# Agencia: Una puerta hacia la convergencia de la Inteligencia Artificial

**Juan M. Corchado**

***Departamento de Informática y Automática***

***Facultad de Ciencias***

***Universidad de Salamanca***

**http://gsii.usal.es**

**http://master-ecom.usal.es**

# Temario

# AGENTES Y SISTEMAS MULTIAGENTE

El término agente es cada vez más conocido y se emplea en campos tan diversos como Internet, los sistemas distribuidos, la inteligencia artificial o la interacción persona-computador. Hoy en día se habla de agentes inteligentes, agentes móviles, agentes software, agentes autónomos, sistemas multiagente. Parece que ha llegado "la era de los agentes", es importante por tanto intentar clarificar algunos aspectos básicos que permitan entender mejor este nuevo modelo y permitan responder a preguntas tales como ¿qué es un agente?, ¿qué lo caracteriza?, ¿cómo se clasifican?, ¿cuáles son sus componentes?, ¿cómo se comunican?, ¿cómo se estructuran?, ¿cómo se analizan y diseñan?, ¿cómo se implementan?, ¿qué lenguajes o herramientas hay disponibles?, etc. A lo largo de este documento se intentarán clarificar algunas de estas cuestiones, aunque como quedará claro en la comunidad científica existe muy poco consenso a la hora de definir la mayoría de los aspectos relacionados con esta tecnología, empezando por la definición de agente.

Esta diversidad de opiniones se debe en parte a que este campo ha atraído a científicos procedentes de áreas muy dispares: psicología, sociología, ingeniería del software, inteligencia artificial, etc. y cada uno de los miembros de estas comunidades tiende a ver el problema desde su perspectiva. Por tanto, realizar una definición de agente o agencia es complicado debido a la diversidad de opiniones que existen en la comunidad científica sobre este tema (Franklin *et al.*, 1996).

Tanto la inteligencia artificial distribuida, como en general el desarrollo de sistemas distribuidos, basados en la resolución de problemas complejos mediante la colaboración de un conjunto de elementos individuales, han evolucionado hacia la utilización de agentes y sistemas multiagente. Actualmente constituyen un área de investigación en pleno desarrollo, a la que cada vez se dedican más recursos.

Los agentes autónomos y los sistemas multiagente representan una nueva forma de analizar, diseñar e implementar sistemas software complejos. Los agentes, en la actualidad, se utilizan en mayor o menor medida en una gran variedad de aplicaciones, y de dominios de aplicación, debido a las ventajas sustanciales que ofrecen, entre las que destaca la naturalidad del modelo para conceptualizar e implementar diferentes tipos de software.

Dentro de la terminología de este campo es importante clarificar la diferencia entre un sistema basado en agentes y un sistema multiagente (Jennings *et al.*, 1998). Un sistema basado en agentes, es aquel que utiliza el concepto de agente como mecanismo de abstracción, pero aunque sea modelado en términos de agentes podría ser implementado sin ninguna estructura de software correspondiente a éstos, lo que puede parecer tan inusual como contraproducente. Un sistema basado en agentes, puede contener uno o más agentes. Por otro lado un sistema multiagente es aquel que se diseña e implementa pensando en que estará compuesto por varios agentes que interactuarán entre sí, de forma que juntos permitan alcanzar la funcionalidad deseada. En este caso hay que hacer un mayor esfuerzo de abstracción, identificar mecanismos de aprendizaje, coordinación, negociación, etc. Como se mostrará en este documento.

Los sistemas multiagente son adecuados para solucionar problemas para los que hay múltiples métodos de resolución y/o múltiples entidades capaces de trabajar conjuntamente para solucionarlos. Por ello uno de los aspectos básicos en estos sistemas es la interacción entre los diferentes agentes que los forman, la cual como se verá más adelante, puede estar basada en modelos de cooperación, coordinación o negociación entre agentes.

Este documento pretende ser una introducción, más o menos sesgada, al campo de los agentes y sistemas multiagente. Primero se introducen conceptos básicos de la teoría de agentes y posteriormente se presentan varias arquitecturas de construcción de agentes. Se hace una breve

descripción de los distintos tipos de agentes que existen y se muestran distintos mecanismos que permiten la comunicación entre agentes. Después se hace una revisión de los distintos métodos de negociación existentes entre agentes y de los mecanismos de aprendizaje que pueden utilizar. Por último se hace una comparación entre la programación orientada a agentes y la orientada a objetos. Aunque el proceloso mundo de la agencia es extraordinariamente joven, es igualmente extenso y por tanto aquí solo se incluye un pequeño número de temas que de algún modo u otro son cercanos al autor del documento.

# 1. TEORÍA DE AGENTES

Definir con precisión el concepto de agente es extremadamente complicado debido al gran número de posibilidades que ofrece este campo y a la falta de consenso entre los integrantes de la comunidad científica que trabaja en este tema. Esto ha hecho que incluso la "definición de agente" haya sido uno de los aspectos que han suscitado mayor debate en los últimos tiempos (Franklin *et al.*, 1996).

En general podemos decir que un agente es un sistema informático, situado en algún entorno, dentro del cual actua de forma autónoma y flexible para así cumplir sus objetivos. Un agente recibe entradas de su entorno y a la vez ejecuta acciones para intentar cambiarlo a su gusto (Figura 1.1).

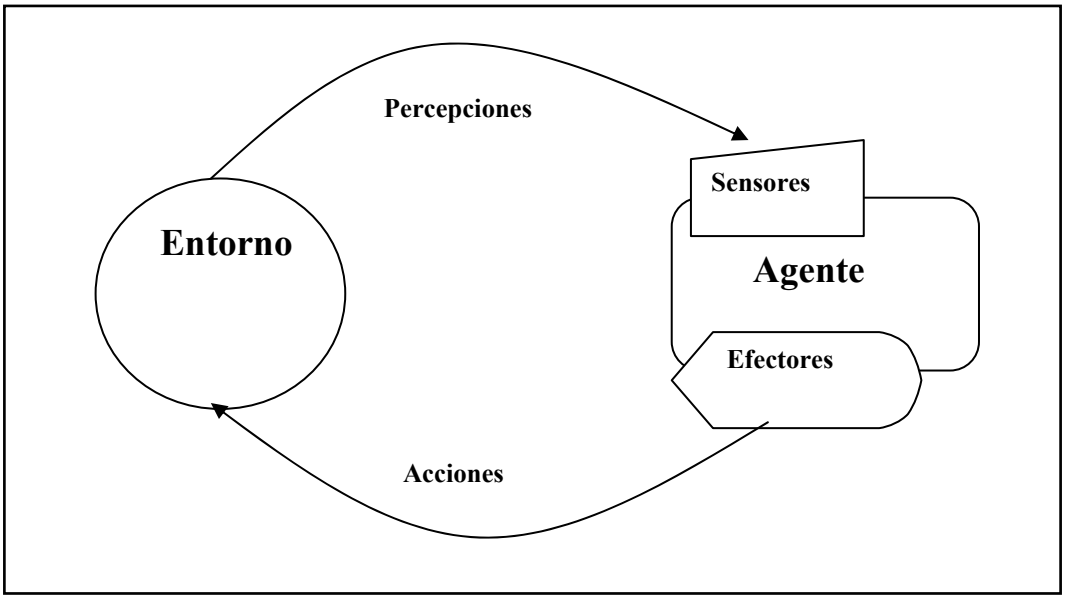


Figura 1.1: Interacción agente/entorno.

Además de la interacción con el medio, un agente se caracteriza por algo más. Desde nuestro punto de vista, y utilizando la definición de Woldridge *et al.* (1995) podemos decir que un agente es todo sistema informático que satisface las siguientes propiedades:

1. ***Autonomía*:** tiene la capacidad de actuar sin intervención humana directa o de otros agentes.
2. ***Sociabilidad*:** capacidad de interaccionar con otros agentes, utilizando como medio algún lenguaje de comunicación entre agentes.
3. ***Reactividad*:** un agente está inmerso en un determinado entorno (hábitat), del que percibe estímulos y ante los que debe reaccionar en un tiempo preestablecido.
4. ***Iniciativa*:** un agente no sólo debe reaccionar a los cambios que se produzcan en su entorno, sino que tiene que tener un carácter emprendedor y tomar la iniciativa para actuar guiado por los objetivos que debe de satisfacer.

Aunque estas propiedades definen las características básicas de un agente, existen numerosos sistemas software que podrían satisfacer algunas de ellas, es el caso por ejemplo de los típicos "demonios" del UNIX, servidores de páginas web, etc. que funcionan de manera autónoma, se comunican con otros procesos (con cualquier equipo que disponga de un navegador) mediante un protocolo de comunicación (httpd), funcionan integrados en un entorno concreto (servidor web conectado mediante una red TCP/IP), reaccionan ante estímulos producidos, como la demanda de una página determinada o la pérdida de un paquete en una transmisión. A estos sistemas les falta, entre otras cosas, cierta capacidad de iniciativa para actuar guiados por los objetivos a satisfacer, y esto es debido a que no disponen de mecanismos inteligentes.

Para eliminar esta ambigüedad y satisfacer las necesidades de los sistemas de agentes, se han definido otra serie de propiedades que se utilizan para caracterizar a los distintos tipos de agentes:

- ***Movilidad*:** habilidad de un agente de trasladarse en una red de comunicación informática.
- ***Veracidad*:** propiedad por la que un agente no comunica información falsa intencionadamente.
- ***Benevolencia*:** un agente no tiene objetivos contradictorios y siempre intenta realizar la tarea que se le solicita.
- ***Racionalidad*:** un agente tiene unos objetivos específicos y siempre intenta llevarlos a cabo.

El carácter de racionalidad de un agente (Russell *et al.*, 1995) depende de cuatro factores:

- De la medida con la que se evalúe el grado de éxito logrado.
- De la secuencia de percepciones, entendiendo por tal todo aquello que hasta ese momento haya percibido el agente.
- Del conocimiento que el agente posea del medio.
- De las acciones que el agente pueda llevar a cabo.

Un agente racional debe emprender todas aquellas acciones que favorezcan obtener el máximo de su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en todo el conocimiento incorporado.

Un agente en este sentido, tiene que ser capaz de aprender de la experiencia (Russell *et al.*, 1995). El diseño de un agente debe basarse en especificar qué tipo de acción deberá emprender como respuesta a una determinada secuencia de percepciones. Si las acciones que lleva a cabo el agente se basan exclusivamente en un conocimiento interno y hace caso omiso de sus percepciones, el agente no dispone de autonomía. La conducta de un agente se debe basar tanto en la experiencia que acumula como en el conocimiento inicial con el que se creó para trabajar en el medio para el que fue concebido. Un sistema será autónomo en la medida en que su forma de actuar está definida por su propia experiencia. Es por tanto razonable dotar a un agente con ciertos conocimientos iniciales y de capacidad para aprender.

Para algunos investigadores el término agente tiene una caracterización más estricta, para que un sistema informático pueda considerarse como un agente tiene que modelarse e implementarse usando aspectos que usualmente se aplican a los humanos. Así, Shoham (1993) define un agente como una entidad cuyo estado está formado por componentes mentales (típicas de los humanos), como creencias, capacidades, elecciones y compromisos. Estados que determinan las acciones que llevan a cabo los agentes y qué están afectados por los mensajes que reciben.

Para que se extienda el uso del modelo de agentes de forma generalizada en el desarrollo de software es necesario disponer de lenguajes y herramientas adecuadas, que permitan tanto la implementación de la estructura y el comportamiento del agente, como la comunicación con otros agentes. Surge así una primera distinción entre los lenguajes de comunicación y los lenguajes de agentes propiamente dichos.

Para desarrollar agentes se pueden utilizar lenguajes de propósito general, entre los que cabe destacar los orientados a objetos, debido al cierto paralelismo entre objetos y agentes, como C++ o Java, o lenguajes específicos de agentes, de los que han aparecido numerosas propuestas en los últimos años, como Agent0, AgentK, Placa, Radl, Lalo, etc.

Los lenguajes de agentes han de permitir definir la estructura, estado y comportamiento de cada agente. Gran parte de estos lenguajes están influenciados por la propuesta de Shoham, de considerar la programación orientada a agentes, como un nuevo paradigma de programación (Shoham, 1993), en donde se representa el estado mental del agente, en base a sus creencias, capacidades, elecciones y compromisos.

El término de agente se comenzó a utilizar por primera vez en el campo de la inteligencia artificial, desde el cual han surgido las principales contribuciones. Aunque la inteligencia artificial (IA) ha proporcionado numerosos éxitos desde hace muchas décadas, el auge y el aumento destacado en la producción científica en el ámbito de la agencia se ha producido en la última década, debido fundamentalmente a la tendencia dentro de la IA a centrarse en abordar los diferentes componentes que dan lugar al comportamiento inteligente por separado.

Los sistemas compuestos de múltiples agentes se desarrollaron inicialmente en el ámbito de la IA distribuida (O'hare *et al.*, 1996), la cual tradicionalmente se ha dividido en dos campos:

- La ***resolución de problemas distribuidos*:** un problema particular puede resolverse por un número de elementos que cooperan y comparten conocimiento sobre el problema y su solución. Las tareas que cada subsistema realiza están prefijadas de antemano, cada agente tiene una conducta fija, y el sistema se centra en el comportamiento global.
- Los ***sistemas multiagente:*** agentes autónomos trabajan juntos para resolver problemas, caracterizados porque cada agente tiene una información o capacidad incompleta para solucionar el problema, no hay un sistema global de control, los datos están descentralizados y la computación es asíncrona. Los agentes deciden dinámicamente que tareas realizan.

Como ya se comentó anteriormente de cara a la utilización generalizada de los sistemas multiagente en el desarrollo de software, se está llevando a cabo un trabajo intenso para disponer de técnicas, métodos y herramientas que garanticen la construcción correcta de tales sistemas. Así mismo, están empezando a aparecer propuestas de metodologías de desarrollo orientadas a agentes, que permiten analizar, diseñar y documentar utilizando un marco común de referencia. En este sentido hay que indicar que existen varias tendencias al respecto, entre las que cabe destacar las que toman como partida metodologías de ingeniería de conocimiento, utilizadas en el campo del desarrollo de sistemas basados en el conocimiento, o las que parten de metodologías orientadas a objetos.

Los sistemas multiagente aun no son una tecnología madura y es necesario solucionar numerosos interrogantes como por ejemplo:

- ¿Cómo formular, describir, descomponer problemas y sintetizar resultados entre un grupo de agentes inteligentes?
- ¿Cómo permitir a los agentes comunicarse e interactuar?
- ¿Qué lenguajes de comunicación y protocolos se pueden usar?
- ¿Qué arquitectura es la más adecuada para construir sistemas multiagente prácticos?
- ¿Qué lenguajes y herramientas de desarrollo se pueden utilizar?
- ¿Cómo construir herramientas para soportar las metodologías de desarrollo?, etc.

Este documento intenta despejar algunas de estas dudas mostrando alguna de las soluciones que se han propuesto.

# 2. ARQUITECTURAS DE CONSTRUCCIÓN DE AGENTES

## 2.1 Introducción

Una arquitectura define los mecanismos que permiten interconectar los componentes tanto software como hardware, que hacen que un agente se comporte como tal. Un hecho evidente, en este sentido, hoy en día es que existen infinidad de propuestas, casi tantas como equipos de investigación centrados en el tema.

Las arquitecturas utilizadas para construir agentes, especifican como se descomponen los agentes en un conjunto de módulos que interactúan entre sí para lograr la funcionalidad requerida. Uno de los aspectos básicos que diferencia una arquitectura de otra es el método de descomposición del trabajo en tareas particulares. En este sentido hay que decir que la planificación es un área muy fuertemente ligada al mundo de la agencia. Este área se centra en el estudio de mecanismos que permitan organizar la ejecución de acciones, y un agente no es más que un sistema que ejecuta acciones en un entorno determinado.

Los sistemas de planificación utilizan modelos de representación del conocimiento y razonamiento de tipo simbólico y su modo de actuación está definido por la necesidad de satisfacer unos objetivos básicos, para lo que elaboran un plan. Estos sistemas tienen la desventaja de que requieren un elevado tiempo de respuesta. Este es un gran inconveniente cuando se utilizan en problemas de tiempo real, ya que los algoritmos de planificación no siempre responden en un tiempo prudencial a las demandas del sistema. Esto es así debido a que los principios básicos de la planificación son indecidibles, lo que hace que no constituyan una opción del todo viable para los agentes. Estas críticas dirigidas fundamentalmente hacia el modelo simbólico utilizado, llevaron a la búsqueda de alternativas que utilizasen otro modelo de representación o razonamiento, como los reactivos o híbridos.

A continuación se presentan tres arquitecturas diferentes que se diferencian en el modelo de razonamiento que utilizan (Wooldridge *et al.*, 1995).

### Deliberativas

Son aquellas arquitecturas que utilizan modelos de representación simbólica del conocimiento. Suelen estar basadas en la teoría clásica de planificación, en las que se parte de un estado inicial de partida, existen un conjunto de planes y un estado objetivo a satisfacer. En estos sistemas parece aceptada la idea de que es necesario dotar a los agentes de un sistema de planificación que se encargue de determinar que pasos se deben de llevar a cabo para conseguir sus objetivos.

Por tanto un *agente deliberativo (*o con una *arquitectura deliberativa)* es aquel que contiene un modelo simbólico del mundo, explícitamente representado, en donde las decisiones se toman utilizando mecanismos de razonamiento lógico basados en la concordancia de patrones y la manipulación simbólica.

Cuando se decide implantar una arquitectura deliberativa hay que buscar, en primer lugar, una descripción simbólica adecuada del problema, e integrarla en el agente, para que este pueda razonar y llevar a cabo las tareas encomendadas en el tiempo preestablecido. Aunque parece una cuestión trivial, debido a la complejidad de los algoritmos de manipulación simbólica, es un aspecto al que hay que prestar mucha atención, especialmente si se tiene en cuenta que los agentes se desenvuelven en dominios reales, en los que tienen que responder a los estímulos en tiempo real.

Por ejemplo, un agente intencional puede implementarse utilizando una arquitectura deliberativa. Los agentes intencionales son sistemas de planificación en los que para definir los

planes se tienen en cuenta sus creencias e intenciones, de forma que el agente puede utilizar sus creencias e intenciones para razonar. Dentro de estas arquitecturas intencionales cabe destacar aquellas que basan su implementación en el modelo BDI (*Belief, Desire, Intention*). Este es uno de los modelos más utilizados hoy en día (Rao *et al.*, 1995) y por ello se estudia en la sección 2.2 con más profundidad.

**Reactivas**

Los numerosos problemas que lleva asociado utilizar una representación simbólica del conocimiento, ha conducido al estudio de modelos más efectivos de representación del conocimiento. Las arquitecturas reactivas, se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo (Brooks, 1991).

Un ejemplo típico de estas arquitecturas es la propuesta de Roodney Brooks, conocida como arquitectura de subsunción (Brooks, 1991). Esta arquitectura se basa en el hecho de que el comportamiento inteligente puede ser generado sin utilizar propuestas del modelo simbólico y en el hecho de que la inteligencia es una propiedad emergente de ciertos sistemas complejos. Las arquitecturas de subsunción manejan jerarquías de tareas que definen un comportamiento. Suelen estar organizados en jerarquías de capas, de menor a mayor nivel de abstracción.

**Híbridas**

Dado que algunos investigadores opinan que para la construcción de agentes no es del todo acertado utilizar una arquitectura totalmente deliberativa, o totalmente reactiva, se han propuesto sistemas híbridos que pretenden combinar aspectos de ambos modelos. Una primera propuesta puede ser construir un agente compuesto de dos subsistemas: uno deliberativo, que utilice un modelo simbólico y que genere planes en el sentido expuesto anteriormente, y otro reactivo centrado en reaccionar a los eventos que tengan lugar en el entorno y que no requiera un mecanismo de razonamiento complejo.

Por su propia naturaleza estas arquitecturas son propicias para una estructuración por capas, que puede ser:

- Vertical, sólo una capa tiene acceso a los sensores y actuadores.
- Horizontal, todas las capas tienen acceso a los sensores y a los actuadores.

De la misma forma que las arquitecturas de subsunción, las capas se organizan jerárquicamente con información sobre el entorno a diferentes niveles de abstracción. La mayoría de las arquitecturas encuentran suficiente tres niveles:

- **Reactivo**: o de más bajo nivel, se toman decisiones acerca de lo que hacer en base a los estímulos recibidos del entorno en tiempo real. Suele estar implementado como arquitecturas de subsunción.
- **Conocimiento**: nivel intermedio, se olvida de los datos que recopila el agente y se centra en el conocimiento que el posee del medio, normalmente con la ayuda de una representación simbólica del medio.
- **Social**: la capa de mas alto nivel, maneja aspectos sociales del entorno, incluyendo tanto información de otros agentes, como deseos, intenciones, etc.

El comportamiento global del agente viene definido por la interacción entre estos niveles. Esta interactuación cambia de una arquitectura a otra. Uno de estos ejemplos es la arquitectura

*TOURINGMACHINES* (Ferguson, 1992), donde cada nivel está continuamente sugeriendo que acción realizar y existe un sistema de control para garantizar el correcto funcionamiento del agente. Otra arquitectura como *INTERRAP* (Müller, 1997) también actúa del mismo modo.

## 2.2 Arquitectura deliberativa BDI

Los humanos a menudo utilizan afirmaciones del tipo: "*Luisa lleva un paraguas porque, cree que va a llover."* Esta forma de estructurar ideas permite a los psicólogos predecir el comportamiento humano y explicarlo mediante actitudes mentales, tales como creencias, deseos, temores,... El filósofo Daniel Dennett (1987) habla de "*sistemas intencionales"* que describen entidades "cuyo comportamiento se puede predecir por el método de atribuir creencias, deseos y perspicacia racional".

¿Qué objetos se pueden describir con esta *postura intencional*? Autores como Seel (1989), Rosenschein y Kaelbling (1986) han llegado a la conclusión, de que se puede definir cualquier cosa. Si el sistema es muy sencillo se le puede atribuir descripciones intencionales consistentes con el comportamiento observado. Sin embargo, en sistemas complejos las nociones intencionales son herramientas de abstracción, que proporcionan una forma adecuada y familiar de describir, explicar y predecir el comportamiento de sistemas complejos.

Por tanto un agente puede verse como un sistema capaz de definirse correctamente mediante posturas intencionales. Un agente contiene actitudes que proporcionan información (actitudes de información) y pro-actitudes.

| ***Actitudes de Información*** | ***Pro-actitudes*** |
|---|---|
| Creencias<br>Conocimiento | Deseo<br>Intención<br>Obligación<br>Compromiso<br>Elección<br>... |

Tabla 2.1: Tipos de actitudes.

Las *actitudes de información* están relacionadas con el conocimiento que un agente tiene sobre su entorno, mientras que las *pro-actitudes* son aquellas que guían de algún modo las acciones del agente. Precisamente, la elección de cual es la combinación de actitudes más adecuada para caracterizar los agentes es motivo de grandes discusiones. Parece razonable que dicha combinación posea, al menos, una actitud información y una pro-actitud (Wooldridge *et al*., 1995). Un agente toma decisiones y posee intenciones en base a la información disponible de su entorno, por tanto, la relación entre las dos categorías de actitudes es un enlace cerrado.

Como ya se indicó, no hay un consenso claro entre el mundo de la inteligencia artificial y la comunidad de filósofos acerca de cual es la combinación entre las actitudes de información y las pro-actitudes más adecuadas para definir un agente. A continuación se incluye una revisión en la cual se indica qué actitudes y pro-actitudes incorporan diferentes autores para caracterizar un agente (Wooldridge *et al*., 1995).

**Conocimiento y acción**: Moore (1990) es un pionero en el uso de lógicas para capturar aspectos de los agentes. Se centra en definir que datos debería conocer un agente para así proyectar el conjunto de precondiciones en el de acciones. De esta manera un agente podría ejecutar las acciones en un orden determinado como consecuencia de un estímulo. Moore formalizó el "*modelo de habilidad"* con una lógica que contiene una forma de modelar el conocimiento, y una lógica dinámica para el modelado de

las acciones. Este formalismo permite que un agente que posea información incompleta sobre cómo alcanzar algún objetivo, pueda hacerlo experimentando y ejecutando acciones en un orden concreto.

**Intención**: Cohen y Levesque (1990) utilizan dos actitudes básicas: creencias y objetivos. Las actitudes, como la intención, se definen en términos de las anteriores. Originalmente, su formalismo se utilizó para desarrollar la "*Teoría de Intención"*. Se ha demostrado posteriormente que la lógica propuesta por estos autores es adecuada para generar mecanismos de razonamiento que los agentes pueden utilizar para el análisis de conflictos, la cooperación entre agentes y, también, como fundamento teórico de cooperación en la resolución de problemas.

**Creencias, Deseos, Intenciones**: Rao y Georgeff (1991) han desarrollado una estructura lógica para la teoría de agentes basada en tres primitivas: creencias, deseos e intenciones. Su formalismo se basa en una extensión del "modelo de tiempo", en el cual los mundos accesibles de creencias, deseos e intenciones son en sí mismo extensiones de ese modelo.

**Intenciones**, **Creencias, Conocimiento, Saber-Como**: Singh (1994) ha desarrollado una familia interesante de lógicas para representar intenciones, creencias, conocimiento, saber-como y comunicación utilizando una extensión del modelo de tiempo. El formalismo de Singh es extremadamente rico y complejo.

Wooldridge (1992) ha desarrollado una familia de lógicas para representar las propiedades de los sistemas multiagente. Sus formalismos se usan en la especificación y verificación de estos sistemas. Desarrolló un modelo simple y genérico de sistemas multiagentes realistas, y mostró ejemplos de cómo estas lógicas se utilizan en la especificación y verificación de protocolos de cooperación.

### 2.2.1 Modelo BDI: Creencias, Deseos e Intenciones

Esta sección se centra en la arquitectura BDI (*Belief, Desire, Intention*) caracterizada porque los agentes que la implementan están dotados de los estados mentales de Creencias, Deseos e Intenciones. Posiblemente ha sido el modelo más difundido y el más estudiado dentro de los modelos de razonamiento de agentes. Hay varias razones para que esto ocurriera pero quizás la más convincente es que el modelo BDI combina elementos interesantes: un apreciable modelo filosófico de razonamiento humano fácil de comprender, un número considerable de implementaciones (Georgeff *et al*., 1987), como por ejemplo sistemas de control de procesos, procesos de decisión en negocios, etc. y se ha desarrollado una semántica lógica abstracta y elegante, la cual ha sido aceptada por la comunidad científica (Rao y Georgeff, 1998; Schild, 1998).

Las nociones de complejidad y cambio tienen un gran impacto en la forma en que se construyen los sistemas computacionales y por tanto en los agentes. Los agentes y en particular los agentes BDI incorporan los componentes esenciales y necesarios para enfrentarse con el mundo real (Georgeff, 1998).

Muchas de las aplicaciones de sistemas informáticos son algorítmicas y trabajan con información exacta. Pero la mayoría de las aplicaciones del mundo real requieren sistemas más complejos, sistemas capaces de relacionarse con un entorno cambiante y con un cierto grado de incertidumbre. Los agentes y los sistemas multiagente tienen por tanto que ser capaces de proporcionar soluciones a este tipo de problemas. El modelo BDI proporciona soluciones en entornos dinámicos, inciertos; en los que el agente o los agentes sólo tienen una visión parcial del problema (el acceso a la información está limitado) y posiblemente manejen un número limitado de recursos (recursos informáticos finitos). Las creencias, los deseos, las intenciones, y los planes son una parte fundamental del estado de ese tipo de sistemas (Georgeff, 1998).

En el campo de la inteligencia artificial, las creencias representan el conocimiento que se tiene del entorno. Desde un punto de vista informático, son la forma de representar el estado del entorno, por

ejemplo el valor de una variable, los valores de una base de datos relacional, o expresiones simbólicas del cálculo de predicados. Las creencias son esenciales porque aunque el medio es dinámico los eventos pasados se tienen que recordar. Además el sistema sólo tiene una visión parcial del entorno (el ámbito de percepción necesita ser recordado). Como es posible que el sistema posea recursos limitados y los agentes deben responder en tiempo real, es conveniente almacenar la información importante en forma de creencias y así evitar el tener que recalcularla a partir de información previa. Por otro lado, puesto que las *creencias* representan información imperfecta acerca del mundo, la semántica fundamental del componente creencia se ajusta a asertos lógicos, aunque la representación computacional no tiene que ser necesariamente lógica.

Los *deseos* (objetivos) son otro componente esencial del estado de estos sistemas. En términos informáticos, un objetivo puede simplemente ser el valor de una variable, un registro, o una expresión simbólica en alguna lógica. Lo importante es que un objetivo representa algún estado final deseado. El software convencional está "orientado a la tarea" en lugar de "al objetivo", de forma que cada tarea (o subrutina) se ejecuta sin ningún recuerdo de por qué ha comenzado su ejecución. Esto significa que el sistema no puede recuperarse ante fallos automáticamente (a menos que esto sea específicamente codificado por el programador) y no puede descubrir ni aprovechar oportunidades que surjan inesperadamente. Por ejemplo, la razón de que un humano recuerde la pérdida de un tren o un pinchazo inesperado de la rueda de su coche, es porque conoce dónde está (a través de sus creencias) y recuerda qué quiere conseguir (a través de sus objetivos). La semántica fundamental de los objetivos, sin tener en cuenta como se representa computacionalmente, se reflejaría en una lógica de deseo.

¿Son suficientes las creencias y los objetivos? Si se ha decidido llevar a cabo una acción (un plan), y el entorno cambia de alguna forma, ¿qué se haría continuar con las decisiones tomadas o replanificar? Los seguidores de la teoría de decisión clásica dicen que siempre se replanificaría, mientras que los seguidores del software convencional orientado a la tarea decidirían continuar. ¿Cuál es la aproximación correcta? Posiblemente ni la teoría de decisión clásica, ni la aproximación convencional son por si solas apropiadas. El sistema necesita comprometerse con los planes y sub-objetivos, pero también ser capaz de reconsiderar éstos en los momentos clave. Estos planes comprometidos constituyen las *intenciones* del agente. Las intenciones son simplemente un conjunto de caminos de ejecución (*threads*) que pueden ser interrumpidos de una forma apropiada al recibir información acerca de cambios en el entorno (Kinny y Georgeff, 1991).

El sistema necesita almacenar sus *intenciones* actuales (porque es un recurso limitado y efímero), además de *planes* para su uso en situaciones futuras (mejor que crear nuevos planes desde el principio). Estos planes desde el punto de vista semántico, se podrían considerar como un tipo de creencias, pero por su importancia computacional, se separan constituyendo otro componente más del estado del sistema.

Los componentes básicos de los sistemas creados para trabajar en un entorno dinámico e incierto incluirían de alguna manera las creencias, representación de creencias, deseos, intenciones y planes, y a estos se le denomina agentes BDI. Los cuales disponen de los componentes necesarios para enfrentarse con los problemas del mundo real (Georgeff, 1998).

La distribución lógica y física de la información y de los mecanismos de procesamiento actuales hacen que resulte eficiente la utilización de sistemas de agentes distribuidos. Los sistemas multiagente además de tener las ventajas proporcionadas por los sistemas distribuidos, tienen el valor añadido de contar con agentes que de forma individual pueden enfrentarse localmente a problemas concretos.

### 2.2.2 Modelo BDI según Rao y Georgeff

Una de las líneas de trabajo básicas en el modelado de agentes BDI es la que lideran Rao y Georgeff (Rao *et al*., 1995). Esta sección se centra en ella, ya que es el modelo más representativo y extendido

dentro de este campo. Proponen una forma de diseñar e implementarlos, así como una base teórica para la construcción de agentes (Kinny *et al.,* 1996).

De acuerdo con estos autores un sistema puede modelarse como un agente con actitudes mentales de creencias (*beliefs*), deseos (*desires*) e intenciones (*intentions*). Aunque no existe un punto de vista unificado en el área con respecto a si estas tres actitudes son suficientes o demasiadas, en el enfoque que se presenta, los autores consideran que, de acuerdo con su experiencia y por razones prácticas, estas tres actitudes son necesarias y suficientes (Rao *et al.*, 1995).

La definición propuesta por Rao y Georgeff (1995), de cada una de las actitudes consideradas en el diseño de los agentes racionales, se introduce a continuación. Un agente ha de ser capaz de actuar de forma autónoma en un entorno cambiante. Para ello es necesario que un componente del sistema almacene la información que se posee del entorno en el que está inmerso el agente, de forma que pueda modificarla cuando los sensores detecten cambios en el medio que le rodea. Esta componente del sistema son las *creencias*.

En segundo término, el sistema debe tener información acerca de los objetivos a alcanzar y las prioridades asociadas a cada uno de los objetivos. En un agente no existe normalmente un único objetivo, como en la mayoría de los sistemas de AI, sino varios que deben ser priorizados y pueden ser incluso incompatibles. Estos objetivos conforman la parte de *deseos* del sistema.

Por último, y puesto que el entorno cambia, puede ser necesario modificar el curso de las acciones a tomar para alcanzar un objetivo determinado. Por ello, se incluye una componente que representa el camino escogido por el sistema para alcanzar los objetivos. Esta componente se denomina *intención*.

Para resumir, se puede decir que las creencias representan la parte de información del sistema, los deseos la parte de motivación y las intenciones la parte premeditada o deliberada del sistema.

A continuación se presentan una serie de aspectos formales relacionados con la arquitectura BDI, una arquitectura abstracta para la implementación de agentes y algunos sistemas prácticos que aplican esta arquitectura.

#### 2.2.2.1 Formalismo teórico del modelo BDI

Las actitudes definidas previamente se representan en un formalismo basado en la lógica modal y el concepto de "mundos posibles" *(possible worlds)*.

De forma intuitiva, la lógica modal se puede definir como aquella que permite razonar sobre lo que podría ser, o se cree que es, en lugar de lo que es realmente. Es decir, permite representar conocimiento del tipo "Es posible que Luis tenga apendicitis". Esto es, desde la situación actual, se puede llegar a una nueva en la que Luis tenga esta enfermedad, aunque también sería posible que se llegase a otras situaciones (mundos) en las que Luis no tuviese apendicitis.

Para conseguir que la lógica incluya estos significados se maneja una semántica de mundos posibles con una relación de accesibilidad entre ellos. Dicha semántica hace necesario ampliar la sintaxis con la incorporación de nuevos operadores: el operador de necesidad y el operador de posibilidad (Chellas, 1980; Hughes *et al*., 1996). La relación de accesibilidad enlaza la situación actual (el mundo actual) con todas las que son posibles a partir de ella. De este modo los conceptos de verdad necesaria y verdad posible se definen a partir de esta relación de accesibilidad. Una proposición es necesariamente verdadera en un mundo si es verdadera en todos los mundos que son accesibles desde él (lo que incluye el caso de que no existiese ningún mundo accesible). Una proposición es posiblemente verdadera en un mundo si es verdadera en alguno de los mundos accesibles desde él (Chellas, 1980).

Como ya se indicó, los agentes BDI se modelan utilizando una estructura, basada en la lógica de mundos posibles, denominada árbol temporal con múltiples futuros y un solo pasado (Rao *et al.,* 1991). Cada nodo del árbol es una situación (o mundo posible) y las ramas del árbol pueden verse como las opciones disponibles para el agente en cada momento del tiempo. Para esta estructura de mundos posibles se definen varias relaciones de accesibilidad (distintas modalidades). Esto es, para cada situación, se definen una serie de mundos accesibles (nuevas situaciones a las que el agente puede llegar) desde el punto de vista de las creencias (los mundos que el agente considera posibles), de los deseos (mundos que el agente desea alcanzar) y de las intenciones (aquellos mundos que el agente ha decidido que intentará alcanzar) (Rao *et al.*, 1995). Aunque, como ya se indicó, los deseos podrían ser incompatibles, este modelo exige que sean consistentes y, además, como se comentará posteriormente, el agente debe creer que dicho objetivo es alcanzable.

Semánticamente, podría decirse que el agente, situado en un mundo accesible por sus creencias, cambia a un mundo accesible por objetivos al *desear* nuevos caminos de acción. De ese mundo evoluciona a otro accesible por medio de las intenciones al comprometerse a realizar las acciones deseadas (Rao *et al.,* 1991).

Basándose en el formalismo teórico anteriormente descrito, se definen las relaciones que deben existir entre las creencias, los deseos y las intenciones del agente. Algunas de las más importantes se enumeran a continuación de manera intuitiva:

- **Compatibilidad entre creencias y objetivos.** Si el agente adopta el deseo de alcanzar un objetivo, debe creer que en alguno de los mundos accesibles por la relación de creencia dicho objetivo es cierto.
- **Compatibilidad entre objetivos e intenciones.** Previamente a que el agente adopte una fórmula como intención debe haberla adoptado como deseo.
- **Las intenciones conducen a acciones.** Si una de las intenciones es una acción primitiva o simple, el agente la ejecuta. Es decir, no se pospone la ejecución de una acción simple.
- **Relación entre creencias e intenciones.** El agente conoce (cree en) sus propias intenciones.
- **Relación entre creencias y objetivos.** El agente conoce sus objetivos o deseos.
- **No hay retrasos infinitos.** Cuando un agente adopta una intención, sigue con ella hasta algún momento del futuro; es decir, no puede haber una posposición infinita en el proceso de alcance de un determinado objetivo.

Un aspecto importante de la arquitectura BDI es la noción de compromiso con las decisiones previas. El compromiso presta una cierta estabilidad al proceso de razonamiento. De este modo, se economiza el esfuerzo informático y, por lo tanto, mejora la ejecución global. Un compromiso tiene dos partes: una es la condición que el agente está comprometido a mantener, llamada condición comprometida; y la segunda es la condición bajo la cual el agente renuncia al compromiso, llamada condición de terminación (Rao *et al.*, 1995).

Como el agente no tiene control directo sobre sus creencias y deseos, no hay forma de que pueda adoptar o realizar un compromiso estratégico sobre estas actitudes. Sin embargo, un agente puede elegir qué hacer con sus intenciones. Así, se restringe la condición de compromiso a las intenciones. Un agente puede comprometerse a una intención basándose en que el objetivo de la intención es satisfecho en un camino futuro o todos los caminos futuros, provocando diferentes condiciones de compromiso y por lo tanto, diferentes comportamientos dinámicos. Las diferentes condiciones de terminación originan nuevos comportamientos.

De este modo las intenciones actuales del agente guían o influyen en sus decisiones sobre futuras intenciones. Dependiendo de cómo afecten las intenciones pasadas a las futuras, se identifican varios tipos de agentes:

- **Ciego**. El agente mantiene sus intenciones hasta que sabe que las ha alcanzado. Por lo tanto, si es necesario rechazaría las creencias o deseos que contradijesen sus compromisos.

- **Firme**. El agente mantiene sus intenciones mientras cree que tiene opciones de alcanzarlas.
- **Imparcial**. El agente mantiene sus intenciones mientras estas se corresponden con sus deseos, es decir, mientras el deseo o deseos que dieron lugar a esa intención no cambian.

Es importante tener en cuenta, que todos los aspectos definidos aquí de un modo intuitivo han sido formalizados empleando elementos de la lógica modal. Por lo tanto, existe una base formal en la que definir y demostrar las propiedades que han sido citadas (Rao *et al.,* 1991). El propósito de esta formalización es construir sistemas prácticos y verificables. Si para un dominio de aplicación se conocen los cambios del entorno y los comportamientos esperados del sistema, se puede usar esta formalización para especificar, diseñar y verificar agentes, que cuando estén en su entorno exhibirán los comportamientos deseados.

El formalismo descrito en los párrafos anteriores se puede ampliar para permitir la inclusión de librerías de planes, que definirán métodos estándar para la consecución de determinados objetivos (Cavendon *et al.,* 1995). Estos planes se componen de una condición de invocación (el objetivo que permiten conseguir) y un cuerpo (una secuencia de acciones para alcanzar el objetivo o incluso subobjetivos, que pueden tener o no asociados nuevos planes).

De este modo y de forma similar al modelo de creencias, deseos e intenciones, se define un mundo de planes *(plan World)*, aunque no es necesaria una relación de accesibilidad entre mundos puesto que los planes son estáticos, es decir, los mismos planes están disponibles en cada una de las situaciones posibles.

Evidentemente debe existir una interrelación entre las intenciones de un agente y los planes de que dispone. Estas relaciones se han definido formalmente (Cavendon *et al.*, 1995), siendo las más importantes las citadas a continuación:

- Las intenciones de un agente están restringidas por sus planes, es decir, las acciones se deben alcanzar exclusivamente utilizando los planes de la librería de planes del agente.
- Si un agente tiene la intención de alcanzar el objetivo $\phi$, debe adoptar también la intención de ejecutar el cuerpo del plan que permite alcanzar $\phi$.
- Por otro lado, se espera que si un agente tiene un plan para alcanzar el objetivo $\phi$, dentro de sus creencias esté el hecho de que el plan logrará el objetivo fijado ($\phi$).

#### 2.2.2.2 Arquitectura Abstracta

La arquitectura propuesta en Rao *et al.* (1995) se basa en la construcción de sistemas de agentes cuyas estructuras de datos se corresponden con cada uno de los componentes de los agentes BDI: creencias, deseos e intenciones. Cada uno de los componentes descritos se guardan aisladamente, o sea, existe una lista de creencias, otra de deseos y otra de intenciones. Además, se trabaja con una secuencia de eventos o acontecimientos ocurridos en el entorno. Cada uno de ellos se implementa como una cola, esto es, mediante una estructura de datos lineal con comportamiento FIFO (primero en llegar primero en salir).

Por lo que respecta a la utilización de estos datos para lograr la reactividad en el agente, parece evidente, que la implementación no puede estar basada en los métodos tradicionales, como la demostración de teoremas. La razón es que, aunque el método pudiese ser ampliado para dar cobertura a todos los aspectos lógico-formales antes descritos, no podría determinarse el límite máximo para el tiempo de computación. Esta característica no permitiría garantizar la reacción en tiempo real ante un entorno cambiante y, por tanto, comprometería la supervivencia del agente.

Por ello, la arquitectura propuesta realiza iterativamente una serie de pasos, cada uno de ellos con una duración limitada y por tanto, con posibilidad de reacción ante el entorno en un tiempo que, a nivel práctico, es suficiente. Al comienzo del ciclo, se lee la cola de eventos y se devuelve una lista de

opciones. Se seleccionan aquellas que se deben adoptar y se añaden a la cola de intenciones. A continuación, se ejecutan todas las intenciones que impliquen la realización de una acción simple. En este momento se comprueba si existen nuevos eventos en el entorno y se incorporan a la cola de eventos. Por último, el agente modifica las estructuras de deseo e intención, eliminando los ya satisfechos y los que son imposibles de alcanzar. Un esquema de este ciclo de ejecución puede verse en la Figura 2.1, tomada de Rao *et al.* (1995).

La arquitectura presentada por Rao es una idealización que representa los conceptos teóricos, aunque no parece completamente implementable a niveles prácticos. Por ejemplo, no se hace referencia a cómo desarrollar los procesos de generación o de selección de opciones, de forma suficientemente eficiente como para lograr las respuestas en un rango de tiempo adecuado.

```
Interprete-BDI
Inicializar_estado();
Repetir
Opciones:= generador_opciones (cola_eventos);
Opciones_seleccionadas :=  decidir (opciones);
Actualizar_intenciones (opciones_seleccionadas);
Ejecutar();
Leer_nuevos_eventos_externos();
Eliminar_actitudes_alcanzadas();
Eliminar_actitudes_imposibles();
Fin repetir
```

Figura 2.1: Ciclo de ejecución del agente BDI.

Por todo ello, dentro de este enfoque general, se han introducido, a la hora de la implementación algunas limitaciones para mejorar el modo de razonamiento, que restringen el poder expresivo de la arquitectura pero mejoran su rendimiento a niveles prácticos (Rao *et al.*, 1995). Por ejemplo, al definir las creencias hay que evitar el uso de disyunciones o implicaciones. Además los agentes suelen incorporar una librería de planes ya definidos (Cavendon *et al.,* 1995).

#### 2.2.2.3 Ejemplos prácticos

Utilizando el formalismo teórico y la arquitectura descritos previamente se han implementado distintas aplicaciones. Entre ellos cabe destacar un sistema gestor de tráfico aéreo, un sistema para la gestión de procesos de negocios y un sistema de modelado de combate aéreo.

El sistema gestor de tráfico aéreo (OASIS), consta de un gran número de agentes construidos siguiendo los esquemas marcados anteriormente, y entre otras cosas permite seleccionar una trayectoria de vuelo de entre varias posibles. Los agentes ayudan a seleccionar el mejor camino para minimizar el consumo de carburante, aumentar la productividad del vuelo, etc. y, a la vez, garantizar que se cumple para todos ellos el tiempo previsto de llegada. El sistema se ha utilizado en el aeropuerto de Sidney.

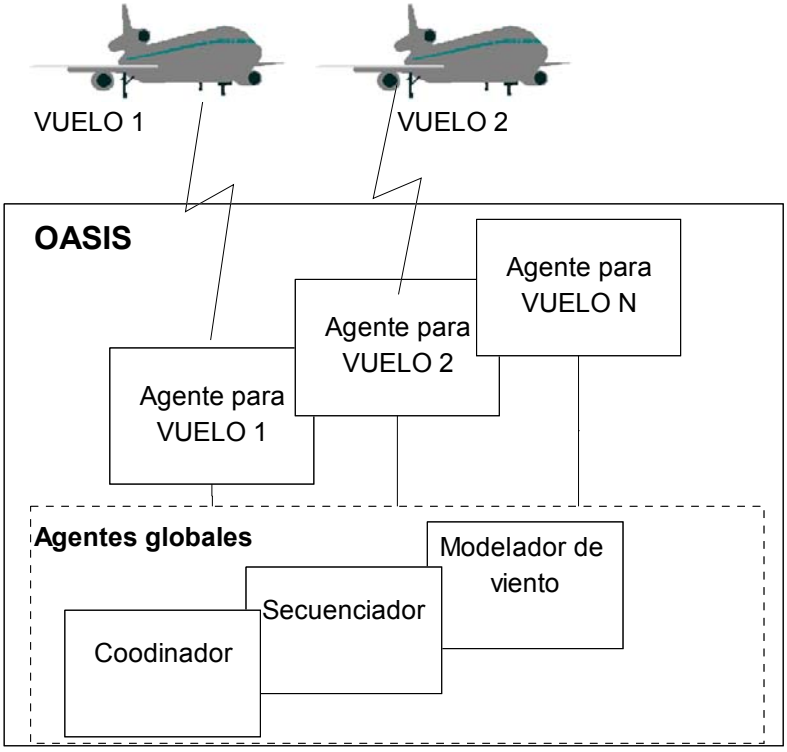


Figura 2.2: La arquitectura de agentes de OASIS.

La arquitectura OASIS divide las tareas de control del tráfico aéreo en grandes bloques y asigna un agente para resolver cada uno de esos subproblemas (Georgeff *et al*., 1998). Cada uno de los agentes resuelve su parte de modo independiente y coopera con el resto para definir el comportamiento global del sistema. En una ejecución habrá un agente por cada uno de los aviones implicados, además de otros agentes que actúan como secuenciadores, modeladores de viento, coordinadores y controladores de trayectoria. En la Figura 2.2 se describe la arquitectura de este sistema (Georgeff *et al*., 1998).

SPOC (Single Point of Contact) es un sistema de gestión de procesos de negocios. Este sistema permite a los empleados del servicio de atención al cliente, de una gran empresa australiana (Optus Communications), responder al 98% de las preguntas de los usuarios en su primer contacto con el servicio. La arquitectura del sistema está compuesta de tres capas:

- La capa de sistema constituida por un grupo de agentes que mantienen la configuración del sistema e interactúan con otros servicios de back-end.
- La capa de equipo, formada por una serie de agentes que gestionan la creación y borrado de nuevos agentes en la capa de usuario.
- La capa de usuario, que consiste en una serie de agentes que gestionan los procesos de cada uno de los usuarios e interactúan con los empleados del departamento de servicio de atención al cliente.

Por último, SWARMM es un sistema de modelado de combate aéreo que permite codificar las tácticas adoptadas por los pilotos en escenarios de combate y crear agentes que pueden participar con pilotos reales en estudios de simulación. El sistema modela tanto los aspectos físicos del avión (armas, sensores, rendimiento…) como el conocimiento táctico y la forma de razonar de los pilotos (Rao *et al*., 1992).

# 3. TIPOS DE AGENTES

## 3.1 Introducción

Los agentes se pueden clasificar teniendo en cuenta diferentes criterios. Como por ejemplo la arquitectura empleada en su construcción, a partir de la cual se habla de agentes deliberativos, reactivos o híbridos, como se comentó en el capítulo anterior.

Otra posible clasificación se puede realizar teniendo en cuenta el tipo de programa empleado para la implementación del agente (Russell *et al*., 1995), que distingue entre:

- **Agente de reflejo simple:** el agente utiliza reglas de condición-acción para establecer la conexión entre percepciones y acciones. El agente actúa encontrando una regla cuya condición coincida con la situación actual, definida por la percepción, y efectuando la acción que corresponda a tal regla.
- **Agente reflejo con estado interno**: el agente mantiene cierto tipo de estado interno, que se actualiza con información de cómo evoluciona el mundo independientemente del agente y de cómo las acciones del agente afectan al mundo.
- **Agentes basados en metas:** el agente para decidir qué hacer requiere información sobre su meta, información que detalle las situaciones deseables. El agente ha de elegir las posibles acciones que le permitan alcanzar su meta, para ello puede utilizar técnicas de búsqueda y planificación.
- **Agentes basados en utilidad:** las metas no bastan para generar una conducta adecuada, si se prefiere un estado del mundo a otro, entonces ese estado ofrece mayor utilidad al agente. La utilidad es una función que a cada estado le asocia su grado de utilidad. La completa especificación de la función de utilidad permite la toma de decisiones racionales cuando satisfacer algunas metas implica un conflicto o cuando el agente puede desear obtener varias metas y no existe la certeza de lograr ninguna de ellas (Russell *et al.*, 1995).

En ocasiones se utiliza una clasificación de agentes teniendo en cuenta las propiedades que caracterizan a un agente, y así por ejemplo se distinguen los agentes autónomos de los agentes reactivos, cuando realmente se está haciendo referencia a la citada propiedad que verifica el agente. Otro ejemplo similar sucede con los agentes móviles, aunque estos tienen especial relevancia debido al protagonismo que están adquiriendo gracias a Internet.

Los agentes móviles, constituyen uno de los campos de investigación más en expansión hoy en día, en donde la informática distribuida cobra especial relevancia gracias al avance en las comunicaciones. Un agente móvil es un programa capaz de viajar por redes de computadoras, como Internet, con el objetivo de evitar una sobrecarga de comunicación (el programa se ejecuta directamente en la máquina donde reside la información) o debido a que no dispone de ciertos recursos. Estos agentes interactúan con el ordenador, para solicitar información y son capaces de regresar a su lugar de origen una vez que han realizado las tareas especificadas. En este campo uno de los principales problemas que hay que solucionar es el de la seguridad, ya que lo ideal sería que el agente se ejecutase sin restricciones, lo cual puede ser peligroso.

Los agentes también se pueden clasificar en función de los modelos biológicos que representan, siguiendo una jerarquía natural (Franklin *et al.*, 1996). Los agentes autónomos se clasifican a nivel de "reino" robóticos o computacionales. Dentro de este último se encuadrarían los agentes software y los agentes de vida artificial. A nivel de "clase" podemos subclasificar los agentes software en agentes con tareas específicas, agentes para el ocio, virus informáticos, etc.

En la actualidad hay empresas de desarrollo de software que están empezando a desarrollar productos comerciales basados en la tecnología de agentes, empezándose a acuñar el nombre de agentes software para identificarlos. Dentro de estos, podemos destacar: agentes de interfaz y los agentes de Internet.

Los agentes de interfaz o asistentes personales, liberan al usuario en tareas habituales, repetitivas y generalmente sencillas. La principal característica es su autonomía y capacidad de aprendizaje para realizar las tareas, además la colaboración con el usuario no requiere necesariamente un lenguaje explícito de comunicación de agentes. Están basados en la idea de la delegación, ya que se supone que el agente tiene conocimientos sólidos sobre el tema y sobre nuestra forma de actuar. Un ejemplo de este tipo de agentes, sería aquel encargado de filtrar el correo, con capacidad para clasificar mensajes y actuar en función del tipo de información recibida o del emisor del mismo.

Los agentes de Internet surgen a partir del problema de la búsqueda y filtrado de información en la Web. Estos agentes resultan útiles debido a la gran cantidad de información que es posible encontrar sobre cualquier tema en este medio. Su tarea consiste en navegar por la red buscando y organizando información. Un ejemplo podría ser un agente que interactuara con buscadores como Yahoo, Lycos o Altavista y organizara la información relativa en función de criterios de búsqueda preestablecidos. Otro ejemplo sería un agente al que le demos información sobre un libro o un CD y nos seleccione la librería donde lo podemos encontrar más barato (podría ser estático o móvil). El siguiente apartado se centra en este tipo de agentes.

### 3.2 Agentes inteligentes en Internet

Como ya se ha comentado un agente es un sistema *software/hardware* que recibe estímulos de su entorno a través de sensores y actúa sobre el mismo a través de una serie de efectores (Russell *et al.*, 1995). Un ejemplo de este tipo de agentes en Internet es Eliza (http://www-ai.ijs.si/eliza/eliza.html) (Baumgartener y Payr, 1995). Este agente analiza las preguntas o respuestas de su interlocutor buscando palabras clave para las que tiene predefinidas ciertas respuestas. Otro programa de este tipo es Julia (http://www.uklaw.net/chatterbot.htm) (Foner, 1993). Sin embargo los agentes inteligentes en Internet normalmente están orientados a asistir a los usuarios de esta red en tareas concretas como por ejemplo búsqueda, navegación, filtrado, etc.. Por ejemplo uno de los primeros agentes que se desarrolló para Internet tenía como objetivo medir el tamaño de la red.

La importancia que está adquiriendo Internet en la actualidad hace que un gran número de estos agentes estén dedicados al tratamiento de la información y en especial a la recuperación de información. En general, para que las aplicaciones que existen en la Web dedicadas a estas u otras tareas se puedan considerar agentes inteligentes tienen que tener entre otras las siguientes características:

- Permitir la referencia a conceptos o palabras imprecisas y ambiguas.
- Poder tratar con información combinada de imágenes, sonido y texto (multimodalidad).
- Interaccionar con el usuario de manera flexible.
- Ser sistemas distribuidos competitivos o cooperativos.
- Realizar búsquedas deductivas.
- Aprender principalmente de manera inductiva.
- Realizar trabajos de minería de datos utilizando técnicas estadísticas sobre bases de datos permitiendo el reconocimiento de patrones y el aprendizaje.

De la misma forma que la inteligencia artificial (IA) ha hecho un gran esfuerzo para que avanzara Internet con la aparición de los agentes inteligentes, la web ha dinamizado algunos campos de la IA como la representación del conocimiento, el aprendizaje en máquinas, el procesamiento del lenguaje natural o los sistemas multiagente.

Podemos clasificar a los agentes inteligentes teniendo en cuenta la función que realizan: monitorización, filtrado, navegación, etc. (King, 1995). Muchos de estos agentes se han utilizado en el ámbito de Internet. Los agentes consejeros por ejemplo, observan el comportamiento de los usuarios,

aportándole sugerencias que ayuden a aumentar la eficiencia y/o eficacia de sus acciones en este u otro medio. Los agentes asistentes actúan en representación del usuario, como ya hemos visto, sobre un problema dentro de un dominio limitado y en el contexto local de operación del usuario, por ejemplo, los filtros de correo electrónico o los que responden a un correo electrónico entrante con otro saliente, indicando, por ejemplo, que el destinatario está de vacaciones.

### 3.2.1 Algunos ejemplos

DailyBriefing realiza una compilación personalizada de noticias procedentes de un buen número de periódicos, revistas y servicios de noticias. Para ello usa tecnología del ámbito de las redes neuronales artificiales. Otros tienen cometidos bastante más concretos, como aquellos que ayudan al usuario-cliente en el proceso de búsqueda de productos comerciales y a los vendedores en el de encontrar potenciales clientes para sus productos. Este es el caso de BargainFinder, un agente desarrollado por Andersen Consulting, que compara los productos a la venta en Internet para ofrecer el mejor precio para un disco compacto dado. Este agente se sitúa realmente en este submundo de Internet, de pujanza creciente, que es el comercio electrónico.

| Tipo de agente | Tarea típica | Características más destacables | Ejemplos |
|---|---|---|---|
| Asistentes de navegación | Recomendaciones sobre qué enlaces seguir durante el uso de navegadores | Uso de heurística, aprendizaje, colaboración entre agentes de usuarios distintos | 1-**WebWatcher**, opera a partir de objetivos especificados por el usuario<br>2-**Letizia**, interfaz basado en comportamiento, que no requiere la especificación a priori de objetivos, ni la evaluación posterior del usuario del resultado de las búsquedas realizadas |
| Asistentes de búsqueda | Recogen y elaboran información para el usuario según sus intereses | Aprendizaje, operación basada en ejemplos | 3-**InfoFinder**, pide al usuario documentos de muestra, con objeto de buscar otros relacionados<br>4-**Amalthaea**, multi-agente, basado en algoritmos genéticos, realizan procesos competitivos/cooperativos orientados al filtrado y descubrimiento de información |
| Monitorización | Realizan procesos de recolección, interpretación y asociación para dar respuesta a consultas del usuario | Reconocimiento y equiparación de patrones | 5-**ContacFinder**, revisa mensajes y obtiene información de expertos en las áreas observadas<br>6-**Shopbot**, crea descripciones de vendedores y asiste en tiempo real al usuario en labores de compra |
| Filtrado | Manejan información, clasificándola o seleccionándola de acuerdo a los requerimientos o preferencias del usuario | Operan, al contrario que los agentes de búsqueda, con consultas estáticas sobre información dinámica | 7-**ProFusion**, aporta nuevos enlaces sobre consultas registradas por el usuario<br>8-**Pefna**, filtra artículos de grupos de noticias a partir de artículos representativos |

Tabla 3.1: Varios agentes inteligentes (Casasola *et al.*, 1997).

Dos de los agentes más populares en aportar información a las consultas de los usuarios son el FaqFinder (Burke *et al.*, 1997) y el Referral Web (Kautz *et al.*, 1997). Ambos emplean métodos muy originales, aunque bien distintos entre si. El primero utiliza los ficheros denominados "Preguntas planteadas frecuentemente" (*Frequently asked questions*) como base de conocimiento sobre la que realizar la búsqueda de respuesta a la consulta de un usuario. El segundo es todavía más original. En este caso el agente sugiere a su interlocutor un experto humano que podría dar cumplida respuesta a su consulta. Para ello realiza una búsqueda apoyada en las relaciones sociales entre individuos. Según indican los propios autores para apoyar su propuesta: "La enorme red de documentos enlazados que

conforman la WWW es sólo una manifestación de un fenómeno mayor y más profundo; denominémosle, la red social que enlaza a todas las personas".

Ahoy! es un agente especializado en la localización de páginas personales en la red (http://www.cs.washington.edu/research/ahoy). A partir del nombre de una persona y la organización a la que ésta está involucrada, Ahoy! filtra la respuesta de múltiples índices de la red con el objeto de aportar una o dos referencias que probablemente "apuntarán" a la página personal de la persona buscada (Shakes *et al.*, 1997). En caso de que a través de esta vía no se encuentren respuestas probables, Ahoy! hace uso de su experiencia anterior para sugerir una dirección URL que podría corresponder a la página buscada. Una búsqueda a través de Ahoy! consume un tiempo medio de 9 segundos, muy inferior al necesario en procesos de búsqueda a través de cualquier otra alternativa usual. Además, y esto es todavía más importante, su precisión y su capacidad de recuperación de información son sensiblemente superiores a los de otros buscadores de más amplio espectro de búsqueda, como AltaVista, Yahoo!, Hotbot o MetaCrawler.

# 4. COMUNICACIÓN ENTRE AGENTES

## 4.1 Introducción

La comunicación entre los agentes les permite sincronizar acciones, enviar y recibir conocimiento, resolver conflictos en la resolución de una tarea, etc. (Russell *et al.*, 95). La comunicación es el soporte de la cooperación, coordinación y de los procesos de negociación que se dan entre los agentes

La comunicación puede proporcionar a los agentes el conocimiento necesario para desarrollar sus acciones con una visión menos local y poder sincronizarse con el resto de los agentes. Sin embargo, una excesiva comunicación puede dar lugar a agentes cuya sobrecarga de comunicación sea mayor que el trabajo efectivo realizado. Se pude distinguir un amplio rango de formas de comunicación (Werner, 1996):

- **No hay comunicación**. Los agentes pueden interactuar sin comunicarse, infiriendo las intenciones de otros agentes. Este modelo de comunicación es el que consigue mejores resultados cuando los objetivos de los agentes no están interrelacionados y por lo tanto no pueden entrar en conflicto (Schelling, 1960).
- **Comunicación primitiva**. En este caso la comunicación está restringida a un número de señales fijas con una interpretación establecida de antemano (Hoare, 1978).
- **Arquitectura de pizarra**. Se usa en la inteligencia artificial como una manera de compartir memoria y conocimiento (Hayes-Roth, 1985). Los agentes pueden escribir mensajes, dejar resultados parciales o encontrar información en una pizarra que todos saben donde está. Se usa esta arquitectura para intercambiar información entre subsistemas de agentes o para modelar el intercambio entre los distintos módulos que componen la estructura de un agente.
- **Intercambio de planes e información**. Dos agentes pueden intercambiarse sus respectivos planes, de esta manera cada uno puede adaptar sus estrategias. Esto presenta algunos incovenientes como el alto coste computacional del intercambio.
- **Intercambio de mensajes**. Consiste en agentes que actúan en respuesta al procesamiento de una comunicación (Agha *et al.*,1988). Las acciones que pueden ejecutar estos agentes son: enviarse mensajes entre ellos mismos, determinar un cambio en el comportamiento como consecuencia de haber alcanzado un nuevo estado después de la comunicación, etc.
- **Comunicación de alto nivel**. El diálogo entre agentes permite la generación e interpretación de palabras o declaraciones, con el objetivo de comunicar la información que el emisor conoce formada por creencias, compromisos e intenciones, para que así el receptor alcance el mismo estado mental que tiene el emisor (Grosz *et al*., 1990).

La comunicación permite la coordinación entre un grupo de agentes de forma que se ejecuten solamente aquellas acciones necesarias. Como no es el propósito de esta sección profundizar en las técnicas de coordinación entre agentes nos limitamos a decir que de forma general hay dos modelos de coordinación: (a) Coordinación global: cuando el sistema multiagente es capaz de determinar y planificar globalmente las acciones de los diferentes agentes. En este caso un agente verifica todos los conflictos posibles entre los agentes del sistema. (b) Coordinación individual: cuando los agentes del sistema multiagente tienen autonomía total para decidir qué hacer y resolver los conflictos con otros agentes.

Cuando se trata de resolver un problema de manera distribuida, entre un grupo de agentes, la solución es el resultado de la interacción cooperativa entre todos ellos. La comunicación facilita por tanto los procesos de cooperación. El grado de cooperación que hay entre agentes puede ir desde una cooperación completa hasta una antagónica u hostil. En el primer caso se paga un alto coste por la total comunicación entre agentes, pero en el segundo puede ocurrir que unos agentes bloqueen los objetivos de otros.

Para que los mecanismos de cooperación y coordinación tengan éxito en un sistema de agentes, debe de existir un mecanismo adicional, por medio del cual, los integrantes de un sistema se puedan poner

de acuerdo cuando cada agente defiende sus propios intereses, llevándolos a una situación que los beneficie a todos, teniendo en cuenta el punto de vista de cada uno. Este mecanismo es la negociación. La quinta parte de este documento se centra en este tema.

### 4.2 Lenguajes de comunicación de agentes

El hecho de que los agentes se comuniquen tanto con otros agentes como con el medio en el que habitan es un elemento representativo de este tipo de sistemas. Además es fundamental que exista un mecanismo adecuado de comunicación entre agentes sobre todo en comunidades multiagente. Incluso se ha llegado a sugerir que una entidad es un agente software si y sólo si se comunica correctamente en un lenguaje de comunicación de agentes (Genesereth *et al*., 1994).

Para que esta interacción e interoperación entre agentes software sea significativa, constructiva e inteligente se requieren tres componentes fundamentales y distintos:

- Un protocolo de transporte: formado por el mecanismo de transporte usado para la comunicación (por ejemplo TCP, SMTP, HTTP, etc).
- Un lenguaje común: es el medio por el cual se realiza el intercambio de información. Este lenguaje indica cual es el contenido de la comunicación y si es una aseveración, una pregunta o una solicitud.
- Un protocolo de interacción: se refiere a la estrategia que sigue el agente para interactuar con otros agentes, la cual puede ir desde esquemas de negociación y protocolos basados en teoría de juegos hasta protocolos muy simples en los que cada vez que el agente no sabe algo busca otro que lo sepa y le pregunta.

#### 4.2.1 Diseño de lenguajes

Los lenguajes de comunicación comunes facilitan la creación de software interoperable. De forma que los agentes o componentes pueden comunicarse independientemente del lenguaje en el que han sido implementados. En este caso se establece una separación entre la implementación de los agentes o componentes y del interface. Existen actualmente dos puntos de vista en el diseño de estos lenguajes:

- Lenguajes procedural: la comunicación se modela como un intercambio de directivas procedurales, pudiendo transmitir además de comandos, programas enteros que permiten a los agentes alcanzar sus objetivos. Las ventajas de estos lenguajes procedurales es que son sencillos y su ejecución es eficiente y directa. Sin embargo, y desafortunadamente, existen desventajas. Estos lenguajes requieren que se tenga información sobre los agentes que recibirán los mensajes, información que a veces no está disponible para el agente emisor. También existe el problema de que los procedimientos son unidireccionales. Mucha de la información que los agentes requieren compartir debe estar disponible en ambas direcciones (transmisión-recepción, y viceversa). Lo anterior puede complicarse cuando un agente recibe varios "*scripts*" provenientes de múltiples agentes que trabajan simultáneamente y que pueden interferir entre sí. La mezcla de información procedural es más complicada que la de información de tipo declarativa. Suelen usarse lenguajes de intérpretes de órdenes (*scripts*) tales como Perl, Tcl (Gray *et al.*, 1997), Apple Events, Telescript (White, 1994) etc. Permiten un rápido prototipado aunque no suelen ser fácilmente escalables ni reciclables. Son especialmente útiles para la construcción de agentes en aplicaciones finales como agentes de usuario o agentes móviles.

- Lenguajes declarativos: la comunicación tiene lugar utilizando enunciados declarativos, es decir intercambio de declaraciones (definiciones, suposiciones, etc), por lo que necesita que el lenguaje sea lo suficientemente expresivo (como para comunicar diferentes clases de información) y compacto. Varios Proyectos *ARPA* han hecho posible que la iniciativa *Knowledge Sharing Effort*

haya permitido definir los componentes del lenguaje de comunicación de agentes declarativo ACL (*Agent Communication Language*) (Genesereth *et al.*, 1994) (ARPA, 1992).

### 4.2.2 ACL

A continuación nos centramos en la definición del lenguaje ACL (*Agent Communication Language*) por ser uno de los más utilizados y porque a nuestro juicio muestra la esencia de este tipo de lenguajes. ACL es un lenguaje que permite la interoperación entre agentes autónomos distribuidos. Un mensaje en ACL es una expresión *KQML* (*Knowledge Query Manipulation Language*) qué consiste en una directiva de comunicación y un contenido semántico en *KIF* (*Knowledge Interchange Format*) (es decir los argumentos de la expresión son términos u oraciones expresados en KIF) construido a partir de términos de un *vocabulario*. En definitiva, ACL tiene tres componentes:

- Vocabulario.
- KIF (Knowledge Interchange Format).
- KQML (Knowledge Query Manipulation Language).

#### 4.2.2.1 Vocabulario. Ontologías

El vocabulario del ACL es un diccionario abierto de palabras amoldadas a cada una de las áreas del dominio de la aplicación. Cada palabra tiene una descripción en lenguaje natural (que facilita el entendimiento de su significado) y una anotación formal (escrita en KIF). El diccionario está abierto de forma que se puedan añadir palabras de nuevas áreas de aplicación.

El término ontología se utiliza para definir la *especificación de una conceptualización* (Grub, 1993), en este caso permite hacer una descripción de los conceptos y relaciones que pueden formar parte del conocimiento de un agente o de una comunidad de agentes. Las ontologías utilizan un vocabulario formal, y un conjunto de definiciones.

Los agentes establecen *compromisos ontológicos*. Un compromiso ontológico es un acuerdo para usar un vocabulario, que es consistente, aunque no completo, respecto a la teoría especificada en la ontología, es decir; se trata de definir un vocabulario común con el que se puede representar el conocimiento compartido. Se construyen agentes comprometidos con ontologías y se diseñan ontologías con las que los agentes pueden compartir conocimiento.

Un área de aplicación se puede describir de muchas formas, el diccionario puede contener múltiples *ontologías* para un área y cada agente utiliza la que más le conviene. Por ejemplo puede definirse un vocabulario para describir la geometría tridimensional en términos de coordenadas rectangulares, coordenadas polares, coordenadas cilíndricas, etc. Las definiciones formales asociadas a las ontologías pueden ser utilizadas por los agentes para transformar mensajes de una ontología a otra ontología.

En las siguientes direcciones electrónicas se puede obtener más información sobre ontologías:

- http://AI-WWW.AIST-NARA.AC.jp/~takeda/doc/html/icot-paper-v2/
- http://www.ksl.stanford.edu/knowledge-sharing/ontologies/README.html

#### 4.2.2.2 KIF

El lenguaje de contenido KIF (*Knowledge Interchange Format*, Formato de intercambio de conocimiento o lenguaje interno) es una versión en notación prefija (es decir, el símbolo que define la operación a realizar se antepone a los operandos) del cálculo de predicados de primer orden, con varias extensiones para incrementar su expresividad. La descripción del lenguaje incluye tanto una especificación para su sintaxis como una para su semántica (Genesereth *et al.,*1992).

KIF permite expresar datos simples e información más complicada (restricciones, negaciones, disyunciones, reglas, expresiones, e información a un metanivel) mediante el uso de términos complejos. Además, KIF incluye una variedad de operadores lógicos para ayudar a codificar información lógica y dos operadores (? y ,) junto a un vocabulario que permita codificar conocimiento acerca del conocimiento. Por último, también se puede utilizar para describir procedimientos, es decir, escribir programas para agentes. Desde el punto de vista sintáctico KIF es similar a Lisp y a Scheme.

En definitiva, KIF define un conjunto de objetos, funciones y relaciones cuyo significado es fijo, aunque KIF también es abierto, es decir, los usuarios tienen la libertad de definir significados de cualquier otro símbolo que no esté predefinido.

(En la dirección electrónica http://www.cs.umbc.edu/kse/kif/ esta disponible información sobre KIF y herramientas asociadas)

#### 4.2.2.3 KQML

Aunque es posible diseñar un marco de trabajo completo para la comunicación en el que todos los mensajes tengan la forma de oraciones en KIF, esto sería ineficiente, ya que se requeriría incluir información implícita sobre el agente que envía el mensaje y sobre el que lo recibe, junto con información complementria como por ejemplo la hora del envío del mensaje, la historia del mensaje, etc, debido a que la semántica de KIF es independiente del contexto. La comunicación es más eficiente si se provee un elemento lingüístico en la que el contexto se toma en cuenta. Esta es la función de KQML (*Knowledge Query Manipulation Language*) (Mayfield *et al.,* 1995).

El lenguaje de comunicación *KQML* (Lenguaje de manipulación y consulta de conocimiento) define un conjunto de protocolos que facilitan el intercambio de información y conocimiento entre agentes (programas autónomos y asíncronos), en tiempo de ejecución. Este lenguaje esta basado en la teoría de actos del habla, que se utiliza para construir una capa lingüística en la que se tiene en cuenta el contexto y para formalizar las acciones lingüísticas de los agentes. Esta teoría se basa en el hecho de que las oraciones expresadas por humanos durante la comunicación no siempre aseveran un hecho, sino que en realidad tratan de transmitir una creencia o conocimiento, una intención o un deseo. Además, esta teoría también ha contribuido al entendimiento de la relación entre el estado interno de un agente y las expresiones que intercambia con otros agentes (Haddadi, 1996).

KQML define el formato de los mensajes y el protocolo que los maneja para permitir que los agentes se identifiquen, se conecten e intercambien información con otros agentes. Sus características son (Mayfield *et al.,* 1996):

- KQML cumple con los estándares definidos por FIPA (*Foundation for Intelligent Phisical Agents*, 1997).
- Los mensajes son opacos al contenido de lo que transportan, es decir, no comunican únicamente oraciones en un lenguaje, sino que comunican una actitud acerca del contenido. Por ejemplo, afirmación, solicitud, pregunta.
- Las primitivas del lenguaje se llaman ejecutivas (*performative*, actuación o realizativos), e indican que acciones u operaciones pueden llevar a cabo los agentes cuando se comunican, es decir; las operaciones que los agentes pueden ejecutar en la base de conocimiento de los otros agentes.
- Para la comunicación entre agentes, se pueden usar agentes especiales llamados *facilitadores (communication facilitators)* que coordinan las relaciones e interacciones entre los agentes, es decir, proporcionan ciertas funciones tales como: asociación de direcciones físicas con nombres simbólicos, registro de bases de datos y/o servicios ofrecidos o buscados por los agentes; y servicios de comunicación como reenvío e intermediación (brokering).

Para poder identificar cada uno de los componentes de ACL, utilizaremos como ejemplo un mensaje en el que aparecen los tres componentes.

Mensaje: Petición de información sobre el trabajador ideal para llevar a cabo una taréa concreta.

**ask-if**
**:content** (<*taréa-concreta*>)
**:reply-with** qj

En negrita aparece la ejecutiva KQML y sus parametros, en cursiva el vocabulario (representan objetos del dominio de la aplicación) y el resto (qj, que es el identificador del mensaje, los parentesis y demás) es KIF.

#### 4.2.2.3.1 Descripción de KQML

El lenguaje KQML conceptualmente está estructurado en tres niveles (Finin *et al.,* 1994):

- **Nivel de contenido**: se relaciona con el contenido real del mensaje escrito en el lenguaje de representación propio de cada agente (se puede usar cualquier lenguaje incluyendo lenguajes expresados como cadenas en ASCII o aquellos expresados usando una notación binaria). Cualquier implementación de KQML ignora la parte del contenido excepto la magnitud que se utiliza para determinar donde termina.
- **Nivel de mensaje**: es el núcleo de este lenguaje pues se utiliza para codificar el mensaje que un agente desea transmitir a otro, es decir; determina las clases de interacciones que se pueden tener con un agente que "hable" KQML. La función principal de este nivel es identificar el protocolo que se va a usar para entregar el mensaje (síncrono o asíncrono) y proveer un *acto del habla* o ejecutiva que el transmisor agrega al contenido. La ejecutiva indica si el contenido es una *aserción*, una *pregunta*, un *comando*, etc. Además, ya que el contenido es opaco a KQML, en esta capa se incluyen características opcionales que describen el contenido: su lenguaje, la ontología que se usa y alguna clase de descripción más general, por ejemplo un descriptor que nombra un tema dentro de la ontología. Estas características permiten que las implementaciones de KQML analicen, ruteen y entreguen el mensaje apropiadamente aunque su contenido sea inaccesible.
- **nivel de comunicación**: codifica un conjunto de características del mensaje que describen parámetros de comunicación de bajo nivel, tales como la identidad del agente que envía el mensaje y del que lo recibe y un identificador único asociado con la comunicación.

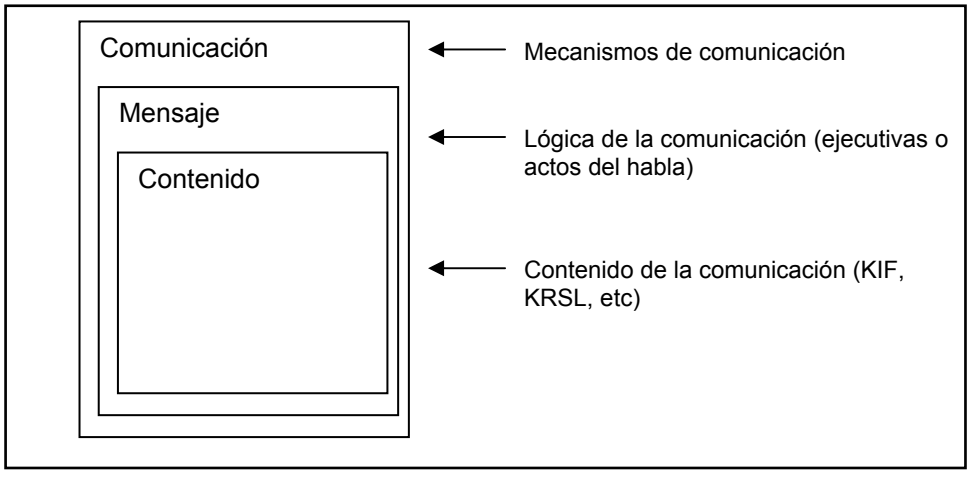


Figura 4.1: Estructuración del lenguaje KQML.

#### 4.2.2.3.2 Sintaxis KQML

El lenguaje KQML es independiente del formato de la información. Un mensaje KQML se inicia con una ejecutiva (indica la intención del mensaje o el acto del habla, y se representa como una cadena ASCII, además invita al receptor a realizar alguna acción), seguida de sus argumentos asociados que incluyen el contenido real del mensaje, y un conjunto de argumentos opcionales. Estos argumentos describen el contenido de una manera independiente de la sintaxis del lenguaje utilizado para expresar el contenido.

La sintáxis de KQML se ayuda de una lista de paréntesis balanceados y se puede considerar como una subclase de la notación prefija del CommonLisp. El elemento inicial de la lista es la ejecutiva y el resto de elementos son los argumentos, que son pares de palabra-clave/valor. Los parámetros están indexados por palabras clave, que pueden usarse en cualquier orden. Las palabras clave, llamadas *parameter names,* deben empezar por dos puntos (:), y deben ir delante de sus correspondientes valores, llamados en la especificación *parameter values*. Debido a que el lenguaje es relativamente simple, la sintáxis actual es relativamente insignificante y se puede cambiar si es necesario en el futuro.

En la siguiente tabla se puede observar la sintaxis de KQML en formato BNF (BACKUS-NAUR FORM). Con respecto a las sintáxis BNF es conveniente tener en cuenta las siguientes indicaciones:

[<x>] significa “opcional <x>”  <x>* significa “cero o más <x>”
<x>+ significa “uno o más <x>”  <x>|<y> significa “<x> or_exclusivo <y>”

```
<performative> ::=(<word> {<whitespace> :<word> <whitespace> <expression>}*)
<expression> ::= <word> | <quotation> | <string> | (<word> {<whitespace>
<expression>}*)
<word> ::= <character> <character> *
<character> ::= <alphabetic> | <numeric> | <special>
<special> ::= < | > | = | + | - | * | / | & | ^ | ~ | _ | @ | $ | % | : | . | ! | ?
<quotation> ::= '<expression> | ,<comma-expression>
<comma-expression> ::= <word> | <quotation> | <string> | ,<comma-expression>
(word> {<whitespace> <comma-expression>}*)
<string> ::= "<stringchar>*" | #<digit><digit>*"<ascii>*
        <stringchar> ::= \<ascii> | <ascii>-\-<double-quote>
```

Tabla 4.1: Sintáxis BNF de la gramática de KQML.

Los nombres de las ejecutivas son palabras reservadas, y existe un conjunto de ellas estándar que se dividen en:

- **ejecutivas de discurso**: se emplean en el contexto de un intercambio de información y conocimiento entre dos agentes.
  *ask-if, ask-all, ask-one, tell, deny, achieve, advertise, subscribe, etc.*

- **ejecutivas de intervención y mecánicas de la conversación**: se usan para intervenir en el curso normal de una conversación.
  *error, sorry, ready, next, discard, rest, standby, etc.*

- **ejecutivas de red y de facilitación**: no son actos del habla, pero permiten a los agentes encontrar otros agentes capaces de procesar sus mensajes.
  *register, forward, broadcast, recommend-one, recruit-all, etc.*

Las ejecutivas definidas pueden no ser suficientes para implementar un sistema multiagente, o puede que estos no necesiten soportar el conjunto entero de ejecutivas. Es decir, los agentes utilizaran un subconjunto y podrán usar ejecutivas que no aparecen en la especificación del lenguaje, definiendolas previamente de modo preciso y con el mismo estilo de las de la especificación. Es decir, el conjunto de ejecutivas de KQML es extensible.

El ANEXO A presenta una relación de las palabras reservadas para las ejecutivas junto con sus respectivos significados.

#### 4.2.2.3.3 Semántica KQML

El modelo semántico de KQML define un contexto simple y uniforme para que los agentes tengan conocimiento acerca de las capacidades de otros agentes (Labrou *et al.,* 1994). Desde fuera, cada agente, aparece como si manejara una base de conocimiento (*Knowledge Base,* KB). La implementación de un agente no siempre requiere la existencia de una base de conocimiento, sino que puede utilizar un sistema simple de base de datos o una estructura de datos propia. Por ello, se dice que cada agente organiza una base de conocimiento virtual (*Virtual Knowledge Base*, VKB). Las VKB tiene dos componentes separados:

- Creencias (almacén de información): codifican información sobre el agente y sobre su entorno, incluyendo las VKB de otros agentes.
- Objetivos o intenciones (almacén de metas): codifican los estados del entorno que el agente quiere alcanzar cuando actúa.

Los agentes se intercambian el contenido de sus VKB y el contenido de las de los otros; pero la codificación del contenido de las distintas VKB puede usar gran variedad de lenguajes de representación.

Las ejecutivas primitivas están definidas en términos de su efecto sobre estos almacenes. Por ejemplo, un *tell(s)* es una aserción que hace el emisor al agente receptor para comunicarle que la sentencia *s* está en su almacén de creencias virtual. Un *achieve(s)* es una petición del emisor al receptor para añadir *s* a su almacén de intenciones.

Las ejecutivas tienen parámetros que se identifican por palabras claves. Todos los mensajes que usen parámetros con estas palabras claves deben ser consistentes con su definición. La siguiente tabla muestra el significado de los parámetros más comunes:

| Palabra clave | Significado |
|---|---|
| :content | Contenido de la ejecutiva |
| :force | El emisor nunca negará el significado de una ejecutiva |
| :in-reply-to | La etiqueta esperada en una respuesta |
| :language | El nombre del lenguaje de representación del parámetro :content |
| :ontology | El nombre de la ontología usada en el parámetro :content |
| :receiver | El receptor del mensaje |
| :reply-with | El receptor espera una respuesta con etiqueta |
| :sender | El emisor del mensaje |

Tabla 4.2: Parámetros de las ejecutivas de KQML.

#### 4.2.2.3.4 Requisitos de Transporte

La especificación de KQML (Finin *et al.,* 1993) no trata de estandarizar cómo ha de ser la infraestructura de transporte de mensajes, ya que normalmente esto depende del lenguaje de programación y de la red. Lo que sí define es una abstracción a nivel de transporte:

- Los agentes están conectados mediante links de comunicación unidireccionales.
- Los links pueden tener un pequeño retraso asociado.
- Cuando un agente recibe un mensaje, sabe desde qué link de entrada ha llegado el mensaje.
- Cuando un agente envía un mensaje puede dirigirlo a un link de salida concreto.
- Los mensajes dirigidos a un destino único han de llegar en el orden en que fueron enviados.
- La entrega de mensajes es segura y fiable.

Esta abstracción se puede implementar de diversas formas. Por ejemplo, los links podrían ser conexiones TCP/IP sobre Internet, que están activos solamente mientras dura la transmisión de un mensaje. Los links podrían ser caminos fijos de correo electrónico. También pueden ser conexiones IPC entre distintos procesos ejecutados en máquinas UNIX. KQML considera que a nivel de agente, la comunicación es un paso de mensajes punto a punto.

#### 4.2.2.3.5 Protocolos del KQML

Existe una variedad de protocolos de intercambio de información entre procesos, pero en orden de complejidad los que soporta KQML son (Finin *et al.,* 1994):

- Comunicación síncrona: una pregunta bloqueante espera por una respuesta esperada. Un proceso (cliente) envia una pregunta a otro proceso (servidor) y espera por una respuesta. Esto ocurre comúnmente cuanto un razonador encadenado-hacia-atrás recupera información de un origen remoto. Cuando necesita datos, coloca preguntas y espera por las contestaciones antes de intentar cualquier inferencia.

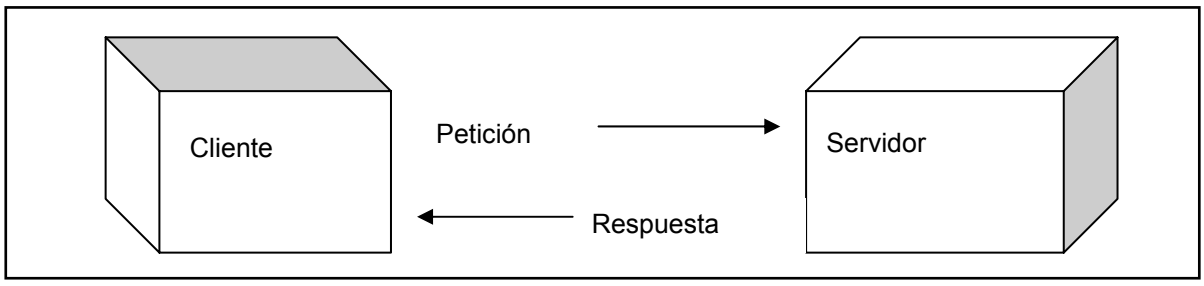


Figura 4.2: Comunicación síncrona: una pregunta bloqueante espera por una respuesta esperada.

Cuando la contestación del servidor(es) está fraccionada, el cliente se ve en la necesidad de encuestar al servidor para obtener la respuesta completa. Un ejemplo de este tipo de intercambio estaría representado por un cliente que interroga una base de datos relacional o un razonador que puede producir una secuencia de instanciaciones en respuesta a una pregunta. Aunque este intercambio requiere que el servidor mantenga algún estado interno, las transacciones individuales se tratan igual que si se tratara de una sola contestación, es decir; cada transacción es un intercambio "enviar-una-pregunta / esperar / recibir-una-contestación". Estas transacciones son como las síncronas porque los mensajes llegan al cliente sólo cuando son esperadas.

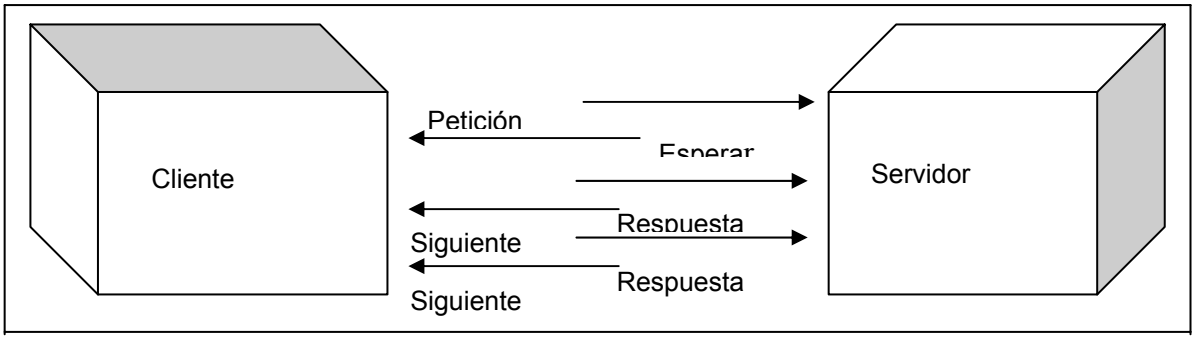


Figura 4.3: Comunicación síncrona: las respuestas se envían individualmente a cada petición del cliente.

- Comunicación asíncrona: Los sistemas de tiempo real trabajan de forma diferente, en ellos el cliente se suscribe a la salida de un servidor(es) y para que así le lleguen un número indefinido de contestaciones a intervalos regulares. En este caso, el cliente no conoce cuando llegará cada mensaje de contestación y puede estar ocupado haciendo cualquier otra tarea cuando lleguen.

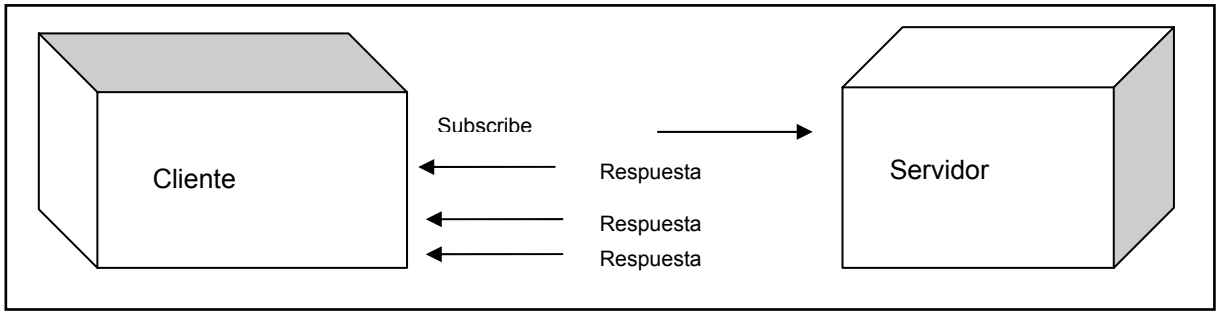


Figura 4.4: Comunicación asíncrona.

Existen más variaciones de estos protocolos. Por ejemplo, los mensajes podrían no estar dirigidos a hosts específicos, sino transmitidos a un número de ellos. Las contestaciones, llegando síncronamente o asíncronamente, se tienen que localizar y, opcionalmente, asociar con la pregunta que contestan.

#### 4.2.2.3.6 Arquitecturas del KQML

Debido a que KQML fue diseñado por un comité de representantes de diferentes proyectos (todos preocupados por la manipulación de una colección de procesos que cooperan y por la simplificación de los requerimientos de programación para implementar un sistema de este tipo) no se ha impuesto una arquitectura particular para el sistema. Más concretamente, uno de los criterios en el diseño del KQML fue producir un lenguaje que pudiese soportar una gran variedad de arquitecturas de agentes.

Finin *et al.* (1994) presentan una arquitectura de comunicación construida alrededor de dos programas especializados: un *router* y un *facilitador*, además de una *librería de rutinas de interface* llamada KRIL. La siguiente figura ilustra esta arquitectura de comunicación:

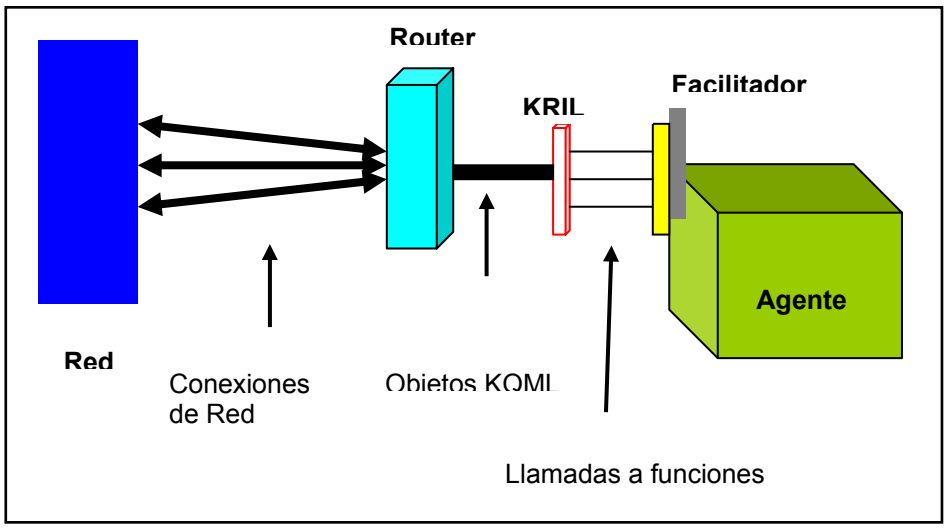


Figura 4.5: Arquitectura de comunicación.

- **Routers.** Cada agente software que "habla" KQML está asociado con su propio proceso router independiente. Todos los *routers* son idénticos siendo todos ellos una copia de uno dado. Un *router* entrega (encamina) todos los mensajes KQML que van a y desde su agente asociado. Debido a que cada programa tiene un proceso *router* asociado, no es necesario hacer extensos cambios en la organización interna del programa(s) para permitirle recibir mensajes de forma asíncrona desde una variedad de orígenes independientes.

  El *router* proporciona este servicio al agente y le suministra un solo punto de contacto con el resto de la red. Proporciona tanto al cliente como al servidor funciones para la aplicación y puede manejar múltiples conexiones simultaneas con otros agentes.

  El *router* nunca examina los campos del contenido del mensaje que encamina. Cuenta solamente con las ejecutivas de KQML y sus argumentos. Si un mensaje KQML especifica una dirección particular de Internet, el *router* dirige el mensaje a ella. Si el mensaje especifica un servicio particular por el nombre, el *router* intentará encontrar una dirección de Internet para ese servicio y le entrega el mensaje. Si el mensaje sólo proporciona una descripción del contenido (por ejemplo la pregunta, : ontology "geo-domain-3", :language "Prolog", etc) el *router* debe intentar encontrar un servidor que pueda ocuparse del mensaje y lo entregará allí, o puede remitirlo a un agente de comunicación más "inteligente" que puede hacerse cargo de él. Los *routers* pueden ser implementados con diversos grados de sofisticación.

  Finin *et al.* (1994) han usado esta arquitectura para realizar dos implementaciones del KQML: una en Common Lisp y la otra en C. Ambas son completamente interoperables y frecuentemente se usan juntas. En la implementación hecha en C, un *router* es un proceso independiente de Unix que lanza la aplicación (el agente) y por lo tanto hijo de esta. El canal de comunicación entre el *router* y la aplicación transporta mensajes KQML. Como hay un canal privado entre el *router* y la aplicación no tiene que atenerse al protocolo de KQML y puede transportar más mensajes de lo que está especificado por el protocolo formal. El *router* sólo tiene que tener en cuenta las reglas formales del KQML cuando habla al entorno. La implementación Lisp usa las primitivas de multitarea de Lucid para implementar el *router* como una tarea independiente de Lisp.

- **Facilitadores** (Finin *et al.,* 1995). Para entregar mensajes que tienen la dirección incompleta, los *routers* cuentan con los facilitadores que son aplicaciones que proporcionan servicios de red útiles. El servicio más simple que proporcionan es mantener un registro con los nombres de los servidores.

Los facilitadores ayudan a los *routers* a encontrar los *hosts*. Los facilitadores son por tanto consultores en el proceso de comunicación.

Los facilitadores son realmente agentes software de red; tienen sus propios routers KQML para mantener su tráfico y tratan exclusivamente mensajes en KQML. Típicamente existe un facilitador para cada grupo local de agentes, aunque puede haber múltiples facilitadores locales para proporcionar mayor seguridad. La base de datos del facilitador puede estar implementada en cualquier formato dependiendo del número de servidores *hosts* y de la calidad de servicio requerida. En sus comienzos la implementación de un facilitador reproducía la base de datos en cada máquina de la red local, para reducir la sobrecarga de la comunicación. Aunque esto ha sido sustituido por una implementación más centralizada para simplificar su implementación, en redes más grandes, en los que es necesario que los facilitadores sirvan múltiples redes, una implementación distribuida (análoga al servidor de nombres del dominio de Internet) puede ser más apropiada.

Cuando cada aplicación/agente se pone en marcha, su router se anuncia al facilitador local para que sea registrado en la base de datos local. De esta forma las aplicaciones se pueden encontrar unas a otras sin tener que mantener manualmente la lista de servicios locales.

- **KRILs.** Como el router es un proceso independiente de la aplicación, es necesario tener una interface de programación entre la aplicación y el router. Esta interface se llama KRIL (KQML Router Interface Library). Mientras el router es un proceso independiente, que no entiende los campos contenidos en el mensaje KQML, el KRIL está empotrado en la aplicación/agente y tiene acceso a las herramientas de la aplicación(es) para analizar su contenido. La meta general del KRIL es hacer accesos al router tan simples como sea posible para el programador.

  Con este fin, un KRIL puede estar empotrado firmemente en la aplicación, o incluso en el lenguaje de programación de aplicación(es), como es deseable. Sin embargo, no es necesario que esté completamente incluido en el lenguaje de programación de aplicación(es). Un simple KRIL para un lenguaje, generalmente proporciona dos entradas programáticas. Para iniciar una transacción hay una función "*send-kqml-message*". Ésta acepta el contenido de un mensaje y tanta información sobre el mensaje y su destino como pueda ser proporcionada. Devuelve una contestación al agente(s) remoto(s) (si la transmisión del mensaje es síncrono y el proceso se bloquea hasta que recibe una respuesta) o un simple código indicando por ejemplo que el mensaje se ha recibido.

  Para manejar la entrada de mensajes asíncronos, se utiliza normalmente la función declare-message-handler. Esta permite al programador de la aplicación declarar que funciones se pueden invocar cuando lleguen los mensajes. Dependiendo de las capacidades del KRIL, los mensajes de entrada se pueden ordenar de acuerdo a la ejecutiva, al tema o a otras características.

# 5. NEGOCIACIÓN

## 5.1 Introducción

La negociación es uno de los conceptos que más se utiliza en el campo de la inteligencia artificial distribuida Müller H. J. (1996). Algunos de los objetivos generales de los agentes dentro del campo de la negociación son: la modificación de los planes de agentes si no se alcanzan resultados adecuados y la identificación de situaciones en las que las interacciones entre agentes son posibles. Los agentes se comunican entre si de muy distintas formas con la intención de alcanzar decisiones consensuadas. La negociación sirve de forma más específica para la definición o identificación de tareas y recursos, el reconocimiento de conflictos, la resolución de objetivos dispares, la determinación de estructuras organizacionales y en definitiva para proporcionar la coherencia que requieren las sociedades de agente. Aunque la negociación es muy importante no existe una definición claramente aceptada de la misma, por la totalidad de la comunidad científica.

Para que los agentes negocien y puedan llevar a cabo procesos de coordinación y resolución de conflictos se necesario determinar que lenguaje de comunicación se utilizará, que directrices seguirá el proceso de negociación y cual es el comportamiento que se espera de los agentes. Por tanto, en función del problema que se quiera resolver se puede definir la negociación de una forma u otra.

Como lo que aquí se pretende es dar una visión general del concepto de negociación en sociedades de agentes dividiremos esta sección en tres apartados. En el primero nos centraremos en los lenguajes de comunicación entre agentes, en el segundo nos centraremos en los modelos de toma de decisión que utilizan los agentes y en el tercero en las políticas globales de negociación de las sociedades de agente.

## 5.2 Principio de negociación

La negociación se puede ver como un principio organizacional utilizado para relacionar adecuadamente tareas y métodos de resolución de problemas. Por negociación se puede entender la discusión en la que las partes interesadas intercambian información con el objetivo de alcanzar un acuerdo. En este marco, la negociación viene definida por tres aspectos importantes: (a) el flujo de información es biridecional, (b) cada negociador evalúa la información desde su propia perspectiva y (c) la decisión final se toma de mutuo acuerdo.

Esta forma de ver la negociación contiene dos elementos principales: la comunicación y la toma de decisiones. Por ejemplo, si se utiliza el "*Contract net protocol* – CNP" la negociación comienza con la oferta de un agente, a esta oferta responden el resto de agentes de la comunidad. En este protocolo el vencedor el es mejor postor. Esta forma de negociar es un tanto ineficaz y rígida al no ofrecer la posibilidad de regateo. Los agentes por tanto, no pueden discutir y ofrecer segundas opciones y la decisión final la toma el agente que hace la oferta, por ello podemos decir que este protocolo es más un método de coordinación que un principio de negociación.

Una segunda forma de negociar puede llevarse a cabo utilizando conceptos de los campos de la psicología y sociología, en este caso también son muy importantes los elementos de comunicación y de toma de decisiones, pero aquí lo fundamental es el proceso de concesiones. En definitiva, la negociación es un proceso en el que los agentes toman decisiones conjuntamente. Los agentes primero demandan y/o proponen y posteriormente modifican sus posturas iniciales mediante un proceso de concesiones o búsqueda de nuevas alternativas para alcanzar un acuerdo. Esta vía de negociación implica que los principales ingredientes de la misma son los aspectos de comunicación, de decisión y los procedurales como ya se ha comentado.

- Lenguaje de negociación: define las primitivas de comunicación que permiten la negociación, su semántica y su uso. También se centran en la definición de la estructura de los temas que se negocian.

- Toma de decisiones: la toma de decisiones en el campo de la negociación se lleva a cabo mediante algoritmos o funciones de decisión que permiten comparar las distintas opciones que se barajan durante la negociación. Las funciones de utilidad, la representación y la estructura de preferencias de los agentes se tienen que identificar antes de definir los algoritmos de toma de decisión. Las estrategias de negociación también caen dentro de esta categoría.
- Proceso de negociación: define los modelos generales del proceso de negociación y el comportamiento general de los agentes que negocian. Estos procesos implican un nivel más alto de abstracción.

### 5.3 Lenguajes de negociación

Los lenguajes de negociación facilitan la comunicación entre agentes y en ellos se pueden identificar cuatro categorías: primitivas de los lenguajes de negociación, estructura de los objetos de negociación, protocolos de negociación y semántica de los lenguajes de negociación. A continuación definimos cada una de estas categorías.

#### 5.3.1 Primitivas de los lenguajes de negociación

Una de las formas más básicas de comunicación entre agentes es el paso de mensajes (Hewitt, 1977). Claramente el poder llevar a cabo simples operaciones para enviar y recibir mensajes ofrece unas posibilidades de negociación muy débiles. Los mecanismos más avanzados de negociación utilizan aspectos relacionados con la teoría del acto del habla. Las primitivas de estos mecanismos se suelen dividir en tres grupos: de comienzo de negociación, de reacción a una propuestas y de finalización a la negociación.

- Comienzo: propuesta, organizar, petición de recursos, información, petición de información, inspección, etc.
- Reacción: respuesta, refinamiento, modificación, rehuso, explicación, etc.
- Finalización: confirmación, promesa, compromiso, aceptación, acuerdo, rechazo, etc.

Las primitivas de comienzo inician el proceso de negociación y posteriormente, durante la etapa de regateo, se emplean las primitivas de reacción para el intercambio de información entre agentes. Este proceso finaliza con la utilización de las primitivas de finalización.

#### 5.3.2 Estructura de los objetos de negociación

Durante la negociación junto con las primitivas del lenguaje de negociación se transmite un contexto especial (Von Martial, 1992). Además de la información contextual del tipo: nombre del agente que envía la información, del que la recibe, identificación del mensaje, hora de transmisión, etc. se envía otro tipo de información muy importante denominada objetos o tópicos. Los objetos o tópicos son por ejemplo planes (secuencias de acciones), intervalos de tiempos para posibles citas, ofertas o tareas, o incluso el coste y el precio de un servicio. Los objetos de la negociación son importantes ya que los agentes los utilizan para llevar a cabo cálculos locales que influyen en los resultados de la negociación. Estos objetos difieren en gran medida de una aplicación a otra y son cruciales en los procesos de negociación.

#### 5.3.3 Protocolos de negociación

Los protocolos de negociación definen, por ejemplo, las posibilidades de iniciar un ciclo de negociación o de responder a un mensaje (Kogan, 1993). Una forma sencilla de definir una posible secuencia de acciones de negociación es utilizar pares como:

(< primitiva de negociación >, <contenido del mensaje>)

En la mayoría de los casos los protocolos se pueden representar como autómatas finitos.

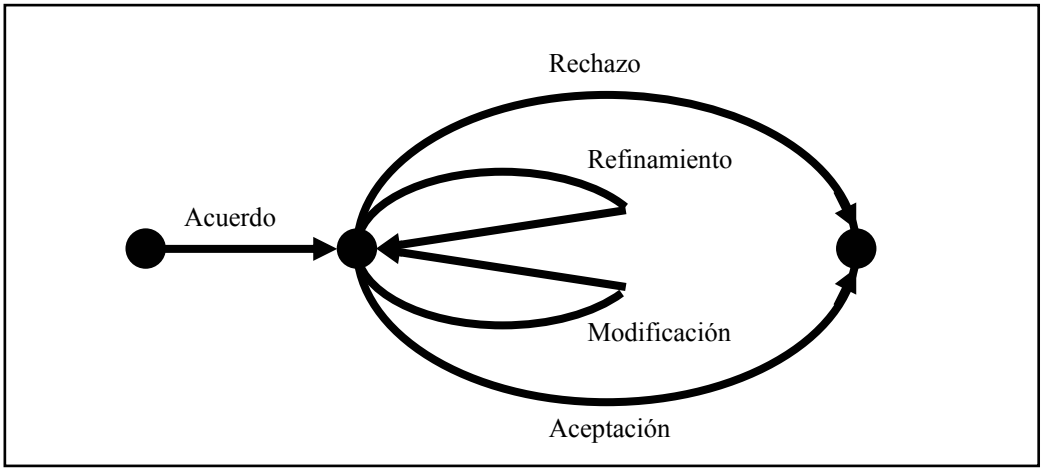


Figura 5.1: Autómata finito representando un protocolo.

Esta figura muestra un autómata finito muy simple utilizado en el entorno de la programación de citas. Indica como después de una propuesta de acuerdo se intercambian mensajes entre los agentes y se llevan a cabo una serie de medidas de refinamiento y modificación que eventualmente finalizarán con la aceptación o no de la hora y la fecha para una cita.

### 5.3.4 Semántica de la negociación

La información que se maneja durante el proceso de negociación se presenta de muy distintas maneras. Un ejemplo de como se puede presentar es el siguiente.

Rechazar (un plan x): este mensaje se utiliza cuando no se acepta un plan propuesto por otro agente.

En ocasiones se utilizan primitivas de negociación mucho más rígidas y precisas. Algunos autores utilizan semánticas de situación como base junto con operaciones modales, como se muestra en el siguiente ejemplo:

INFORM (S, R, L)
precondición:
<<cree, S,<< conoce, R, <<cree, S, a; 1>>; 0>>; 1>>
$\wedge$ <<quiere, S, << conoce, R, << cree, S, a; 1>>; 1>>
efecto:
<<habla, S, R, a; 1>> ~>
<<escucha, R, a; 1>> ➔ << conoce, R, << cree, S, a; 1 >>; 1)

De manera informal esto quiere decir lo siguiente:

> Si suponemos que un agente S cree que un agente R no sabe que S cree que la secuencia ”a” es cierta y que S quiere que R conozca que S cree que ”a” es verdadero entonces mantiene que después de que S hable con R, R conocerá que S cree que ”a” es verdadero siempre y cuando R haya escuchado que ”a” es verdadero.

Algunos autores utilizan también lógicas formales puras para definir la semántica de las primitivas de negociación.

### 5.4 Toma de decisiones durante la negociación

Aunque un agente o una comunidad de agente conozcan un lenguaje de negociación, sus protocolos, su semántica y sus objetos de negociación, no sería posible una negociación adecuada sin la definición de una estrategia de negociación (Zlotkin y Rosenschein, 1993).

Los mecanismos de toma de decisión sirven para decidir que protocolos se utilizan y que primitivas maneja de cada protocolo. Para llevar a cabo la toma de decisiones durante la negociación se pueden utilizar almenos cuatro elementos diferentes: las funciones de utilidad, las funciones de comparación y proyección, preferencias y estrategias de negociación.

#### 5.4.1 Funciones de utilidad

Las funcines de utilidad que se utilizan en el campo de los agentes y sistemas multiagente proceden del campo de la teoría de juegos (Bussman y Müller, 1993). La utilidad se expresa en la forma de matrices de decisión, que los agentes consultan para determinar el valor de una acción concreta. Dependiendo de la estrategia a seguir el agente realiza la acción que represente el mayor o el menor valor. En el contexto de la negociación, las funciones de utilidad representan el precio o coste de las actividades.

Los agentes intentan optimizar sus actividades a lo largo del tiempo y tratan de maximizar su utilidad total. El proceso de negociación está fuertemente ligado a los objetivos internos de los agentes de maximizar sus utilidades. Los precios, costes y valores deben ser los objetivos de la negociación, pero las decisiones se deben tomar teniendo en cuenta que el agente siempre pretende optimizar su función de utilidad.

#### 5.4.2 Funciones de Comparación y Proyección

El término “negociación” se utiliza en problemas de resolución de conflictos y reparto de tareas. En estos casos, los agentes toman sus decisiones basándose en complejos “objetos de negociación” como por ejemplo planes. Una vez se evaluan los planes la decisión final se toma teniendo en cuenta cálculos de utilidad. Para ello hay que calcular funciones que permitan comparar la utilidad de diferentes objetos.

#### 5.4.3 Preferencias

Los agentes negociadores en ocasiones tienen expectaciones acerca del resultado final de la negociación. En este caso pueden preferir una solución más que otra, lo que les dota de un criterio de selección. Estas preferencias se pueden expresar de forma relativa o absoluta. Por ejemplo: una propiedad concreta (predicado) está presente (verdadero), o es menos que, igual que, mayor que un valor concreto. Las preferencias se pueden expresar como fórmulas de primer orden:

Visitar (Pedro) V (Hoy (Sábado) ir (cine))

Lo cual quiere decir que el agente prefiere tanto visitar a Pedro como, si es sábado, ir al cine. Si las estrategias de los agentes oponentes son conocidas, se pueden aplicar una serie de estrategias para proponer soluciones del gusto de todas.

#### 5.4.4 Estrategias de negociación

En el campo de la psicología y la sociología son muy importantes las estrategias de negociación y los comportamientos en la negociación. Por tanto no es de extrañar que elementos de estas dos disciplinas

tengan influencia en las estrategias de negociación entre agentes artificiales. Entre otras se pueden distinguir cinco estrategias de negociación:

1. Acuerdo unilateral.
2. Comienzo competitivo: manteniendo una postura firme y utilizando técnicas de presión.
3. Comienzo cooperativo: buscando soluciones a base de acuerdos.
4. Impasible.
5. Ruptura.

El acuerdo unilateral no facilita el que se alcance un acuerdo favorable entre todos los participantes. Sólo es interesante en el caso de que la negociación pierda su sentido o sea superflua para el agente. Parece claro que la impasibidad o la ruptura no favorece el proceso de negociación. Los comportamientos cooperativos y los competitivos son los extremos del espectro de posibilidades que ofrece la negociación. Durante el comienzo de la negociación un conflicto se puede resolver siguiendo una de las siguientes estrategias:

1. Seleccionar las opciones de una en una.
2. Seleccionar la opción más importante.
3. Seleccionar la menos diferente.
4. Reducir todos los criterios de selección a uno (honor, dinero, etc).
5. Empaquetar, por ejemplo identificar dos o más criterios como equivalentes en valor.
6. Etc.

La táctica más adecuada, de entre las anteriores, no se puede determinar de antemano y a menudo depende de la situación en la que el agente se encuentre. En situaciones en las que la negociación tienda a fallar, el agente puede seleccionar la opción que más interese a todos para tratar de reavivarla. La reducción de todos los objetos puede no ser posible y el empaquetamiento puede ser una opción. Una táctica habitual es la de seleccionar los objetos por orden de importancia y negociar uno por uno.

### 5.5 El proceso de negociación

Los lenguajes de negociación facilitan la comunicación entre agentes. Definen como se debe establecer la comunicación cuando aparecen situaciones conflictivas. La toma de decisiones durante la negociación sirve para controlar la marcha de la negociación. Determina como encontrar las palabras adecuadas durante la comunicación. El proceso de negociación complementa los anteriores aspectos de la comunicación y define un metanivel que permite tener una visión totalitaria del proceso de negociación.

El propósito de este componente de la negociación es el de describir el proceso total, analizándolo desde el punto de vista de las sociedades de agentes.

El modelo de negociación procedural define el comportamiento de un agente durante el proceso de negociación. Se puede definir de forma abstracta utilizando nociones de estrategias y preferencias o puede venir definido por algoritmos, que básicamente describen los niveles de decisión, los planes, la base del conocimiento, etc.

Es necesario hacer un análisis exhaustivo del problema y del sistema que se pretende describir para así determinar el comportamiento de la sociedad de agentes desde el punto de vista de la negociación. Por ejemplo es siempre interesante estudiar la eficacia de los procesos de resolución de problemas utilizados por agentes y compararlos con los utilizados en el campo de la informática centralizada.

Existen numerosos modelos de negociación. Por ejemplo un algoritmo de negociación se puede presentar como un modelo con dos niveles. El ciclo de negociación puede comenzar proponiendo una solución a un conflicto determinado, con el análisis crítico de otras posturas teniendo en cuenta

nuestras propias preferencias y con la actualización del espacio de soluciones y de nuestras preferencias. En paralelo a este ciclo se ejecuta otro en el que se intentan solucionar conflictos, para ello primero hay que encontrarlos y después seleccionar aquellos que requieran mayor prioridad planificando posibles soluciones.

Otro modelo similar consistiría en definir un conjunto de estados que representan el estado de la negociación y en especial la calidad de las soluciones propuestas. Los estados se cambian aplicando una serie de operaciones de inicialización, crítica, relajación de soluciones y de finalización.

### 5.5.1 Análisis y comportamiento del sistema

Dentro de esta categoría se incluye el estudio de aspectos socio/psicológicos de agentes y sociedades de agentes como por ejemplo la "Justicia". Esta forma de describir las sociedades de agente implica un nivel mayor de abstracción y complejidad. El análisis de estos sistemas permite entre otras cosas determinar si esta forma de concebir sistemas informáticos facilita el trabajo de los analistas y programadores. Varios estudios han demostrado que la construcción de sociedades de agentes en las que se aplican los principios de la negociación requiere el mismo esfuerzo que construir sistemas coordinados centralizados on-line y off-line.

Un sistema coordinado y centralizado on-line es aquel en el que los agentes deciden quien realiza que tarea según le van llegando, con la condición de que cada tarea se complete antes de que llegue la siguiente y off-line quiere decir que todas la tareas se conocen a priori y que las tareas se asignan a los agentes en un orden óptimo.

# 6. APRENDIZAJE Y RAZONAMIENTO

## 6.1 Introducción

El aprendizaje en sistemas multiagente es un campo de gran diversidad muy relacionado con otras áreas de investigación. En esta sección se describen las tres ramas básicas del razonamiento y aprendizaje en sistemas multiagentes partiendo de la surgida en primer lugar, en la cual los agentes aprenden por programación; pasando por las nuevas técnicas de inteligencia artificial, que permiten al agente aprender directamente del entorno; para terminar con los enfoques que permiten al agente aprender por adaptación de su código utilizando tanto la experiencia almacenada en su interior como el entorno en el que habita.

Como ya se indicó, el aprendizaje en sistemas multiagente está relacionado con muchas áreas de investigación. Esto ha llevado a la creación de una gran variedad de modelos de razonamiento y aprendizaje (Maes *et al.*, 1993). Se han diseñado tanto agentes como sistemas de agentes que pueden aprender en base a órdenes, a ejemplos, por observación, gracias a la comunicación con otros agentes, por refuerzo, por clasificación de información, por construcción de modelos y, también, por formas especializadas de comparación (Woolridge *et al.*, 1994). Hoy en día, es más frecuente el desarrollo de agentes que utilizan técnicas de inteligencia artificial, como clasificadores simbólicos, redes de neuronas, razonamiento basado en casos, algoritmos genéticos y algoritmos neuronales-difusos (Nwana, 1996).

Esta sección presenta una revisión del estado del arte con respecto a los mecanismos de razonamiento y aprendizaje que utilizan los agentes y los sistemas multiagentes. En un principio se habla de los agentes que hay que programar cuando es necesario que adquieran nuevo conocimiento. Posteriormente se presentan mecanismos clásicos que facilitan la adaptación del agente al medio que les rodea y la adquisición semiautomática de conocimiento. Para finalizar se presentan mecanismos de aprendizaje que utilizan nuevas metodologías de intelgencia artificial.

## 6.2 Aprendizaje por programación

El método más simple y primitivo para hacer que un agente aprenda y razone es mediante la modificación de su conocimiento de forma manual (Maes, 1989). Para ello, cada vez que el agente o sistema multiagente requiere una adaptación al medio, simplemente se altera su código para introducirle la nueva información. Para dotar a los agentes de este tipo de aprendizaje de bajo nivel es necesario utilizar sistemas basados en reglas, métodos de "planning", métodos de conocimiento de dominios y sistemas que permitan la programación del agente en tiempo real.

### 6.2.1 Sistemas basados en reglas

Los agentes que utilizan sistemas basados en reglas se apoyan en un método que implica la utilización de construcciones del tipo if-then (Maes, 1989). Este tipo de agentes es adecuado para entornos donde las reglas no cambian frecuentemente y donde no se necesitan demasiadas reglas. Para adaptar un agente basado en reglas puro, el programador debe cambiar el conjunto de reglas que definen el comportamiento del sistema. Esto implica que los agentes que utilizan reglas no manejan la incertidumbre del entorno demasiado bien. De todos modos, esta forma de diseñar agentes puede combinarse de manera efectiva con otros métodos de aprendizaje con efectos muy positivos. IBM ha creado un agente que se utiliza como asistente para médicos, por ejemplo, y que usa un proceso basado en reglas para verificar las interacciones conocidas entre drogas.

### 6.2.2 Planificación

Los agentes que utilizan métodos basados en planificación (*planning*) son similares a los que utilizan métodos basados en reglas, en el sentido de que siguen un cierto orden de ejecución definido a priori.

También en este caso, su rendimiento mejora en combinación con otros métodos de aprendizaje (Plunkett *et al.*, 1999). Kaebling y Rosenschein combinan la programación embebida o empotrada (consciente de su entorno y reactiva ante él) con técnicas de "planning" en el diseño de agentes (Kaebling *et al.*, 1991).

### 6.2.3 Conocimiento del dominio

Otro enfoque seguido se basa en la idea de pre-programar al agente con conocimiento exhaustivo de su entorno o dominio. "Cyc" es un gran esfuerzo para proporcionar al agente "conocimiento de sentido común" por el método de introducir en él información sobre estrategias de toma de decisiones humanas, emociones, gusto por el arte, historia, literatura, actualidad, etc (CYCORP). Aunque se trata, en cierto sentido de un enfoque holístico, puede ser peligroso crear agentes cuya base es el conocimiento que del dominio tienen las personas que los programan con unas opiniones sociales o raciales pre-definidas.

### 6.2.4 Programación del usuario final

Este método implica la programación por parte del usuario final de lo que quiere que haga el agente. Aunque normalmente las interfaces se diseñan para personas que no tienen conocimientos sobre programación, este es un método bastante complejo para la mayoría de los usuarios. De todos modos, el permitir que los usuarios finales programen los agentes ha llevado a la obtención de grandes éxitos especialmente cuando se combina con otras técnicas (Maes *et al.*, 1993).

## 6.3 Razonamiento y aprendizaje a partir del entorno y la experiencia

Cuando se estudia la forma de diseñar agentes con capacidad de aprendizaje surgen diversas cuestiones. Cuando se considera el aprendizaje a partir de la experiencia, es necesario preguntarse: ¿cómo puede un agente mejorar su rendimiento utilizando las experiencias acumuladas en un cierto período de tiempo?, ¿cómo decide cuando debe ejecutar la mejor opción de las disponibles en ese momento en lugar de continuar la búsqueda de métodos mejores para alcanzar sus objetivos?, ¿cómo se puede integrar la experiencia externa en su tarea interna de ejecución de módulos?, ¿cómo puede corregir y mejorar sus comportamientos no efectivos? (Corchado, 1997).

Estas cuestiones conducen al problema de la selección de acciones y a una lista de preguntas aún mayor: ¿Cómo puede el agente decidir cual es la mejor acción para realizar a continuación?. ¿Cómo puede manejar las situaciones inesperadas o las crisis?. ¿Cómo decide entre objetivos contradictorios?. ¿Cómo se diseña para que maneje entradas con ruido o incertidumbre?. ¿Cómo puede responder dentro de un rango temporal oportuno? (Maes *et al.*, 1993).

Recientemente, ciertos investigadores que trabajan en el diseño de agentes han incorporado un enfoque más holístico, para intentar resolver las cuestiones antes planteadas (Corchado *et al.*, 1997). La investigación en agentes autónomos adaptativos se ha basado fundamentalmente en dos "principios-guía":

1. El hecho de conocer el agente y el entorno en el que habita, cambia la dinámica del problema favorablemente.
2. La dinámica de la interacción entre el agente y su entorno implica un aumento del grado de complejidad.

Al tomar un enfoque holístico, un agente necesita una menor carga de planning y razonamiento complejo, puesto que puede aprender de su entorno realizando tests con sus sensores y actuadores. Así, en lugar de construir un modelo interno grande y detallado para manejar cada situación posible el agente puede usar el "mundo que conoce para crear su propio modelo" (Brooks, 1991). Además, cuando un Agente evoluciona a lo largo del tiempo puede mejorar su rendimiento y aprender de la experiencia. Finalmente, cuando el Agente se diseña siguiendo un enfoque holístico, puede considerar todos los recursos disponibles en su entorno, incluidos usuarios y otros Agentes.

La capacidad de aprender a partir de la experiencia debería ser una propiedad de los agentes que pretendan mantener un grado de robustez y autonomía a lo largo de un período de tiempo significativo. Como los entornos del mundo real cambian continuamente y de forma impredecible, el aprendizaje debe ser un proceso dinámico y continuo. Los agentes necesitan aprender de cada experiencia. Esto significa que han de tener la capacidad de aprender y realizar sus tareas concurrentemente. De todos modos, el aprendizaje debe ser específico en función de los objetivos del agente y capaz de hacer frente a tecnología con ruido o fallos. Idealmente, el aprendizaje debería ser no supervisado. A los agentes hay que dotarlos de unos conocimientos mínimos, para que a partir de ellos pueda aprender de manera rápida y eficaz (Corchado, 1997).

A continuación, se incluyen varios modelos generales de aprendizaje de agentes seguidos de una serie de métodos que incorporan mecanismos capaces de aprender a partir de la interacción con su entorno o de la experiencia adquirida por el agente.

### 6.3.1 Programación por ejemplos

El ejemplo clásico de esta aproximación es KidSim, una herramienta diseñada para enseñar a niños las construcciones básicas de programación (Smith *et al.*, 1994). En un mundo bidimensional, los niños pueden crear caracteres usando un editor gráfico que ilustra los cambios en el estado de los programas. El sistema obtiene reglas abstractas (o código de programa) a partir de los objetos gráficos creados. Los usuarios pueden además cambiar ciertos estados programando el sistema, una vez que están familiarizados con él.

### 6.3.2 Observación del usuario

Este tipo de aprendizaje se denomina también aprendizaje por imitación, e implica que el agente descubra patrones repetitivos en el comportamiento del usuario. Cuando el agente ha visto una serie de patrones las veces suficientes (este número puede fijarse como parámetro), puede sugerir al usuario realizar la tarea en su lugar. Este es el método utilizado por el agente MAXIMS de Maes (1995).

### 6.3.3 Cooperación con otros agentes

Esta técnica se utiliza en sistemas Multiagentes y conlleva la construcción de agentes que aunque no conozcan la solución a un problema particular, tienen la opción de hacer una interrogación general a otros agentes que creen que pueden tener la solución. Por ejemplo, un agente dedicado al control del correo electrónico puede preguntar a otros agentes cómo clasificar un mensaje en particular. Goldman y Rosenschein (1996) usan una extensión de este método en el diseño de un sistema Multiagente donde cada Agente actúa como maestro de sus compañeros en el esfuerzo de agilizar el proceso de coordinación requerido para resolver un problema.

### 6.3.4 Refuerzo

En este tipo de aprendizaje, al agente se le proporcionan incentivos numéricos cuando toma una decisión positiva como respuesta a la influencia del entorno. El agente trata de maximizar estas recompensas numéricas y con el tiempo mejora su rendimiento. Las interacciones entre el agente y el entorno ocurren en una secuencia de puntos temporales discretos. En cada momento, el agente recibe información del estado del entorno y selecciona la acción apropiada (Sutton *et al.*, 1998).

### 6.3.5 Sistemas de clasificación

Los sistemas de clasificación han sido descritos como “mecanismos de aprendizaje basados en algoritmos genéticos” (Sen *et al.*, 1994). De todos modos, puesto que se requiere que el agente aprenda de su entorno más próximo, se incluye en esta sección. Los sistemas de clasificación se construyen sobre árboles de decisión para aprendizaje y son un caso especial de sistemas de

aprendizaje por refuerzo. Como en el caso de aprendizaje por refuerzo, el agente intenta incrementar la recompensa que recibe por realizar la mejor acción en cada situación. Sin embargo, en el sistema de aprendizaje por clasificación, el agente tiene una serie de reglas (clasificadoras) para cada situación de la que tiene datos de rendimiento. También se almacena en el sistema de información sobre el "peso" de cada clasificadora. Cuando el agente realiza una acción considerada positiva y es premiado, esta recompensa incrementará el peso de las clasificadoras que han llevado a esa acción. Este proceso asegura que, con tiempo, las clasificadoras que más a menudo contribuyen a una acción exitosa serán premiadas. Este tipo de aprendizaje se ha utilizado con éxito para coordinar sistemas Multiagentes (Sen *et al.*, 1994).

### 6.3.6 Construcción de modelos

En este método el agente desarrolla un modelo probabilístico en el que se incluyen los efectos que se producirían al realizar determinadas acciones en ciertas situaciones. Este modelo se puede utilizar para decidir qué acciones son las mejores en determinadas circunstancias y dependiendo del objetivo a alcanzar. Una gran ventaja de los agentes constructores de modelos es que son capaces de transferir los modelos creados en un contexto particular a otro contexto diferente, usando, por tanto, los modelos construidos previamente como guía para el logro de otros objetivos. Como resultado de ello, los agentes con este método de aprendizaje tienen unos rendimientos mejores en entornos donde los objetivos cambian con el tiempo. Este tipo de agente puede ser construido con un mínimo de conocimiento de base, incluyendo inicialmente algún modelo causal. El agente puede posteriormente corregir este modelo de estimación si es necesario.

### 6.3.7 Filtrado colaborativo

El filtrado colaborativo se conoce también como filtrado social. Es un método que ha sido aplicado con un porcentaje de éxito variable. Depende de una única técnica: "agrupar gran número de usuarios para obtener correlaciones significativas entre ellos". (Página de Agentes Inteligentes de IBM). Se basa en el principio de que las personas que tienen similares características (como vivir en un adosado, beber licores y leer libros) tendrán también preferencias similares en otras áreas. "Waldo the WebWizard's LifestyleFinder" es un agente que emplea esta técnica. Al usuario se le hacen una serie de preguntas sobre sus hábitos, las respuestas a esas preguntas se comparan con una base de datos de usuarios previos de esa categoría. Desgraciadamente, las categorías posibles son bastante restringidas y el agente utiliza poca información como para realizar la clasificación con un alto grado de precisión. La base de datos de usuarios encuestados, usada luego para la comparación es todavía demasiado pequeña para incluir los gustos menos típicos. Esta técnica ha sido usada con gran éxito para establecer servicios de medios de comunicación por petición (Nygren *et al.*, 1996). La combinación de esta técnica con otras formas de aprendizaje, como redes neuronales, permite la producción de resultados más precisos.

## 6.4 Nuevas técnicas de inteligencia artificial

Aunque las técnicas de aprendizaje han sido objeto de estudio en el campo de la inteligencia artificia desde antes del 1950, se han aplicado a la tecnología de agentes software, sólo en fechas muy recientes. De todos modos, su relativa novedad en el campo no resta valor a su potencial y éxito. Las redes neuronales, el razonamiento basado en casos, los algoritmos genéticos y los algoritmos difusos han realizado aportaciones importantes al diseño de agentes software.

### 6.4.1 Redes neuronales

Las redes neuronales son un campo de la inteligencia artificia que se basa en los conceptos básicos de la biología humana. El sistema neuronal del cerebro humano tiene una gran capacidad para procesar información (Hayken *et al.*, 1994).

Existen varias formas de lograr el aprendizaje en redes de neuronas artificiales; dos de las más populares en Agentes Software son las redes basadas en el modelado Hebian (Hayken *et al.*, 1994) y las ART (Cayglaayan *et al.*, 1996). El aprendizaje de Hebian se basa en el aumento de los pesos entre las neuronas de entrada y las de salida que han tenido éxito, es decir, "favoreciendo al fuerte". Esto significa que si los pesos entre una neurona de entrada y otra de salida son ya altos, tienen una mayor probabilidad de crecer todavía más. Las redes ART, por otro lado, usan una variante del modelo competitivo. Estas redes utilizan un algoritmo que permite tanto determinar si las neuronas están trabajando en su punto óptimo como definir un método que recompensa a las neuronas con pesos más altos (Fyfe, 1996).

Las redes neuronales artificiales son robustas, tolerantes a fallos por su diseño paralelo, flexibles y adaptables y pueden manejar datos difusos, probabilísticos, con ruido e inconsistentes. Por lo tanto, son modelos adecuados para usarse en problemas con un cierto grado de incertidumbre. Se pueden utilizar en problemas de clasificación de patrones, optimización, compresión de datos, aproximación, asociación, predicción, etc. (Fyfe, 1996). De este modo, las redes de neuronas artificiales son muy útiles para cualquier Agente Software que se diseñe para realizar cualquiera de las funciones antes citadas.

Las redes de neuronas artificiales se usan en varios Agentes Software. Canamero (1997) las utiliza como sistemas de ayuda, que permiten a entes autónomos aprender y desarrollarse. Los agentes de reconocimiento en una criatura artificial utilizan redes neuronales de aprendizaje rápido (ART) para reconocer importantes características del entorno. La aplicación "Creatures", disponible comercialmente, utilizan redes neuronales con aprendizaje Hebian para permitir que las criaturas se adapten a lo largo de su vida (Grand *et al.*, 1997). También el agente Open Sesame User Interface Learning adquiere su conocimiento sobre el usuario utilizando una variante del reconocimiento de patrones de las redes de neuronas ART (Cayglaayan *et al.*, 1996).

### 6.4.2 Razonamiento basado en casos

El razonamiento basado en casos (*Case Based Reasoning*) es un método relativamente nuevo que encuentra soluciones a los problemas reutilizando información almacenada y conocimiento obtenido en situaciones similares previas. En esencia, permite resolver nuevos problemas mediante la adaptación de soluciones que ya habían sido utilizadas con éxito para solucionar problemas anteriores. Técnicamente un sistema CBR analiza un problema, utiliza algoritmos de indexación para recuperar casos almacenados previamente y adapta los casos recuperados para solucionar nuevas situaciones. Los sistemas CBR no necesitan un modelo exacto de su entorno, sólo adquirir un número razonable de casos históricos.

Los sistemas CBR tienen muchas aplicaciones en los agentes software, se pueden utilizar en interfaces adaptables, en negociación entre agentes y en emparejamiento. El grupo de Dublín Review (Green *et al.*, 1997) sugiere la utilización de un CBR para un agente con Interface adaptable a múltiples usuarios en el que las clases de usuarios pueden ser consideradas casos generales. Cada usuario nuevo se incluiría en un grupo o caso y la respuesta del agente se adaptaría posteriormente de forma adecuada.

Un sistema multiagente ha utilizado con éxito un sistema CBR para reducir el coste de comunicación asociado con la emisión de avisos a grupos de agentes. Con la utilización del CBR, el sistema multiagente es capaz de extraer conocimiento de los mensajes y seleccionar un robot capaz de solucionar una tarea particular (Ohko *et al.*, 1996).

Además, los CBR han sido utilizados en el diseño de Sistemas Multiagente para identificar el supervisor más adecuado para un estudiante durante su proyecto de Máster. En este sistema, cuando un agente CBR recibe datos relativos a un nuevo estudiante (un nuevo caso), se comparan con casos previamente almacenados; el sistema recupera el caso más similar y devuelve el nombre del supervisor más adecuado para el estudiante (Corchado y Lees, 1997).

### 6.4.3 Razonamiento basado en memoria

El razonamiento basado en memoria (MBR - *Memory Based Reasoning*) es una variante directa de los sistemas CBR. Mientras en el segundo caso se guardan todos los datos como casos, cada uno de los cuales se representa por un registro separado, en los MBR se almacenan sólo las relaciones entre los atributos de los casos. Esto supone una mejora, porque se necesita menos espacio de almacenamiento y un menor tiempo de comparación entre los casos. Los MBR hacen posible la utilización de algoritmos como el de los "K vecinos más próximos". El agente de IBM *Memory Agent* utiliza este tipo de aprendizaje para descubrir las preferencias personales de los usuarios de aplicaciones de negocios (página de IBM de agentes inteligentes).

Un Agente de Colaboración denominado Firfly utiliza también los sistemas MBR. En este caso el agente encuentra grupos de patrones para construir un álbum y facilitar recomendaciones al artista, basándose en grupos de evaluación de música en CDs. De este modo, los agentes software, que incorporan el uso de sistemas MBR, son capaces de proporcionar recomendaciones similares a las sugerencias de humanos.

### 6.4.4 Algoritmos genéticos

Los algoritmos genéticos centran su estrategia de solución de problemas en las características de la evolución natural. Específicamente, se utiliza la teoría de Darwing sobre la supervivencia del más fuerte que propugna que los individuos que mejor se adaptan de una sociedad tienen una probabilidad mayor de sobrevivir y reproducirse. Incluso, una especie completa puede hacerse más fuerte cuando sus genes más débiles se eliminan y los fuertes prevalecen.

Los agentes software utilizan los algoritmos genéticos para crear un esquema de codificación capaz de decidir si el agente es adecuado para solucionar un problema o no. Posteriormente, se crea una población de agentes codificados de modo idéntico, que se evalúa a medida que se desarrolla. Los agentes fuertes (genes) pueden reproducirse y mutar, y gradualmente el sistema global de agentes se hace más fuerte a medida que los Agentes débiles terminan. Los algoritmos genéticos proporcionan las mismas ventajas que las redes neuronales artificiales: robustez y adaptabilidad, sin embargo, el proceso de adaptación suele llevar un tiempo mayor (Fyfe, 1996).

Los algoritmos genéticos se utilizan cada vez más en el desarrollo de agentes. Dos de los ejemplos más recientes se dan en el ámbito de los agentes de información y en el movimiento de imágenes.

PAWS es un sistema multiagente "*Personal Adaptive Web Sentenial*" que actualiza automáticamente una librería Web de información personal (Falk *et al.*, 1996). Usa los algoritmos genéticos para implementar de forma adaptativa la búsqueda de palabras clave y los módulos de filtrado.

Una estrategia evolutiva se utiliza también en un ente-agente con 6 piernas para optimizar la búsqueda de la posición de sus articulaciones mientras camina. La experimentación posterior concluyó que los agentes autónomos que utilizaban este método eran robustos y eficientes al buscar el mejor movimiento durante la ejecución (Liu *et al.*, 1997).

### 6.4.5 Lógica difusa

La lógica difusa trata la complejidad del mundo real de un modo relativamente sencillo, computacionalmente hablando, facilitando la transformación de conceptos difusos en reglas. Los algoritmos difusos son especialmente útiles en situaciones donde la información es incompleta o ambigua.

Los diseñadores de agentes software han comenzado recientemente a utilizar este tipo de algoritmos. Un ejemplo de su uso es Camargo Silva, que ha desarrollado un modelo para el diseño de agentes

usando técnicas de memoria asociativa difusa (Silva, 1996). Una Memoria Difusa asociativa (*Fuzzy Asociative Memory*, FAM) es una matriz en la que se guardan una serie de reglas que representan todas las combinaciones de entradas. Las entradas al sistema difuso se pueden evaluar usando la FAM para determinar qué reglas son verdaderas. A medida que se incrementa el número de entradas, la potencia de las reglas resultantes también se hace mayor. En el diseño de Silva, la interface de aprendizaje del agente almacena los datos de acuerdo con la experiencia con un usuario y utiliza la FAM para realizar prediciones sobre futuras acciones del usuario.

# 7. AGENTES Y OBJETOS

## 7.1 Introducción

La programación orientada a agentes (POA) puede considerarse una extensión o especialización de la programación orientada a objetos. Un sistema orientado a objetos está formado por un conjunto de objetos con estado y comportamiento definidos, que interactúan entre sí a través de mensajes. La programación orientada a agentes supone una extensión de este enfoque ya que cada agente está compuesto por estados "mentales" como creencias, capacidades, deseos, intenciones, decisiones, etc.

La Programación orientada a objetos (POO) utiliza conceptos como clase e instancia, y basa el desarrollo de aplicaciones en establecer relaciones entre clases: herencia, agregación y asociación. Estos elementos son la base del éxito de la POO en el modelado conceptual y la reutilización por ejemplo, de manera que parte de la comunidad científica que trabaja en este campo piensa que dichos conceptos se pueden exportar al ámbito de la agencia.

Tanto las herramientas de desarrollo utilizadas en el campo de la POO como las técnicas de modelado y metodologías, y los conceptos de clases, instancias, herencia, agregación, etc. podrían facilitar la construcción de agentes. En el seno de la comunidad científica preocupada por este problema se han propuesto numerosos puntos de vista y aunque no se ha alcanzado ningún acuerdo parece claro que puede ser positivo el presentar a los agentes como una extensión de los objetos.

Los agentes presentan un mayor grado de abstracción que los objetos y los modelos conceptuales que con ellos se crean presentan una mayor complejidad que los modelos utilizados en la POO. Una de las principales ventajas de ver la POA como una extensión de la POA es que la primera proporciona soporte directo para los principales mecanismos de abstracción: clasificación (instaciación), agregación (descomposición), generalización (especialización) y agrupación (individualización) y por tanto la segunda puede hacer suyos dichos mecanismos. Cada uno de estos principios está bien definido y tiene una utilidad determinada dentro del modelo de objetos. Aunque la POA puede aprovecharse de estos mecanismos en algún caso será necesario hacerlo desde una perspectiva diferente, como se verá en las siguientes secciones.

## 7.2 Casos y agentes

La clasificación es un principio de abstracción fundamental en el campo de la POO. Consiste en agrupar elementos que satisfacen ciertas condiciones. Cada clase es la abstracción de un elemento o concepto y normalmente contiene elementos que la hacen diferentes del resto. El principio inverso a la clasificación es la instanciación. Al instanciar una clase se generan elementos concretos que satisfacen la descripción intencional de su clase.

La práctica totalidad de los lenguajes de POO soportan estos dos tipos de abstracción, facilitando la construcción y definición de clases e instancias. Una clase define la estructura y comportamiento de un conjunto de objetos. Es una plantilla en la que se describen los atributos y métodos que compartirán sus instancias. Cada objeto concreto creado a partir de una clase, tiene en cada instante un estado propio definido por el valor de sus atributos. Los objetos se instancian a partir de las clases y una vez creados, no pueden cambiar de clase.

En el modelo de agentes la clasificación/instanciación se soporta a través de la definición de clases y de instancias de agentes. Una clase de agentes define la estructura y el comportamiento de un conjunto de agentes, teniendo en cuenta que cada agente particular va a estar formado por un conjunto de componentes mentales. Un sistema orientado a agentes debe dar soporte a la definición de clases de agentes y a la creación de instancias particulares de agentes. Los conceptos de clase e instancia se han

utilizado en varios modelos de agentes (Kinny *et al.*, 1996; Kendall *et al*., 1997; Iglesias *et al*., 1998; DeLoach, 1999; Odell, 1999; Wooldridge *et al*., 1999; Wagner, 2000).

En general el concepto de clase de agente está relacionado con el de role. Por ejemplo las propuestas de Kinny *et al*. (1996) y Wooldridge *et al*. (1999) parten de la descomposición del problema en base a los roles claves en una aplicación. Para el primero el proceso comienza con la identificación de roles y sus relaciones. A partir de ellos se define la jerarquía de clases de agentes. Los agentes concretos son instanciaciones de las clases identificadas. Cada agente o grupo de agentes se identifica por tanto con un role.

La diferencia entre la POO y la POA aparece a nivel de interfaz. Cada agente tiene capacidad para decidir si realizar o no una tarea que le ha sido encomendada por otro agente. Un objeto no tiene esta capacidad. Los *roles* son los bloques que identifican clases de agentes y capturan los objetivos del sistema durante la fase de diseño. En este sentido podemos asegurar que un sistema cumplirá con su cometido si se identifican todos los objetivos del mismo, se asocia un role a cada uno de ellos y se diseña un agente para cada role. En las sociedades de agente, las relaciones entre ellos se pueden representar por las conversaciones que mantienen, por ello al final de la etapa de diseño se debe generar un diagrama de clases de agente que represente las clases y los posibles diálogos entre ellas (Wood y DeLoach, 2000).

Un agente podría representar varios roles al mismo tiempo o cambiar de role durante su ejecución. Aunque la capacidad de representar varios roles y cambiar de role no es una característica que aparezca en la mayoría de los lenguajes de POO, si se contempla esta posibilidad. Sin embargo, en el caso de la POA sería una característica muy deseable que proporcionaría un mayor grado de autonomía y flexibilidad a los sistemas desarrollados (Odell, 1999).

### 7.3 Objetos y agentes complejos

La agregación es el principio de abstracción que permite definir colecciones de conceptos como elementos simples de un nivel superior. La agregación permite describir conceptos en función de sus partes. Una parte puede visualizarse como un todo en si mismo. Por ejemplo, los lenguajes de POO permiten que un objeto pueda utilizarse como una variable de otro objeto. De esta forma se crean los objetos complejos o agregados, que son aquellos que están compuestos por otros objetos. Cada uno de los componentes de un objeto complejo tiene su propio estado y comportamiento, y en conjunto determinan el estado y comportamiento del objeto complejo. La operación inversa a la agregación es la descomposición que produce los componentes individuales de una agregación.

Este tipo de abstracción se puede utilizar en el ámbito de los agentes. De forma que se puedan identificar clases de agentes compuestas a partir de otras clases más primitivas. Así se generarían agentes capaces de solucionar problemas complejos a partir de sus componentes: agentes más sencillos. En este caso, las relaciones entre agentes complejos y sus agentes componentes son más complejas y requieren un mayor nivel de abstracción que en el caso de los objetos.

Los agentes en general tienen un alto grado de autonomía y sentido de privacidad, por ellos tienen limitada su capacidad de compartir las bases de conocimiento. La agregación se puede utilizar para crear agentes BDI complejos a partir de subagentes BDI más simples de manera que cada subagente disponga de sus propias creencias, objetivos y planes propios y no tenga acceso a las componentes mentales del resto de los subagentes. La arquitectura (deliberativa, reactiva, etc.) que define cada agente o el conjunto de agentes marcará de manera sustancial la relación entre ellos.

La agregación, en el modelo de agencia, facilita la creación de agentes a partir de otros que lleven a cabo tareas concretas. Los subagentes pueden actuar en representación del agente complejo de manera que cada uno de ellos tenga asociado una serie de compromisos que le son transferidos por delegación.

De esta manera cada subagente estaría capacitado para reaccionar ante determinados estímulos llevando a cabo determinadas acciones.

### 7.4 Herencia y Agregación

La generalización permite definir nuevos conceptos a partir de otros que comparten ciertas similitudes. La generalización permite identificar clases generales capaces de capturar las similitudes y suprimir parte de las diferencias de un número de clases.

Un concepto X es una especialización de otro concepto Y, si los componentes que forman parte de X pertenecen a Y. Aunque Y y X son similares, X puede contener elementos adicionales, que de alguna manera lo diferencien de Y. En el caso de la POO podríamos decir que si X e Y son objetos, X se habría heredado de Y. Por tanto la especialización es el equivalente de alto nivel de la herencia. El principio opuesto a la generalización es la especialización.

La herencia constituye uno de los pilares básicos de la programación orientada a objetos, es la base para la reutilización y una de las principales ventajas del modelo OO. Esta propiedad que ha demostrado tener tantos aspectos positivos en el ámbito de la POO, claramente se puede extrapolar al campo de la agencia, de manera que se puedan definir nuevos agentes a partir de otros previamente definidos. Este concepto se ha incluido en el modelado conceptual de agentes (Kinny *et al*., 1996; Iglesias *et al.*, 1998; Wagner 2000). Por ejemplo, Kinny utiliza el concepto de jerarquía, para determinar que clases de agentes se pueden construir a partir de otras clases de agentes previas, mediante relaciones jerárquicas. En este contexto se utiliza la herencia y agregación como herramientas para organizar la abstracción del problema en clases.

La herencia permite definir nuevos objetos a partir de otros ya existentes, de forma que las clases derivadas extiendan o modifiquen las definiciones de las clases base. Cualquier instancia de una clase derivada tiene por defecto las mismas propiedades que su clase base posteriormente, mediante un proceso de especialización, cada descendiente podrá caracterizarse y así diferenciarse de su padre agregando, eliminando o redefiniendo propiedades.

La herencia puede convertirse en un aspecto fundamental dentro del modelo de agentes ya que en la actualidad se está trabajando mucho sobre esta idea. Si se consideran los agentes como objetos con unas características adicionales, puede pensarse que la herencia de agentes es similar a la de objetos. Sin embargo, la estructura de un agente y su grado de abstracción es diferente a la de un objeto y por tanto, la herencia de agentes tiene sus propias características. En el caso de los agentes deliberativos, estructurados en base a sus componentes mentales: creencias, deseos, capacidades, intenciones, etc., la herencia permitirá construir nuevos agentes a partir de otros previamente definidos de forman que aunque tengan una estructura similar se puedan especializar en por ejemplo la resolución de un problema concreto. Cuando se define una nueva clase de agentes sólo se necesitan declarar explícitamente las propiedades que difieren de las propiedades de las clases a partir de las que se construye. El resto de propiedades se extraen automáticamente de las clases de partida y son incluidas en la nueva clase. De la misma forma que para el resto de los principios de abstracción, la herencia estará determinada por la arquitectura de los agentes.

Aunque desde un punto de vista clásico la herencia en el modelo de objetos es un mecanismo de estructuración jerárquico pensado para la especialización conceptual, realmente la correspondencia entre herencia y especialización conceptual no es tan simple. La mayoría de los lenguajes de POO no proporcionan ninguna garantía de que la herencia se use realmente para la especialización conceptual. Es decir, si la herencia se utiliza de tal forma que permita añadir, eliminar o redefinir propiedades en las clases derivadas, no hay nada que garantice que se siga manteniendo una correspondencia conceptual con sus clases bases. Lo que viene a constatar que las abstracciones construidas usando herencia raramente son verdaderas especializaciones conceptuales de sus padres. Parece por tanto que hay una diferenciación importante entre considerar la herencia como un mecanismo del lenguaje y

como una herramienta para el modelado conceptual. Como parte de la fase del análisis y diseño de software, la herencia sirve como un medio de modelar las relaciones de generalización/especialización que existen entre las diferentes abstracciones (relación “es-un”). En la fase de implementación, la herencia tiene más que ver con la reutilización de clases existentes utilizadas para la definición de nuevas clases.

Dado el mayor nivel de abstracción de los agentes parece necesario mantener una relación conceptual adecuada entre el concepto de creencia y las relaciones de generalización/especificación. Si de lo que se trata es de utilizar el concepto de agencia para modelar un problema y facilitar de alguna manera el análisis, el diseño y la implementación de aplicaciones, además de los procesos que esto lleva consigo como por ejemplo la reutilización, parece esencial que la herencia se utilice para generar subclases de agentes que estén fuertemente relacionadas con sus superclases a nivel conceptual.

### 7.5 Agrupación y agentes

La agrupación consiste en reunir objetos, de manera coherente, para facilitar la conceptualización del problema, de manera que aunque los objetos agrupados no tengan las mismas propiedades, sea conceptualmente interesante mantenerlos juntos.

El agrupamiento permite representar colecciones de entidades. El principio opuesto a la agrupación es la individualización, la cual permite acceder a cada uno de los miembros individuales de una colección.

La POO soporta este principio permitiendo la definición de colecciones, más o menos arbitrarias, de objetos como por ejemplo listas, conjuntos, etc. Normalmente las instancias de una clase tienen alguna relación contextual dentro del dominio del problema que requiere que varios objetos sean agrupados para formar una entidad de mayor nivel (por ejemplo una familia estaría constituida por un conjunto de personas).

Al igual que en el modelo de objetos, en el modelado de agentes la agrupación de agentes/subagentes en colecciones facilita la organización y conceptualización del problema. De forma que cada colección se pueda tratar como un agente superior.

## Anexo A: Ejecutivas del lenguaje KQML

La tabla siguiente presenta una breve descripción de las ejecutivas del lenguaje KQML.

| Nombre | Significado |
|---|---|
| *Achieve* | S quiere que R haga algo verdad en su entorno |
| Advertise | S es particularmente adecuado para procesar una ejecutiva |
| ask-about | S quiere todas las sentencias relevantes de la VKB de R |
| ask-all | S quiere todas las respuestas de R a una pregunta |
| ask-if | S quiere conocer si la sentencia esta en la VKB de R |
| ask-one | S quiere una de las respuestas de R a una pregunta |
| Break | S quiere que R rompa el pipe establecido |
| Broadcast | S quiere que R envie una ejecutiva por todas las conecsiones |
| Broker-all | S quiere que R coleccione todas las respuestas a una ejecutiva |
| Broker-one | S quiere conseguir ayuda respondiendo a una ejecutiva |
| Deny | La ejecutiva incluida no se aplica a S (nada más) |
| Delete | S quiere que R quite una sentencia de su VKB |
| Delete-all | S quiere que R quite todas las sentencias emparejadas de su VKB |
| Delete-one | S quiere que R quite una sentencia emparejada de su VKB |
| Discard | S no querra las restantes contestaciones de R a una ejecutiva previa |
| Eso | Fin de un flujo de respuestas a una pregunta prematura |
| Error | S considera que el mensaje prematuro de R esta mal formado |
| Evaluate | S quiere que R simplifique la sentencia |
| Forward | S quiere que R encamine una ejecutiva |
| Generator | El mismo que una lista de un flujo-completo |
| Insert | S pide a R que añada el contenido a su VKB |
| Monitor | S quiere actualizar la respuesta de S a un flujo-completo |
| Next | S quiere la respuesta de R a una ejecutiva previamente mencionada |
| Pipe | S quiere dirigir todas las ejecutivas adicionales a otro agente |
| Ready | S esta listo para responder a la ejecutiva previamente mencionada de R |
| Recommend-all | S quiere todos los nombres de los agentes que puedan responder a una ejecutiva |
| recommend-one | S quiere el nombre de un agente que pueda responder a una ejecutiva |
| recruit-all | S quiere que R consiga todos los agentes convenientes para responder a una ejecutiva |
| recruit-one | S  quiere que R consiga otro agente para responder a una ejecutiva |
| register | S puede repartir ejecutivas a algún agente nombrado |
| reply | Comunica una contestación esperada |
| rest | S quiere las respuestas restantes de R a una ejecutiva previamente mencionada |
| sorry | S no puede proporcionar una contestación más informativa |
| standby | S quiere que R este listo para responder a una ejecutiva |
| stream-about | Versión multiple-respuesta de ask-about |
| stream-all | Versión multiple-respuesta de ask-all |
| subscribe | S quiere actualizar la respuesta de R a una ejecutiva |
| tell | La sentencia en la VKB de S |
| transport-adress | S asocia el nombre simbólico con la dirección de transporte |
| unregister | Negación de un registro |
| untell | La sentencia no está en la VKB de S |