor

En este momento ya es posible ejecutar el programa testProveedor.exe. La interfaz de este programa se ve en la figura 65.

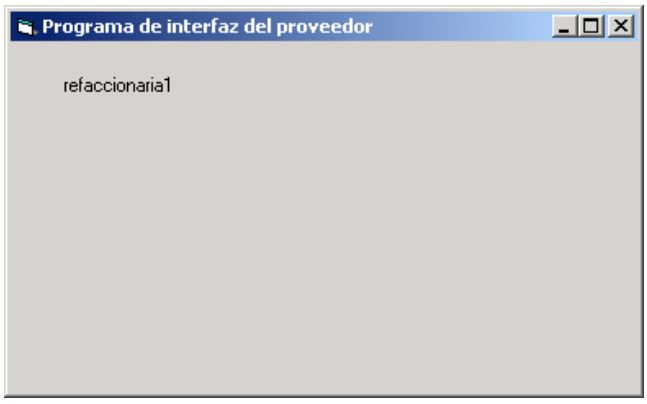

Figura 65 Aplicación del Agente proveedor en ejecución

Los dos programas necesarios por la aplicación ya están ejecutándose en procesos separados y en máquinas distintas. Para iniciar el Sistema MultiAgentes el usuario debe de dar clic en el botón Iniciar del programa test.exe. Esto hace que el agente integrador comience a comunicarse con el agente de dominio para buscar las partes del producto (auto) que se desea formar. Después de eso, genera los agentes de coalición necesarios para cada una de las partes que forman el auto. En la figura 66 se ve la ventana del agente Ag_integrador_auto_carrocería, que es uno de los agentes de coalición formados, y se va a encargar de comunicarse con el agente proveedor para solicitarle los servicios de venta de las partes del auto que conforman la carrocería. Este tipo de agentes son creados de manera dinámica durante la ejecución del sistema. Cuando tiene las propuestas del proveedor las acepta y le informa al agente integrador los datos del servicio que contrató para que se forme la SCN.

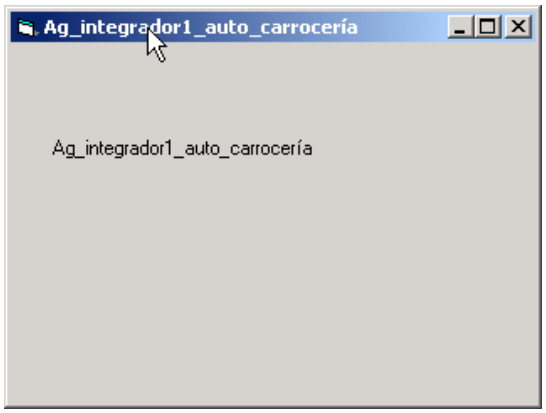

Figura 66 Agente de coalición creado

c) Resultados. El prototipo de SCN llega hasta la fase en que los agentes de coalición contratan los diferentes servicios requeridos para formar el producto que se demanda y le informan al agente integrador los datos respectivos. Como se ve en la figura 67, los datos que se despliegan en la pantalla de interfaz de la aplicación son el nombre del proveedor, el servicio negociado, la sub-parte que se compra con dicho servicio y su costo.

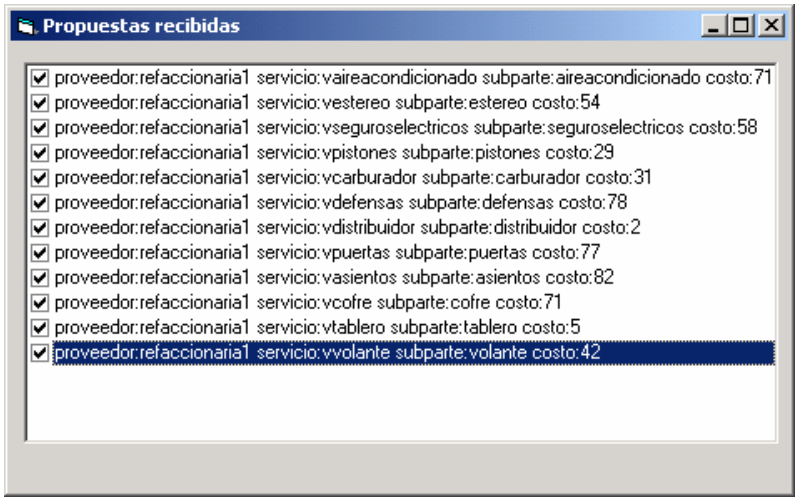

Figura 67 Resultados del SMA de SCN

CAPÍTULO 9. CONCLUSIONES

Resumen

En este apartado se presentan los resultados y contribuciones obtenidos durante el desarrollo del trabajo. Además, se mencionan las conclusiones generales que tienen que ver con la solución presentada y con respecto a la problemática planteada. Por último se comentan las limitaciones y algunas líneas de investigación identificadas para continuar con el trabajo relacionado a esta tesis, como trabajo a futuro, para cubrir la investigación y desarrollo de los temas que no se contemplaron como parte de la solución.

Objetivos del Capítulo

Mencionar los resultados obtenidos con el presente trabajo.

Listar las conclusiones generales a las que se ha llegado con la investigación.

Establecer las líneas de investigación abiertas para trabajo futuro relacionado con esta tesis.

9.1 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 conjunto de herramientas que permiten a los desarrolladores de aplicaciones utilizar las ideas orientadas a agentes, en ambientes de programación basados en Windows; dichas herramientas permiten crear los programas de agentes según las especificaciones de FIPA para la instanciación, despliegue y operación de agentes de software que funcionan bajo las reglas administrativas de una plataforma compatible con FIPA, como es el caso de la plataforma CAP.

Los agentes componentes que se pueden crear están basados en el modelo COM y DCOM de Microsoft, a través de la tecnología de los controles ActiveX. De esta manera se pueden aprovechar las bondades que los controles ActiveX ofrecen a los programadores que los utilizan, pero desde el punto de vista de agentes, como por ejemplo su facilidad de uso, la reusabilidad y extensibilidad. Esto viene a ayudar a que el paradigma orientado a agentes pueda estar más accesible para los entornos de programación de Windows, lo cual es algo que se puede considerar novedoso, si tomamos en cuenta que la mayor parte de las herramientas de programación de agentes están basadas en la plataforma de Java.

El concepto central de la tesis es el concepto de agente. Aunque la finalidad principal es crear Sistemas MultiAgente, no se puede conceptualizar un SMA sin antes definir las características básicas de un agente y su entorno computacional. Por ello, merece especial interés mencionar que se estableció un marco general de ideas que permite ubicar el contexto de las aplicaciones de agentes que se quiere desarrollar con esta infraestructura. Este marco conceptual se basa en las ideas centrales del modelo de agentes colaborativos para la creación de aplicaciones distribuidas heterogéneas, cuya comunicación está dada a través de los componentes de la plataforma CAP utilizando en todo momento un lenguaje de comunicación de agentes. Estas definiciones y modelos sirven de base para la implementación de agentes y de SMA a través de las herramientas.

Un concepto importante que forma parte de las herramientas construidas es la plantilla de agente básico. Por medio de esta plantilla es posible que cada clase de control de agente nuevo herede el comportamiento y funcionalidad de agente mínimo que se les ha atribuido desde su diseño como controles ActiveX de agentes. La plantilla contiene los elementos principales que permiten que un control ActiveX sea utilizado como un agente debido a que se le añaden los mecanismos para su comunicación con los componentes COM de CAP (sea ésta local o remota), propiedades de configuración del control, detección de mensajes ACL que son enviados entre agentes, eventos para programar la reacción del agente a los mensajes recibidos, invocación de acciones remotas, funciones para la manipulación de la base de conocimientos y de acciones públicas de los agentes, etc. Todas estas características son propias de todo agente que se crea a través de esta plantilla.

Se construyó la herramienta CAP-AgentTool para permitir la creación de nuevos controles ActiveX como agentes. Por medio de CAP-AgentTool es posible ir especificando las

características que van a formar parte del nuevo tipo de control de agente. La motivación de esta herramienta es facilitar el proceso de creación de agentes basados en controles ActiveX para que el programador se concentre en las características propias de su agente y no tenga que preocuparse por los detalles de la implementación inherentes a la tecnología subyacente. De esta manera se pueden especificar detalles como el conocimiento inicial de cada clase de agente, las acciones públicas que ofrece a otros agentes, las ontologías que necesita en determinados dominios, aunado a las características establecidas en la plantilla de agente básico, de la cual la herramienta hereda su código fuente para cada nuevo control.

Se desarrolló una variante del lenguaje de contenido FIPA-RDF0 para expresar el conocimiento manejado por los agentes. La implementación del lenguaje de contenido FIPA-RDF0 que se propone es una adaptación que se hace para permitir a los agentes trabajar con contenido explícitamente definido en los mensajes ACL. Los elementos principales del lenguaje de contenido son objetos, proposiciones o hechos y acciones.

Por otra parte, se implementaron las clases utilitarias para complementar el proceso de programación de la funcionalidad del conocimiento de los agentes. Estas clases ayudan a manipular hechos y acciones. Además, habilitan a los programadores para que sus agentes puedan crear objetos de contenido de acuerdo con los elementos del lenguaje utilizado, de manera que puedan manejar sus hechos y acciones internos en los diferentes momentos que sea necesario, de acuerdo con las necesidades de cada aplicación.

Se creó un conjunto de objetos de contenido que permiten manipular el contenido de los mensajes ACL que se comunican entre agentes. Estos objetos de contenido están basados en la semántica y sintaxis de los *performatives* o actos comunicativos soportados en el lenguaje FIPA-ACL. Se han creado objetos de contenido para 12 actos comunicativos. La característica común entre estos actos comunicativos es que han sido considerados para su implementación ya que especifican en su contenido proposiciones o acciones o ambos, que son los elementos retomados del lenguaje de contenido.

Los objetos de contenido para el conocimiento de los agentes y para el contenido de los *performatives* implementados utilizan el esquema de codificación en XML por default. Para ello, cada objeto de contenido implementado tiene una función para crear la descripción del objeto de contenido en formato XML para su envío; a su vez, tiene una función para leer de XML y obtener el objeto respectivo.

Se desarrolló la especificación del *performative success* para manejar la acción de informar el resultado de la ejecución de acciones. A su vez, este *performative* se está proponiendo para añadirse a la biblioteca de actos comunicativos de FIPA. La justificación de la necesidad de este acto comunicativo es que el ambiente colaborativo de los agentes está fuertemente relacionado a la solicitud de acciones entre agentes, por lo cual se detectó la importancia de tener una forma de comunicación de los resultados de la ejecución de las acciones solicitadas entre agentes que sea más clara y eficiente que lo que hasta ahora FIPA establece para este caso de comunicación.

Por último, se presenta un caso de estudio o prototipo que muestra la forma en que se pueden utilizar las herramientas y trata de guiar en el proceso de creación de una pequeña

aplicación de SMA para ilustrar los puntos centrales del trabajo. El propósito de este prototipo es que los usuarios de estas herramientas conozcan los diferentes pasos y la secuencia en que se deben llevar a cabo para implementar una aplicación que use este conjunto de ideas que se presentan.

9.2 Contribuciones

En base a los resultados obtenidos, podemos considerar que las principales contribuciones de la presente tesis son las siguientes:

1.- La herramienta CAP-AgentTool que facilita el proceso de desarrollo de SMA compatibles con las especificaciones de FIPA sobre entornos de programación para Windows.

2.- Un modelo y una plantilla de agentes para programar agentes colaborativos sobre la plataforma CAP en diferentes lenguajes de programación que soporten los contenedores ActiveX.

3.- Implementación de un subconjunto del lenguaje de comunicación de agentes FIPA-ACL. Los agentes pueden utilizar la semántica y sintaxis establecida por FIPA para la comunicación de alto nivel, necesaria para la interoperabilidad en sistemas de agentes. Esto es posible a través de los objetos de contenido dados para 12 de los 22 *performatives* de la biblioteca de actos comunicativos de FIPA.

4.- Implementación de un lenguaje de contenido basado en la especificación del lenguaje de contenido FIPA-RDF0 para expresar los diferentes tipos de contenido que se comunican entre los agentes. Esta implementación está basada en los elementos proposición y acción de RDF0. Por medio de este lenguaje de contenido es posible que los agentes puedan interactuar más autónomamente a través de su lenguaje de comunicación facilitando así la interoperabilidad entre agentes heterogéneos.

5.- Propuesta del acto comunicativo *success* para expresar la acción en que un agente informa a otro el resultado de haber ejecutado una acción satisfactoriamente. Este acto es necesario en ambientes colaborativos y distribuidos en donde las aplicaciones requieren conocer los resultados que otros agentes les pueden facilitar a través de la ejecución de acciones remotamente y en donde la eficiencia en el intercambio de información es muy importante.

9.3 Conclusiones

En el presente trabajo se ha desarrollado una herramienta de software para el diseño y creación de agentes a través de la tecnología de controles ActiveX. Su uso facilita el proceso de desarrollo de SMA compatibles con FIPA sobre entornos de programación para Windows. De esta manera se habilita a los programadores en los diversos lenguajes de programación no Java más comunes en Windows, a utilizar la programación basada en

agentes en sus aplicaciones. Los lenguajes de programación que soporten contenedores para controles ActiveX podrán beneficiarse de la tecnología de agentes.

Al estar basado en estándares y especificaciones internacionales se pueden construir sistemas de agentes colaborativos en ambientes abiertos, distribuidos y heterogéneos con un fuerte soporte a la interoperabilidad. Tomando en cuenta estas características, este trabajo nos permite tener una base más sólida en el campo de la programación de agentes y sistemas multiagente.

Otro aspecto importante de señalar es que se ha avanzado en el desarrollo de la infraestructura de programación y despliegue de agentes basada en componentes, iniciada con la plataforma CAP. En el proceso de desarrollo del presente trabajo con la primera versión de la plataforma CAP, se detectaron las limitaciones de la misma que motivaron el análisis y diseño de su segunda versión que contempla la solución de problemas mencionados. En ese sentido, significa que con esta infraestructura se amplían las posibilidades de desarrollo de diferentes aplicaciones de agentes dentro del Laboratorio de Agentes del CIC en el ámbito del desarrollo experimental con herramientas propias.

9.4 Limitaciones

1.- El tamaño de los agentes puede ser grande cuando el número de acciones y su tamaño de implementación es grande. Esto es debido a que los agentes heredan el código fuente desde la plantilla de agente básico. Un agente sencillo puede tener un tamaño desde los 150 Kb. Esto puede ocasionar que este tipo de agentes no sea muy práctico (por lo menos hasta la versión actual) al instalar el control ActiveX que los contiene. Esto también puede afectar el uso de agentes en páginas html dónde la carga y descarga de los controles debe hacerse a través del web.

2.- Hasta este momento los agentes que se crean con la herramienta no tienen añadida la funcionalidad de agentes móviles. Esto limita a las aplicaciones que los utilizan ya que no pueden obtener los beneficios de la movilidad de agentes.

3.- Debido a que los agentes utilizan la tecnología de los controles ActiveX esto hace difícil tomar ventaja de algunas de las características de la programación orientada a objetos, sobre todo la herencia de clases y el polimorfismo. El diseño de la arquitectura de agentes está basado, por eso mismo, en una composición de objetos y herencia de código fuente.

4.- La implementación del lenguaje de contenido FIPA-RDF no está completa. Las referencias a objetos del dominio de las aplicaciones no están consideradas en esta versión. Esto hace que varios de los *performatives* del lenguaje de comunicación de agentes que requieren una referencia a objetos en su contenido, no estén implementados.

9.5 Publicación de los resultados de la tesis

1. Matias Alvarado, Leonid Cheremetov, Ernesto Germán, Erick Alva, Logic of Interaction for Multiagent System, C. Coello et. Al., (Eds.) MICAI 2002: Advances

in Artificial Intelligence, *Lecture Notes in Artificial Intelligence (Proc. of the MICAI'02)*, Springer Verlag, 2313: 387-400, 2002.
2. Leonid Sheremetov, Miguel Contreras, Ernesto Germán Soto, Manuel Chi, Matias Alvadaro. Towards the Enterprise Information Infraestructure Based on Components and Agents. ICEIS'02. (En revisión).
3. Ernesto Germán, Leandro Balladares, Miguel Contreras. Propuesta de un Ambiente Virtual Colaborativo MultiAgente Basado en Internet para la Construcción y Simulación de Circuitos Eléctricos con Fines de Aprendizaje y Entrenamiento. ICPI'02.
4. Metodología de desarrollo de sistemas de agentes sobre una plataforma compatible con FIPA (reporte técnico del CIC, en proceso).
5. Implementación de protocolos de interacción en CAP-AgentTool: Fipa-request, Fipa-query y Fipa-ContractNet (reporte técnico del CIC, en revisión).

9.6 Trabajos futuros

1.-Desarrollar la implementación de FIPA-RDF completa considerando las especificaciones de los sublenguajes FIPA-RDF1 para sistemas basados en reglas y FIPA-RDF2 para la implementación de agentes lógicos.

2.- Terminar de implementar los objetos de contenido de los actos comunicativos que faltan, para que puedan ser utilizados de acuerdo con la semántica y sintaxis de la especificación dada por FIPA-ACL.

3.- Desarrollar e implementar un modelo de operación de agentes utilizando la arquitectura deliberativa, que permita la creación de agentes inteligentes basados en especificaciones de creencias, metas, planes y motivaciones.

4.- Implementar en los agentes un mecanismo eficiente para la manipulación del conocimiento. Podría desarrollarse una implementación de mecanismos de revisión de creencias y de razonamiento orientado a aplicaciones que requieran un manejo más amplio de lógicas y en donde el aprendizaje colaborativo pudiera ser muy importante.

5.- Implementar una clase de control de agente de ontologías utilizando las herramientas de creación y programación de agentes presentadas, con la finalidad de aprovechar sus servicios en aplicaciones de SMA distribuidas y con agentes heterogéneos. Esto puede llevar a la creación de agentes más autónomos e inteligentes.

6.- Experimentar con el desarrollo de aplicaciones distribuidas, con agentes de diferentes clases y arquitecturas y en entornos de desarrollo diversos, en busca de lograr resultados que permitan avanzar en el campo de la interoperabilidad y escalabilidad de SMA.

7.- Formalizar una metodología de ingeniería de software para el desarrollo de sistemas de agentes sobre plataformas compatibles con FIPA.

8.- Estudiar la tecnología de .NET de Microsoft para evaluar las posibilidades que ofrece desde la perspectiva de las ideas de agentes que se han desarrollado en este trabajo. Un punto de partida necesario para esto sería que la plataforma de agentes migrara a esta tecnología de software. Sin duda que el uso de esta tecnología podría beneficiar mucho al desarrollo de algunos conceptos y técnicas de programación basada en agentes, sobre todo, en ambientes distribuidos y en el web y que pueden ayudar a resolver algunas de las limitaciones de la plataforma, como por ejemplo, la descentralización de los servicios, soporte a tolerancia a fallas y facilidades en el desarrollo y despliegue de aplicaciones de agentes en Internet. Este trabajo ya se ha comenzado a desarrollar y se encuentra en las fases de análisis y diseño. De igual forma, la herramienta CAP-AgentTool debe extender su funcionalidad para ayudar en la creación de clases de agentes basados en la tecnología .NET.

APÉNDICES

Apéndice A: Diseño de CAP-AgentTool

Este apéndice comprende una serie de diagramas de UML desarrollados como parte de la especificación de diseño de los elementos principales que se desarrollaron como parte de esta tesis. Incluye tres secciones: En la sección 1 se muestra el diseño de la herramienta CAP-AgentTool. En la sección 2 del apéndice se presentan los diagramas que especifican el diseño de algunas de las clases para manejo de contenido en la comunicación de los agentes. Por último, la sección 3 la conforman los modelos y diagramas que ayudan a entender mejor el prototipo de SMA que se presentó en el capítulo 8. En el CD que acompaña a la tesis, este apéndice se encuentra en la carpeta “ApéndiceA-Diseño”, en el archivo “ApéndiceA-Diseño.doc”.

Apéndice B: Implementación de los protocolos de interacción *request, query* y *Contract-net*

Las conversaciones que se llevan a cabo entre los agentes caen en determinados patrones regularmente. En tales casos, se espera que ocurra cierta secuencia de mensajes y en algún momento de la conversación, otros mensajes se espera que sigan. Estos patrones típicos de intercambio de mensajes son llamados protocolos de interacción (IP).

Con el uso de estos IPs no se intenta cubrir cada tipo de interacción deseable ya que no resuelven un buen número de temas del mundo real comunes en la interacción entre agentes como por ejemplo manejo de excepciones, llegada de mensajes fuera de la secuencia, mensajes perdidos, tiempos de espera, cancelación, etc. Más bien, los protocolos deben ser vistos como patrones de interacción que deben ser usados de acuerdo con el contexto de las aplicaciones.

Por su naturaleza misma, los agentes pueden participar en múltiples diálogos, quizá con diferentes agentes simultáneamente. El término conversación es usado para denotar alguna instancia particular de tales diálogos. En particular, en este material se trata el problema de la interacción bajo el control de un protocolo individual, y se refiere a una conversación particular.

Los agentes que se pueden crear con las herramientas que aquí se presentan pueden incorporar en tiempo de su diseño tres protocolos de interacción. En este documento se presenta la implementación de estos tres protocolos de interacción basados en la especificación de protocolos de FIPA: *FIPA-Request*, *FIPA-Query* y *FIPA-ContractNet*. Además se muestra la forma en que los agentes incorporan esta funcionalidad y el papel que juega el programador del agente al usar cada protocolo.

Además de seguir la secuencia de mensajes predefinida en cada protocolo, la implementación que se propone en este trabajo considera el uso del lenguaje de comunicación de agentes FIPA-ACL y la sintaxis y semántica de su contenido están basadas en la representación de objetos de contenido definidos en el lenguaje de contenido FIPA-RDF0, propuesto y desarrollado como parte de la tesis.

Después de explicar el funcionamiento de la implementación se muestra la forma de utilizar los protocolos en pequeños ejemplos que tienen la finalidad de indicar las partes del protocolo que deben ser implementadas de acuerdo a la aplicación particular que los utiliza.

En el CD de apéndices de la tesis se encuentra la carpeta “ApéndiceB-Protocolos de interacción”. En esta carpeta está el archivo “ApéndiceB-Protocolos de interacción.doc” en el cual se explican los detalles de la implementación propuesta para los protocolos mencionados.

Apéndice C: Código fuente de agentes y programas de ejemplo de la implementación de los protocolos.

El objetivo de este apéndice es presentar el código fuente de los agentes que tienen implementado los protocolos de interacción considerados en la tesis, junto con los programas de ejemplo que utilizan dichos agentes. Con esto se trata que el lector pueda revisar y entender mejor los aspectos y detalles de la implementación y la forma de utilizar los protocolos.

En el CD de apéndices existe una carpeta llamada "ApéndiceC-Código Fuente de los Protocolos de Interacción" y en su interior tiene el siguiente contenido:

Carpeta	Contenido	Descripción
\AgenteCN	Protocolo *FIPA-ContractNet*	Agente que implementa la funcionalidad del protocolo de interacción *FIPA-ContractNet*.
\cn	Programa de prueba	Programa de prueba que sirve de ejemplo para mostrar la funcionalidad del protocolo *ContractNet*.
\AgReq	Protocolo *Request* y Protocolo *Query*	Agente que implementa la funcionalidad de los protocolos de interacción *FIPA-Request* y *FIPA-Query*.
\query	Programa de prueba	Programa de prueba que sirve de ejemplo para mostrar la funcionalidad del protocolo *Query*.
\request	Programa de prueba	Programa de prueba que sirve de ejemplo para mostrar la funcionalidad del protocolo *Request*.

Apéndice D: Código fuente de CAP-AgentTool

El código fuente de la herramienta CAP-AgentTool para crear agentes está incluido en el apéndice D. En el CD de Apéndices se encuentra la carpeta “ApéndiceD-Codigo Fuente de CAP-AgentTool”. Su contenido tiene los siguientes elementos:

Elemento	Descripción
Carpeta \AgentTool	Carpeta que contiene todo lo necesario para la implementación de la herramienta CAP-AgentTool
\AgentTool\AgentTool.vbp	Código fuente del proyecto de la herramienta. Es un proyecto en Visual Basic 6.0
\AgentTool\Ayuda.hpj	Código fuente del programa de ayuda de la herramienta. Este archivo de ayuda se genera con la herramienta *Help Magician* 3.0 para Windows.
\AgentTool\Clases	Grupo de clases utilitarias para manipular el conocimiento de los agentes y objetos de contenido para expresar el campo *content* de los mensajes ACL en la comunicación de agentes.

Apéndice E: Instalación de CAP-AgentTool

La herramienta CAP-AgentTool sirve para crear clases de agentes como controles ActiveX. Cada vez que se crea una clase de agente se crea un nuevo proyecto de tipo control ActiveX con código fuente en el lenguaje Visual Basic 6.0.

Los agentes que se pueden instanciar a partir de las clases de agentes utilizan la funcionalidad básica descrita en la plantilla de agente básico. Algunas de las funciones que son comunes a todos los agentes se refieren a varios aspectos como por ejemplo la comunicación con los componentes de la plataforma CAP, el proceso de inicialización de los agentes, el manejo de su base de conocimiento para hechos y acciones, principalmente, y la manipulación de mensajes ACL.

Además, los agentes tienen la posibilidad de utilizar un conjunto de objetos de contenido para expresar el contenido de los mensajes ACL que manejan. En este sentido, los objetos de contenido utilizan el esquema de codificación basado en XML y RDF para representar los objetos que se requieren en cada acto comunicativo implementado. Por lo tanto, se requieren algunos esquemas de validación de los objetos de contenido en XML a través de Definiciones de Tipos de Documentos (DTD).

En este manual de instalación se muestra la forma en que se debe instalar y configurar tanto la herramienta CAP-AgentTool como algunos aspectos del sistema en general, para que los agentes puedan utilizar los objetos de contenido.

En el CD de Apéndices está la carpeta “ApéndiceE-Instalación de CAP-AgentTool”. Ahí se encuentra el archivo “ApéndiceE-Instalación de CAP-AgentTool.doc” en donde se pueden revisar los detalles de la instalación.

Apéndice F: Código fuente del prototipo y agentes utilizados

En el capítulo 8 de la tesis se habla acerca de un prototipo de SMA para explicar el proceso completo de programación de aplicaciones al utilizar las herramientas que implementan las ideas centrales del presente trabajo.

El prototipo lo forman 4 clases de controles de agente diferentes y dos aplicaciones independientes que incluyen los controles de agente. Todos los elementos del prototipo están programados en Visual Basic 6.0.

En la siguiente tabla se puede ver el contenido y descripción de los programas que forman parte del prototipo. En el CD de Apéndices aparece una carpeta llamada “ApéndiceF-Codigo Fuente del Prototipo” que contiene lo siguiente:

Elemento	Contenido
\AgDominio	Agente de Dominio
\Coalición	Agente de Coalición
\Integrador	Agente Integrador
\Refaccionaria	Agente proveedor
\test.vbp	Programa principal del prototipo de SMA que contiene instancias de los agentes del dominio, integrador y de coalición.
\testProveedor.vbp	Programa principal que contiene instancias del agente proveedor. Este agente colabora con los agentes de coalición del programa test.vbp.

Apéndice G: Ejemplo de la Documentación del agente creado con CAP-AgentTool

'Control de agente creado con la herramienta CAP-AgentTool
'Laboratorio de Agentes CIC-IPN, México D.F.

Nombre de la clase de Agente: AgConReporte

Datos Generales:

Ubicación de la clase de Agente: C:\maestria\cuartosemestre\pruebas\agConReporte
Dueño del agente: Const m_def_owner = "Agentes/Agente2000"
Plataforma CAP: Const m_def_plataforma = "Agente2000"

Grupo de acciones públicas definidas en el agente:

  Public Function Dividir (args As Collection, res As Collection) As Boolean
    Argumentos de la función:
      Dividendo
      Divisor

    Resultados de la función:
      Cociente

  Public Function Sumar (args As Collection, res As Collection) As Boolean
    Argumentos de la función:
      Sumando
      Sumando

    Resultados de la función:
      Suma

Protocolos incluidos en el agente:
  Protocolo FIPA-Request
  Protocolo FIPA-Query-If
  Protocolo FIPA-ContractNet

Configuración de comunicación:
  Lenguajes de comunicación de agentes:
    fipa-acl

  Ontologías:
    pruebas

  Lenguajes de contenido:
    fipa-rdf0

Lista de Hechos iniciales:
predicado: h1 Sujeto: s1 objeto: o1 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas
predicado: h2 Sujeto: s2 objeto: o2 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas
predicado: h3 Sujeto: s3 objeto: o3 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas
predicado: h4 Sujeto: s4 objeto: o4 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas
predicado: h5 Sujeto: s5 objeto: o5 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas
predicado: h6 Sujeto: s6 objeto: o6 belief: Verdadero owner: Agentes/Agente2000 ontología: pruebas

BIBLIOGRAFÍA Y REFERENCIAS

(Alvarado et al., 2002) Alvarado, M., Cheremetov, L., Germán, E. & Alva, E. Logic of Interaction for Multiagent System, C. Coello et. Al., (Eds.) MICAI 2002: Advances in Artificial Intelligence, *Lecture Notes in Artificial Intelligence (Proc. of the MICAI'02),* Springer Verlag, 2313: 387-400, 2002.

(Alvarado, M. & Sheremetov, L., 2001). Alvarado, M. & Cheremetov, L. Interaction Modal Logic for multi-agent systems based on BDI architecture, *In Proc of the 3er. Encuentro Internacional de Ciencias de la Computación (ENC' 01),* Aguascalientes, México, del 15 al 19 de Septiembre del 2001, pp.13-22.

(Appleman, 2000). Appleman D. 2000. Desarrollo de componentes COM/ActiveX con Visual Basic 6, Editorial Prentice Hall, Madrid España.

(Bellifemine, Caire, Trucco & Rimaza, 2000). Bellifemine, F., Caire, G., Trucco, T. & Rimaza, G., 2000. Jade Programmer's Guide, CSELT.

(Bellifemine, Poggi & Rimaza, 1999). Bellifemine, F., Poggi, A., Rimaza, G. 1999. JADE - A FIPA-compliant agent framework Proceedings of PAAM'99, London, pag.97-108.

(Cohen & Levesque, 1990). Cohen, P. and Levesque, H. "Intention is Choice with Commitment", Artificial Intelligence 42, pages 213261, (1990).

(Collis, Ndumu, Nwana & Lee, 1998). Collis J. C., Ndumu D. T., Nwana H. S. and Lee L. C., 1998. The ZEUS agent building tool-kit. British Telecomm Technology Journal Volume 16 #3.

(Collis, 1999). Collis J., 1999. *The Zeus Technical Manual*, British Telecommunications, BT Labs.

(Contreras, 2001). Contreras Montoya, M, 2001. CAP. Desarrollo de una infraestructura MultiAgentes para organizaciones virtuales.. CIC-IPN, México D.F.

(Delphi, 1999). Ayuda en línea de Borland Delphi Usando Controles ActiveX.

(DCOM, 1998). DCOM Architecture, Microsoft Windows, Withe Paper, 1998, Microsoft Corporation.

(Ferber, 1999). Ferber J, 1999. *Multi-Agent Systems*, Editorial Addison-Wesley, E.U.A, pag. 8-15.

(Finin, Labrou & Mayfield, 1997). Finin, T., Labrou Ya., y Mayfield J. 1997. KQML as an agent communication language en J. Bradshaw, ed., Software Agents, AAAI/MIT Press.

(FIPA-1, 2000). Especificaciones de FIPA. http://www.fipa.org

(FIPA-2, 2001). FIPA Agent Management Specification, http://www.fipa.org/specs/fipa00023/. 15/08/2001

(FIPA-3, 2001). FIPA ACL Message Structure Specification, http://www.fipa.org/specs/fipa00061/ 15/08/2001

(FIPA-4, 2001). FIPA Communicative Act Library Specification. http://www.fipa.org/specs/fipa00037/. 15/08/2001

(FIPA-5, 2001). FIPA Content Language Library Specification, http://www.fipa.org/specs/fipa00007/. 15/08/2001

(FIPA-6, 2001). FIPA RDF Content Language Specification, http://www.fipa.org/specs/fipa00011/. 15/08/2001

(FIPA-7, 2001). FIPA Protocolos de interacción. http://www.fipa.org/specs/fipa00025/. 15/08/2001

(FIPA-8, 2001). FIPA Request Interaction Protocol Specification. . http://www.fipa.org/specs/fipa00026/. 15/08/2001

(FIPA-9, 2001). FIPA Query Interaction Protocol Specification http://www.fipa.org/specs/fipa00027/. 15/08/2001

(FIPA-10, 2001). FIPA Contract-Net Interaction Protocol Specification http://www.fipa.org/specs/fipa00029/. 15/08/2001

(FIPA-OS, 2001). Nortel Networks. http://fipa-os.sourceforge.net/
Documentación de JADE CSELT. http://sharon.cselt.it/projects/jade/

(Fonseca, Griss & Letsinger, 2001). Fonseca S. P., Griss M. L. & Letsinger R., 2001. Evaluation of the Zeus MAS Framework. Software Technology Laboratory. HP Laboratories Palo Alto HPL-2001-154

(Gauvin, Marchal & Saldaña, 1997) Gauvin, D., Marchal, H., Saldaña, C., 1997. *LALO: un environnement de programmation ouvert pour des syst mes multi-agents*, acte des 5.me joun.es francophones JFIADSMA Ô97.

(Genesereth & Ketchpel, 1994). Genesereth, M. R. & Ketchpel, S. P.,1994. Software Agents, Communications of the ACM 37 (7), 48-53.

(Hindriks, de Boer, van der Hoek & Meyer, 1999). Hindriks K.V., de Boer F.S., van der Hoek W. and Meyer J.-J.Ch., 1999. Semantics of communicating agents based on deduction and abduction. IJCAI'99 Workshop on Agent Communication Languages.

(Jamison, 1998). Jamison, W., 1998. Dynamic agent collaboration for open sys-terra using concurrent interpretation of role-based coordination protocols, Thesis, Syracuse University.

(Jamison & Lea, 1999). Jamison, W. C. & Lea, D., 1999. Scripting Distributed Agents. ACM SIGAPP Applied Computing Review, Volume 7, Issue 1. ACM Press, pag. 18-22

(JAS, 2002). Java agent Services, JAS http://www.java-agent.org/, http://www.java-agent.org/JAS_Intro.htm , JAS Specification PR 1.13 05032002.pdf

(MSDN, 1999). Randell, B. 1999. A Beginner's Guide to the XML DOM. Microsoft MSDN.

(Mueller, 1997). Mueller, J. P., 1997. ActiveX from the ground up, McGraw Hill, Berkeley California, E.U.A, pag. 6,7,9,13,14,15,16,17

(Nwana & Ndumu, 1997). Nwana, H.S., Ndumu, D.T., 1997. *Software agents and soft computing. Towards enhancing machine intelligence. Concepts and applications*, ISBN 3-540-62560-7, Springer-Verlag, Berlin, Germany, 1997, pp 3-26.

(Nwana & Wooldridge, 1996). Nwana H. S. and Wooldridge M., 1996. Software agent technologies. British Telecomm Technology Journal Volume 14 #4.

(Nwana, 1996). Nwana, H. S., 1996. The Potential Benefits of Software Agent Technology to BT, *Internal Technical Report*, Project NOMADS, Intelligent Systems Research, AA&T, BT Labs, UK.

(Petroutsos, 1999). Petroutsos, E., 1999. ActiveX Development with Visual Basic 6, Editorial Ventana, E.U.A pag. 230,231.

(Poslad, Buckle & Hadingham, 2001). Poslad S., Buckle P.& Hadingham R., 2001. Open Source, Standards and Scaleable Agencies.

(Poslad, Buckle, Hadingham, 2000). Poslad, P. Buckle & R. Hadingham, 2000. The FIPA-OS agent platform: Open Source for Open Standards published at PAAM2000, Manchester, UK.

Programación en VB. Ayuda en línea de Visual Studio 6.0 y Manual de referencia del lenguaje Visual Basic 6.0

(RDF, 1999). RDF Specification Development. http://www.w3.org/RDF/

(RETSINA,2002). RETSINA AFC Programmers Guide. Laboratorio de Agentes del Instituto de Robótica de la Universidad de Carnegie Mellon. AFC Developer Martin van Velsen. http://www-2.cs.cmu.edu/~softagents/afc/Manual_rev_15.htm

(Romero, Sheremetov, 2001). Romero Cortés, J. & Sheremetov L., 2001. Model of Negotiation in Multi Agent Systems for Fuzzy Coalition Formation. In Proc of the Second

International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS'01, Cracow, Poland, pp. 237-246.

(Searle, 1969). Searle, J. 1969. *Speech Acts: An essay in the Philosophy of Language*. Cambridge University Press, Cambridge UK.

(Sheremetov & Contreras,2001). Sheremetov, L. & Contreras, M., 2001. Component Agent Platform. In Proc of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS'01, Cracow, Poland**,** pp. 395-402.

(Sheremetov & Contreras,2001). Sheremetov, L. & Contreras, M., 2001 Development of Agent Platform based on Distributed Object Technology. In Proc of the 3er. Encuentro Internacional de Ciencias de la Computación (ENC' 01), Aguascalientes, México, pp.803-812.

(Shoham, 1990) Shoham, Y. "Agent-Oriented Programming", Technical report No. STAN-CS-90-1335, Computer Science Department, Stanford University, 1990.

(Sing, 2000). Sing, Li, 2000. Profesional Jini, Wrox Press Ltd, Canadá, pag. 705-761

(Smith, 1996). Smith, R. 1996, Software Agent Technology, Proceedings of The First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology, London, UK, 557-571.

(Smith, 1980). Smith, R. G. ,1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers* C29 (12).

(Subrahmanian, Bonatti, Dix, Eiter & Ozcan, 2000). V. S. Subrahmanian, Piero, Bonatti, Jurgen Dix, Thomas Eiter, Fatma Ozcan. *Heterogeneous Agent Systems*, MIT Press.

(Sycara, 1998). Sycara, K., 1998. MultiAgent Systems. AI magazine Volume 19, No.2 Intelligent Agents.

(Sycara, Decker, Pannu, Williamson & Zeng, 1996). Sycara K., Decker, K., Pannu A., Williamson M., and Zeng D. Distributed Intelligent Agents." IEEE Expert: Intelligent Systems and Their Applications. Vol. 11, No. 6, Dec., 1996, pp. 36-46.

(Tambe, 1997). M. Tambe. Towards Flexible Teamwork. Journal of AI Research, 7(1997), 83-124.

(XML, 1999). WWWC Architecture Domain, eXtensible Markup Language. http://www.w3.org/XML/

GLOSARIO

ACC	Agent Communication Channel. Componente que funciona como Canal de Comunicación entre Agentes.
ACL	Agent Communication Language. Lenguaje de Comunicación de Agentes, en especial, el propuesto por la FIPA.
Activación COM	Es el proceso de crear un objeto real en la memoria, a partir de una llamada de un cliente COM sin importar el lugar en que se encuentra la clase concreta.
ActiveX	Es un conjunto de tecnologías de Microsoft que lo habilitan para el Internet.
AFC	Agent Foundation Classes. En la arquitectura RETSINA, API para la programación de agentes en Visual C++.
AMS	Agent Management System. Sistema Administrador de Agentes. Se encarga, entre otras cosas, de dar de alta y de baja a los agentes de una plataforma.
API	Application Program Interface. Interfaz de Programas de Aplicación.
BDI	Belief-Desire-Intention. Arquitectura de comportamiento de agentes basada en actitudes mentales como creencias, deseos e intenciones.
CAP	Component Agent Platform. Plataforma de Agentes Componentes.
CCL	Constraint Choice Language. Lenguaje de contenido especificado por FIPA para manejo de restricciones.
Cliente COM	Es el código que llama las interfaces para obtener los servicios requeridos desde un servidor COM. Por ejemplo un control ActiveX al comunicarse con una plataforma CAP.
COM	Component Object Model. Modelo Componente Objeto.
CORBA.	Common Object Request Broker Architecture.
DAI	Distributed Artificial Intelligence. Inteligencia Artificial Distribuida.
DDE	Dynamic Data Exchange. Intercambio de datos dinámico.
DF	Directory Facilitator. Agente Facilitador de Directorios.
DLL	Dynamic Link Library. Biblioteca de Ligado Dinámico utilizada en

	ambientes Windows.
DOM	Document Object Model. Modelo de Objeto Documento.
DTD	Document Type Definition. Definición de tipo de documento.
EXE	Executable. Archivo ejecutable en el ambiente Windows.
FIPA	Foundation for Intelligent Physical Agents. Fundación para los Agentes Inteligentes. Tiene como una de sus funciones proponer estándares de comunicación entre agentes.
FIPA-RDF0	Lenguaje de contenido para agentes especificado por FIPA basado en RDF.
GUID	Global Unique Identifier. Identificador Global Único.
IDL	Interface Definition Language. Lenguaje de Definición de Interfaces.
In-Process	Proceso de activación COM a través de un servidor COM que se encuentra en el mismo espacio de proceso que el cliente COM.
Interfaz COM	Es la forma en la que un objeto expone sus servicios externamente a los clientes.
IPMT	Internal Platform Message Transport. Transporte de mensajes interno de la plataforma en la herramienta FIPA-OS.
IIOP	Internet Inter-ORB Protocol.
JESS	Java Expert System Shell. Shell de Sistema Experto de Java que acompaña a FIPA-OS y JADE para el desarrollo de agentes basados en reglas.
KIF	Knowledge Interchange Format. Formato de intercambio de conocimiento.
KQML	Knowledge Query and Manipulation Language. Lenguaje ampliamente utilizado para comunicar agentes.
Lenguaje de Contenido	Ayuda a tener mayor eficiencia, flexibilidad e interoperabilidad en la comunicación entre agentes heterogéneos.
MFC	Microsoft Foundation Classes. Biblioteca desarrollada por Microsoft para facilitar el desarrollo de aplicaciones en ambiente Windows.
MIDL	Microsoft Interface Definition Language. Lenguaje de definición de interfaces de Microsoft.
MTS	Message Transport Service. Servicio de transporte de mensajes.

OCX	OLE Control eXtension. Extensión de control OLE.
OLE	Object Linking and Embedding. Es un estándar de documentos compuestos desarrollado por Microsoft que permite crear objetos con una aplicación y enlazarlos o embutirlos en otra aplicación.
Ontología	Es una descripción de los conceptos y relaciones que pueden existir para un dominio de agentes o una comunidad de agentes.
ORPC	Object Remote Procedure Call. Llamada a Procedimiento Remoto utilizando objetos componentes en el modelo COM.
OutOf-Process	Proceso de activación COM a través de un servidor COM que se encuentra en otro proceso distinto que el cliente COM, tal vez en otra computadora.
PA	Plataforma de Agentes, (Agent Platform).
Performative	Acto comunicativo
PROGID	Programmatic Identifier. Identificador asociado a un componente.
Proxy	Es un objeto que se crea del lado del cliente COM y representa al servidor COM durante la comunicación entre procesos en DCOM.
RDF	Resource Description Framework. Marco de descripción de recursos.
RMI	Remote Method Invocation. Invocación de Métodos Remotos.
RPC	Remote Procedure Call. Llamada a procedimiento remoto.
SCM	Service Control Manager. Administrador de Servicios de Control.
SCN	Supply Chain Network. Red de cadenas de suministros.
Servidor COM	Es un módulo, dll, exe u ocx, que contiene el código para un objeto COM.
SL	Semantic Language. Lenguaje semántico propuesto por la FIPA para servir como lenguaje de contenido del lenguaje ACL.
SL0	Semantic Language Subset 0. Subconjunto mínimo del lenguaje SL.
SMA	Sistemas Multiagente.
Software legado	Software convencional que se integra en sistemas de agentes.

Stub	Es un objeto que se crea del lado del servidor COM y representa al cliente COM en la comunicación entre procesos.
VC++	Visual C++. Compilador, conjunto de bibliotecas, herramientas y ambiente de desarrollo para construir aplicaciones con C++ fabricado por Microsoft.
XML	eXtensible Markup Language. Lenguaje de marcado extensible.
XML-DOM	Objeto COM que se utiliza para la manipulación de documentos XML en diversas aplicaciones.