CAPÍTULO 2

MODELO DE INTERACCIÓN ENTRE AGENTES (MIA)

El uso de agentes de software está motivado en la posibilidad de que las personas cuenten con herramientas que tomen decisiones y acciones en forma proactiva y autónoma para alcanzar propósitos, reduciendo el tiempo que un humano dedica a la interacción con computadoras. Hasta ahora se han desarrollado diferentes modelos y sistemas de agentes, algunos siguiendo un enfoque de la escuela cognitiva y otros de la escuela reactiva (véase la sección 1.2 en este documento).

En este capítulo se propone un modelo de agentes que intentan alcanzar varios propósitos tomando papeles de escenarios (interacciones) que los contienen en donde algunas de estas acciones se realizan en paralelo (multihilos). En estas acciones se dan interacciones entre agentes y con su ambiente. Las interacciones entre agentes son a través de la comunicación de mensajes codificados en la ontología de cada agente. Para determinar los papeles que un agente habrá de ejecutar para lograr sus propósitos, se genera un plan con base en sus recursos y los del ambiente considerando los papeles disponibles. Dado que para interactuar los agentes requieren entender los mensajes que reciben y dado que cada dos agentes podrían estar usando ontología diferentes, se considera un comparador de ontologías mixtas para encontrar la equivalencia entre los significados utilizados en los conceptos de agentes diferentes. En el modelo también se consideran aquellos eventos que pueden favorecer o dificultar el logro de los propósitos, de estos eventos se desconoce el momento de su ocurrencia.

La *modelación* [Kaindl 1999], es una actividad que permite abstraer información y conocimiento de un dominio específico para obtener un modelo que contiene los componentes esenciales (desde la perspectiva de los modeladores y de los objetivos que propician su obtención). En este caso un *modelo se entiende aquí como una abstracción de la realidad, en donde se consideran los detalles que son relevantes con el propósito de comprender, predecir o controlar el comportamiento de la realidad modelada*. La abstracción es la percepción selectiva de los detalles relevantes prescindiendo de aquellos que o no afectan o no son importantes dentro de los aspectos considerados. Como parte de la abstracción de la realidad, es conveniente hacer notar que, a través de una metodología se puede llegar a representarla. Una metodología es un conjunto sistemático de pasos que permiten tomar una parte de la realidad y mediante procesos de abstracción, análisis y síntesis, obtener un modelo. Dentro de los sistemas de información la metodología de desarrollo de sistemas consiste en general, de análisis, diseño, implantación y liberación. La etapa de análisis es donde generalmente se obtiene un modelo conceptual y en el diseño un modelo computacional que se desarrolla durante la implantación.

En la sección 2.1 se explican los elementos del Modelo de Interacción entre Agentes (MIA) propuesto en esta tesis, en los que se pretende representar a las entidades participantes del comercio electrónico consideradas en base a los propósitos que pretenden alcanzar, sus recursos, características, acciones iniciales, los eventos inesperados que perciben y las reacciones que han de desarrollar cuando se presentan estos. Los participantes en las interacciones pueden ser compradores, repartidores, vendedores, cargadores, entre otros. En el ambiente del modelo además de los agentes interactuando se consideran los recursos y las características del mismo.

Para alcanzar sus propósitos, un agente necesita mapear los recursos que tiene y sus propósitos contra los papeles disponibles en las interacciones (escenarios), dado que pueden ser varios los papeles que ha de ejecutar, es posible que se encuentre un camino formado por la secuencia de papeles que lo lleve de la situación inicial a una situación esperada.

Cada agente se considera con una forma interna de representación del conocimiento y que utiliza palabras de un lenguaje común entre agentes, por lo que, para lograr entender un concepto que otro intenta referirle se utiliza un comparador de ontologías mixtas que busca la equivalencia entre los conceptos dadas. La motivación para utilizar ontologías mixtas es para permitir que personas distintas sin necesidad de conocer al detalle el conocimiento de otro desarrollador puedan hacer posible la interacción entre los agentes creados entre ellos.

Otro aspecto de interés en el modelo es, la capacidad de los agentes para reaccionar ante eventos inesperados que se pueden presentar en el ambiente en que se encuentran. Cada agente en el modelo se considera capaz de realizar varias acciones en paralelo cuando existe compatibilidad entre ellas, esto es, por ejemplo, comprar un producto mientras otro agente le informa de una promoción. Como cada papel genera una hebra, decimos que los agentes son multihebras. El problema que enfrenta ante eventos inesperados radica en que hay un número infinito de estos y no es posible escribir un número igual de reacciones de emergencia para enfrentarlos y por lo tanto se debe contar con alguna forma de encontrar un "mejor comportamiento ante los mismos".

A diferencia de otros modelos como BDI o Agent0 (en donde los agentes ya cuentan con capacidades específicas), los agentes que modelamos toman sus habilidades dependiendo de sus propósitos. En el trabajo de Alvarado et. al. [Alvarado 2002a] se propone un modelo formal de interacción entre agentes donde cada agente está definido por sus creencias, metas y acciones en contraste, nuestro propuesta considera a cada agente descrito por sus recursos y características además de sus propósitos. Si bien existen algunas similitudes en cuanto a los elementos de interacción, existen diferencias en cuanto al tratamiento de planes, dado que en el modelo de Alvarado dichos planes indican las acciones para realizar colaboración, competición o negociación entre los agentes participantes y en nuestro modelo, los planes únicamente establecen los papeles que pueden servir para que un agente alcance sus propósitos.

En cuanto a los sistemas de comercio electrónico que se han realizado, hasta ahora la diferencia de esta propuesta con respecto a ellos radica en que nuestros agentes pueden realizar acciones que se especifican en los papeles del sistema, teniendo de esta forma compradores, vendedores, papeles de logística, entre otros; en otros sistemas los papeles están fijos en cada agente y los objetivos que alcanzan y las capacidades de un agente ya están especificadas. Un propósito es algo que se quiere lograr con un carácter general en donde no se especifica la forma ni los detalles para hacerlo. Un objetivo también es algo a lograr pero con la diferencia de que es cuantificado y se tienen alternativas en el agente para alcanzarlo. Una meta se considera como una parte o uno de los pasos que conducen a alcanzar un objetivo. Una acción es la realización de una actividad para lograr una meta. Por ejemplo, una agente puede tener el propósito de preparar tortillas, en base a este se generan

varios objetivos («goals») como: conseguir 5 kilos de maíz en 2 horas, moler el maíz en 1 hora, conseguir un comal y una estufa, obtener masa de maíz agregando 1 litro de agua por cada kilo de maíz, hacer las tortillas de la masa obtenido y luego cocerlas. A su vez, cada objetivo puede tener varias metas que el agente va alcanzando. Otro objetivo puede ser obtener todos los productos derivados del maíz visitando una empresa ubicada en la zona industrial de Toluca. En nuestro caso se maneja el concepto de propósito en contraste con el de creencias, deseos e intenciones, en donde las creencias corresponden a la información que el agente tiene respecto al mundo que lo rodea, en nuestro caso debido a la dinámica de las acciones realizadas por otros agentes no se modelan las creencias, sino únicamente el estado interno del agente. El ambiente se modela en forma independiente del agente. Los deseos representan los actos que el agente en un mundo ideal quiere realizar. En nuestro caso no modelamos deseos. Los papeles que el agente toma para alcanzar sus propósitos se ejecutan cuando los agentes de nuestro modelo cumplen con sus requisitos, es decir aunque ocurran eventos inesperados.

A partir del modelo propuesto y con la intención de hacer posible representar agentes en un ambiente de simulación en donde se puedan especificar tanto estos como su ambiente se desarrolla en la sección 2.2 el Lenguaje de Interacción entre Agentes (LIA), indicando sus elementos léxicos, sintácticos y semánticos. Estos sistemas descritos en LIA requieren de un ambiente para su ejecución, el cual es motivo de la sección 2.3 en donde se explican los módulos del Sistema de Ejecución de Agentes (SEA).

La intención de crear LIA es para facilitarle al desarrollador concentrarse en los aspectos relevantes de un problema en donde se tienen agentes descritos mediante propósitos, recursos y características y que toman papeles de interacciones (escenarios) ejecutándolos en paralelo cuando hay compatibilidad entre ellos.

En SEA un proceso de planeación encuentra el papel que a un agente le permite alcanzar sus propósitos, durante el intercambio de mensajes se activa el comparador de ontologías mixtas para encontrar la equivalencia en el significado de un conjunto de palabras que se refieren a un concepto, cuando ocurre un evento inesperado les es posible reaccionar ante estos buscando para cada uno la mejor reacción. Estas características en conjunto, actualmente están en proceso de investigación en algunas plataformas como JADE o Zeus como puede observarse en el anexo A, al final de este documento.

2.1 MODELO CONCEPTUAL

Como se ha comentado al inicio de este capítulo, los agentes representan entidades del mundo real y los modelamos en base a sus propósitos, recursos y características. Cada agente tiene una ontología con la que relaciona las palabras (que utiliza durante la comunicación con otros agentes) con los conceptos que maneja. Durante el intercambio de palabras, el comparador de ontologías busca la equivalencia de conceptos referidos por dos agentes. Cada agente utiliza sus recursos durante la ejecución de papeles, durante esto también obtiene otros. Los papeles que le permiten a un agente lograr sus propósitos se obtienen de un plan que se obtiene considerando sus recursos y los papeles disponibles. Cada agente se le definen algunos papeles que utiliza durante la ocurrencia de eventos inesperados.

En esta sección se indican los elementos que se utilizan para la representación de los componentes del modelo.

Los componentes generales del modelo son:

a) El ambiente, donde existen los agentes y las interacciones en que participan los primeros, tomando de estos los papeles que les permiten alcanzar sus propósitos.

b) Los agentes, descritos por sus propósitos, recursos, características y los eventos que perciben.

c) Las interacciones, conteniendo los papeles que pueden utilizar los agentes.

En el modelo una variable representa un recurso o una característica.

2.1.1 AMBIENTE

El ambiente se modela mediante recursos a los que pueden acceder cualquier agente. Los recursos se representan mediante variables globales y regionales. Una variable global se considera accesible a cualquier agente o hebra en el modelo, por ejemplo: Reloj, Catálogo. Las variables regionales son accesibles únicamente a aquellos agentes que las declaran como accesibles, por ejemplo: localidad, producto, horaLocal.

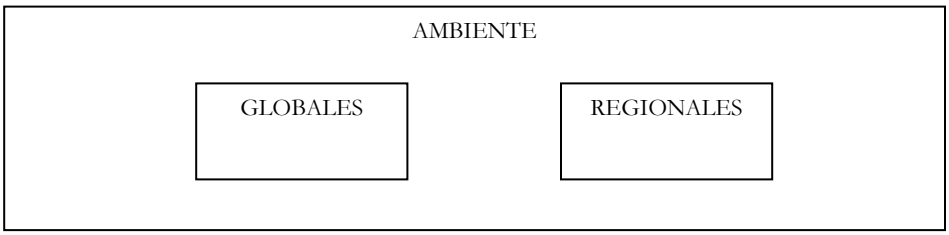


Figura 2.1 Componentes del Ambiente

2.1.2 AGENTES

En el modelo, un *agente* (figura 2.2) *representa una entidad del mundo real la cual posee propósitos*, por ejemplo: persona, animal, camión.

Un *propósito es una proposición que puede ser falsa o verdadera y cuenta con un atributo que indica si se ha alcanzado o no*. Cada agente tiene varios propósitos que inicialmente están marcados como no alcanzados.

Cada agente posee *variables internas que definen sus características y sus recursos*. Las variables internas se caracterizan porque son accesibles únicamente por el mismo agente y sus hebras, esto significa que es imposible que otros las puedan accesar, pero es posible enviar el contenido de una variable interna como un mensaje a otro agente (el envío y recepción se explica en 2.1.3). Cada agente tiene una lista de las *variables regionales* a las que tiene acceso para almacenar o recuperar su contenido. Existe una variable distinguida dentro de aquellas que son internas (del agente) que indica la ontología utilizada para expresar el contenido de los mensajes del mismo, llamada *ontology*. Si dicha variable no aparece,

se considera que el agente utiliza una ontología común para la codificación de los mensajes que intercambia con otros.

Un papel es un conjunto de instrucciones que un agente puede realizar y se encuentra dentro de las interacciones (escenarios). Cada agente tiene asignado un conjunto de papeles que se ejecutan automáticamente cuando se crea, llamados *papeles iniciales*.

Un evento inesperado es un acontecimiento que representa situaciones del mundo real que, en la mayoría de las ocasiones no son previsibles en tiempo y espacio, por ejemplo: un temblor, la caída de un servidor, reducción del ancho de banda en una transmisión, encontrar algo, ganarse la lotería. Cada agente tiene una lista de aquellos eventos inesperados que percibe y que en un momento dado pueden causarle una afectación. La afectación puede ser positiva, negativa o neutra. Una afectación positiva puede causar en el agente el acceso a más recursos o adquirir características que le permiten alcanzar sus propósitos más fácilmente. Una afectación negativa se considera cuando el agente pierde la oportunidad de alcanzar uno o varios de sus propósitos. Es neutra sino afecta en nada al agente. Para responder ante los eventos inesperados, un agente tiene un conjunto llamado "papeles de emergencia" que corresponden con las reacciones que puede utilizar ante los eventos inesperados que lo afecten. Estas reacciones se seleccionan en base a un árbol de Eventos Inesperados buscando siempre la más específica.

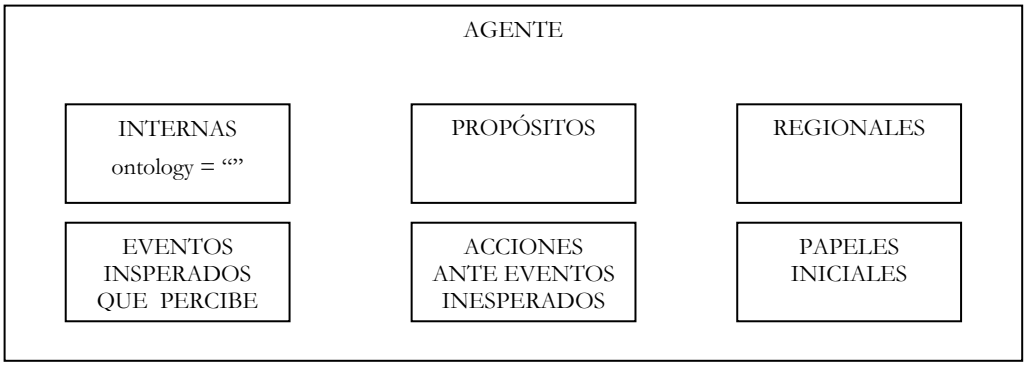


Figura 2.2 Componentes de un agente

2.1.3 INTERACCIONES

Una *interacción (escenario)* (figura 2.3) *representa lugares del mundo real donde se llevan a cabo algunas acciones por agentes, por ejemplo, un restaurante, o centro comercial*; ahí los individuos adquieren un papel determinado para alcazar el propósito que los llevo al lugar, por ejemplo, en el centro comercial hay vendedores y compradores cada uno con un propósito diferente, esto es, en dos vendedores uno puede tener como propósito comprar una casa, mientras que, otro quiere dinero para alimentar a su familia. Los papeles se ocupan por agentes que cumplen sus requisitos, no cualquier agente ocupa un papel, solamente aquellos que cuentan con las características o recursos que el papel necesita. Cada interacción con sus papeles se describen en forma independiente de los agentes que los ocupan, esto significa que los papeles no pertenecen de antemano a determinado agente, sino que la asignación se hace dinámicamente (como resultado del proceso de planeación que determina los papeles más apropiados para que un agente alcance sus propósitos) con aquellos que cumplen con los requisitos del papel. A un papel en ejecución se le llama hebra, y cada agente cuenta con la capacidad de ejecutar varias hebras en paralelo [Alvarado 2002]. Las hebras pueden provenir de tomar papeles de

diferentes interacciones (los agentes en el modelo son multihebras). La ejecución en paralelo es posible cuando existe compatibilidad entre los requisitos de los papeles.

Cada papel dentro de una interacción (figura 2.4) consta de un conjunto de requisitos que pueden cubrirse para que un agente pueda ocuparlo. Además, contiene un atributo del máximo de instancias que puede tener para ser ocupado. Esto último indica que aunque un agente cubra los requisitos, si el máximo de ocupación de un papel dentro de una interacción se ha alcanzado entonces no puede ocuparse. Cada papel de una interacción además contiene un conjunto de variables locales mismas que son accesibles únicamente a la hebra que ocupa una instancia del papel, estas no son accesibles a otras hebras en un mismo agente.

En los papeles se encuentran descritas las instrucciones que debe realizar un agente (como en una obra de teatro) pero no se dice exactamente cual de todos será quien ocupe el papel. Para poder acotar quien es el agente que puede ocupar un papel, cada uno tiene un conjunto de requisitos que deben cumplirse (por parte del agente) en el momento en que el papel, se pasa a ejecución como una hebra.

Cuando un agente ejecuta una instrucción de un papel se dice que realiza una acción.

Un plan es una secuencia de papeles de una o varias interacciones que permiten a un agente alcanzar uno o varios propósitos.

La selección de los papeles que un agente ha de interpretar, se realiza mediante un proceso de planeación (descrito en el capítulo 4). El proceso de planeación utiliza la información del ambiente particular definido. En este caso, los agentes, sus propósitos, las interacciones y los papeles disponibles. El resultado es una secuencia de papeles que se pueden ejecutar en paralelo cuando sea pertinente y exista compatibilidad con los que se encuentren en ejecución en ese momento.

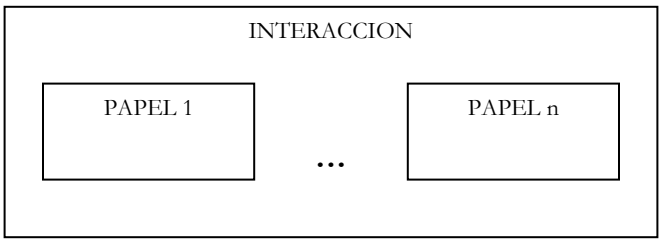


Figura 2.3 Componentes de una interacción

Un papel es compatible con otro al momento de ejecutarlo si es una excepción en una tabla donde se tienen las incompatibilidades entre papeles, esto es, si dos papeles no se encuentran en la misma, entonces pueden ejecutarse en paralelo.

Para que un agente pueda iniciar la ejecución de un papel dentro de un plan para alcanzar un propósito, se verifica que cumpla en ese momento con los requisitos y además, que sea compatible con los que ya se encuentra ejecutando; es decir, que si un agente impresora ejecuta el papel "imprimiendo un documento" será imposible que tome el papel "reimprimir documento" al mismo tiempo, para esto será necesario que abandone el papel en proceso o bien que espere a concluirlo para iniciar el siguiente, pero si podrá por ejemplo realizar el papel "informar actividad actual".

Los agentes acceden a recursos comunes del ambiente, como se dijo, expresados como variables globales o regionales. Una forma alternativa para intercambiar información o conocimiento entre agentes es, mediante *puertos de comunicación* que se declaran como variables globales y en donde pueden dejar o tomar mensajes. Un agente tiene acceso a varios puertos en forma simultanea y por lo tanto, puede enviar o recibir mensajes de diferentes agentes. Cada mensaje que se envía contiene el agente emisor y por lo tanto otro u otros agentes pueden enviarle la respuesta.

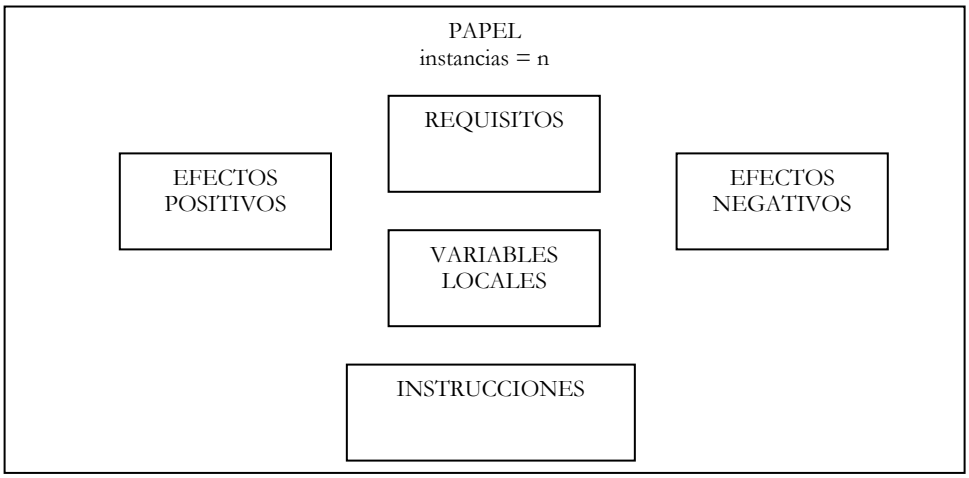


Figura 2.4 Componentes de un papel

Los mensajes que se envían mediante los puertos de comunicación pasan por el proceso de comparación de ontologías en forma transparente al agente (véase el capítulo 3 en este documento); por lo tanto, los mensajes que recibe un agente se encuentran en su ontología; cuando no se puede mapear a su ontología un mensajes se incluyen las palabras `not-found` para cada elemento desconocido con lo que el agente que recibe un mensaje identifica que existen elementos incomprensibles y así tome alguna acción.

A partir de la cuantificación de los propósitos alcanzados por un agente y de los que quedaron pendientes se obtiene su grado de satisfacción, por ahora como un porcentaje de propósitos alcanzados. Esto significa que un agente está cien por ciento satisfecho cuando alcanza sus propósitos y estará insatisfecho en alguna medida en función de cuantos propósitos le falten o quedaron pendientes, asignando un valor entre 0 y 99% de satisfacción redondeado al entero más próximo.

2.1.4 Interacción entre Agentes

La interacción entre agentes puede darse mediante el acceso a recursos comunes (expresados en variables globales o regionales) o mediante la comunicación de mensajes. Los recursos a los que se refiere un agente durante una interacción se encuentran expresados en los papeles que forman parte de ella.

Para establecer la secuencia de las interacciones entre los diferentes papeles se pueden utilizar los diagramas de interacción de AUML (Agent Unified Modeling Language, «Lenguaje Unificado de Modelación para Agentes») [Odell 2001]. En estos diagramas, el nombre de la interacción se escribe en la parte superior con mayúsculas, el nombre del papel se escribe en un rectángulo, para cada uno se tiene una línea de tiempo que avanza hacia abajo. En una interacción puede haber varios papeles. Las líneas horizontales con punta de flecha son los mensajes que se intercambian entre los agentes que toman el papel, la punta de flecha indica la dirección del mensaje. Generalmente se presentan únicamente los mensajes más representativos. En la figura 2.5 se muestran dos interacciones, una para cuando se carga un camión y otra que representa las interacciones entre un chofer y el camión que conduce a un destino.

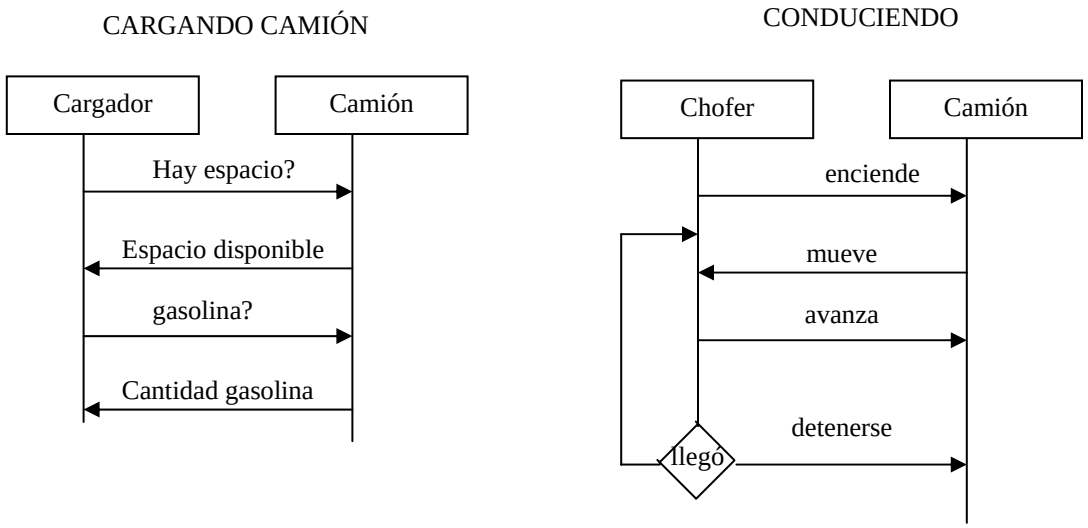


Figura 2.5 Diagramas de Interacción para la carga de un camión y conducción del mismo. Cargador, Camión y Chofer son los agentes que interactúan

Los agentes durante sus interacciones pueden comunicarse para intercambiar información y conocimiento, uno actúa como un emisor y otro como receptor. El emisor envía un mensaje al receptor utilizando palabras de un lenguaje común entre ellos. En nuestro modelo la comunicación entre agentes se hace mediante el intercambio de mensajes. Cada mensaje tiene varios atributos que determina la forma en que llega a su destino como su cardinalidad y duración. La duración se refiere al tiempo que esperará un mensaje en caso que el receptor no se encuentre en ese momento. La cardinalidad se refiere al número de agentes que pueden leer el mismo mensaje.

Los mensajes se colocan en puertos donde en caso que el agente receptor no se encuentre en el momento en que llega, puedan esperar, además si son varios mensajes pueden encolarse. Un agente para tomar un mensaje puede referirse al identificador del mensaje o tomar el primer mensaje que aparezca en la cola. Como un agente puede referirse a varios puertos y a su vez puede leer de varios de ellos; el resultado es una red de comunicación (figura 2.6).

El identificador del mensaje es útil en caso que lleguen mensajes por pares o tríos, como cuando se aceptan parejas de mercancía junto con una persona encargada de transportarla, al momento de que coinciden los identificadores se garantiza que viajan juntos el encargado con su carga (figura 2.7).

Aunque los mensajes pueden estar codificados con diferentes lenguajes y dado que pueden utilizarse traductores asumimos que los mensajes están en un mismo lenguaje, dando entonces lugar

a otro tipo de problemas, como el hecho que para que el mensaje se entienda, debe poder integrarse en la representación interna de un agente, este tema se trata en el capítulo 3 en este documento.

En la comunicación entre agentes se pueden presentar dos problemas los cuales tampoco pretenden resolverse en esta tesis:

a) *Deadlock*, cuando un agente espera en forma indefinida. Inicialmente, se pensó en un límite de tiempo («time-out») para terminar hebras que han esperado mucho, pero no se siguió esta línea de investigación.

b) *Livelock*, cuando por más que interaccionan los agentes no se da progreso en la misma. Igualmente se pensó en un límite de tiempo («time-out») para terminar hebras que no progresan hacia sus objetivos, pero no se siguió esta línea de investigación.

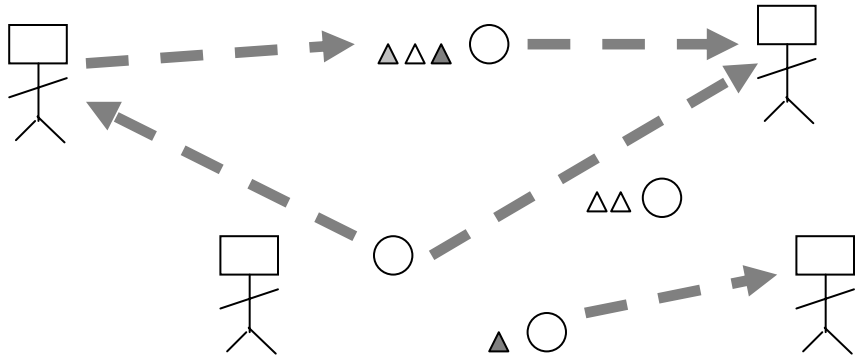


Figura 2.6 En el ambiente de MIA-LIA existen varios puertos donde los agentes pueden dejar o tomar mensajes. Los triángulos son mensajes, los circulos puertos

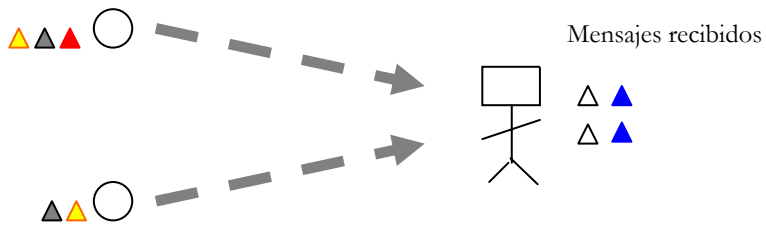


Figura 2.7 Los mensajes contienen un identificador que se utiliza para enviar y recibir mensajes relacionados. Aquí se usa el color del mensaje para marcar a los relacionados

2.2 MODELO NOTACIONAL

El modelo notacional tiene el propósito de identificar los componentes del modelo de interacción entre agentes. De esta forma al representarlos mediante una notación de conjuntos los componentes forman una tupla como sigue:

$$\text{MIA ::- } \langle \mathbf{A}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{I}, \mathbf{C}, \mathbf{E}, \mathbf{H} \rangle$$

el símbolo ::- se lee "está formado por", de donde:

- **A** es un conjunto de Agentes y cada uno representa una entidad del mundo real con propósitos.
- **P** es un conjunto de Propósitos, que al instanciarse se establecen objetivos que un agente puede alcanzar, cada propósito tiene el formato variable_valor, equivalente a un predicado de primer orden variable(valor) y donde variable es un recurso o una característica del agente y el valor corresponde con las expectativas del agente.
- **R** es un conjunto de Recursos encontrados en el ambiente o son propiedades de los agentes, se representan cada uno por una variable.
- **O** es un conjunto de Ontologías, donde cada una representa una estructuración del conocimiento en la que puede estar expresada la información y el conocimiento usados por un agente.
- **I** es un conjunto de Interacciones, donde cada una contiene papeles que utilizan los agentes para alcanzar sus propósitos.
- **C** es un conjunto de Papeles que puede un agente tomar para alcanzar sus propósitos.
- **E** es un conjunto de Eventos Inesperados que pueden afectar a los agentes.
- **H** es un conjunto de instrucciones que son parte de los papeles.

El ambiente en que se encuentran los agentes y las interacciones es:

$$\textbf{Ambiente} \subseteq \textbf{R}$$

a su vez está formado por dos subconjuntos:

$$\textbf{Globales} \subseteq \textbf{Ambiente}$$

$$\textbf{Regionales} \subseteq \textbf{Ambiente}$$

donde:

$$\textbf{Globales} \cap \textbf{Regionales} = \varnothing$$

Un agente $\mathbf{a} \in \mathbf{A}$ tiene la estructura $\mathbf{a}$ ::- (P, <V, o>, E, S)

donde:

$P \subseteq \mathbf{P}$, $V \subseteq \mathbf{R}$, $E \subseteq \mathbf{E}$, $o \in \mathbf{O} \subseteq \mathbf{R}$, $S \subseteq \mathbf{C}$

se dice que P son los propósitos del agente; V es el conjunto de sus propiedades y recursos, o es la ontología en que se codifican los mensajes que envía y es una característica individual; E es el conjunto de eventos inesperados que puede percibir y S es el conjunto de papeles con que inicia su ejecución y los de emergencia que puede utilizar para atender los eventos inesperados que lo afecten.

Cada interacción tiene la estructura siguiente:

$$I ::- (C)$$

donde

C es el conjunto de papeles que contiene la interacción ($C \subseteq \mathbf{C}$) y cada papel $c_i \in C$ tiene la estructura:

$$c_i ::- (<\text{cupo, Requisitos}>, \text{Instrucciones, Efectos})$$

donde:

cupo $\subseteq \mathbf{R}$, Requisitos $\subseteq \mathbf{R}$, Efectos son aquellos que pueden ser positivos o negativos, los positivos son los que indican que un agente logró sus propósitos, Efectos $\subseteq \mathbf{R}$, Instrucciones es una secuencia ordenada de elementos $<h_i \mid h_i \in \mathbf{H}>$, cada instrucción h_i puede causar cambios en el ambiente, en los recursos del agente y en las variables locales de los papeles que ejecuta el agente, esto es:

$$hi: \mathbf{Ambiente}_i \times V_i(\mathbf{a}) \times C_i(\mathbf{a}) \rightarrow \mathbf{Ambiente}_j \times V_j(\mathbf{a}) \times C_j(\mathbf{a})$$

donde se cumple una o las condiciones siguientes:

$$\mathbf{Ambiente}_i = \mathbf{Ambiente}_j \vee \mathbf{Ambiente}_i \neq \mathbf{Ambiente}_j$$

$$V_i(\mathbf{a}) = V_j(\mathbf{a}) \vee V_i(\mathbf{a}) \neq V_j(\mathbf{a})$$

$$C_i(\mathbf{a}) = C_j(\mathbf{a}) \vee C_i(\mathbf{a}) \neq C_j(\mathbf{a}).$$

Para que un agente alcance sus propósitos debe desempeñar algunos papeles que forman su plan. Un plan de un Agente **a** es:

$$\text{Plan}(\mathbf{a}) = <c_1, c_2, ..., c_a>...<c_1, c_2, ..., c_n>$$

donde **a** es un agente, $c_i \in \mathbf{C}$, cada secuencia $<c_1, c_2, ..., c_x>$ le permite al agente alcanzar uno o varios propósitos P al efectuar los papeles. Un agente puede alcanzarlos parcialmente aprovechando los efectos positivos de un papel.

Para encontrar el plan de un agente **a** se utiliza un planificador que toma los propósitos de un agente, los recursos del agente y los papeles de las interacciones.

$$\text{Planificador}(P(\mathbf{a}), V(\mathbf{a}), \mathbf{I}) \rightarrow \text{Plan}(\mathbf{a})$$

donde:

$P(\mathbf{a})$ son los propósitos del agente **a**, $V(\mathbf{a})$ son los recursos y características del agente **a**, **I** son las interacciones con sus papeles de donde el planificador selecciona los que forman parte del plan del agente.

Incompatibles es el conjunto de pares de papeles $<c_i, c_j>$ que son incompatibles y cuya ejecución simultanea es imposible. $c_i, c_j \in \mathbf{C}$.

La ejecución de un plan se realiza en forma simultanea, siempre que ci de una secuencia y c_j de otra secuencia sean compatibles, es decir el par $<c_i, c_j> \in$ Incompatibles.

Un evento inesperado $e \in \mathbf{E}$ afecta a un agente **a** si el agente los percibe, es decir si $e \in E(\mathbf{a})$. La afectación de un agente puede reflejarse modificando sus variables internas, suspendiendo la ejecución de algunas hebras, adicionándole papeles de emergencia o puede dejar inalterado al agente. Para que reaccione ante un evento inesperado e utiliza sus papeles de emergencia $S(\mathbf{a})$.

La comunicación entre agentes se realiza mediante el intercambio de mensajes expresados en su ontología.

Un mensaje $m\mathbf{a}$ de un agente **a** se dice que está expresado en una ontología o_a y se denota:

$$o_a(\mathbf{a}, m\mathbf{a})$$

Para que un agente **a** entienda un mensaje m_a de un agente **b** se requiere que exista un mapeo (COM) entre las ontologías de ambos agentes.

$$\text{COM}\ (o_b(\mathbf{b}, m_b), o_a(\mathbf{a}, m_a))$$

donde COM encuentra que:

$$m_a \approx m_b \vee m_a \neq m_b$$

En el mapeo se considera el intercambio de palabras que pueden ser ambiguas, para ayudar en el mapeo se utilizan los ancestros y los hijos de los conceptos en cuestión (véase el capítulo 3 para mayor detalle sobre el mapeo de conceptos).

2.3 LENGUAJE DE INTERACCIÓN ENTRE AGENTES (LIA)

Con el propósito de obtener una forma traducible a programas de computación del modelo expuesto en la sección anterior, se consideró la conveniencia de proponer un lenguaje porque, al intentar mapearlo en lenguajes existentes como C o Java resultaba en metaprogramas que complican la visualización de los aspectos que se quieren destacar en el modelo. Por lo tanto en LIA los aspectos

relevantes se perciben directamente haciendo natural[14] la expresión de los mismos, en este caso: el ambiente, los agentes con sus recursos y características, las interacciones con los papeles que forman a cada una. En los papeles se tienen las instrucciones que cada agente que toma ese papel realiza en caso que las condiciones del ambiente sean favorables (que no ocurran eventos inesperados o que otros agentes estén usando los recursos que el agente necesita).

A continuación se indican fragmentos de un programa escrito usando LIA para ilustrar la forma en que se definen los elementos del modelo de agentes (MIA).

Descripción de los recursos de un *ambiente* mediante variables globales y regionales:

```
global {  // VARIABLES GLOBALES
    int Reloj ;
    port pcliente ; // PUERTO DONDE ESCUCHA MENSAJES EL CLIENTE
    port pprovedor ; // PUERTO DONDE ESCUCHA MENSAJES EL PROVEEDOR
    char id_msg[30] ;
    int duracion, lectores ;
}
regional { // VARIABLES REGIONALES
     float localidad ;
     char  sucursal[40] ;
}
```

Declaración de un *agente*:

```
agent Cliente {  // AGENTE CLIENTE
    regional { // VARIABLE REGIONAL QUE PUEDE REFERENCIAR
      localidad ;
    }
    internal { // VARIABLES INTERNAS
       char tiene[30], dinero[30], comidaAporta[30],
            obtiene[30], quiere[30], desea[30],
            ofrece[30], alimento[30]  ;
       char articulo[100] ; int comprar_a ;
    }
    purpose { // PROPOSITOS DEL AGENTE
        vacaciones_cancun ; // SE INTERPRETA COMO vacaciones(cancun)
     }
    percieve { // EVENTOS INESPERADOS QUE PERCIBE EL AGENTE
       lluvia, terremoto, caida  ;
    }
    action { // ACCIONES ANTE EVENTOS INESPERADOS
       pedirAyuda  ;
    }
    initial { //  PAPELES INICIALES
       esperar  ;
    }
}
```

[14] Respecto a lenguajes como C o Java

Declaración de una *interacción:*

```
interaction Buscando {
    role buscador(producto, oferta)[instances 1] {
    requisite {
      ofrece_oferta, dinero_oferta, desea_producto, alimento_producto ;
    }
    positive {
     quiere_producto ;
    }
    negative {
      ofrece_oferta ;
    }
    local  {
       char logro[100] ;
       char linea[100] ;
       int encontro ;
    }

   linea := "Buscando ";
   append(linea, articulo);
   print(linea);
   mensaje := "( BUY ARTICULO ";
   append(mensaje ,articulo);
   tmp := " )";
   append(mensaje ,tmp);
   id_msg := "mensaje1";
   lectores = 10;
   duracion = 120; // EN SEGUNDOS
   // EMITE EL MENSAJE DE LO QUE QUIERE EL CLIENTE
   out(pcte, id_msg, mensaje, lectores, duracion);
   // ESPERA UN TIEMPO PARA QUE AGENTES VENDEDORES SE ENTEREN
   wait(10);
   encontro = 0;
   while( encontro == 0 )  {
      // ACEPTA UNA OFERTA
      id_msg := "acuerdo";
      espera = 100;
      accept(pcte, id_msg, linea, espera);
      // REVISA SI LO QUE PIDEN LO PUEDE SURTIR
      dato:= "ARTICULO";
      extract(solicitud, dato, cual);
      compare(cual, articulo, resultado);
      // EN CASO QUE LO PUEDE SURTIR
      if( resultado == 1 )  {
          encontro = 1;
      }
   }
   logro := "quiere_";
   append(logro, desea); // 'ARTICULO'
   goal(logro);  // INDICA QUE SE HA OBTENIDO UN ARTICULO
                 // SI ES EL DESEADO ENTONCES SE ALCANZO EL PROPÓSITO
  }
}
```

Sección principal donde se crean instancias de agentes e interacciones

```
main()
{
   print("Industria del Maíz");
   // CREACION DE INTERACCIONES
   addinteract(Casa ,casaAntonio );
   addinteract(Mercado,CentralAbasto);
   addinteract(Buscando, Buscar);
   // CREACION DE AGENTES
   addagent(Cliente , Antonio, // ANTONIO ES UN AGENTE CLIENTE
         obtiene_energia,  // SE ADICIONAN PROPÓSITOS
         ofrece := "dinero"; // SE ASIGNAN VALORES A SUS VAR. INTERNAS
         desea := "tortillas"; alimento := "tortillas";
         dinero := "dinero"; comidaAporta := "energia";
         articulo := "MAIZ-SEMILLA";
         ontology := "cliente" ); // SE INDICA LA ONTOLOGÍA QUE USA

   addagent(Provedor, Rafael,
         gana_dinero,
         ofrece := "maíz"; desea := "dinero"; recurso := "dinero";
         persona := "humano"; capta := "humano";
         producto := "MAIZ-SEMILLA-BLANCO"; precio := "80.00";
         id := "1"; ontology := "provedor" );

   addagent(Provedor, Juan,
         gana_dinero,
         ofrece := "maíz"; desea := "dinero"; recurso := "dinero";
         persona := "humano"; capta := "humano";
         producto := "MAIZ-JARABE"; precio := "15.00"; id := "2";
         ontology := "provedor" );

   addagent(Provedor, Esteban,
         gana_dinero,
         ofrece := "maíz"; desea := "dinero"; recurso := "dinero";
         persona := "humano"; capta := "humano";
         producto := "MAIZ-FRUCTUOSA"; precio := "12.00"; id := "3";
         ontology := "provedor" );
}
```

2.3.1 PALABRAS RESERVADAS DE LIA

LIA es un lenguaje estructurado para definir agentes y crear instancias de los mismos en donde se tienen varias *palabras reservadas* que no pueden utilizarse como identificadores ni en otro contexto de un sistema, estas son: global, int, char, float, port, regional, agent, purpose, internal, local, interaction, role, requisite, accept, out, SI, NO, if, else, while, stop, addagent, addinteract, addpurpose, start, print, main, self, true, false, optional, instances, print, wait, percieve, initial, action, unexpected_events, positive, negative, append, extract, compare, delmessage, goal.

2.3.2 SINTAXIS DE LIA

Para describir la *gramática de LIA* se utiliza la notación de Backus Naur Form (BNF) [Naur 1960] introducida para describir la sintaxis del lenguaje Algol60 y actualmente difundida para otros.

Los meta-símbolos de BNF son:

::= que significa "definido como".

| que significa "ó"

<> indica un nombre de categoría, las palabras encerradas en estos se conocen como símbolos no terminales.

Los símbolos de puntuación (coma, punto, punto y coma, etc.) se denotan a si mismos.

Los símbolos terminales se escriben con minúsculas (en este caso, palabras reservadas de LIA) o son signos de puntuación.

Un ejemplo de producción BNF es:

```
<programa> ::=  programa
                    <secuencia_declaraciones>
                begin
                    <secuencia_instrucciones>
                end ;
```

Esto indica que un programa consiste del símbolo terminal "programa" seguido de una secuencia de declaraciones, el símbolo terminal "begin" una secuencia de instrucciones y finalmente los símbolo terminales "end" y ";".

Algunos autores han incluido ciertas extensiones de la BNF para facilitar su uso:

a) Los elementos opcionales se encierran en los metasímbolos **[** y **]**, por ejemplo:

```
<if_statement> ::=  if <boolean_expression> then
                         <statement_sequence>
                    [ else
                         <statement_sequence> ]
                    end if ;
```

b) Elementos repetitivos (cero o más veces) se encierran en los meta símbolos **{** y **}**, por ejemplo:

```
<identificador> ::= <letra> { <letra> | <dígito> }
```

lo cual equivale a la regla recursiva:

```
<identificador> ::= <letra> | <identificador> [ <letra> | <dígito> ]
```

c) Los terminales de un solo carácter se encierran entre comillas (“”) para distinguirlos de los metasímbolos.

```
<secuencia_instrucciones> ::= <instrucción> { “;” <instrucción> }
```

Un sistema en LIA se compone de la descripción del ambiente formado por variables globales y regionales, los agentes que interactuan, un proceso de inicio llamado main y la descripción de las interacciones que se pueden realizar donde cada una contiene varios papeles.

```
<sistema_LIA> ::= <globales> <regionales> <agentes> <main>
    <interacciones>
```

Las *variables globales* se identifican porque inician con la palabra reservada global y a continuación entre llaves se da la lista de las mismas, las cuales inician con un identificador de tipo seguido por una lista de identificadores. Cada identificador está formado por una letra y una sucesión de letras y números. En el caso que se trate de un arreglo de identificadores el factor de repetición se escribe entre corchetes.

```
<globales> ::= global “{”  <global> { <global> } “}”
<global> ::= <tipo> <variables> “;”
<tipo> ::= int | char  | float | port
<variables> ::= <variable> { “,” <variable> }
<variable> ::= <identificador> |
    <identificador> “[” <entero> “]” [ “[” <entero> “]” ]
<identificador> ::= <letra> { <letra> | <digito> | “_”  }
<letra> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n |
    o | p | q  |  r  | s | t | u | v | w | x | y | z
<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <dígito> { <dígito> } [  "." { <dígito> } ]
<entero> ::= <dígito> { <dígito> }
```

Las *variables regionales* se definen en forma similar a las globales, por lo que comparten algunas producciones sintácticas.

```
<regionales> ::= regional "{" <regional> { <regional> } "}"
<regional> ::= <tipo> <variables> ";"
```

Los agentes se definen mediante una lista de ellos y cada uno se identifica por la palabra reservada agent y luego sus componentes, que en este caso son las variables regionales que puede accesar, sus variables internas, sus propósitos, los eventos inesperados que puede percibir, los papeles que puede utilizar como reacciones ante los imprevistos y los papeles con que inicia su ejecución. Las variables internas se definen en forma similar a las globales y regionales. Los propósitos, variables, regionales, eventos que percibe, reacciones y papeles iniciales son listas de identificadores.

```
<agentes> ::= agent "{" <lista_regionales> <internas> <propositos>
    <percibe> <acciones> <iniciales>  "}"
<lista_regionales>  ::= regional "{" <identificador> { ","
    <identificador> } "}"
<internas> ::= internal "{" <interna> { "," <interna> } "}"
<interna> ::= <tipo> <variables> ";"
<propósitos> ::= purpose "{" <predicado> { "," <predicado> } "}"
<percibe> ::= percieve "{" <identificador> { "," <identificador> } "}"
<acciones> ::= action "{" <identificador> { "," <identificador> } "}"
<iniciales> ::= initial "{" <identificador> { "," <identificador> } "}"
```

La *sección principal* con que inicia la ejecución de un sistema en LIA se identifica por la palabra reservada `main` y contiene las instrucciones que se efectuan al inicio de la ejecución de un sistema.

```
<main> ::= main "(" ")" "{" <instrucción> { <instrucción>  } "}"
```

Las *interacciones* (escenarios) agrupan a los papeles que pueden tomar los agentes. En estos las interacciones entre agentes se indican en las instrucciones de los mismos. Cada uno tiene un máximo de ocupación que establece el número de agentes que pueden ocupar el mismo papel al mismo tiempo, por ejemplo solamente puede haber un director en una empresa o 20 empleados de limpieza en un edificio. Cada papel tiene variables locales que se definen en forma similar a las globales, regionales e internas. Las producciones gramaticales que representan esto son:

```
<interacciones> ::= <interacción> { <interacción> }
<interacción> ::= interaction <identificador> "{" <papeles> "}"
<papeles> ::= <papel> { <papel> }
<papel> ::= role <identificador> <parametros> "[" instances <número>
    "]" "{" <requisitos> <locales>   <instrucciones> "}"
<parametros> ::= "(" <predicado> { "," <predicado> } ")"
<requisitos> ::= requisite "{" <predicado> { "," <predicado> } "}"
<locales> ::= local "{" <local> { <local> } "}"
<local> ::= <tipo> <variables> ";"
```

Las instrucciones que forman al proceso inicial (main) y a cada papel, permiten crear agentes (addagent), interacciones (addinteract), desplegar información (print), llevar el control de la ejecución (while, if), indicar que se ha alcanzado un propósito (goal), suspender una hebra durante un intervalo de tiempo (wait), enviar o aceptar mensajes (out, accept, deltemessage), extraer información de un mensaje (extract), comparar dos cadenas de caracteres (compare), anexar datos a una cadena (append) o efectuar operaciones aritméticas y lógicas.

```
<instrucciones> ::= <instruccion> { <instruccion> }
<instruccion> ::= print "(" <cadena> ")" ";" |
    print "(" <identificador> ")" ";" |
    if "(" <condicion>  ")" "{" <instrucciones> "}" |
    while "(" <condicion>  ")" "{" <instrucciones> "}" |
    addagent "(" <identificador>  "," <identificador>  ","
        <asigna_propositos> "," <asigna_recursos>  ")" |
    addinteract "(" <identificador>  "," <identificador>  ","
        <entero>  ")"
    goal "(" <identificador>  "," <identificador> ")"
```

```
    out "(" <identificador> "," <identificador> "," <identificador>
        "," <identificador> "," <identificador> ")"
    accept "(" <identificador>  "," <identificador>  "," <identificador>
        "," <identificador> ")"
    deletemessage "(" <identificador> "," <identificador> ")"
    append "(" <identificador> "," <identificador>  ")"
    extract "(" <identificador> "," <identificador>  ")"
    compare "(" <identificador> "," <identificador> "," <identificador>
        ")" wait "(" <entero> ")"
    <identificador> "=" <expresion_aritmetica>
    <identificador> ":=" <expresion_cadena>
<asigna_propositos> ::= <predicado> { ";" <expresiones> }
<asigna_recursos> ::= <expresiones> { ";" <expresiones> }
<expresiones> ::= <identificador> "=" <expresion_aritmetica> |
    <identificador> ":=" <expresión_cadena>
<expresion_aritmetica> ::= <termino> <operador_termino>
    <expresion_aritmetica> | <termino>
<termino> ::= <factor> <operador_factor> <termino> | <factor>
<factor> ::= "(" <expresion_aritmetica>  ")" | <entero> |
    <identificador>
<operador_factor> ::= "*" | "/"
<operador_término> ::= "+" | "-"
<condicion> ::= <termino_logico> <operador_disyuncion> <condicion> |
    <termino_logico>
<termino_logico> ::= <factor_logico> <operador_conjuncion>
    <termino_logico> | <factor_logico>
<factor_logico> ::= "(" <condicion>  ")" | true | false | <comparacion>
<operador_disyuncion> ::= "||"
<operador_conjuncion> ::= "&&"
<comparacion> ::= <expresion_aritmetica> <operador_relacional>
    <expresion_aritmetica>
<operador_relacional> ::= "==" |  "!=" | "<" | ">" | "<=" | ">="
```

2.3.2 SEMÁNTICA DE LIA

En LIA, un ambiente (como se mencionó en la sección 2.1) se compone por variables globales y regionales. Las producciones sintácticas `<global>` y `<regional>` declaran a las variables globales como una lista donde cada una tiene un tipo que puede ser entero (`int`), real (`float`), cadena de caracteres (`char`) o un puerto de comunicación entre agentes (`port`). En el caso de los puertos, únicamente pueden declararse como variables globales.

Los agentes en LIA se declaran mediante la palabra reservada `agent` seguido de la lista de variables regionales que reconoce, sus variables internas, sus propósitos iniciales, los eventos inesperados que percibe, los papeles que puede utilizar ante los eventos inesperados, y los papeles iniciales con que inicia su ejecución. Las variables internas corresponden a sus recursos y características. Para declarar las variables internas se usa la palabra reservada internal y a continuación la lista de variables precedidas por su tipo que puede ser: entero, real o cadenas de caracteres.

Los propósitos de un agente se indican mediante una lista precedida por la palabra reservada purpose. Cada propósito debe escribirse indicando el predicado, luego un guión bajo y a continuación el valor del predicado. Por ahora los predicados que se manejan en LIA se consideran con un argumento. Ejemplo de propósitos: `comprar_radio`, `ganar_100`, `encontrar_roberto`.

Las variables regionales van precedidas por la palabra reservada regional seguida de una lista de identificadores únicamente pues ya fue definido su tipo.

Los eventos inesperados que percibe un agente van en forma de lista después de la palabra reservada `percieve`. Los papeles que puede realizar el agente se indican a continuación de la palabra reservada `action`.

Cuando un agente inicia su ejecución se arrancan los papeles declarados delante de la palabra reservada `initial` en un agente, luego se invoca al planificador. Después que se tiene el plan, se activan los primeros papeles del plan.

Un sistema de LIA tiene una sección `main` donde inicia su ejecución. Esta sección es independiente y genera una hebra de ejecución independiente de cualquier interacción o agente. Típicamente contiene instrucciones de LIA para crear agentes e interacciones.

Las interacciones se indican mediante la palabra reservada `interaction` y luego, entre llaves se indican los papeles que pertenecen a ella.

Cada papel inicia con la palabra reservada `role` y un identificador, a continuación los argumentos de sus requisitos y el número máximo de instancias que pueden tenerse en paralelo de este. Por ejemplo, solamente se puede generar una instancia del papel "presidente de México", pero es posible generar varias instancias del papel "comensal" en una interacción restaurante.

Cada papel contiene a su vez, variables locales que se crean dentro de la hebra del agente.

Las instrucciones que puede contener un papel se indican en la figura 2.8.

Dado que el lenguaje que ejecuta una computadora debe ser lineal (como ensamblador), para cada declaración o instrucción de LIA se generan directivas que al interpretarse en el sistema SEA, dan origen a generar una estructura de datos en memoria o bien, realizar una operación que modifica dichas estructuras de datos.

Para generar directivas, se emplean algunas transformaciones en las instrucciones `if` y `while`. En el caso del `if` se tiene la condición y un grupo de instrucciones que van entre llaves, por lo tanto la transformación es:

INSTRUCCIÓN	TRANSFORMACIÓN LINEAL

`If( condicion)`
`{`
`  i1;`
`}`
	`evalúa condición`
	`CMP resultado 0`
	`JEQ fin-ifN`
	`I1;`
	`:fin-ifN`

En la instrucción while ocurre la transformación a un lenguaje lineal es:

INSTRUCCIÓN	TRANSFORMACIÓN LINEAL

`while( condición )`
`{`
`  i1;`
`}`
	`:inicio-whileN`
	`evalúa condición`
	`CMP resultado 0`
	`JEQ fin-whileN`
	`I1;`
	`JMP inicio-whileN`
	`:fin-whileN`

INSTRUCCIÓN	DESCRIPCIÓN
`if`	esta instrucción es de control del flujo de ejecución, si la condición que evalúa es verdadera se ejecutan las instrucciones que forman su cuerpo.
`while`	es otra instrucción de control de flujo, que hace que se repitan las instrucciones contenidas en el bloque mientras la condición que se evalúa es verdadera.
`print`	despliega el contenido de una variable o una cadena constante.
`addagent`	se utiliza para crear instancias de un agente, se coloca el identificador del agente del que se crea su instancia, luego el nombre del agente creado, algunos propósitos que se le adicionan y se asignan los valores iniciales a sus variables internas. Se usa sólo en `main`.
`addinteract`	se utiliza para crear instancias de una interacción, se indica la interacción de la que se crea la instancia y el nombre de la interacción creada. Se usa sólo en `main`.
`goal`	indica cuando un propósito se ha alcanzado, el predicado indicado se busca en los que tiene un agente y causa que los papeles subsecuentes del plan generado para el propósito, se cancelen en ejecución.
`out`	sirve para que un agente coloque un mensaje en un puerto, se explica más adelante en este capítulo.
`accept`	sirve para que un agente acepte un mensaje en un puerto, se explica mas adelante en este capítulo.
`deletemessage`	sirve para que un agente retire voluntariamente un mensaje que el mismo haya colocado en una cola.
`append`	permite concatenar cadenas, en este caso se anexa la segunda cadena en la primera, dando como resultado una cadena concatenada.
`extract`	extrae el valor de una cadena utilizando como prefijo el contenido de la primera .
`compare`	indica el resultado de comparar el contenido de las cadenas indicadas por los primeros dos identificadores y dejando el resultado en la variable que aparece como tercer identificador.
`wait`	indica que se haga una pausa de los segundos indicados en la instrucción.
expresiones aritméticas	permiten realizar cálculos aritméticos y asignarlos a una variable.
expresiones de cadena	permite asignar valores a las variables de cadena.

Figura 2.8 Instrucciones de LIA

Las directivas generadas a partir de LIA se listan en la figura 2.9 indicando en la primera columna la directiva y en la segunda columna la descripción de la misma. Se hace distinción entre mayúsculas y minúsculas.

En la interacción entre agentes, se requiere de mecanismos de comunicación. En muchos casos donde interactuan agentes de comercio electrónico, el nivel de detalle de los mensajes varía dependiendo de las necesidades de información de las entidades, en algunos casos puede resultar necesario saber si existen descuentos por volumen, si el producto puede entregarse en un punto diferente a donde lo ofertan sin cargo extra; por ejemplo, un cliente puede querer comprar unos

lentes de seguridad con un precio entre tres y cuatro dólares, mientras que otro puede requerir que los lentes tengan el marco de color oscuro y que en la compra de más de diez piezas se aplique algún descuento en el precio además, de que le entreguen el producto en su domicilio.

DIRECTIVA	DESCRIPCIÓN
G	declara una variable global de un tipo indicado; el tipo puede ser E (int), A (char), F (float), P (port).
R	declara una variable regional de un tipo indicado
L	declara una variable local de un tipo indicado
P	declara un propósito de un agente
E	declara un evento que percibe un agente
A	declara un agente
a	declara una acción que puede utilizar un agente ante cualquier evento inesperado.
S	declara una acción que debe ejecutarse cuando se crea un agente
r	declara un requisito de un papel
P	declara un papel que va dentro de la última interacción declarada
I	declara una interacción
+	declara un predicado que es efecto positivo de un papel
-	declara un predicado que es efecto negativo de un papel
m	indica el inicio del papel main
INICIO	indica el inicio de las instrucciones de un papel
STOP	indica el final de las instrucciones de un papel
Pr	indica el despliegue una cadena
PrId	indica que se despliegue el contenido de la variable indicada por el identificador
CMP R 0	indica que se compare el contenido del registro R con el valor cero, si es verdadero lo indica en el registro de resultado de la hebra en ejecución
:finif<número>	es una etiqueta que indica el final de un if
:whi<número>	etiqueta que indica el final de una instrucción while
CLRSTK	limpia la pila de ejecución de una hebra
POP	extrae un dato de la pila
PUSH	introduce un dato en la pila
JEQ	cambia la ejecución de una hebra a la posición indicada si es que la condición última evaluada tuvo el valor diferente de cero
CREAA	directiva para crear una instancia de un agente
CREAI	directiva para crear una instancia de una interacción
O	indica que se adicione un propósito en el agente indicado
SUMA	indica que se haga la suma de dos valores y el resultado lo asigna en el segundo
RESTA	indica una operación de resta dejando el resultado en segundo identificador
MULT	indica la operación de multiplicación entre dos variables dejando el resultado en la segunda
DIV	divide el contenido de una variable entre la segunda dejando el resultado en la segunda, si hay división por cero se envía un mensaje de error y se asigna como cero el resultado
WAIT	indica que se haga una pausa en una hebra
STRCAT	concatena el contenido de dos variables dejando el resultado en la segunda
MOVSTR	mueve el contenido de un identificador de tipo cadena del primero al segundo
OUT	emite un mensaje
ACCEPT	indica que se acepte un mensaje
EXTRACT	indica que se extraiga el un fragmento de un mensaje
COMPARE	compara el contenido de dos cadena y deja el resultado en la variable que corresponde al tercer identificador de esta directiva
DELMSG	indica que se borre un mensaje de la cola
PLAN	indica que se inicie el planificador para un agente
PARAM	indica los parámetros de un papel
GOAL	indica que se marque como alcanzado el propósito que se indica en esta directiva

Figura 2.9 Directivas de LIA generadas con el traductor

Para que la comunicación se de entre dos agentes es necesario que ambos manejen el mismo lenguaje o que cuenten con alguna forma de traducción para que sea un mismo lenguaje en el que se intercambien los mensajes.

Además del problema del lenguaje se tiene el de la comprensión, aquí consideramos que dos agentes se entienden si es posible asociar los componentes del mensaje a un elemento de su representación interna (concepto).

En otros casos cuando los participantes ya han manejando operaciones anteriormente, puede ocurrir que el nivel de detalle requerido para solicitar un producto sea mínimo; por ejemplo, en algún caso únicamente se requerirá de indicar el producto y la cantidad.

Esta capacidad de expresión para ser automatizada requiere de una nomenclatura en donde los agentes involucrados, puedan comprender el contenido de los mensajes; en este caso es importante que la información que se intercambia sea de tamaño variable y que considere las relación entre los diferentes componentes de los objetos descritos.

Los puertos se utilizan como mecanismo de comunicación entre agentes. Los mensajes pueden ser utilizados por el agente para proporcionarle información a otro. Las instrucciones que se utilizan para el manejo de puertos son: `out` para enviar un mensaje y `accept` para recibirlo.

Los mensajes que se intercambian pueden tener un formato que compartan los agentes, como por ejemplo KIF, KQML o FIPA-ACL. Cada mensaje emitido pasa por el COM (Comparador de Ontologías Mixtas) cuando las ontologías que utilizan los agentes son diferentes, este módulo se activa en forma transparente para la ejecución de las hebras e intenta encontrar las equivalencias entre los conceptos de las ontologías de los agentes mediante el uso de las palabras que los describen.

La instrucción `out` tiene varios parámetros una vez que se encuentra en el interprete (SEA) (figura 2.10):

```
out(puerto, identificador, mensaje, cardinalidad, duracion)
```

donde

`puerto`, es donde se coloca el mensaje, si existe algún mensaje antes del que llega el nuevo mensaje se coloca al final generando así, una cola de mensajes.

`identificador` del mensaje, es un etiqueta para poder referenciar su contenido; por ejemplo: “pedido”, “producto”, “solicitud”, etc.

`mensaje,` es el contenido en forma de una cadena de caracteres que puede contener números, letras de acuerdo a un lenguaje o protocolo de comunicación (como KQML, KIF, FIPA-ACL).

`cardinalidad`, indica cuantas veces será leído el mismo mensaje, cuando se ha leído el número de veces esperadas, entonces se retira del puerto. Mediante este atributo se maneja la distribución del mensaje a un agente (unicast) o a muchos agentes (multicast).

`duracion` (Timeout), es el tiempo en segundos que el mensaje dura en el puerto, esto mientras su cardinalidad sea mayor que cero. Cuando el tiempo se cumple, el mensaje se retira automáticamente aunque no haya sido leído.

Figura 2.10 Componentes de un mensaje de salida (`out`)

El `tipo` igual a 1 indica que es un mensaje de salida. La `marca` la utiliza el eliminador del mismo cuando se cumple el tiempo (`llegada + duracion`). El `identificador` es una etiqueta que referencia el mensaje que llega. El `emisor` apunta al agente que envió el mensaje. La `cardinalidad` es el número de agentes que pueden leer el mensaje. Las `lecturas` es el número de agentes que han leído el mensaje. La `duracion` es el tiempo en segundos que debe estar vigente el mensaje. El `contenido` es el mensaje que será leído por algún agente cuando utilice alguna instrucción `accept`.

Los mensajes esperados se reciben mediante la instrucción `accept`, estos tienen los argumentos:

```
accept(puerto, identificador, variable, duracion)
```

donde:

`puerto`, es donde se lee el mensaje.

`identificador` del mensaje, es un etiqueta que identifica al que se quiere recuperar, la analogía es que es un elemento que permite tomar uno o varios mensajes; por ejemplo, si se reciben mensajes de dos puertos diferentes, es posible recuperar pares mediante el mismo. Es alfanumérico, por ejemplo: "numeroFactura", "destino", "recursos", etc..

`variable` del agente donde se almacena el contenido del mensaje recuperado.

`duracion` (Timeout), es el tiempo en segundos que se puede esperar el mensaje. La ejecución del agente se detiene durante el tiempo especificado en duracion. Cuando el valor de `duracion` es cero, si el mensaje solicitado no está al momento de la llegada de la solicitud del agente, no se encola y continua la ejecución de la hebra del agente pero sin recibir el mensaje; de otra forma, se coloca la solicitud en el puerto a esperar un mensaje.

Cuando el `accept` se encuentra en el interprete (SEA) y tiene que esperar a que llegue algún mensaje, se suspende su hebra y se almacena la información de la figura 2.11.

Figura 2.11 Componentes de un mensaje de entrada (`accept`)

El `tipo` igual a 2 indica que es un mensaje de entrada. El `identificador` es una etiqueta que referencia el mensaje que se espera. La `variable` es donde se almacenará el mensaje en el agente que lo recibe. El `receptor` apunta al agente que espera recibir el mensaje. La `duracion` indica el tiempo que esperará el agente antes de omitir el mensaje y continuar con la ejecución de su hebra. `hebra` apunta a la hebra del agente que espera el mensaje.

Cuando las entradas estén "ligadas", es necesario que los puertos reciban mensajes coloreados (las entradas ligadas tienen el mismo color —identificador—), como en el caso de un matrimonio, en donde por alguna razón el marido puede demorar más en llegar al puerto de entrada "hombres", en tanto que la esposa ya llegó al puerto de entrada "mujeres" de la hebra. Cuando se deba dar entrada a las parejas (o eneadas) completas, el postulado `accept` debe leer de varios puertos simultáneamente eneadas completas del mismo color (identificador), en tanto que las incompletas se quedan esperando, por ejemplo: marido_rojo, marido_verde; si llegua primero esposa_verde, la instrucción que acepta el mensaje del puerto sabrá encontrar al marido apropiado en este caso el verde (marido_verde).

2.4 SISTEMA DE EJECUCIÓN DE AGENTES (SEA)

El Sistema de Ejecución de Agentes (SEA) tiene el propósito de efectuar la ejecución de la simulación de las interacciones entre agentes establecidos en un sistema escrito en LIA (véase la sección 2.3) que tiene los elementos del Modelo de Interacción entre Agentes (MIA) (véase la sección 2.1). El procesamiento de un sistema escrito en LIA (figura 2.12) se hace tomando la descripción de un sistema contenida en archivo de texto; a este, el traductor lo convierte a una serie de directivas (equivalente a un lenguaje ensamblador), el constructor toma las directivas para crear estructuras de datos en la memoria del ejecutor que representan recursos, agentes, interacciones, contenido de los papeles; el ejecutor entonces crea una cola de ejecución, en donde primero coloca el papel main y a partir de este las hebras de los agentes que se crean después de pasar por el planificador. El ejecutor llama cuando corresponde al planificador, comparador de ontologías y al manejador de eventos inesperados..

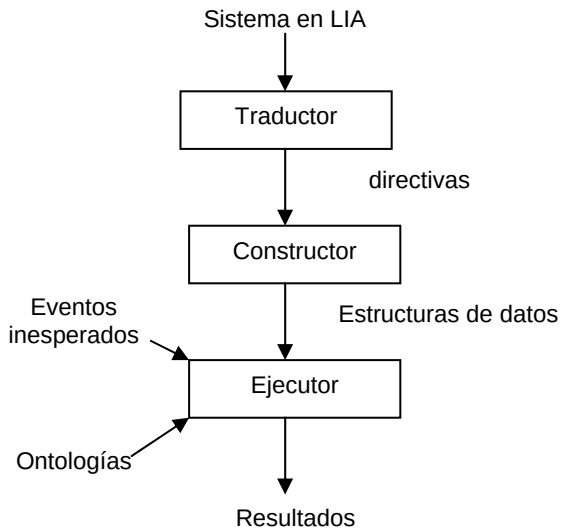


Figura 2.12 Diagrama del flujo que sigue un sistema escrito en LIA para ejecutarse

El Diagrama de Flujo de Datos (DFD) de los componentes de SEA se ilustra en la figura 2.13 y cada uno de sus módulos se describen en las siguientes subsecciones. Los módulos de ingreso de información son:

a) El *editor de sistemas*, permite editar archivos de texto con la definición de un sistema de agentes que incluye su ambiente, los agentes, la sección principal y las interacciones con sus papeles. Con este mismo módulo se editan los papeles de emergencia y el diccionario que relaciona conceptos con palabras para el comparador de ontologías.

b) El *editor de ontologías*, es el módulo que permite editar la estructura de árbol de los conceptos que forman una ontologías.

c) El *generador de eventos inesperados*, permite crear varios archivos conteniendo eventos inesperados, en cada uno se indican los datos del evento: fecha (d/m/a) y tiempo (h:m:s) de su ocurrencia y su descripción.

El traductor toma un sistema, lo divide en unidades léxicas, valida que sean las propias del lenguaje LIA y en caso que sea correcto léxicamente procede a revisar que la organización de las palabras esté congruente con la gramática de LIA, si la cumple, se genera una lista de directivas que es el programa en forma lineal.

El Ejecutor tiene dos partes, en la primera (llamado constructor), toma las directivas generadas en el módulo anterior para construir las estructuras de datos que se utilizarán durante la ejecución. En la segunda parte, se inicializa la cola de ejecución con los eventos inesperados y el proceso principal del sistema (`main`). Durante la ejecución se crean las instancias de agentes e interacciones. Cada vez que se crea un agente se invoca al planificador para que obtenga un plan para el agente. Cuando se ejecuta alguna instrucción de comunicación entre agentes, se llama al comparador de ontologías mixtas para que mapee el mensaje en término de los conceptos del agente receptor. El manejador de eventos inesperados se invoca cuando inicia o termina cada uno de estos.

Los archivos que se utilizan durante la traducción, construcción y ejecución de un sistema de agentes son:

a) *Sistema en LIA*, contiene los elementos que forman un sistema de agentes e interacciones.

b) *Papeles de emergencia*, son los papeles que utilizan los agentes cuando se presentan eventos inesperados, tienen extensión `.rol` y respetan el código LIA.

c) *Unidades Léxicas*, tiene las palabras reservadas de LIA.

d) *Gramática*, contiene las producciones gramaticales utilizadas en el análisis sintáctico de programas en LIA.

e) *Directivas*, contiene las directivas del sistema traducido.

f) *Eventos Inesperados*, tiene las tablas de eventos inesperados que han de ocurrir durante la ejecución de un ambiente de agentes e interacciones.

g) *Ontologías*, está formado por un archivo de palabras-concepto y otro con un árbol de conceptos y sus características.

h) *Resultados*, contiene la secuencia de eventos que efectuados durante la ejecución de la simulación de un sistema de agentes.

Los módulos que componen a SEA (figura 2.13) que se desarrollan en este trabajo se indican con color azul, los archivos y las tablas con color amarillo. Las flechas indican las entradas y salidas de cada módulo. Los módulos y tablas que aparecen de color blanco corresponden al trabajo de tesis de [Domínguez 2001] pero forman parte del sistema de ejecución de agentes, motivo por el cual se indican en el diagrama.

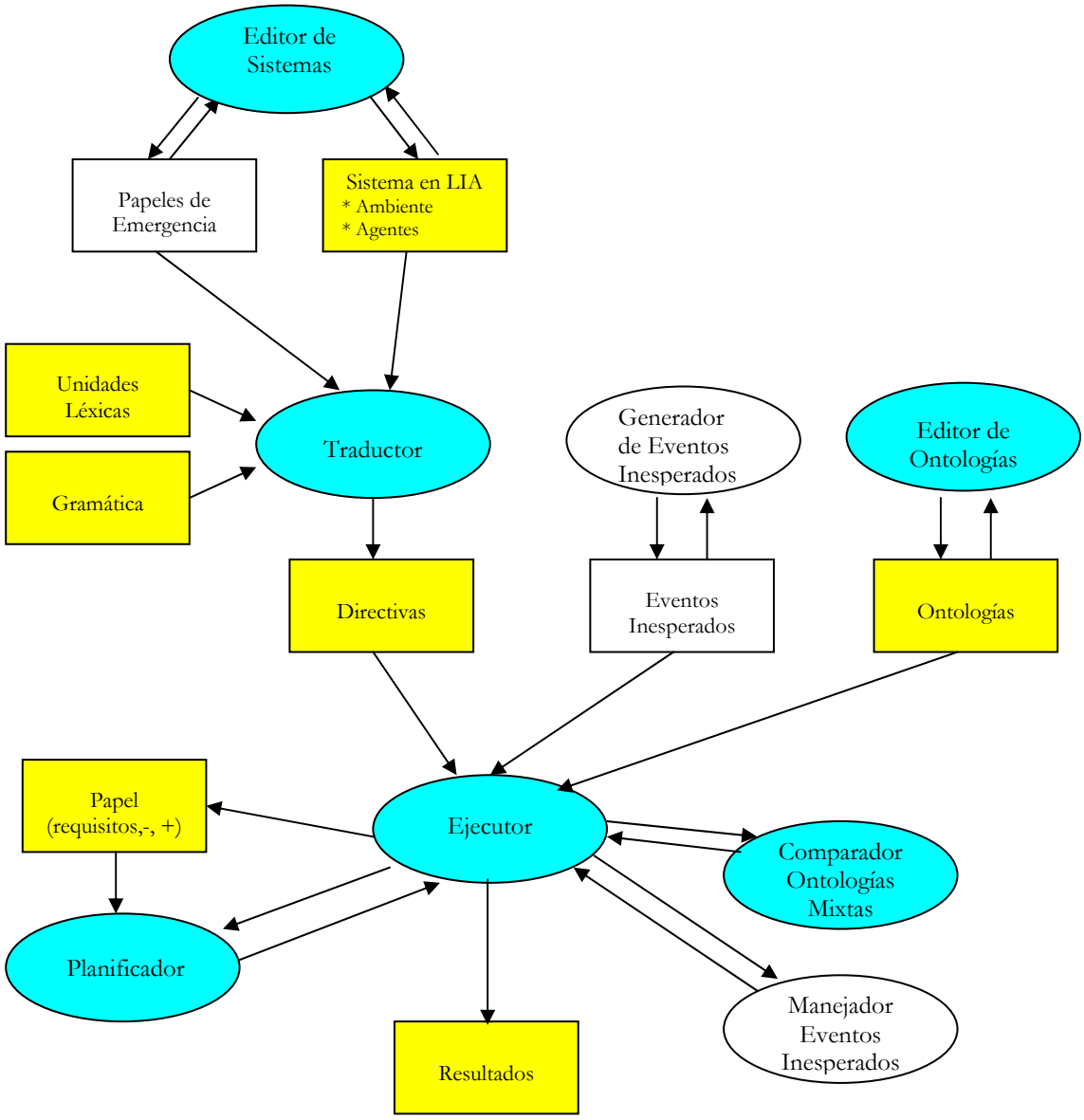


Figura 2.13 Módulos del Sistema de Ejecución de Agentes (SEA)

2.4.1 EDITOR DE SISTEMAS

El *Editor de sistemas* es un módulo que permite editar archivos de texto, en este caso los sistemas descritos en LIA, los papeles de emergencia y los archivos de palabras-concepto utilizados por el comparador de ontologías mixtas.

Consta de una pantalla (figura 2.14) con una barra de menús y un área de edición. Las opciones de la barra de menú son: `File`, `Edit`, `LIA-System`, `Unexpected-Events`, `Ontologies`.

Con la opción `File` se puede abrir, guardar, guardar como y salir. La opción `Edit` permite cortar, pegar y buscar textos en un archivo. La opción `LIA-System` se utiliza para invocar al traductor de sistemas descritos en LIA y al ejecutor. La opción `Unexpected-Events` contiene las subopciones para compilar los papeles de emergencia y generar los eventos inesperados. La opción `Ontologies` permite editar los árboles de ontologías y ejecutar el comparador de ontologías mixtas fuera del ambiente de ejecución de un sistema completo.

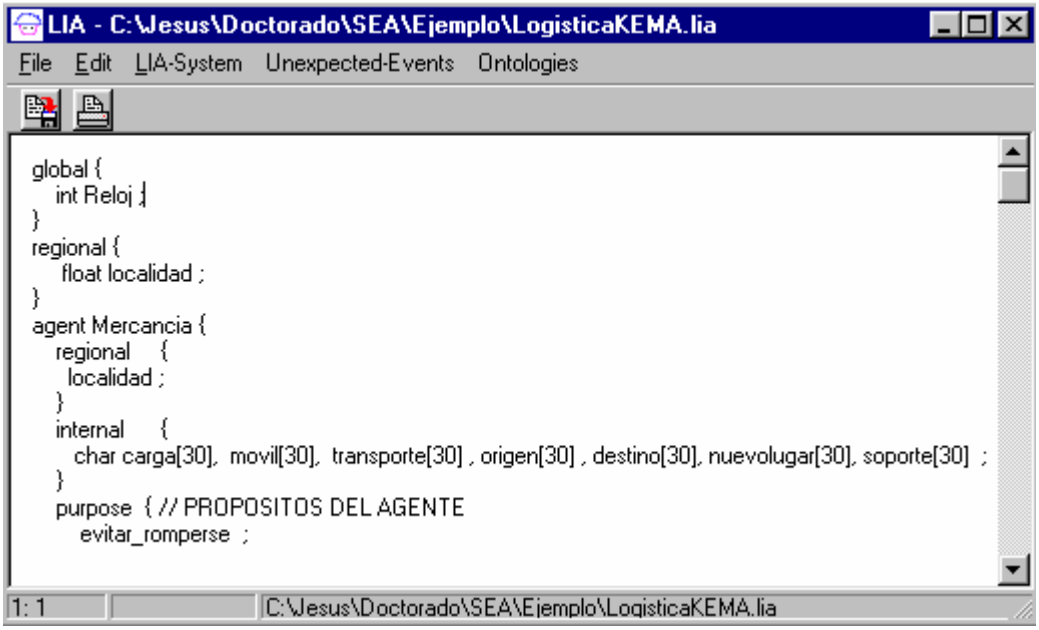


Figura 2.14 Editor de sistemas escritos en LIA, consta de un menú y un área de texto

Generalmente cuando se realiza un sistema de LIA se crea un documento con la opción `File/New` y se guarda con `Save`, a continuación se crean los eventos inesperados y los archivos de ontologías que utilizan los agentes. El sistema se traduce para generar el archivo de directivas y se invoca al ejecutor. El ejecutor interactúa con el planificador para generar el plan de cada agente que se crea, el planificador toma la información de los papeles de las interacciones activas y los propósitos de un agente.; con el comparador de ontologías mixtas se busca un concepto equivalente de una ontología con otra, cada vez que se intercambia un mensaje entre agentes. Cuando se invoca el mapeador de ontologías, este accede a las variables internas del agente para determinar cual es la ontología de cada uno y en base en esta, tomar el árbol de conceptos correspondiente a cada uno y su diccionario de palabras. Cuando se activan los eventos inesperados se invoca al manejador de eventos inesperados. Los resultados de la ejecución se almacenan en un archivo en donde se indica la fecha y la hora de cada acción que ocurrió durante la ejecución del sistema en LIA y reporta el

número de propósitos que tuvo cada uno y cuantos fueron alcanzados con la intención de dar un índice del grado de satisfacción de cada agente.

2.4.2 EDITOR DE ONTOLOGIAS

Con el editor de ontologías se crea o modifica un árbol de conceptos en donde cada uno contiene sus características expresadas también como conceptos. El árbol se almacena como un archivo con extensión `.tree`, el nombre externo con que se guarda coincide con el contenido de la variable interna `ontology` de cada agente. Las relaciones entre palabras y conceptos se almacenan en otro archivo aparte, debido a que pueden existir mapeos 1:1, 1:M y N:M, donde el primer 1 y N es número de palabras y el segundo 1 y M es el número de conceptos, el nombre de este archivo lleva el mismo nombre de la ontología y la extensión `.dic`.

En la figura 2.15 se muestra un ejemplo de una pantalla donde se esta edita un árbol de conceptos. El boton `LOAD` permite cargar algún árbol existente, el botón `SAVE` almacena los cambios generados en un árbol editado (si es la primera vez se crea el archivo del árbol). El botón `ADD` permite adicionar un concepto con sus propiedades escrito en los campos de texto Concept y Properties. El botón `CHANGE` permite respectivamente realizar cambios en un concepto y sus propiedades. El botón `DELETE` elimina un concepto seleccionado (con una barra azul). El botón `EXIT` sale de este editor.

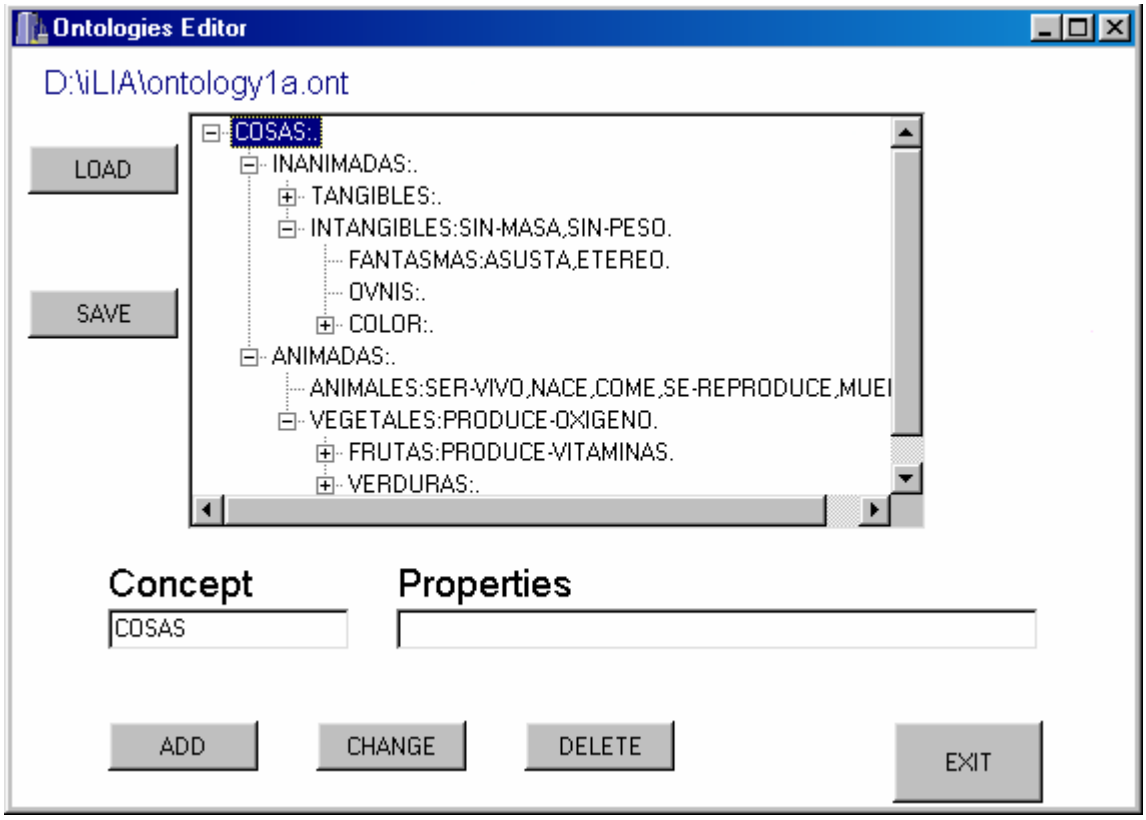


Figura 2.15 Un ejemplo del editor de ontologías los botones son para editar agregando nodos, eliminando o cambiando. En este caso los conceptos se escriben con mayúsculas.

2.4.3 GENERADOR DE EVENTOS INESPERADOS

La opción de eventos inesperados (`Unexpected-Events`) de la pantalla principal, consta de tres opciones, el editor del árbol (figura 2.16a) de eventos inesperados que se almacenan en un archivo con extensión `.ue`, la edición de eventos individuales (figura 2.16b) mismos que se almacenan en un archivo con extensión `.uef` y la invocación al traductor de papeles de emergencia a directivas (similar a la traducción de sistemas en LIA pero en este caso sólo se traduce el papel de emergencia).

Cada nodo del árbol de eventos inesperados (figura 2.16a) se compone del nombre del evento y una lista de papeles que se utilizan para que un agente lo supere. Estos papeles se buscan en las reacciones que tiene un agente y en caso que coincidan se procede a adicionarla como una hebra más del agente, en este caso la hebra de emergencia causa la suspención de hebras normales del agente debido a que las de emergencia tienen prioridad sobre las normales.

Un evento inesperado se compone de (figura 2.16b) la fecha, tiempo de ocurrencia, su nombre y duración en segundos (esto es porque si dura semanas o años es posible mediante aritmética transformar a segundos).

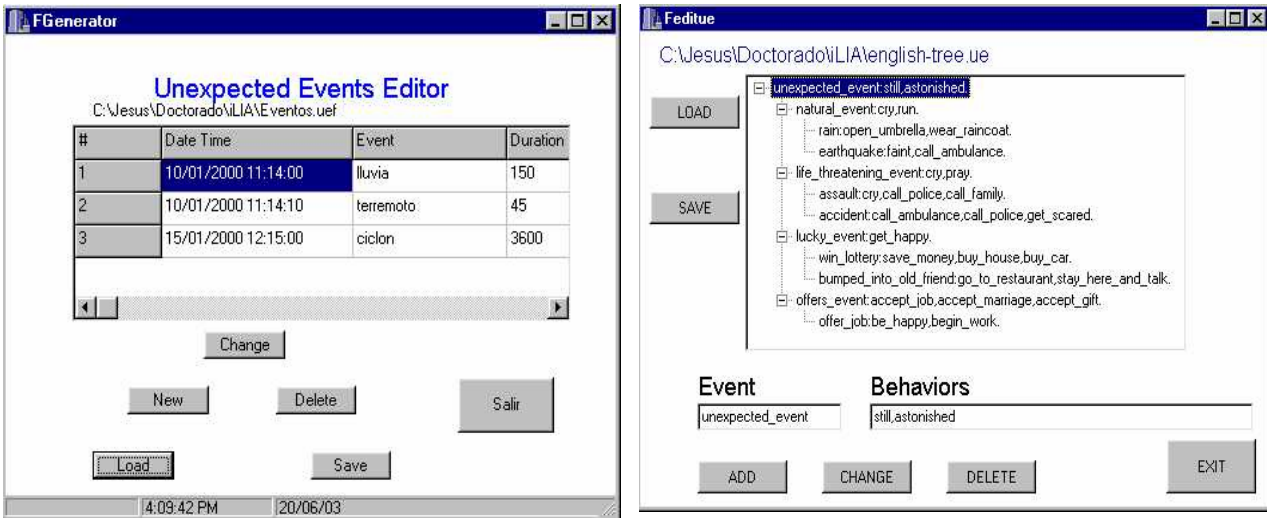


Figura 2.16 Pantallas del editor de eventos inesperados. En a) es el editor del árbol y en b) el editor de eventos

2.4.4 TRADUCTOR

El traductor (figura 2.17) toma un ambiente de interacción entre agentes escrito en LIA o algún papel de emergencia; carga las unidades léxicas y la gramática de un sistema o de un papel de emergencia según el caso para realizar la traducción a directivas. El archivo de entrada lo separa en unidades léxicas verificando que sean válidas. Si existe algún error léxico, se detiene el proceso de traducción. Después de verificar las unidades léxicas, se usa la gramática para revisar la sintaxis del archivo en proceso. Si hay errores de sintaxis, se detiene el proceso. Si la sintaxis es correcta entonces se generan las directivas y se deja el resultado en un archivo que utilizará el Cargador/Ejecutor.

El traductor es dirigido por sintaxis (si se cambia el archivo de la gramática se cambiar el lenguaje que reconoce el traductor). Las directivas corresponden al sistema descrito en LIA pero en forma lineal.

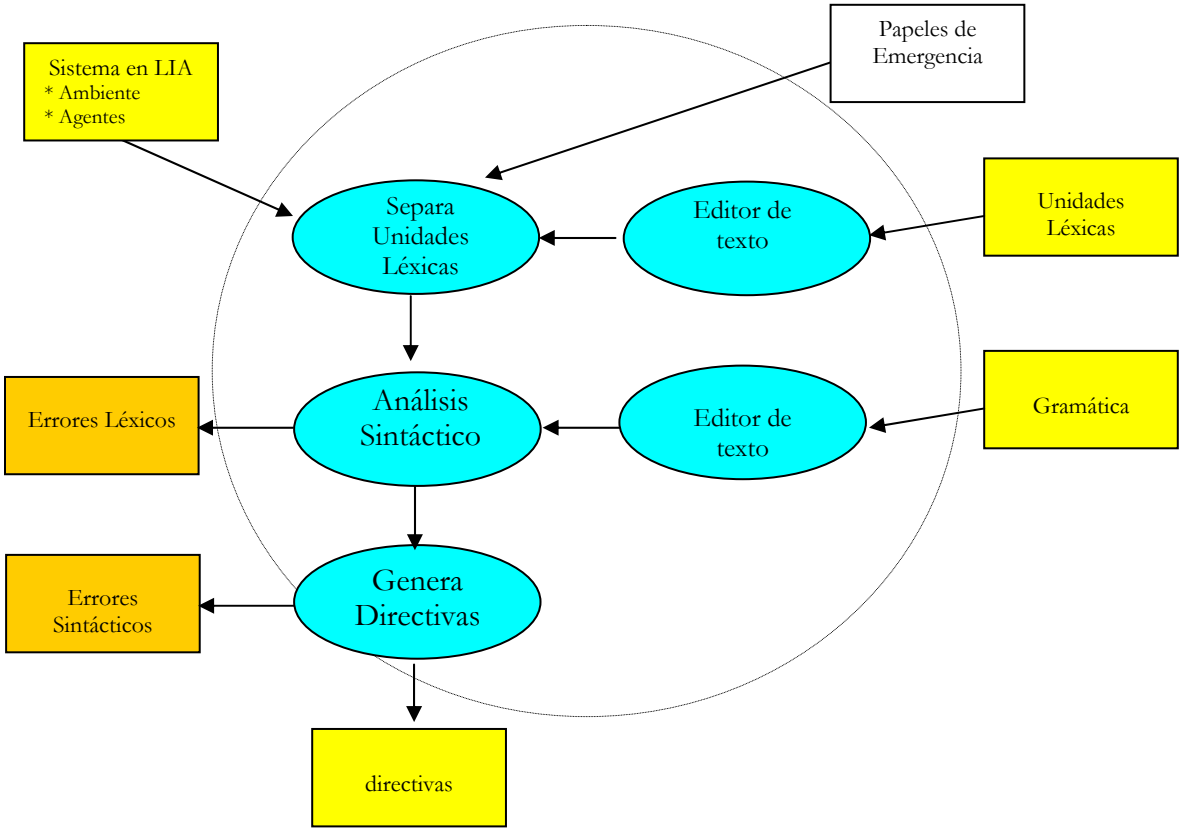


Figura 2.17 Componentes del Traductor. Los ovalos azules son procesos, los rectángulos amarillos son archivos y los naranja son mensajes. El círculo punteado es el traductor. El cuadrado blanco es del módulo de eventos inesperados

La traducción se hace dividiendo el archivo de entrada en unidades léxicas, las cuales se clasifican en las categorías:

a) Palabras reservadas
b) Cadenas
c) Números
d) Hora

Después de asignarle su categoría a cada unidad léxica (análisis léxico) se tienen pares (unidad léxica, categoría) a los que se aplica el análisis sintáctico para determinar la validez de su estructura respecto a la gramática.

El análisis sintáctico se hace mediante una gramática siguiendo un proceso Top-Down a partir de un símbolo terminal inicial. La gramática completa de LIA se encuentra en el Anexo B.

El analizador sintáctico toma una producción correspondiente a un símbolo no terminal, la primera vez toma el símbolo `INICIO`, y revisa cada uno de los símbolos de la producción, si es un símbolo terminal se compara contra el símbolo de la categoría léxica, este debe ser el mismo, si es diferente marca error de sintaxis; si es un símbolo no terminal se invoca recursivamente al analizador sintáctico pero ahora con el nuevo símbolo no terminal; si es una rutina semántica se almacena una acción en una tabla; si es un error se reporta y se detiene el proceso.

Al terminar el análisis sintáctico se tiene una tabla de acciones; la cual, sirve para generar las directivas que utilizará el constructor. En el caso de la estructura if-then-else siguiente:

```
if( dato > 5 )
{
    print("es mayor");
}
else
{
    print("es menor o igual");
}
```

se convierte en las directivas:

```
        CMP>  dato 5
        PUSHs R
        POPs  R
        CMP   R 0
        JEQ   elseif
        Pr    $0
        JMP   finif
:elseif
        Pr    $1
:finif
```

Se tienen directivas para cada componente o instrucción del lenguaje LIA.

Al realizar la traducción aparece una pantalla (figura 2.18) donde se indica el número de líneas para el análisis léxico del sistema, el porcentaje de avance del análisis sintáctico. En una caja de texto se indican los eventos ocurridos durante la traducción; como el inicio, la carga de palabras reservadas, la carga de la gramática y la validación de la sintaxis correcta o los errores existan. Si no hay errores léxicos o sintácticos en la traducción, entonces se genera el archivo `prog.asm` que contiene las directivas.

2.4.5 EJECUTOR

Este módulo utiliza las directivas generadas por el traductor para construir en memoria las estructuras de datos utilizadas en la creación de agentes con sus propósitos, recursos y características

representados estos con variables internas y regionales; las interacciones con sus papeles; cada papel contiene su cupo, requisitos e instrucciones. El ejecutor consiste de dos módulos: el constructor y el intérprete (figura 2.19).

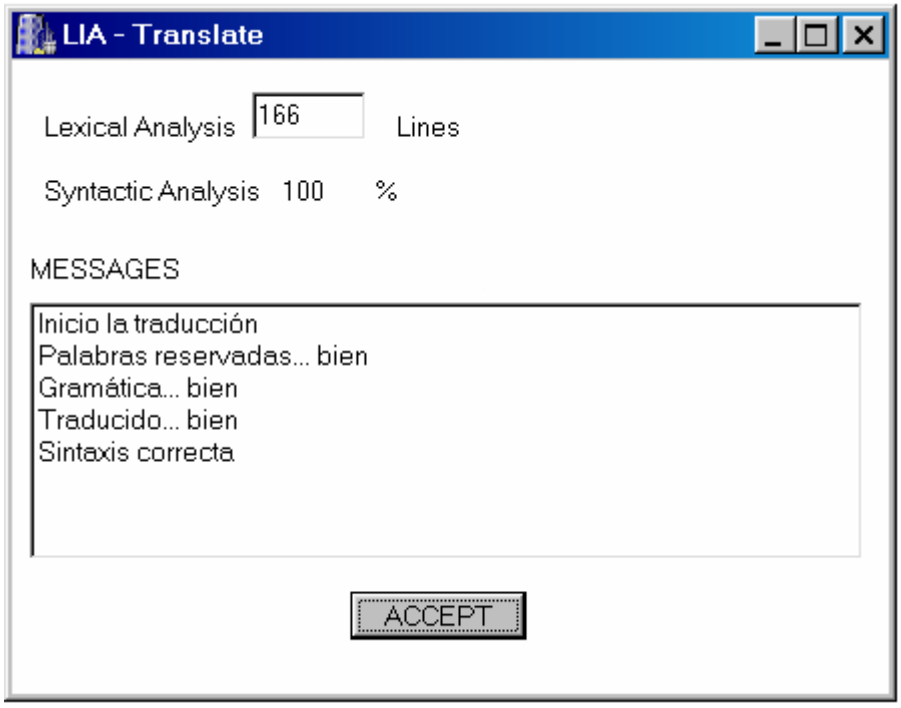


Figura 2.18 Pantalla del traductor en donde se indica que un sistema escrito en LIA está correcto

2.4.5.1 Constructor

Este módulo utiliza las directivas generadas por el traductor para construir en memoria las estructuras de datos que representan a los componentes de LIA, como son, variables globales y regionales, agentes e interacciones (escenarios). A su vez los componentes de cada uno de estos. En los Agentes, sus variables internas, propósitos, eventos inesperados percibidos, acciones para atender los eventos inesperados y los papeles iniciales. En las interacciones, sus papeles con los requisitos y sus instrucciones.

La descripción de cada una de las componentes que se utilizan en la creación de las estructuras de datos se indican a continuación con su nombre y los atributos que contiene cada nodo. Las variables globales y las regionales se almacenan en listas de variables. Los agentes e interacciones se almacenan en forma de listas. Este módulo deja listo el ambiente para la ejecución de instrucciones y la carga de eventos inesperados.

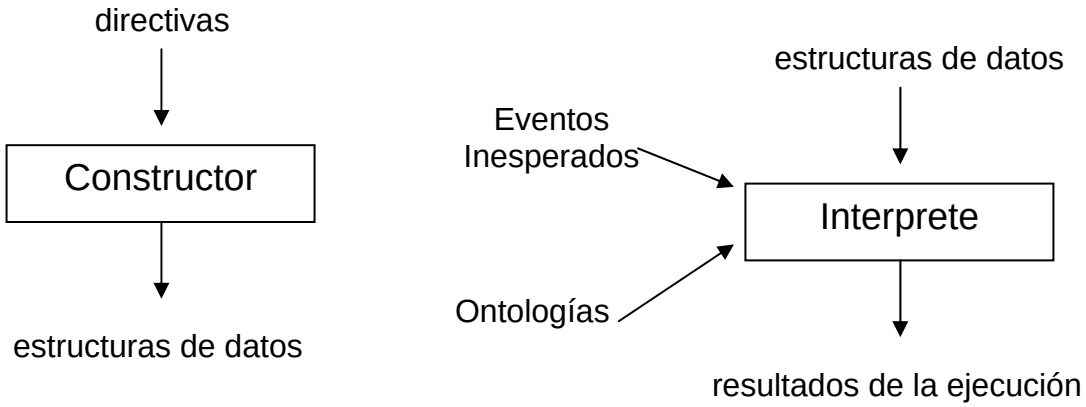


Figura 2.19 Módulos del ejecutor: constructor e interprete

Cada una de las variables globales (accesibles a cualquier hebra activa) tiene la estructura siguiente:

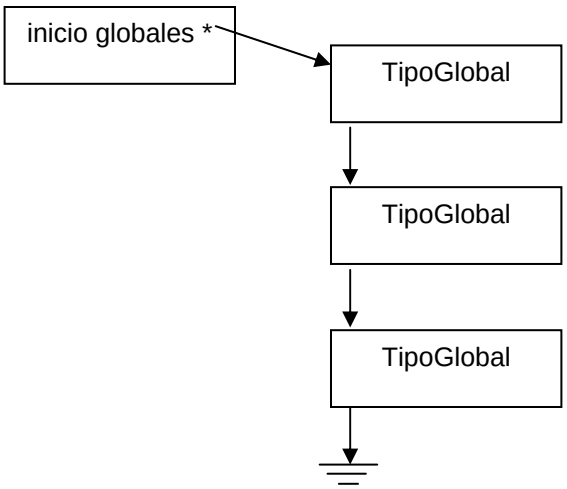
TipoGlobal

`char nombre[MAX_PALABRA] ;`	Nombre de la variable
`char tipo ;`	Tipo = char, int, float
`int dimension1;`	Utilizada cuando hay arreglos de una dimensión
`int dimension2;`	Utilizada para arreglos bidimensionales

`valor`
`  long entero;`
`  double flotante;`
`  char byte;`
`  char *cadena;`
`  long *enteros;`
`  double *flotantes;`
	Área de memoria que se debe reservar, en el caso de los apuntadores, la cantidad es el producto de `dimension1 * dimension2`

`long tiempo;`	Disponibilidad a partir de este tiempo
`struct EstructuraGlobal *sig ;`	Apuntador a la variable siguiente

Las variables globales se guardan en una lista ligada, donde un apuntador hace referencia al inicio de la misma. Para cada una se asigna su nombre, tipo, longitud y se reserva la memoria de acuerdo al tipo de variable que contiene:



Las variables regionales (accesibles a las hebras de los agentes que las declaran) tienen la estructura:

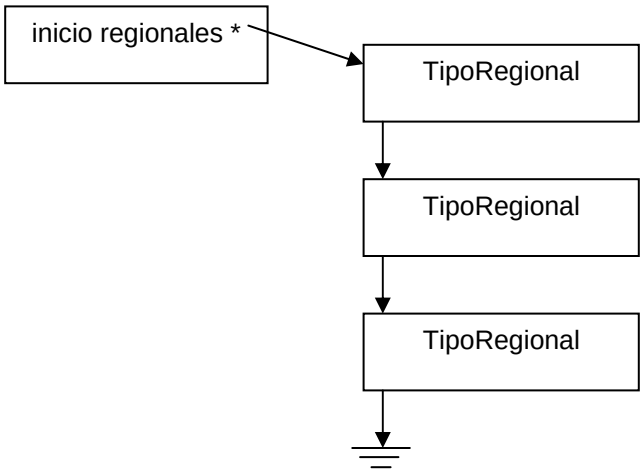
TipoRegional

`char nombre[MAX_PALABRA] ;`	Nombre de la variable
`char tipo ;`	Tipo = char, int, float
`int dimension1;`	Utilizada cuando hay arreglos de una dimensión
`int dimension2;`	Utilizada para arreglos bidimensionales

`valor`
`long entero;`
`double flotante;`
`char byte;`
`char *cadena;`
`long *enteros;`
`double *flotantes;`
	Área de memoria que se debe reservar, en el caso de los apuntadores, la cantidad es el producto de `dimension1 * dimension2`

`long tiempo;`	Disponibilidad a partir de este tiempo
`struct EstructuraRegional *sig ;`	Apuntador a la variable siguiente

Las variables regionales se guardan en una lista ligada, a cada variable se le asigna su nombre, tipo y se reserva la memoria necesaria para representar el tipo de valor que contiene.



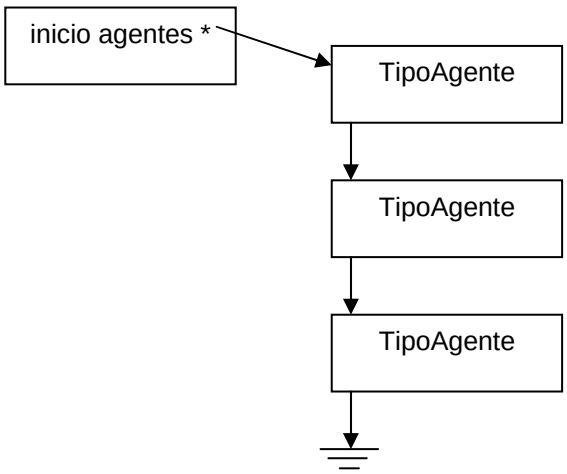
Los agentes tienen varios componentes: recursos, características, propósitos que se indican en la estructura TipoAgente. Cada agente puede llevar a cabo un plan mediante la ejecución en paralelo de papeles, algunos papeles se le asignan como iniciales, otros los toma de las interacciones y otros más son hebras de emergencia.

Cada agente se representa internamente en el ejecutor por una estructura de datos como la que sigue:

TipoAgente

`long id ;`	Identificacion unica de cada agente
`char nombre[MAX_PALABRA] ;`	Nombre del agente
`TipoProposito *proposito ;`	Lista de propositos
`TipoInterna *interna ;`	Lista de variables internas
`TipoVRegion *regional ;`	Lista de variables regionales accesables
`TipoEvento *evento ;`	Lista de eventos que percibe
`TipoAccion *accion ;`	Lista de acciones que puede usar en EI
`TipoInicial *inicial ;`	Lista de papeles con los que nace
`TipoPlan *plan ;`	Plan que puede desarrollar
`TipoIdHebra *hebra ;`	Lista de papeles activos (hebras)
`struct EstructuraAgente *instancia ;`	Lista de instancias de este agente
`struct EstructuraAgente *agente ;`	Agente padre del actual
`struct EstructuraAgente *sig ;`	Apunta al siguiente agente

Los agentes se guardan en una lista ligada donde un apuntador indica el inicio de la misma. Cada agente a su vez, puede tener instancias, mismas que heredan los componentes de un agente (variables internas, propósitos, etc.):

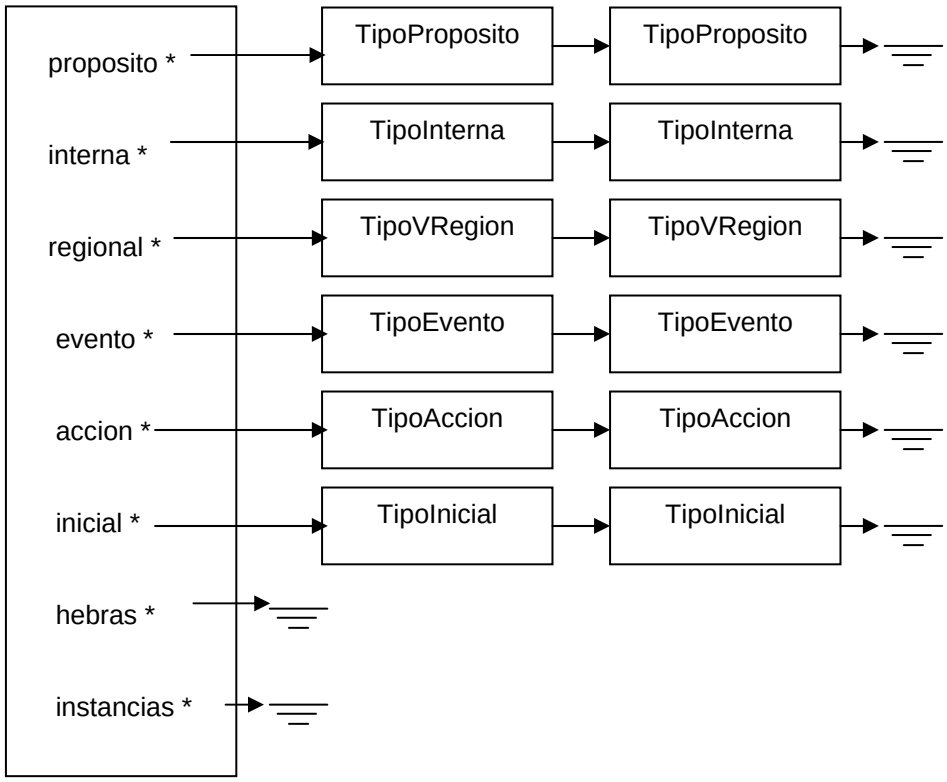


Cada componente de un agente se representa con listas de estos, el apuntador inicial de cada una de estas listas se encuentra en el nodo del agente. Las listas que forman a un agente son:

a) propósitos
b) variables regionales que el agente puede referenciar
c) eventos inesperados que percibe
d) acciones que puede usar para reaccionar ante los eventos inesperados
e) papeles iniciales con los que nace el agente.

Estas listas cuando se han cargado las directivas de un agente generan una estructura de datos como la siguiente:

TipoAgente



Las instancias de agentes (`addagent`) y las hebras se generan en momento de la ejecución del interprete.

Cada variable interna de un agente se representa mediante un nodo como el siguiente:

TipoInterna

`char nombre[MAX_PALABRA] ;`	Nombre de la variable
`char tipo ;`	Tipo = char, int, float
`int dimension1;`	Utilizada cuando hay arreglos de una dimensión
`int dimension2;`	Utilizada para arreglos bidimensionales

`valor`
`long entero;`
`double flotante;`
`char byte;`
`char *cadena;`
`long *enteros;`
`double *flotantes;`
	Área de memoria que se debe reservar, en el caso de los apuntadores, la cantidad es el producto de `dimension1 * dimension2`

`struct EstructuraInterna *sig ;`	Apuntador a la variable interna siguiente

Cada variable regional que puede accesar un agente tiene la estructura:

TipoVRegion

`char variable[MAX_PALABRA] ;`	Nombre de la variable
`struct EstructuraVRegion *sig ;`	Apuntador a la siguiente variable regional

Los propósitos de un agente se representan mediante nodos conteniendo la estructura:

TipoProposito

`char proposito[MAX_PALABRA] ;`	Nombre del propósito (predicado)
`int estado;`	0=sin cumplir, 1=cumplido
`struct EstructuraProposito *sig ;`	Apuntador al propósito siguiente

Los eventos inesperados que percibe un agente se agrupan en una lista formada por nodos de la forma:

TipoEvento

`char evento[MAX_PALABRA] ;`	Nombre del evento inesperado
`struct EstructuraEvento *sig ;`	Apuntador al evento inesperado siguiente

Las reacciones que un agente tiene para responder ante los eventos inesperados consta de la estructura:

TipoAccion

`char accion[MAX_PALABRA] ;`	Nombre de la acción
`struct EstructuraAccion *sig ;`	Apuntador a la siguiente acción

Los papeles con que inicia su ejecución un agente tienen la estructura:

TipoInicial

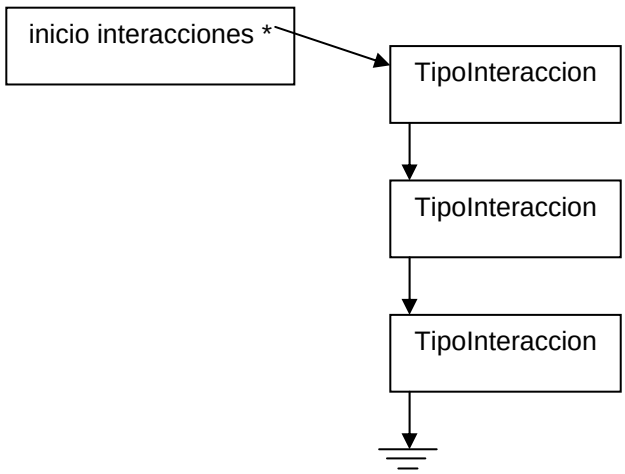
`char inicial[MAX_PALABRA] ;`	Nombre del papel inicial
`struct EstructuraInicial *sig ;`	Apuntador a la siguiente acción inicial

La identificación de las hebras que apuntan a las hebras de un agente tienen la forma:

TipoIdHebra

`TipoHebra *hebra ;`	Apuntador a la hebra del agente
`Struct EstructuraIdHebra *sig ;`	Apuntador al siguiente identificador de hebra

Las interacciones se forman por un conjunto de papeles y cada uno contiene a su vez, una lista de requisitos y sus instrucciones. Las interacciones forman una lista ligada referenciadas por un apuntador al inicio de esta:

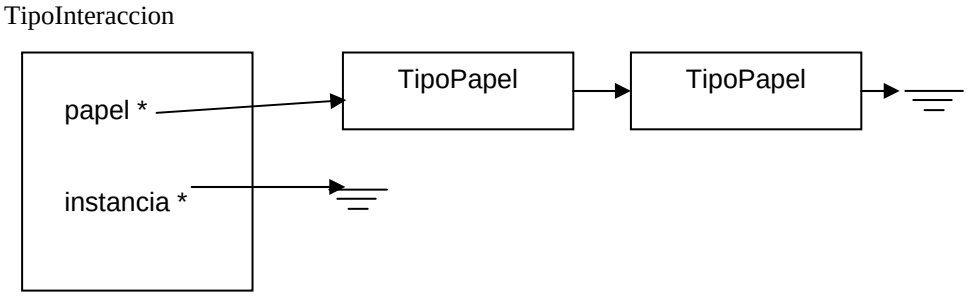


Las instancias de las interacciones se crean durante la ejecución mediante la instrucción de LIA `addinterac`. El nodo describe a cada interacción tienen la estructura siguiente:

TipoInteraccion

`char nombre[MAX_PALABRA] ;`	Nombre del espacio de interacción
`TipoPapel *papel ;`	Apuntador al inicio de los papeles
`struct EstructuraInteraccion *instancia ;`	Apuntador a las instancias de la interacción
`struct EstructuraInteraccion *sig ;`	Apuntador a la siguiente interacción

Cada interacción contiene papeles que forman una lista ligada, el inico de esta se encuentra en el nodo de la interacción como se indica a continuación:



A su vez cada papel consta de varias listas: variables locales, requisitos, y además tiene sus instrucciones. El nodo de cada papel tienen la estructura siguiente:

TipoPapel

`char nombre[MAX_PALABRA] ;`	Nombre del papel
`int cupo ;`	Máximo de instancias de este papel
`int usados ;`	Instancias creadas
`TipoRequisito *requisito ;`	Lista de requisitos
`TipoLocal *local ;`	Lista de variables locales
`int u_instruccion ;`	Número de instrucciones
`TipoInstruccion *instruccion ;`	Instrucciones LIA del papel
`struct EstructuraPapel *sig ;`	Apuntador al papel siguiente

Los requisitos de cada papel tienen la estructura:

TipoRequisito

char requisito[MAX_PALABRA] ;	Descripción del requisito
struct EstructuraRequisito *sig ;	Apuntador al requisito siguiente

La estructura de las variables locales tiene la estructura:

TipoLocal

`char nombre[MAX_PALABRA] ;`	Nombre de la variable
`char tipo ;`	Tipo = char, int, float
`int dimension1;`	Utilizada cuando hay arreglos de una dimensión
`int dimension2;`	Utilizada para arreglos bidimensionales

`valor`
`long entero;`
`double flotante;`
`char byte;`
`char *cadena;`
`long *enteros;`
`double *flotantes;`
	Área de memoria que se debe reservar, en el caso de los apuntadores, la cantidad es el producto de `dimension1 * dimension2`

`struct EstructuraLocal *sig ;`	Apuntador a la variable local siguiente

Las instrucciones que forman a un papel tienen la estructura:

TipoInstruccion

`char opcode[8];`	Operación
`char op1[MAX_PALABRA];`	Primer operando
`char op2[MAX_PALABRA];`	Segundo operando
`char op3[MAX_PALABRA];`	Tercer operando

La sección principal de ejecución contiene un apuntador al papel `main` y otro apuntador a la hebra que corresponde al mismo papel `main`. Los Eventos Inesperados se adicionan como elementos de las listas papel y hebra. El nodo que corresponde al elemento principal es:

TipoMain

TipoPapel *papel ;	Apuntador a las instrucciones de `main`
TipoHebra *hebra ;	Apuntador a la hebra de `main`

Una vez que se han aplicado las directivas en el constructor, asignándole memoria a los componentes de un sistema descrito mediante LIA, el ambiente está listo (figura 2.20) para que se inicie el proceso de interpretación donde se cargan los eventos inesperados y se crean instancias de agentes e interacciones.

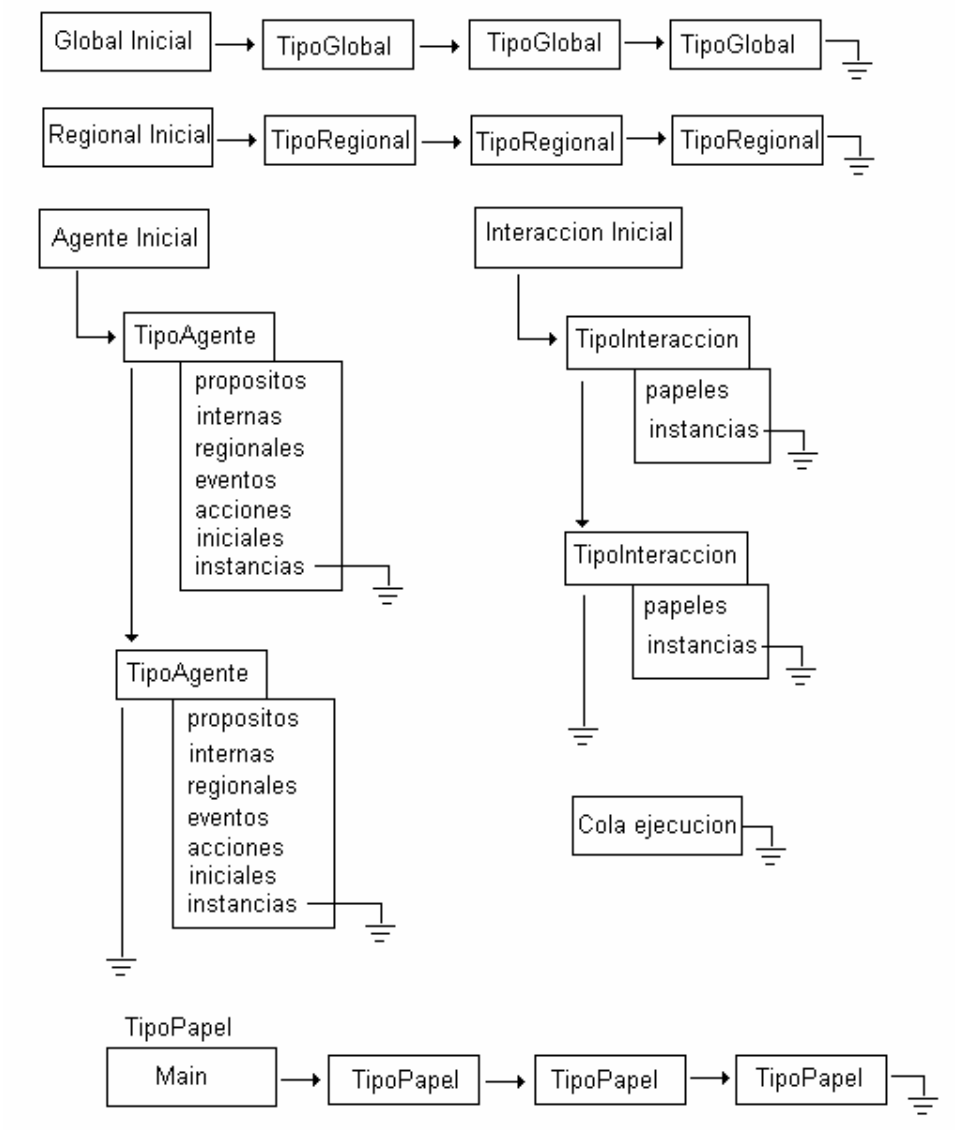


Figura 2.20 Ambiente generado por el Constructor usando las directivas de un sistema escrito en LIA

2.4.5.2 Interprete

El interprete es el módulo del SEA que aprovecha el ambiente creado por el constructor y lleva a cabo la ejecución de las instrucciones del papel `main` y de los papeles que toman los agentes descritos en los programas de LIA. Cuando inicia el ejecutor se despliega la pantalla mostrada en la figura 2.21, en donde la ejecución inicia cuando el usuario presiona el botón `start`, mientras tanto, el usuario puede colocar el inicio de la simulación en el reloj que se encuentra en la parte superior izquierda, cargar los eventos inesperados que ocurrirán (`Load Events`) y el árbol de atención a eventos inesperados (`Load Tree`). Cada acción ocurrida durante la ejecución de las directivas se visualiza en la ventana de acciones (abajo del letrero Current Action). En la pantalla en la parte inferior derecha está la ventana donde se despliegan las etapas del proceso de planeación. En la parte derecha se encuentran algunas ventanas donde aparecen las variables globales y regionales, los agentes y las interacciones que se crean como resultado de la ejecución de las directivas.

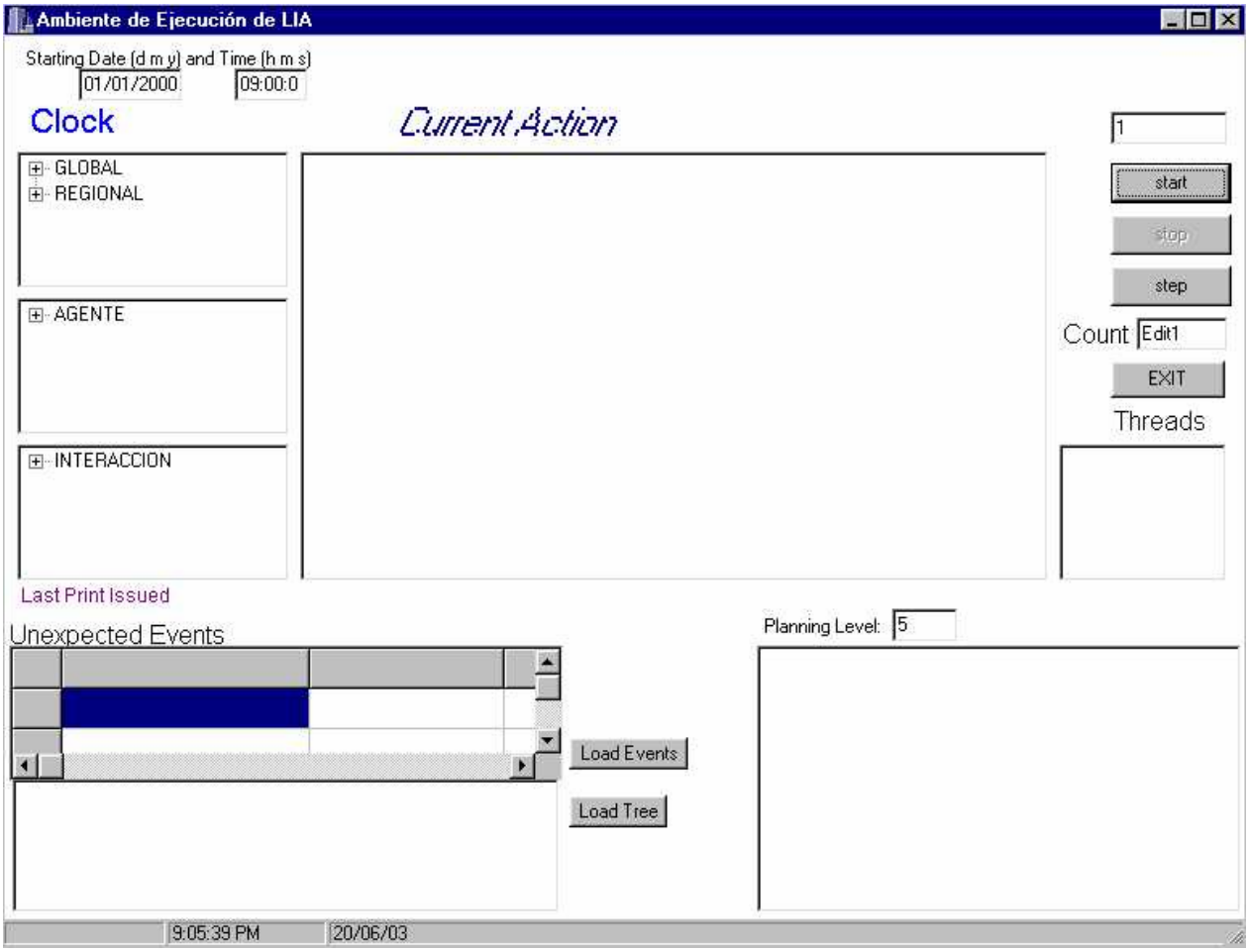
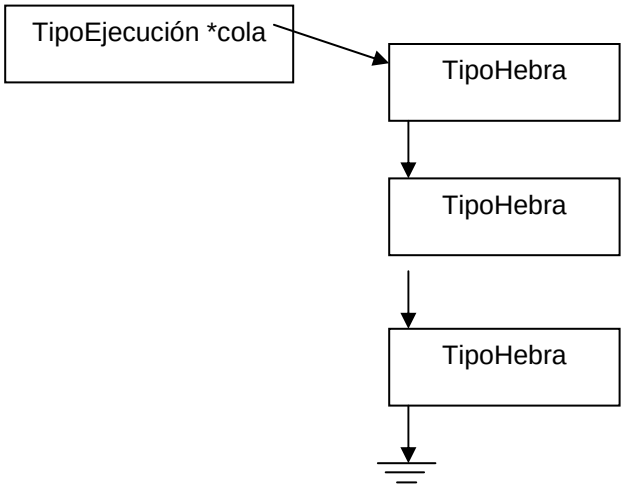


Figura 2.21 Pantalla de ejecución de SEA después de que el cargador ha creado las estructuras de datos

La primera acción del interprete consiste en cargar en la cola de ejecución el papel `main` y los eventos inesperados (nombre y el tiempo de ocurrencia). A continuación se pone en marcha el reloj que marca el tiempo de la simulación. Con las instrucciones de `main` se crean instancias de agentes e interacciones. Para cada agente se invoca al planificador que se encarga de determinar los papeles que le permiten alcanzar sus propósitos y a cada uno de ellos conforme se ejecuta se genera una hebra que se coloca en la cola de ejecución.

La cola de ejecución es una lista ligada como sigue:



Los elementos de la cola de ejecución tienen la estructura siguiente:

TipoEjecucion

TipoHebra *activo ;	Apuntador a la hebra
Struct EstructuraEjecucion *sig ;	Apuntador a la siguiente elemento de la cola de ejecución

Cada nodo ("papel activo") se llama hebra y contiene su program counter que apunta a la instrucción en ejecución, sus registros de proceso y su stack (útil para ejecución de las operaciones aritméticas). Una hebra contiene un apuntador al papel donde se encuentran descritas sus instrucciones y dado que varios agentes pueden utilizar el mismo papel, se sigue la estrategia de *código reentrante*, por lo tanto cada hebra tiene su propia lista de variables locales. Cuando se ejecuta una operación aritmética se empuja un valor al stack o se extrae para depositarlo en un registro del proceso.

La estructura de datos de cada elemento de la pila (stack) es:

TipoStack

`double valor ;`	Lugar donde se almacena cada valor de la pila (stack)
`Struct EstructuraStack *sig ;`	Apuntador al siguiente elemento de la pila

TipoHebra

`long id ;`	Identificación única del papel en ejecución
`long R ,tA ,tB ;`	Registros internos de cada papel activo
`long tiempo ;`	Tiempo de la siguiente instrucción, -1=espera
`int PC ;`	Program counter, instr. en que se encuentra
`int estado ;`	0=ejecutandose, 1=suspendido, 2=terminado
`TipoPapel *papel ;`	Papel que contiene sus instrucciones
`TipoProposito *proposito ;`	Lista propositos que mantienen activo al papel
`TipoStack *stack ;`	Stack de ejecucion
`TipoLocal *local ;`	Lista de variables locales
`Struct EstructuraAgente *agente ;`	Apunta a su agente propietario
`Struct EstructuraActivo *sig ;`	Apuntador a la siguiente hebra del agente

Durante la ejecución se el reloj (variable global) tiene el tiempo en que se van ejecutando las operaciones de cada agente. Cada vez que se ejecuta una operación, se determina el tiempo en que la próxima instrucción se va a ejecutar. Esto permite ver que el tiempo avanza de acuerdo a los eventos que se van presentando y no en forma lineal continua.

El ambiente de la ejecución contiene estructuras de datos adicionales a como se encontraba después del constructor (figura 2.22).

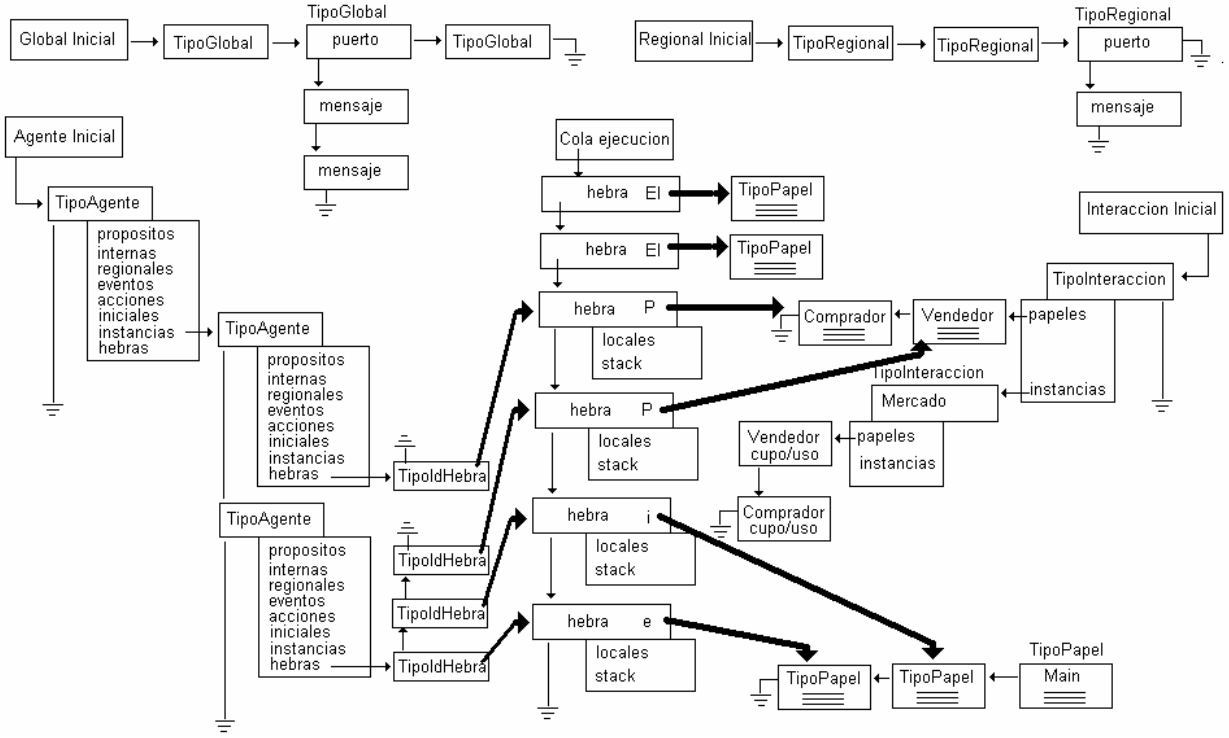


Figura 2.22 Estructuras de datos utilizadas en la ejecución

Las instrucciones que hacen referencia a las variables requieren que estas se resuelvan, para esto se utiliza un `Buffer` como un área donde se guardan los datos en modo texto y de donde se recuperan.

Las variables que participan en las expresiones aritméticas y lógicas se resuelven mediante dos rutinas que sirven para recuperar un valor o almacenarlo. Para leer el contenido de una variable se usa `LeeVar` y para guardarlo se utiliza `EscribeVar`. Estas rutinas devuelven `-1` cuando no se encuentra la variable. La rutina `LeeVar(variable, ambito, agente)` tiene como parámetros el nombre de la variable de interés; el ámbito en que se quiere localizar la variable (en caso que no importe se coloca asterisco) y el agente al cual pertenece la variable (en caso que no importe el agente, se coloca asterisco). Por ejemplo si se quiere el contenido de la variable interna `importe`, la invocación que se hace es `LeeVar("paraguas", 'I', "JuanPerez")`. La contraparte de la lectura es la escritura la cual se hace mediante la rutina `EscribeVar(variable, ambito, agente)`, en esta se utiliza también el `Buffer`.

El `Stack` (pila) se utiliza para realizar cálculos aritméticos. El `Buffer` seutiliza como área donde se guardan los datos en modo texto. La operación `push` agrega en la pila un operador del tipo que se indique entero o real (float), los operadores `* - + /` toman dos operandos de la pila y dejan el resultado en la misma, como sigue: E E = E, F E = F, E F = F, F F = F donde E significa entero y F flotante; si una asignación extrae (`pop`) un dato F en una variable entera, se trunca su valor. El valor contenido en `Buffer` se asigna a la variable que se resuelve. Durante la evaluación de una expresión, cada vez que llega un operador se empuja en la pila, y cuando llega un operando se extrae la cantidad de operadores necesario; por ejemplo, dos para la multiplicación y se realiza la operación, el resultado se vuelve a empujar a la pila. La pila se utiliza para cálculos numéricos y lógicos.

En la parte superior derecha de la pantalla de ejecución (figura 2.23) se tienen tres botones y un campo; estos se utilizan para controlar la velocidad de la ejecución, `start` inicia la ejecución o la continua cuando se suspende con `stop`, `step` permite ejecutar una instrucción de la hebra en proceso. El campo indica la velocidad de ejecución; entre más pequeño sea este valor, más rápida será la ejecución; las unidades de este campo son milisegundos. El contador que aparece debajo de los botones de control de la ejecución a la derecha indican el número de ciclos de ejecución utilizados.

Durante la interpretación de un ambiente de interacciones, se despliegan los eventos significativos como la creación de un agente, creación de interacciones, su plan, cuando un agente toma o deja un papel, cuando se alcanza un propósito, o el inicio y terminación de un evento inesperado.

Al ejecutarse las hebras de un sistema, estas pueden estar en diferentes estados (figura 2.24) dependiendo de los eventos que ocurren; por ejemplo, si una hebra está esperando un mensaje entonces se suspende pasando al estado 2, si vuelve a ser activada, entonces pasa al estado 0. Las hebras en estado 0 y 4 se ejecutan, mientras que en otro estado la hebra permanece suspendida.

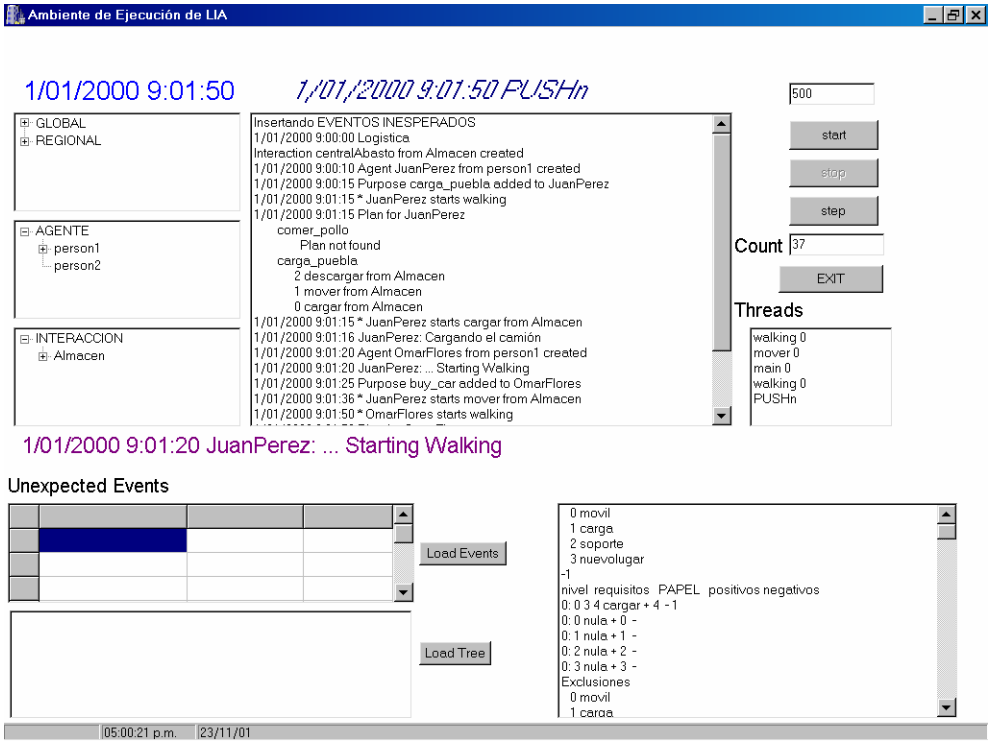


Figura 2.23 Pantalla del ejecutor durante la ejecución de un sistema (aquí sin eventos inesperados)

ESTADO	SIGNIFICADO
0	Hebra Activa
1	Suspendida por un Evento Inesperado (EI)
2	Suspendida por un Accept
3	Suspendida por un Accept y un EI
4	Hebra de Emergencia
5	Suspendida por un EI

Figura 2.24 Estados en que puede estar una hebra

A las variables se les asignan valores o bien, se les toma un valor. La resolución de los nombres de variables se da en tiempo de ejecución. La forma de resolver una se da en forma jerárquica, primeramente se busca la variable en las locales, luego en las internas, posteriormente en las regionales y finalmente en las globales. La búsqueda en las variables regionales se hace siempre y cuando un agente haya declarado a dicha variable. En caso de no encontrar una variable se le asigna un valor nulo y se continua el proceso de ejecución.

2.5 CONCLUSIONES DEL MODELO DE INTERACCIÓN PROPUESTO

Se ha mostrado la propuesta de un modelo de interacción entre agentes en donde se tiene entre otras características agentes que intentan alcanzar varios propósitos ejecutando papeles en paralelo (multihebras), los agentes manejan ontologías mixtas, mediante un proceso de planeación obtienen sus papeles y reaccionan ante eventos inesperados. Este modelo integra componentes de la escuela cognitiva y de la reactiva.

A partir del modelo y con el propósito de obtener una simulación de un ambiente multiagente con las características propuestas se desarrolla el Lenguaje de Interacción entre Agentes (LIA) y su interprete Sistema de Ejecución de Agentes (SEA). En SEA se editan los sistemas descritos mediante LIA y se traduce a directivas que el módulo Cargador utiliza para construir las estructuras de datos que sirven para dejar listo el ambiente para la ejecución. La ejecución del sistema implica la creación de otras estructuras de datos para administrar las hebras activas que pertenecen a un agente. Una vez que está el ambiente de ejecución se ejecutan las instrucciones contenidas en los papeles que el módulo de planeación selecciona para los agentes, además se ejecutan los eventos inesperados. El interprete invoca al Comparador de Ontologías Mixtas cuando hay intercambio de mensajes entre agentes.

Estos componentes en forma integrada son una alternativa a las propuestas que se han realizado sobre agentes dentro de los cuales se han hecho algunas aplicaciones específicas para el comercio electrónico donde se modelan compradores y vendedores, subastadores, vendedores. En nuestro caso es posible modelar diferentes sistemas de agentes que pueden ejecutarse utilizando LIA y SEA.

CAPÍTULO 3

MAPEO DE UN CONCEPTO AL MÁS SIMILAR EN OTRA ONTOLOGÍA

En este capítulo se plantea el problema de la comunicación entre dos agentes que tienen internamente su propia organización de conceptos y externamente expresan sus mensajes utilizando un lenguaje común entre ellos. Para lograr la comprensión entre ambos ellos desarrollamos un algoritmo donde se intercambian palabras que en algunos casos son ambiguas porque se refieren a conceptos diferentes, pero utilizadas en pares (refiriéndose al concepto y a su padre respecto a una taxonomía jerárquica) permiten reducir la ambigüedad.

La comunicación de mensajes entre dos entidades (como dos personas, sistemas de información o agentes), expresados mediante un código (lenguaje) que ambas partes comprenden, involucra ocasionalmente fallas en el canal de comunicación o que los involucrados manejen diferentes lenguajes. En este trabajo se considera el caso en que los mensajes siempre llegan y que se utiliza un mismo lenguaje. Entonces el problema a considerar es relativo al entendimiento entre las entidades. Como se mencionó, ocasionado porque dos agentes manejan diferentes conceptos en sus representaciones internas y los tienen organizados en formas diferentes (porque cada uno puede tener una especialidad diferente: médicos, artistas, agricultores, químicos, abogados, etc.)

En los sistemas informáticos la representación del conocimiento es una manera de codificar la realidad en la cual se encuentra inmerso el mismo. Estos estudios se han hecho para obtener modelos del mundo real para manipularse y obtener resultados aprovechables en el contexto donde proviene la información. Existen diferentes formas de estructurar el conocimiento como se explicó en la sección 1.3, esto implica que una entidad puede estar especializada en el manejo de información visual mientras que otra puede serlo en base de datos. En cuanto al contenido, dependerá de los temas que resulten de interés para cada una de las entidades, de esta forma para una persona que trabaje con agricultura, lo más probable es que su conocimiento esté estructurado en conceptos como tierra de hoja, temporal, lechugas, manzanas, peras, ejotes, entre otros. A la forma particular de organizar los conceptos por parte de una entidad le llamaremos *ontología*.

En las transacciones comerciales es necesario intercambiar información y conocimiento para hacer posible la comercialización de productos. Por lo tanto, es necesario que los participantes establezcan

acuerdos acerca de lo que quieren intercambiar. Para esto se toma en consideración que cada entidad tiene su propia ontología y utiliza un lenguaje común mediante el cual comunica sus intereses. Los conceptos son independientes del agente que los maneja y se consideran únicos, esto es, el concepto maíz-cereal será el mismo para todos los agentes aún cuando algunos utilicen las palabras maíz, corn, milpa u otras para referirse al mismo.

A pesar de que se utiliza un lenguaje común, los conceptos que se establecen cuando se intenta la comprensión entre agentes, no necesariamente corresponden a los conceptos a que se refieren las entidades en comunicación, esto se debe a que cada entidad representa de manera propia sus conceptos y utliza diferentes palabras para referirse a los mismos. Si se utilizan palabras únicamente puede ocurrir que dos entidades se estén refiriendo a dos cosas distintas, por ejemplo si alguien pregunta por una manzana, una entidad puede referirse a la manzana de una colonia mientras que otra a una fruta. Por esto, diferenciamos entre palabras ordinarias (que pueden ser ambiguas al hacer referencia a conceptos) y conceptos que son únicos.

En otros términos, se intenta resolver el problema entre dos agentes que intercambian palabras en un lenguaje común, y nos interesa la comprensión que los participantes logran alcanzar (la manera en que llevan las palabras intercambiadas a su representación interna cada uno). Como ya se mencionó en la sección 1.3 cuando nos referimos a un concepto usamos palabras minúsculas subrayadas y las palabras que lo referencian en minúsculas sin subrayar. En ambos casos las palabras se expresan en singular. Los conceptos se organizan en una jerarquía, el nodo superior es el concepto más general respecto a los de nivel (la raíz del árbol es el concepto más general) (figura 3.1). Los conceptos están unidos por relaciones es-un u otra similar, por ejemplo en la figura 3.1 cosa agrupa a planta y animal donde cada uno es-una cosa, a su vez el concepto planta agrupa a planta-comestible y planta-ornato y ambos tienen la relación es-una, un nodo es una instancia de su nodo superior.

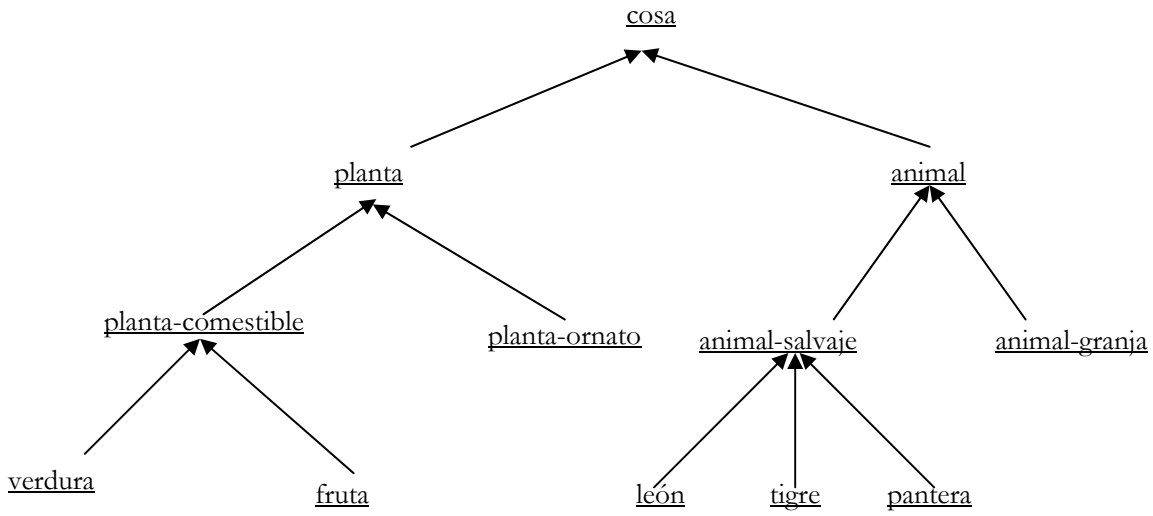


Figura 3.1 Organización de una Ontología

La relación entre palabras y conceptos puede estar sujeta a ambigüedad, debido a que una palabra puede mapear a muchos conceptos, muchas palabras a un concepto o una palabra mapea a un concepto. Algunos ejemplos de esto son:

a) *una a muchos*, la palabra perico se puede referir al concepto perico-animal o a perico-herramienta;

b) *muchas a uno*, las palabras melocotón y chabacano se refieren al concepto chabacano;

c) *una a uno*, una palabra mapea a un concepto único, por ejemplo en el caso de: π (la palabra π mapea al concepto π), 5, Mar Caspio, polinomios como $(x + 4)^2$, posiciones geográficas como (19°N, 90°W), un dibujo, un diagrama arquitectónico.

Existe también la posibilidad que una entidad no maneje algunas palabras porque no tiene conceptos con quien asociarlas.

Por ejemplo si una persona utiliza la palabra perico se tiene que revisar el sentido de la misma porque puede referirse al concepto perico-animal, (ave verde), o al concepto perico-herramienta (especie de llave), o quizá perico-apodo (sobrenombre de una persona). De esta forma cuando un agente quiere comprar un perico (animal), al utilizar el concepto y ubicarlo en su ontología se utiliza la información del concepto y de su padre (en este caso el concepto animal) para reducir la ambigüedad. *Cuando un agente es capaz de llevar a su representación interna las palabras relacionándo un concepto con las palabras que se le indican, decimos que el agente entendió el mensaje.*

Los agentes que consideramos, intercambian palabras ordinarias para comunicar sus conceptos de interés (figura 3.2).

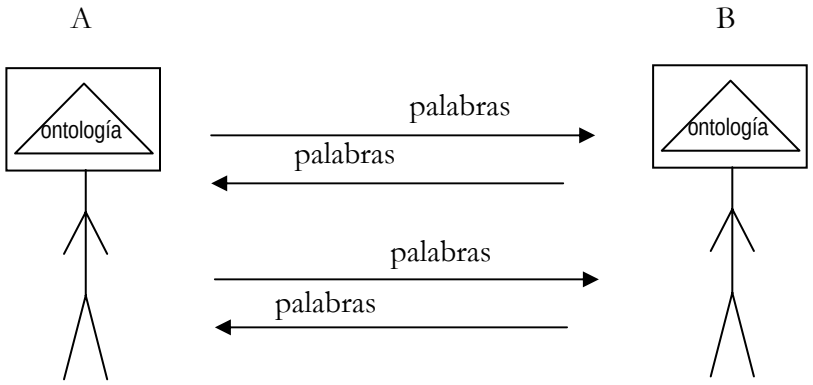


Figura 3.2 Intercambio de palabras (ambiguas) para encontrar los conceptos entre dos agentes

Las ontologías que consideramos se componen de nodos organizados en una jerarquía formada por relaciones *es-un*, donde el nodo de nivel superior corresponde con la clase más general y a partir de ella se van obteniendo nodos que son subconjuntos del nodo de nivel anterior en clases más específicas. De esta forma el padre de un nodo se considera un concepto más general, mientras que un nodo hijo se considera como una especialización del primero.

Cada concepto tiene una lista de propiedades expresadas como pares en donde el primero es el atributo y el segundo su valor (ambos expresados como conceptos) (atributo valor), en estas listas se aplica la recursividad, por ejemplo: (maíz (color azul) (tamaño ((valor 1) (unidad cm))) (dureza duro-macizo)).

Estas relaciones de padre a hijo se utilizan para encontrar el concepto al que se refiere un agente al utilizar palabras [Guzmán 2000] [Guzmán 2000a], por ejemplo, para dos agentes que se comunican, sus conceptos no necesariamente están organizados de la misma manera.

Considerando la forma en que las personas llevan a cabo el proceso de interacción se observa que inicialmente crean un contexto de la información de interés para ambos y comienzan con un intercambio de mucha información (palabras), luego cuando ya se entienden necesitan menos información para que quede claro lo que quieren comunicarse, finalmente, llegan a un punto en donde utilizan palabras clave y se comunican una gran cantidad de conocimiento de interés para ambas partes [Guzmán 2002], por ejemplo puede ser típico un mensaje de la forma "necesito maíz como siempre", lo cual se puede interpretar como "necesito una tonelada de maíz en forma de elotes entregado en la ciudad de Mérida a finales de octubre y el pago se realizará en efectivo a contraentrega".

Al considerar el caso del entendimiento entre dos agentes que intercambian palabras para intentar mapear los conceptos que tienen en su representación interna existen diferentes alternativas desarrolladas hasta ahora como se comentó en la sección 1.3, una de ellas consiste en que utilicen la misma ontología y se intercambien los mismos conceptos, lo cual implica que los agentes que participan en las interacciones comparten la misma representación interna y utilizan elementos de la misma (como apuntadores a nodos). Otra alternativa es crear una ontología de nivel superior o utilizar alguna existente donde se mapean hacia la superior los conceptos de las ontologías de nivel inferior. En general, no se puede esperar que dos agentes compartan una misma ontología, principalmente, al considerar que existen muchas empresas que realizan comercio con otras tantas y solamente para algunas transacciones, por lo que el costo de hacer coincidir sus ontologías implica un costo mayor al esperado para realizar operaciones de comercio electrónico. Por esto se requiere de alguna herramienta con la que realice el mapeo de conceptos entre dos ontologías respetando la estructuración individual.

Cada agente en el modelo presentado en el capítulo 2 tiene una representación interna (llamada ontología) y un conjunto de palabras que representan a cada concepto dentro de la ontología, a partir de esto se hace el mapeo de los conceptos sin necesidad de integrar los conceptos que forman las dos ontologías. La ventaja de esto es que, solamente se mapean entre dos y diez conceptos y no 30 mil o 50 mil que contiene una ontología; tampoco se necesita integrar la ontología con otras 4 o 5 ontologías. Lo que estamos proponiendo es que si se tiene una ontología compleja y únicamente interesa que un agente se ponga de acuerdo sobre algunos conceptos de interés para llevar a cabo una transacción comercial, únicamente se haga el mapeo de estos conceptos aprovechando que los agentes utilizan palabras de un lenguaje común y que cada concepto es único dentro del sistema de agentes. Esto está apoyado en el hecho de que un agente no le puede copiar su representación interna a otro agente porque si esto fuera posible el problema ya estaría resuelto, de la misma manera que un ser humano no le puede copiar parte de su cerebro a otro para darse a entender, situación similar en muchos sistemas informáticos, donde tampoco es posible transferirle el conocimiento a otro programa porque cada uno maneja sus estructuras de datos propias.

Muchas veces la razón de utilizar diferentes estructuras es porque cada programa fue hecho por un grupo de desarrollo diferente, cada uno con una filosofía, idiosincrasia o intención diferente y difícilmente estos se pueden pegar de una forma simple. Dado entonces que en el mundo abunda una heterogeneidad de desarrolladores es que se justifica el que tratamos de aprovechar el conocimiento que ya esta codificado de alguna forma en ontologías existentes.

El algoritmo que presentamos en 3.1 y del cual damos ejemplos en 3.2, tuvo como antecedente otro algoritmo en el cual se ubica el concepto candidato enviando sus palabras al segundo agente, ante lo cual, este busca los conceptos con los cuales asocia estas palabras, (en este caso ùnicamente se envìaba las palabras del concepto de interés, en el nuevo algoritmo también se incluyen las palabras del padre del concepto). Lo que encontramos es que varios conceptos mapeaban a las palabras, por lo que en caso de duda el agente enviaba las palabras correspondientes a los padres de cada uno, mismos que a su vez al llegar al primer agente generaba un mapeo múltiple que requería mucho más interacciones para determinar cual concepto es el referido. Sobre la base de este proceso de confirmar y refutar nos dimos cuenta que se podría simplificar enviando desde la primera vez tanto las palabras que corresponden con el concepto de interés como las de su padre, por lo tanto es esta la versión que se expondrá en 3.1.

El comparador de ontologías mixtas (COM) es un modulo en el cual se considera la estructuración de una onde una ontología con su diccionario donde se asocian las palabras con los conceptos. Estos elementos los utiliza el algoritmo `hallar`, el cual toma un concepto de una "ontología A" busca sus palabras asociadas, toma al padre del mismo concepto y sus palabras asociadas y con estos conjuntos de palabras busca el conceptos más similar en una "ontología B".

Un fragmento de un diccionario de palabras-concepto se muestra a continuación, en el mismo se observa que el concepto cosa tiene dos palabras asociadas, la palabra maíz se refiere a los conceptos maiz_planta y maiz_semilla.

Palabra	Concepto
cosa	cosa
algo	cosa
fantasma	fantasma
espanto	fantasma
animada	animada
viva	animada
animal	animal
vegetal	vegetal
planta	vegetal
fruta	fruta
comida	fruta
fruta_caliente	fruta_tropical
fruta_templada	fruta_templada
fruta_fria	fruta_fria
vegetal	verdura
verdura	verdura
mamey	mamey
mango	mango
manzana	manzana
durazno	durazno
melocoton	durazno
pera	pera
tejocote	tejocote
maiz	maiz_planta
maiz	maiz_semilla
frijol	frijol
trigo	trigo
cereal	cereal
semilla	cereal

3.1 ALGORITMO HALLAR

Como se mencionó en la sección anterior, el algoritmo `hallar` es el resultado de varios experimientos realizados para encontrar el concepto más similar de una ontología A en otra B. En la última versión en este algoritmo se consideran cuatro casos con base a los elementos que se involucran en la búsqueda: el concepto y su padre (o algún ancestro significativo); se considera posible construir algoritmos más complejos en donde se intercambie una sección mayor de una ontología como por ejemplo el concepto sus hijos y los ancestros, esto es motivo de una línea de investigación diferente a la que seguimos. A continuación explicamos la solución que hemos encontrado:

Considerando el concepto y su padre (ancestro) se tienen cuatro casos:

a) Cuando *coinciden* las palabras de los *conceptos* y las respectivas de su *ancestro*.

b) Cuando *coinciden* las palabras de los *conceptos* pero *no las de su ancestro*.

c) Cuando *coinciden* las palabras de su *ancestro* pero *no del concepto*.

d) Cuando no coinciden *ni las palabras de su ancestro ni las del concepto*.

Sean A y B dos ontologías, para encontrar un concepto cb en B lo más cercano a otro ca en A se utiliza el algoritmo `hallar` en donde se indica un concepto de A que se quiere encontrar en B mediante el envío de las palabras que describen al concepto de A, el concepto más similar de B se devuelve o se indica que no se encontró, el resultado es único, por lo que el algoritmo elige el mejor concepto. El algoritmo usa dos ontologías distintas A y B (figura 3.3). La invocación al algoritmo `hallar` es:

```
cb = hallar (ca, A, B)
```

que significa: hallar el concepto cb en la ontología B más cercano al concepto ca de la ontología A.

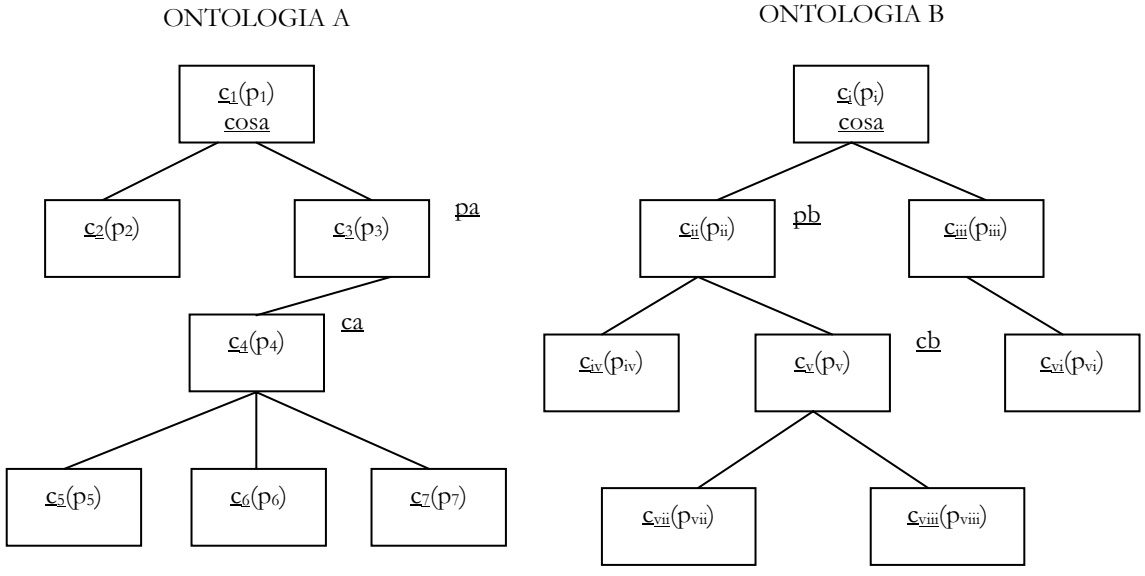


Figura 3.3 Dos ontologías diferentes. Cada cada concepto c_x se referencia por palabras comunes a ambas ontologías. Se intenta encontrar dos conceptos padre-hijo (pb-cb) en la ontología B que correspondan con el par (pa-ca)

En forma abreviada el llamado a `hallar` lo escribimos como:

```
cb = hallar(ca)
```

El algoritmo `hallar` utiliza las palabras que se relacionan con cada concepto, es decir no los conceptos directamente. En principio envía las palabras correspondientes al concepto de interés y de su padre. Luego en la ontología B se buscan conceptos que tengan alguna relación ancestro-nodo para encontrar al posible padre y concepto equivalente. Si la información es insuficiente se utiliza recursivamente el algoritmo para encontrar algunos ancestros apropiados. El propósito de hallar es encontrar en B los conceptos cb y pb que corresponden a los conceptos ca y pa de A, o cuando menos, hallar cb.

El algoritmo `hallar` consta de los pasos siguientes:

a) En B se buscan dos nodos (conceptos) pb y cb, de manera que las palabras de pb coinciden con la mayoría de las palabras (recibidas) de pa y las palabras de cb coinciden con la mayoría de las palabras (recibidas) de ca; pb es el padre, abuelo o bisabuelo de cb. Con esto se considera el concepto cb encontrado como el más cercano a ca y por lo tanto la respuesta es cb y termina el algoritmo (figura 3.4).

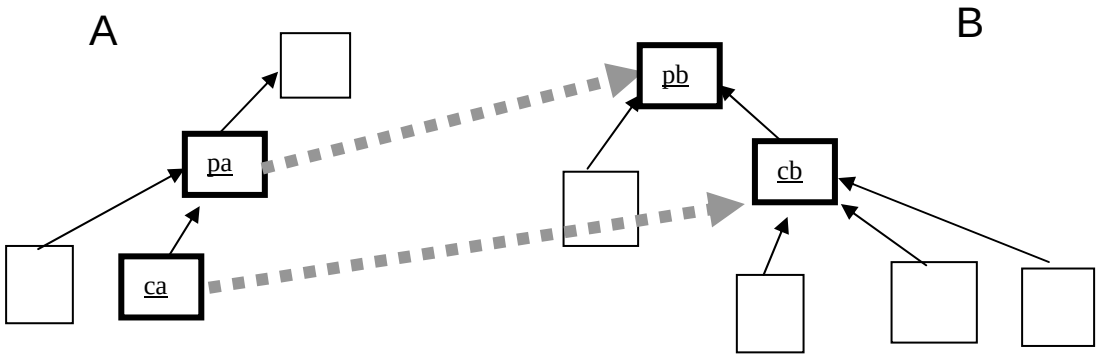


Figura 3.4 Caso a) (ca, pa) en A hallan un par (cb, pb) en B que son hijo y ancestro

b) Si se encuentra pb pero no cb. En este caso, se llama recursivamente al proceso hallar de la forma `pb' = hallar(pa)` para confirmar que pb es ancestro del concepto de interés (ca). Si el pb' hallado es cosa, es decir, el concepto de más alto nivel, el algoritmo concluye sin éxito; si no, se busca en B para cada hijo de pb, aquél, cuya mayoría de propiedades y valores, coincidan (recursivamente vía hallar) con las correspondientes de ca. Es decir, se busca en B al hijo de pb que tenga la mayor cantidad de las propiedades (y valores correspondientes) del concepto ca. Se rechazan hijos de pb que coinciden sólo en unas cuantas propiedades y en unos cuantos valores con las de ca. Si el candidato cb' analizado tiene además hijos, se ve si éstos coinciden (usando recursivamente hallar) razonablemente con los hijos de ca. Si se encuentra un cb' con las propiedades deseadas, el algoritmo concluye con éxito indicando el concepto buscado. En otro caso, se intenta hallar el cb' entre los hijos del padre (en B) de pb, es decir, entre los hermanos de pb; en caso necesario, entre los hijos de los hijos de pb, o sea, entre los nietos de pb. Sino se encontró un cb', entonces el más cercano a ca es pb, por lo que hallar devuelve la anotación

`hijo_de_pb` y el algoritmo concluye. Por ejemplo, A puede enviar a B el par (kiwi, fruta)[15], B puede tener el concepto fruta pero no el concepto kiwi ni alguna fruta que se le parezca, en este caso, kiwi (la palabra kiwi, enviada por A y recibida por B) se traduce por B a `hijo_de_fruta`, o "alguna fruta que no conozco, que no tengo en mi ontología", lo más cercano en B a kiwi en A (figura 3.5).

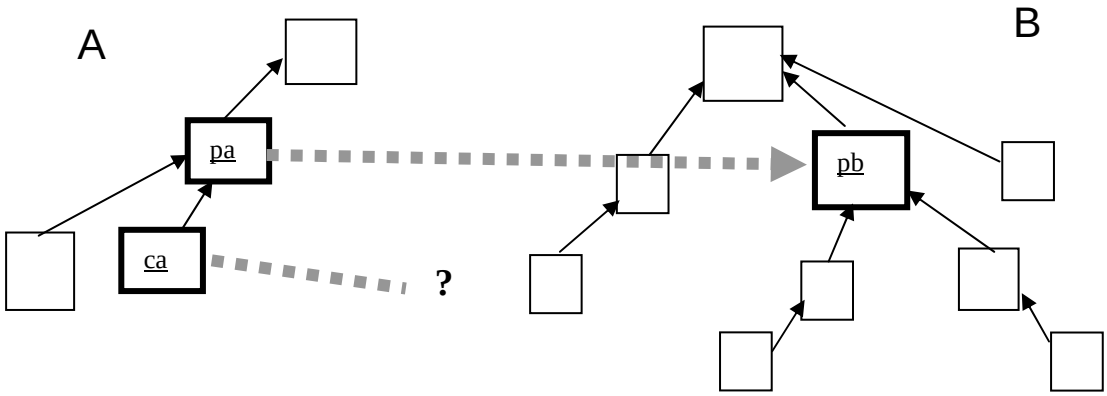


Figura 3.5 Caso b) Se halla en B al padre pb que corresponde a pa en A, pero no se halla concepto cb en B que corresponde a ca en A

c) Si se halla cb pero no pb, se busca si el abuelo en B de cb es similar a pa, o si el bisabuelo de cb en B es similar a pa (esto ya fue comentado en el caso b). Si se halla, entonces el concepto en B más similar a pa es el abuelo o el bisabuelo de cb y concluye el algoritmo. Si no se halla, se verifica si la mayoría de las propiedades (y sus valores correspondientes) de ca coinciden con las de cb y si la mayoría de los hijos de ca coinciden (usando `hallar`) con la mayoría de los hijos de cb; si las propiedades y los hijos coinciden, entonces la respuesta es cb y concluye el algoritmo aunque no se haya hallado en B al pb que corresponda al concepto pa en A. Si solamente una parte de propiedades e hijos son similares entonces la respuesta es `probably_cb` y concluye el algoritmo. Si ninguna propiedad ni hijos son similares la respuesta es `no_hay` y concluye el algoritmo (figura 3.6).

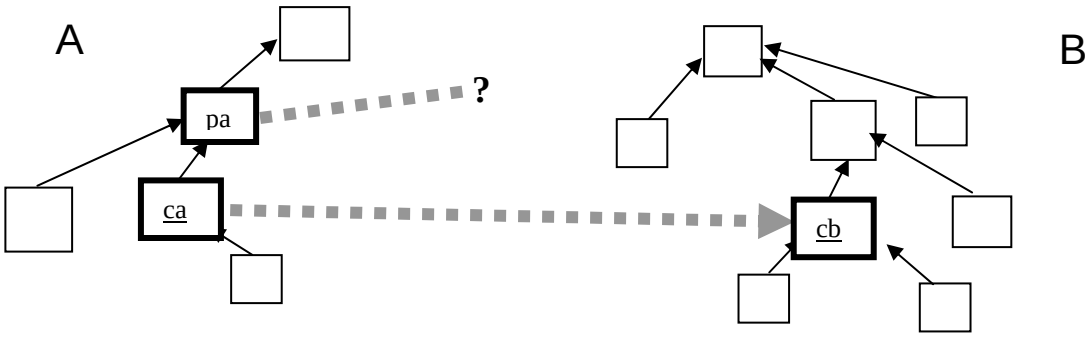


Figura 3.6 Caso c) se encuentra el concepto cb en B similar a ca en A, pero no se encuentra pb en B similar a pa

[15] Son dos palabras, no dos conceptos y un agente no puede envíar conceptos a otro agente (representación interna)

d) Si no se encuentra cb ni pb, entonces la respuesta es `no_hay` y concluye el algoritmo (figura 3.7).

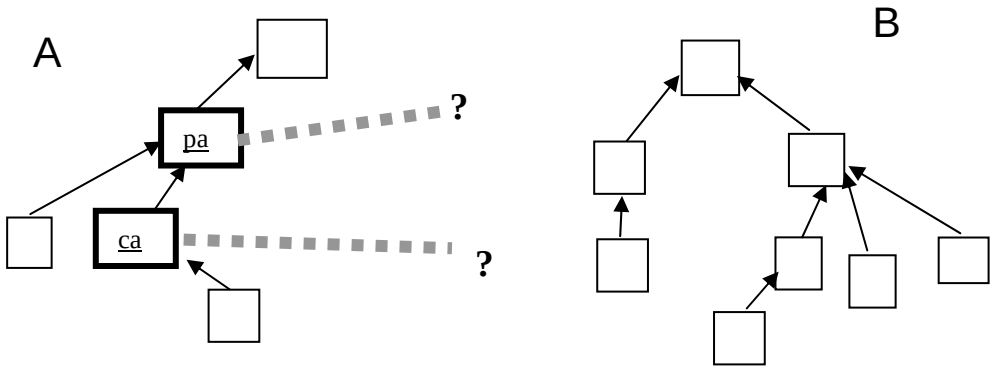


Figura 3.7 Caso d) no se encuentra ni cb, ni pb en B similares a ca y pa en A

3.2 EJEMPLOS

Una ontología generalmente consiste de cientos o miles de conceptos, pero como la intención aquí es ilustrar el funcionamiento se usan secciones de ontologías donde únicamente se tienen los conceptos de interés tomando en cuenta esto.

En el caso a), donde se encuentra el mapeo (usando palabras ambigüas) entre el concepto y su padre de una ontología A con el concepto y su padre de una ontología B (figura 3.8).

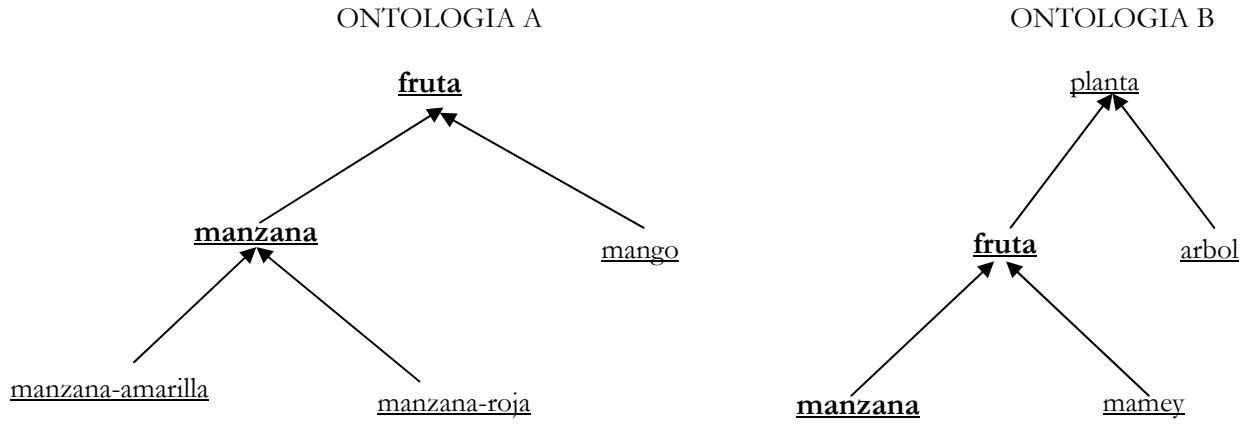


Figura 3.8 Ejemplo del caso a) donde se encuentra el mapeo entre los conceptos y sus padres

En el comparador de ontologías implementado (figura 3.9) se le indican las ontologías que se utilizan mediante los botones `Load Ontology A` y `Load Ontology B` presionando el botón `Ontology Match` se activa el algoritmo hallar implementado. En el campo `Result` aparece el resultado del mapeo entre las ontologías en cuestión.

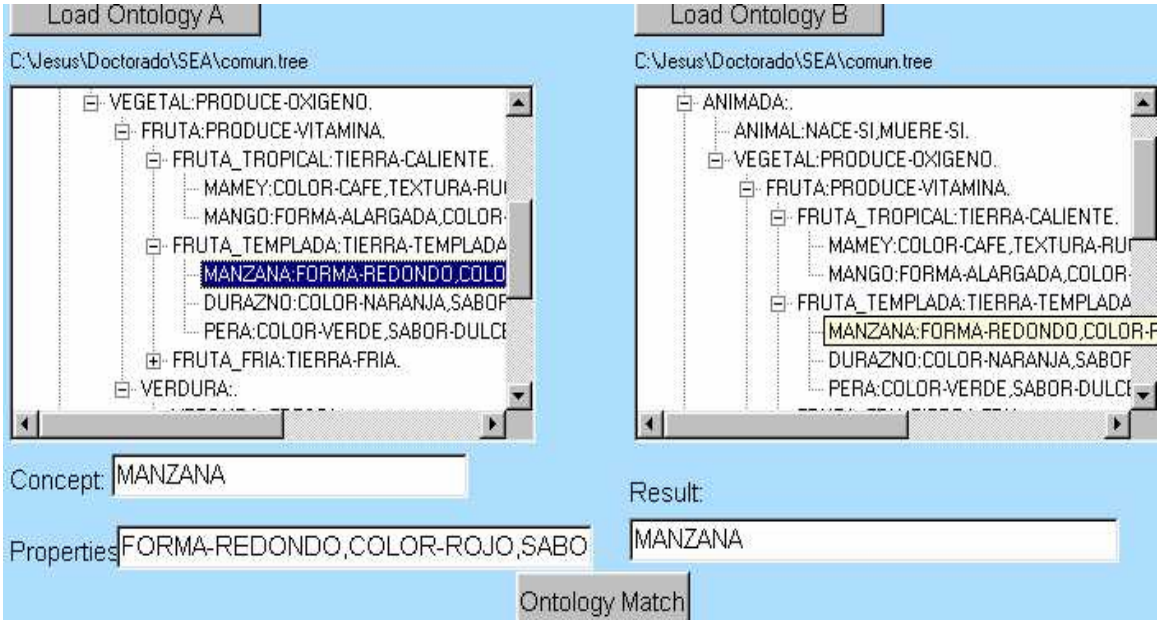


Figura 3.9 Ejecución del COM para el caso a) donde se encuentra la equivalencia de ca y pa con el respectivo cb y pb

El caso b) se encuentra un concepto pero no el padre del mismo por lo que se aplica recursivamente el algoritmo hallar para encontrar otro ancestro equivalente en la ontología del agente A, la figura 3.10 se muestra la sección de las ontologías en donde se presenta esta situación.

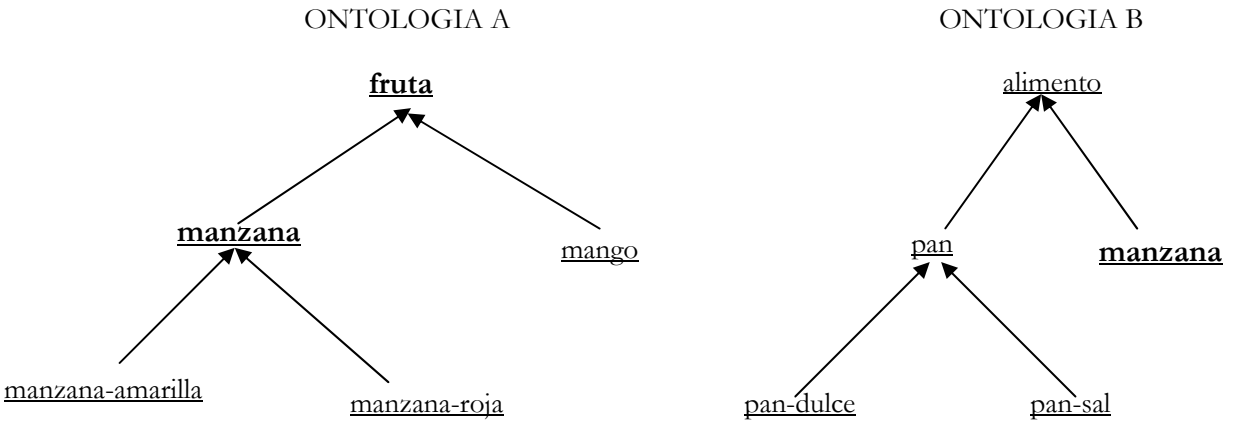


Figura 3.10 Caso b) se encuentra el mapeo entre los conceptos ca de A con cb de B pero no de sus ancestros

En la figura 3.11 se muestra la pantalla del COM en donde se presenta el caso b).

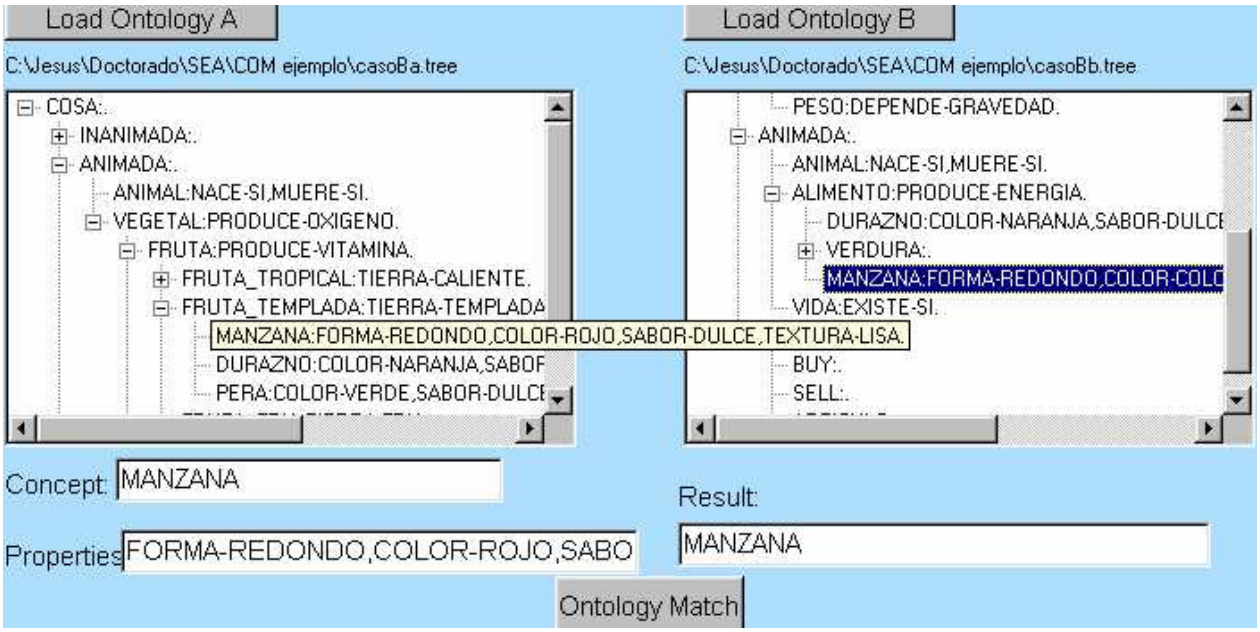


Figura 3.11 Ejecución del COM para el caso b) se encuentra la equivalencia de ca y cb pero no de sus ancestros

Un ejemplo para el caso c) (figura 3.12) en donde se encuentra el mapeo entre los padres del concepto pero no el concepto de interés. En este caso se utilizan las propiedades de los conceptos y sus hijos.

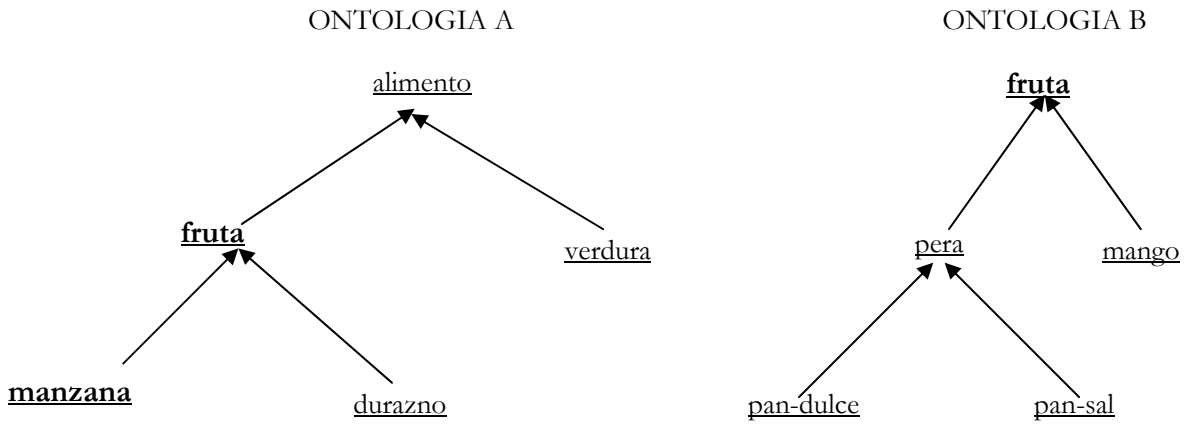


Figura 3.12 Caso c) en este caso se encuentra el mapeo entre los padres pa y pb pero no entre los conceptos

La pantalla del COM donde se muestra un caso c) se muestra en la figura 3.13.

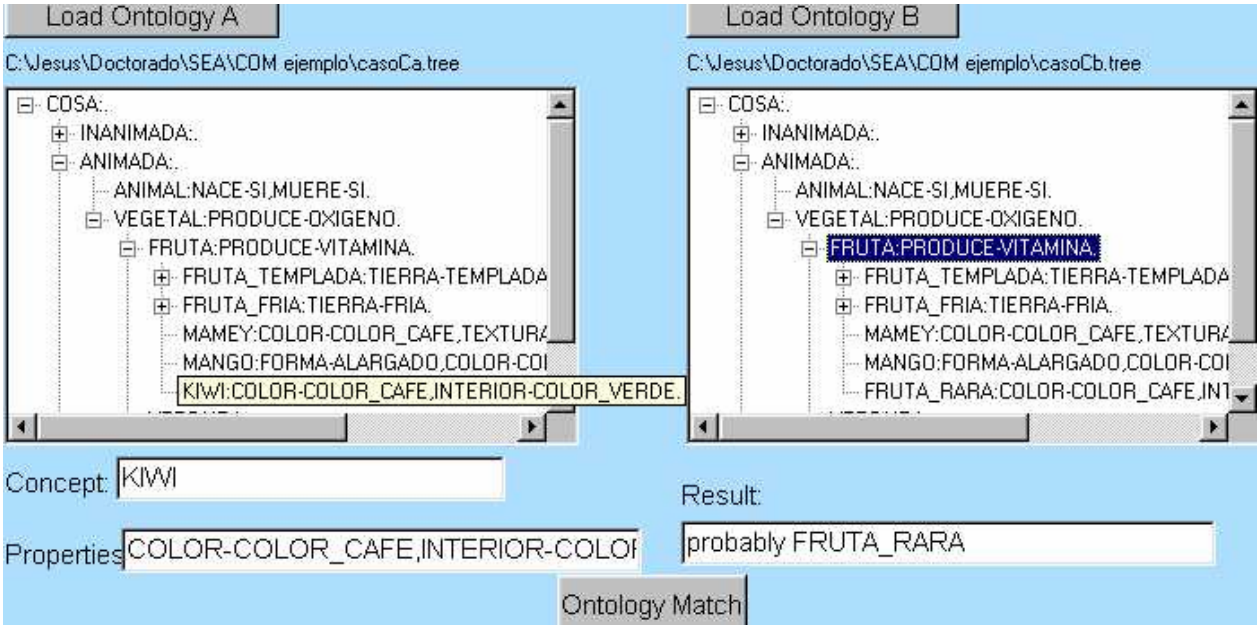


Figura 3.13 Ejecución del COM para el caso c) se encuentra la equivalencia de pa y pb pero no de los conceptos ca y cb

El ejemplo para el caso d) es cuando las ontologías son diferentes y no contienen conceptos comunes, esto es típico cuando los agentes tienen diferentes especialidades, como en el caso de un agricultor y un informático (figura 3.14).

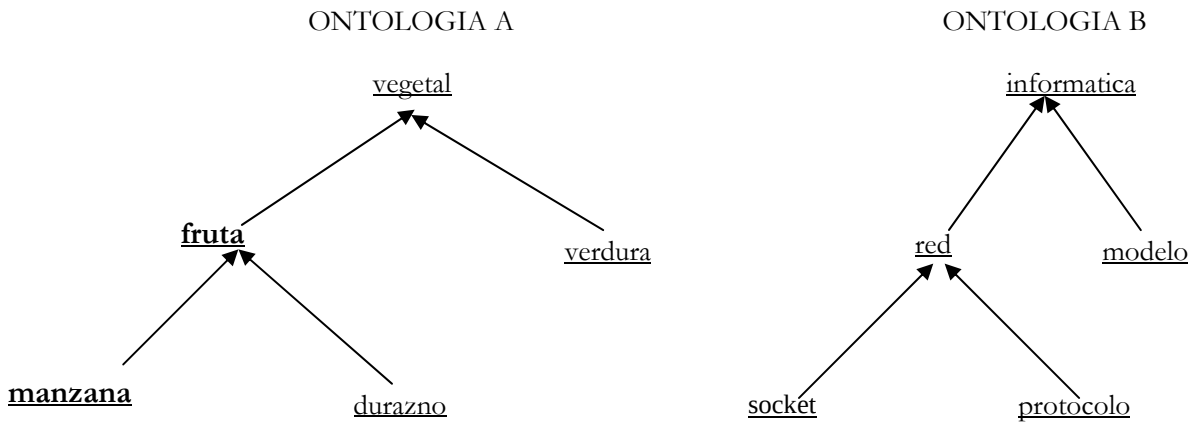


Figura 3.14 Caso d) en donde dos ontologías son diferentes y no existen conceptos en común

La ejecución del COM para el caso d) se ilustra en la figura 3.15.

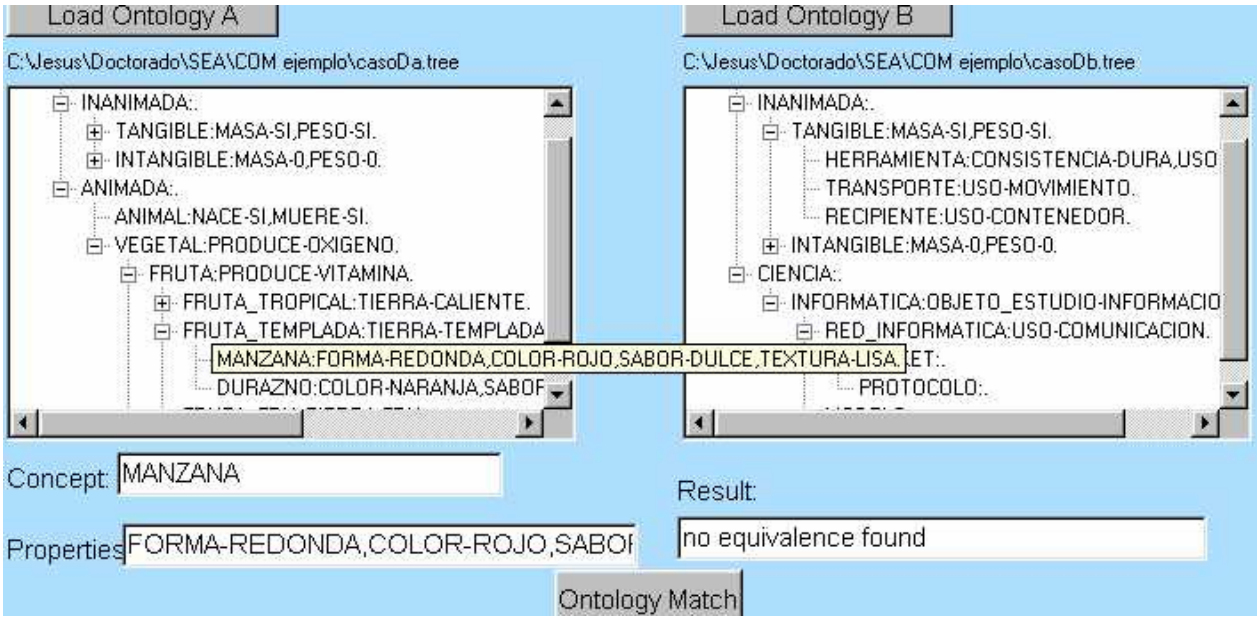


Figura 3.15 Ejecución del COM para el caso d) no existen conceptos comunes

3.3 CONCLUSIONES DE MAPEO DE CONCEPTOS

Se propuso en este capítulo una forma de establecer un mapeo de conceptos entre ontologías diferentes, dos a la vez. Específicamente hemos presentado un algoritmo constructivo que halla, para un concepto ca en una ontología A, el concepto cb más similar de otra ontología B (organizadas ambas en base a una estructura jerárquica) utilizando las palabras que son de un lenguaje común entre agentes pero que el mapeo de palabras a conceptos es ambiguo debido a que una palabra mapea a varios conceptos y que un concepto se puede referir con diferentes palabras. El algoritmo hallar, responde con las palabras del concepto encontrado o con `no_hay`, `duda_`cb, `hijo_de_`pb.

Nuestra propuesta es una alternativa a los trabajos que se están realizando sobre manejo de ontologías, con la ventaja de que se toman dos ontologías intactas y se trabaja únicamente con los conceptos de interés. Al manejar un mínimo de información es posible entonces aplicar el comparador de ontologías en sistemas de comercio electrónico, bases de datos, base de conocimientos donde muchas veces solamente se requiere realizar pocas transacciones y no se quiere incurrir en el costo que implica generar una ontología común para dos entidades (como empresas). Nuestra propuesta también es útil en la interacción entre agentes ligeros como los móviles o los que se encuentran embebidos en smart cards («tarjetas inteligentes») u otros sistemas distribuidos.

El algoritmo aquí propuesto no usa de una tercera ontología "común" (tipo CYC) para resolver ambigüedades. Empero, el uso de estos recursos de conocimiento (ontologías comunes, tesauros, diccionarios, etc.) podría producir algoritmos más eficientes lo cual, es motivo de otra línea de investigación.

La propuesta que se hace en este capítulo es importante por la cantidad de empresas en el mundo que día con día requieren realizar transacciones entre ellas y necesitan establecer con precisión los productos y condiciones a intercambiar. La forma en que se integra esta propuesta a ontologías existentes es mediante la adición de un diccionario de palabras de un lenguaje común hacia los conceptos descritos en la ontología, con esto establece una independencia entre los conceptos y el lenguaje de comunicación (por ejemplo podrían utilizarse diferentes lenguajes para interactúar con diferentes agentes).

CAPÍTULO 4

PLANEACIÓN EN EL MODELO DE INTERACCIÓN ENTRE AGENTES

En este capítulo se desarrolla la planeación para los agentes del modelo propuesto en esta tesis. El módulo de planificación elige los papeles con que alcanzan sus propósitos los agentes, en base a sus recursos y los papeles disponibles en el ambiente. Para seleccionar un papel es necesario que sus requisitos se cubran. Una vez que un agente tiene su plan, se ejecutan los papeles de las interacciones tomando en cuenta la disponibilidad de sus recursos y las condiciones del ambiente. El plan que se obtiene para cada agente es particular, esto es que, para dos agentes diferentes, con los mismos propósitos, se generan dos planes distintos debido a sus diferencias en sus recursos o papeles disponibles. Durante la planeación se puede detectar si un agente no alcanzará sus propósitos.

En el modelo, algún agente que no tenga la capacidad para obtener un plan debido a sus recursos puede suceder que, algún evento inesperado los modifique y pueda estar en condiciones de tenerlo. Lo contrario también es posible, esto es, que un agente que cuenta con recursos suficientes, algún evento inesperado se lo impida.

Algunas de las técnicas tradicionales de planeación se basan en el desarrollo de un árbol de búsqueda en donde se examinan las posibles alternativas que pueden dar lugar a un plan, esto tiene una complejidad exponencial. En nuestro caso, en vez de un árbol de búsqueda, se utiliza una gráfica en donde las relaciones de exclusión limitan la generación de opciones que resultan contradictorias y por lo tanto se reduce la complejidad a una exponencial constante.

En este caso el método de planeación obtiene un árbol AND-OR para cada propósito y por lo tanto, un bosque de árboles cuando un agente tiene varios propósitos. Las hojas de los árboles son los papeles y para iniciar la ejecución de cada uno se deben ir cumpliendo los requisitos de cada uno. Cuando un papel que está en arcos AND y no puede ejecutarse ocasiona que los demás tampoco lo hagan. En el caso de arcos OR se pueden ejecutar en paralelo.

Nuestro modelo de planeación no intenta maximizar una función objetivo en donde se involucran todos o algunas de los agentes sino, sólo se intenta generar un plan para cada uno, procurando con el mismo alcanzar los propósitos del agente.

Debido a que existen varios agentes en el modelo, cada uno conteniendo diferentes propósitos la planeación se puede efectuar para cada agente en paralelo al igual que la ejecución de los papeles de los planes, esto significa que mientras uno esté generando su plan otros ya pueden estar ejecutando papeles, al que se está obteniendo su plan se integra al ambiente de ejecución en cuanto lo tiene.

En este capítulo se integra la planeación al modelo de interacción en donde el planificador toma los papeles y los recursos del agente para seleccionar aquellos que les permiten alcanzar sus propósitos.

En el caso del comercio electrónico se tienen agentes que compran, venden, subastan mercancía y la transportan. El proceso de distribución y transporte se propone con una mínima intervención de personas, por ejemplo que solamente le especifique lo que quiere con sus características. Este problema consiste en que dada una localidad y un transporte, la mercancia solicita su carga al transporte para viajar a su destino y una vez que llega pide se descargue. Automatizar la entrega de la mercancía con el uso de transponders o tarjetas inteligentes «smartcards» contribuye a evitar los problemas que se generan por el uso de etiquetas legibles por personas o código de barras que en ocasiones sufren corrupción o daños, lo que trae problemas en el envío de paquetes. En este caso si un agente se integra a la mercancía el planificador le da las acciones que la mercancía seguirá para llegar de su origen a su destino.

4.1 PLANEACIÓN MEDIANTE UNA GRÁFICA POLINOMIAL

El planificador que se describe en este capítulo tiene similitud con la planeación gráfica de Blum [Blum 1995]. Esta se aplica para cada propósito de un agente. Dado que es posible obtener diferentes árboles AND-OR es conveniente un proceso de factorización de la misma, (está línea de trabajo no se siguió en esta tesis). Para buscar la gráfica de planeación se desarrollan n niveles en donde se encuentra un plan o se decide que no existe alguno, es un dato con que se parametriza al planificador.

En este caso un problema de planeación tiene:

a) Un *conjunto de papeles* con requisitos y efectos positivos y negativos.

b) Un *conjunto de variables internas de un agente* (similares a predicados) que representan la situación inicial.

c) Un *conjunto de propósitos* (similares a predicados) que están sin alcanzar y se quiere que se alcancen como consecuencia de la ejecución del plan.

Cada uno de los papeles tiene un conjunto de requisitos en forma de predicados (variable_valor, variable(valor)) que se deben cumplir para aplicarlo, también un conjunto de efectos positivos y negativos. Los efectos son predicados. Los papeles tienen la estructura mostrada en la figura 4.1. Se diferencian los efectos porque se definen en forma independiente cada uno.

```
Papel(parámetros)
Requisitos         {    Predicados   }
Efectos positivos  {    Predicados   }
Efectos negativos  {    Predicados   }
```

Figura 4.1 Estructura de los papeles, en donde se tienen los requisitos que deben cubrirse para tomar el papel y los efectos que pueden darse al ejecutarlo, estos pueden ser positivos o negativos

Algunas extensiones a la planeación gráfica consideran los requisitos con negaciones [Anderson 1998], en el caso que presentamos únicamente se trata con predicados positivos, aunque es posible

integrar requisitos o efectos con negaciones dada la separación que hacemos en la descripción de los papeles en positivos y negativos.

La técnica de planeación gráfica se realiza por niveles, alternando predicados y papeles instanciados. El nivel inicial de predicados (llamado situación inicial) consta de los recursos del agente. Las papeles no crean ni destruyen objetos y el tiempo se representa de forma discreta.

El algoritmo de la planeación gráfica se muestra en la figura 4.2, el máximo de iteraciones (Máximo) es un parámetro. Los papeles se instancian con los recursos con que cuenta un agente. El plan se encuentra revisando si en los predicados del nivel generado alguno(s) corresponde(n) con la situación esperada (propósito a alcanzar).

```
Planeacion( )
{
  Se coloca el primer nivel de predicados
  Nivel = 0
  Itera hasta encontrar un plan o nivel = Maximo
  {
    Se integran a la gráfica los papeles que se puedan instanciar
    Para cada papel se colocan sus efectos positivos y negativos
    Se registran las relaciones de exclusión en los papeles y predicados
    Se revisa si hay un plan
  }
  Si hay un plan
    se indica
  sino
    se marca falla
}
```

Figura 4.2 Algoritmo de planeación gráfica polinomial (se indican los pasos generales)

El *primer nivel de predicados* se genera utilizando los parámetros de los papeles que se encuentran activos en las interacciones (existentes en el ambiente al momento de hacer planeación), asignándoles su valor de acuerdo con los recursos de un agente. Se seleccionan aquellos donde coinciden los recursos del agente.

Con base en los predicados de un nivel dado se buscan los papeles que los utilizan para revisar si unifican con los recursos del agente, en aquellos en que esto se cumple se utilizan para generar un nuevo nivel de papeles. Aquellos *que unifican sus argumentos se integran a la gráfica.*

Cada papel integrado en la gráfica se utiliza a su vez para generar un nuevo nivel de predicados empleando los argumentos de entrada para *asignar valores en sus efectos positivos y negativos*. Los efectos positivos son los que se utilizan como el nuevo nivel de predicados.

En los papeles y predicados generados se *revisan sus relaciones de exclusiones*, registrándolas en la gráfica. Los diferentes tipos de exclusiones se utilizan para generar un árbol más compacto al eliminar aquellos papeles que juntos es imposible su ejecución al generar un plan.

Cada vez que se genera un nivel de papeles y efectos, se examina la gráfica generada para determinar si se *encuentra un plan* con n pasos, mismos que son los que se han desarrollado hasta ese momento. Si se encuentra un plan, mediante el encadenamiento hacia atrás de los nodos de la gráfica

que lo forma, se indica esto. En caso contrario, se revisa si aun pueden desarrollarse más niveles en la gráfica para generar otro nivel de papeles y efectos.

Si se va a generar un nuevo nivel de papeles, se propagan los recursos del nivel actual mediante un papel llamado 'Nulo', útil solamente para esto y el cual implica que en ese nivel no se realizan acciones.

.

4.1.1 RELACIONES DE EXCLUSIÓN

Las relaciones de exclusión se dan cuando en un cierto nivel, un plan válido no puede contener a dos papeles al mismo tiempo. En un nivel de proposiciones, las exclusiones se dan cuando dos proposiciones en un mismo nivel, un plan válido no las puede generar como un efecto positivo el mismo predicado que es un requisito al mismo tiempo. Las relaciones de exclusión sirven para ayudar a generar un plan válido.

Una gráfica de planeación guarda las relaciones de exclusión mutua propagándolas a través de la gráfica usando reglas simples. Esas reglas no garantizan encontrar todas las relaciones de exclusión mutua, pero usualmente se encuentran un gran número de ellas. Algunas de las formas en las cuales dos papeles A y B en un nivel de acción se marcan como exclusivo uno de otro son:

a) Si un papel A tiene un efecto negativo de una proposición p que es requisito o efecto positivo de otro papel B. Se dice que A y B son exclusivas por interferencia (figura 4.3).

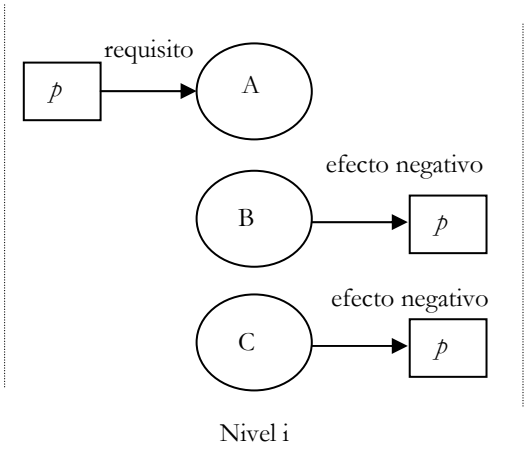


Figura 4.3 Exclusión por Interferencia. A es exclusivo de B porque p en B es requisito de A en el mismo nivel i. A es exclusivo de C porque p en C es requisito de A en el mismo nivel i

b) Si hay una requisito p de la acción A y un requisito q de la acción B que están marcadas como mutuamente exclusivas una de la otra en el nivel de proposiciones anterior (i - 1), entonces A y B son exclusivas por necesidades competitivas en el nivel actual (i) (figura 4.4).

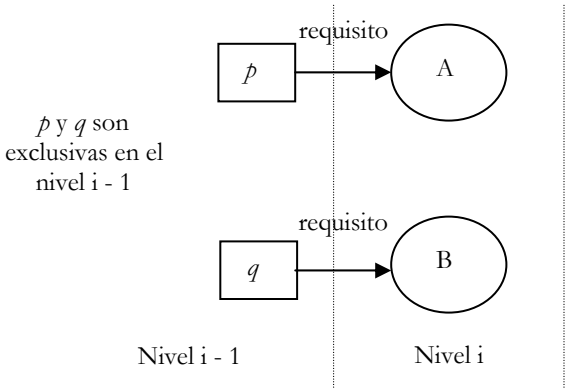


Figura 4.4 Exclusión por Necesidades Competitivas. Dado que p y q son exclusivas en el nivel i - 1, entonces en el nivel i las acciones A y B son exclusivas

c) Dos proposiciones p y q en un nivel de proposiciones se marcan como exclusivas si todas las formas de crear la proposición p son exclusivas de todas las formas de crear la proposición q. Es decir, si los papeles A y B son exclusivos y la acción A genera a p con un efecto positivo y la acción B genera a q con efecto positivo, p y q son exclusivas por propagación (figura 4.5).

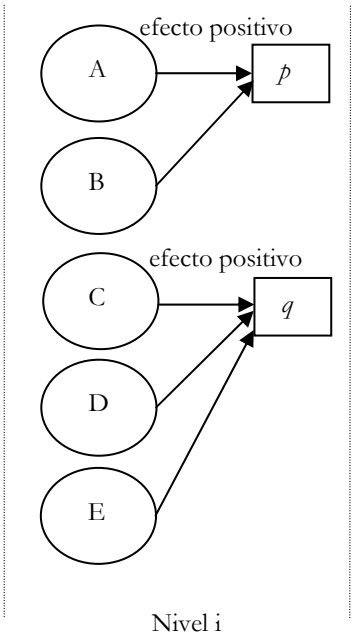


Figura 4.5 Exclusión por Propagación. Si A es exclusivo de C, D y E además de que B es exclusivo de C, D y E, entonces p y q son exclusivos por propagación

Una vez que se han marcado las relaciones de exclusión, la búsqueda de un plan válido usa una estrategia de encadenamiento hacia atrás utilizando un enfoque de nivel por nivel para hacer un mejor uso de las restricciones de exclusión mutua. En particular dado un conjunto de propósitos al tiempo t, se intenta encontrar un conjunto de papeles (incluidos los 'Nulo') en el tiempo t - 1 que tenga esas metas como efectos positivos. Los requisitos a esas acciones forman un conjunto de submetas en el tiempo t - 1 teniendo la propiedad de que si esas metas pueden alcanzarse en t - 1 pasos, entonces los propósitos originales pueden lograrse en t pasos. Si el conjunto de propósitos en el tiempo t - 1 resulta sin solución, entonces se intenta encontrar un conjunto diferente de papeles, continuando hasta que tiene éxito o se prueba que el conjunto de propósitos no es soluble en el tiempo t.

Para implementar esta estrategia se usa el siguiente método recursivo. Para cada propósito en el tiempo t, en cualquier orden arbitrario, se selecciona una acción en el tiempo t - 1 logrando que la meta no sea exclusiva de algunas acciones para las cuales ha sido seleccionada. Continúa recursivamente con el siguiente propósito en el tiempo t. (por supuesto, si por fortuna una meta ya se alcanzó por una acción seleccionada previamente, no se requiere seleccionar un papel nuevo) si nuestra llamada recursiva regresa "falla", entonces se intenta una acción diferente para lograr el propósito actual, y así sucesivamente. Una vez que se termina con todas las metas en el tiempo t, los requisitos de las acciones seleccionadas constituyen un nuevo conjunto de propósitos en el tiempo t - 1. Continúa este procedimiento en el paso t - 1. El resultado es un conjunto de papeles que forman el plan del agente.

4.2 EJEMPLO

Consideremos problema de transportar un pedido de un lugar a otro, por ejemplo, de Veracruz a Puebla. El camión para transportarlos y los artículos A y B se encuentran en el lugar de origen, el propósito es que los artículos lleguen a Puebla, lo cual se expresa como:

```
En(A, Puebla), En(B, Puebla)
```

lo cual significa que se quieren las cargas A y B en Puebla.

La situación inicial está formada por los predicados:

```
En(Camión Veracruz), En(A Veracruz), En(B Veracruz), Combustible(Camión)
```

significando que las cargas `A` y `B` se encuentran en `Veracruz` junto con el camión y este tiene combustible, el resultado que se quiere es que `A` y `B` se encuentren descargadas en `Puebla`.

En este problema se tienen 3 papeles aplicables a la situación inicial para llegar a la situación esperada, el conjunto de papeles en este caso son `Carga`, `Descarga` y `Mueve`, cada uno de ellos tienen requisitos que deben satisfacerse para que se apliquen y entonces sus efectos positivos se señalan en la situación nueva y los efectos negativos se eliminan de la situación de donde se aplican los mismos:

```
Carga(movil lugar carga)
Requisitos {
    En(movil, lugar),
    En(carga, lugar)
}
Efectos positivos{
    Dentro(carga, movil)
}
Efectos negativos{
    En(carga, lugar)
}
```

Carga se realiza a un *movil* en un *lugar* determinado, después de subir la *carga* al movil, esta ya no se encuentra en el origen

```
Descarga(movil lugar carga)
Requisitos {
     En(movil, lugar),
     Dentro(carga movil)
}
Efectos positivos {
    En(carga, lugar)
}
Efectos negativos {
    Dentro(carga, movil)
}
```

Descarga se aplica para descargar de un *movil* ubicado en un *lugar* determinado una *carga*

```
Mover(movil origen llegar)
Requisitos {
  En(movil, origen),
  Combustible(movil)
  Destino(llegar)
}
Efectos positivos {
    En(movil, destino)
}
Efectos negativos {
    En(movil, origen),
    Combustible(movil)
}
```

Mover se aplica cuando se cambia de lugar un *movil*, esto es de *origen* a *llegar*, durante el movimiento se consume combustible

Como se estableció en el algoritmo de la figura 4.2 inicialmente se genera el primer nivel de predicados describiendo la situación actual:

Nivel	Situación Inicial
I	1 En(Camión Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camión) 5 Destino(Puebla)

Utilizando los hechos de la situación inicial, estos se unifican con las acciones existentes (`carga`, `descarga`, `mueve`), obtieniendo la activación de los papeles siguientes (los números indican el predicado y la letraNúmero indica un papel a incluir en el plan):

Nivel	Situación	Papeles
I	1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Pueblall)	1 2 a1 Carga(Camion Veracruz A) 1 3 a2 Carga(Camion Veracruz B) [a2] 1 4 5 a3 Mueve(Camion Veracruz Puebla)

De los papeles que unificaron se desarrollan sus efectos positivos y negativos, obteniendo de esta forma dos columnas de predicados adicionales. La columna de efectos positivos es en este caso la que se considera como situación inicial para la unificación con papeles en el nivel siguiente:

Nivel	Situación	Papeles	Positivos	Negativos
I	1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla) EXCLUSIONES **Acciones:** a3 con a1 y a2 por interferencia con el predicado 1	1 2 a1 Carga(Camion Veracruz A) 1 3 a2 Carga(Camion Veracruz B) 1 4 5 a3 Mueve(Camion Veracruz Puebla)	6 Dentro(A Camion) 7 Dentro(B Camion) 8 En(Camión Puebla)	2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camión) 1 En(Camión Veracruz)

Una vez aplicados los papeles se busca en los efectos positivos los predicados esperados para determinar si se ha generado el estado final (`En(A, Puebla), En(B, Puebla)`), como no es el caso, se procede a propagar los predicados de la situación inicial hacia los efectos positivos (que dan lugar a la siguiente situación inicial) mediante el papel '`Nulo`'.

Después de hacer la propagación se identifican las relaciones de exclusión, en este caso, la acción a3 tiene el efecto negativo 1 que es un requisito para las acciones a1 y a2. Por lo tanto, las acciones a1 y a2 son exclusivas con la acción a3; esto significa que, en este nivel solamente se puede realizar a3 o en forma exclusiva a1 y a2: entre a1 y a2 no hay exclusión.

Los predicados 6 y 8 son exclusivos entre ellos porque todas las formas de generarlos son exclusivas. Lo mismo ocurre para los predicados 7 y 8; en forma gráfica esto es:

Nivel	Situación	Papeles	Positivos	Negativos
I	1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla) EXCLUSIONES **Acciones**: a3-a1 a3-a2 **Predicados**: 6 y 8, 7 y 8	1 2 a1 Carga(Camion Veracruz A)_____ 1 3 a2 Carga(Camion Veracruz B)_____ 1 4 5 a3 Mueve(Camion Veracruz Puebla)_ 1 a4 Nulo _ _ _ _ _ _ _ _ _ _ _ _ 2 a5 Nulo _ _ _ _ _ _ _ _ _ _ _ _ 3 a6 Nulo _ _ _ _ _ _ _ _ _ _ _ _ 4 a7 Nulo _ _ _ _ _ _ _ _ _ _ _ _ 5 a8 Nulo _ _ _ _ _ _ _ _ _ _ _ _	6 Dentro(A Camion) 7 Dentro(B Camion) 8 En(Camion Puebla) 1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla)	2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 1 En(Camion Veracruz)

En la tabla la línea continua señala los predicados de los efectos positivos y negativos generados a partir de un papel, mientras que las líneas punteadas señalan a los efectos positovos generados como efecto de propagación a partir de la situación inicial en el actual nivel.

En el nivel I no se generaron los predicados, en los efectos positivos, que permitan alcanzar los propósitos deseados, los predicados de la columna "positivos" se consideran ahora como la situación inicial para el nivel II.

En el nivel II se obtienen los papeles que unifican con los predicados de su situación inicial (Situación), obteniéndose lo siguiente:

Nivel	Situación	Papeles	Positivos	Negativos
II	1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla) 6 Dentro(A Camion) 7 Dentro(B Camion) 8 En(Camion Puebla) EXCLUSIONES **Acciones:** b5 con b3 y b4 por interferencia con el predicado 8 b5 con b1 y b2por interferencia con el predicado 1 **Predicados**: 9 y 10 con 8 por propagación 5 y 6 con 8 por propagación	1 2 b1 Carga(Camion Veracruz A)_____ 1 3 b2 Carga(Camion Veracruz B)_____ 6 8 b3 Descarga(Camion Puebla A)_____ 7 8 b4 Descarga(Camion Puebla B)_____ 1 4 5 b5 Mueve(Camion Veracruz Puebla)_ 1 b6 Nulo_ _ _ _ _ _ _ _ _ _ _ _ 2 b7 Nulo_ _ _ _ _ _ _ _ _ _ _ _ 3 b8 Nulo_ _ _ _ _ _ _ _ _ _ _ _ 4 b9 Nulo_ _ _ _ _ _ _ _ _ _ _ _ 5 b10 Nulo_ _ _ _ _ _ _ _ _ _ _ 6 b11 Nulo_ _ _ _ _ _ _ _ _ _ _ 7 b12 Nulo_ _ _ _ _ _ _ _ _ _ _ 8 b13 Nulo_ _ _ _ _ _ _ _ _ _ _	5 Dentro(A Camion) 6 Dentro (B Camion) 9 En(A Puebla) 10 En(B Puebla) 8 En(Camion Puebla) 1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla) 6 Dentro(A Camion) 7 Dentro(B Camion) 8 En(Camion Puebla)	2 En(A Veracruz) 3 En(B Veracruz) 6 Dentro(A Camion) 7 Dentro(B Camion) 4 Combustible(Camion) 1 En(Camion Veracruz)

En este nivel II se encuentran entre los efectos positivos los predicados de la situación esperada (`En(A, Puebla), En(B, Puebla)`) por lo que se procede a recorrer la gráfica hacia atrás hasta llegar a la situación inicial del nivel I donde se deben cumplir los requisitos de los papeles utilizados en ese nivel. Al hacer esto se nota que al llegar al nivel I no se cumple esta situación por lo que resulta necesario generar un nuevo nivel de papeles con los efectos positivos de la situación II como situación inicial para el nivel III:

Nivel.	Situación	Papeles	Positivos	Negativos
III	1 En(Camion Veracruz) 2 En(A Veracruz) 3 En(B Veracruz) 4 Combustible(Camion) 5 Destino(Puebla) 6 Dentro(A Camion) 7 Dentro(B Camion) 8 En(Camion Puebla) 9 En(A Puebla) 10 En(B Puebla)	1 2 c1 Carga(Camion Veracruz A)_____	5 Dentro(A Camion)	2 En(A Veracruz)
		1 3 c2 Carga(Camion Veracruz B)_____	6 Dentro (B Camion)	3 En(B Veracruz)
		1 6 c3 Descarga(Camion Veracruz A)___	2 En(A Veracruz)	5 Dentro(A Camion)
		1 7 c4 Descarga(Camion Veracruz B)____	3 En(B Veracruz)	6 Dentro(B Camion)
		6 8 c5 Descarga(Camion Puebla A)_____	<u>8 En(A Puebla)</u>	5 Dentro(A Camion)
		7 8 c6 Descarga(Camion Puebla B)_____	<u>9 En(B Puebla)</u>	6 Dentro(B Camion)
		1 4 5 c7 Mueve(Camion Veracruz Puebla)_	1 En(Camion Puebla)	4 Combustible(Camion) 1 En(Camión Veracruz)
		1 c8 Nulo_ _ _ _ _ _ _ _ _ _	1 En(Camion Veracruz)	
		2 c9 Nulo_ _ _ _ _ _ _ _ _ _	2 En(A Veracruz)	
		3 c10 Nulo_ _ _ _ _ _ _ _ _ _	3 En(B Veracruz)	
		4 c11 Nulo_ _ _ _ _ _ _ _ _ _	4 Combustible(Camion)	
		5 c12 Nulo_ _ _ _ _ _ _ _ _ _	5 Dentro(A Camion)	
	EXCLUSIONES **Acciones:** c7 con c1 y c2 por interferencia con el predicado 1	6 c13 Nulo_ _ _ _ _ _ _ _ _ _	6 Dentro(B Camion)	
		7 c14 Nulo_ _ _ _ _ _ _ _ _ _	7 En(Camion Puebla)	
		8 c15 Nulo_ _ _ _ _ _ _ _ _ _	8 En(A Puebla)	
		9 c16 Nulo_ _ _ _ _ _ _ _ _ _	9 En(B Puebla)	

En este nivel se observa que se obtienen los predicados de la situación esperada (subrayados) y entonces al satisfacer los requisitos de los predicados con el nivel anterior, esto es posible y así sucesivamente hasta llegar al nivel I donde los requisitos de los papeles que se aplican en ese nivel se satisfacen con los predicados de la situación inicial. Por lo tanto los papeles que se involucran en este recorrido hacia atrás forman el plan del agente, estos se organizan en forma de un árbol AND-OR como se muestra en la figura 4.6. Algunos papeles de esta es posible ejecutarlos en paralelo por lo que al aplicar factorización sobre el árbol (o árboles generados cuando se tienen árboles para

multipropósitos) se obtiene el resultado mostrado en la figura 4.7 que corresponde con una gráfica acíclica.

Paso	Situación	Papeles
I	1 En(Camión Veracruz) 2 En(A Veracruz) 3 En(B Veracruz)	Carga(Camión Veracruz A) Carga(Camión Veracruz B)
II	4 Combustible(Camión) 5 Dentro(A Camión) 6 Dentro(B Camión)	Mueve(Camión Veracruz Puebla)
III	5 Dentro(A Camión) 6 Dentro(B Camión) 7 En(Camión Puebla)	Descarga(Camión Puebla A) Descarga(Camión Puebla B)

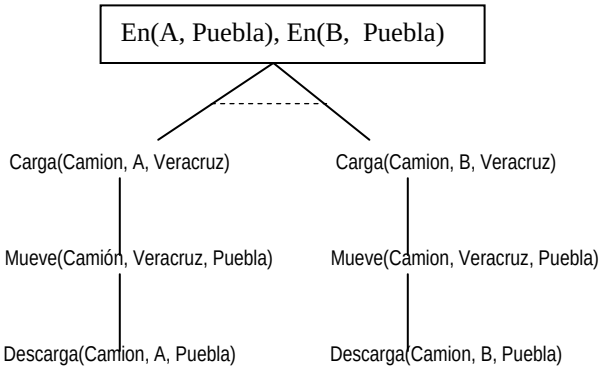


Figura 4.6 Resultado de la planeación en el problema de transporte (árbol AND-OR)

En la gráfica 4.7 para ejecutar los papeles primeramente se colocan en la cola de ejecución aquellos del nivel más superior y conforme se van ejecutando (considerando su relación AND-OR) entonces se van insertando a ejecución aquellos que de niveles más profundos. En caso que algunos de los papeles de arco AND falle entonces se considera que el arco completo ha fallado y esa rama si es que pertenece a su vez a una rama OR falló y se espera si alguno de los arcos OR da un resultado satisfactorio, si no hay opciones (OR) entonces se considera que el plan para alcanzar el propósito fallo.

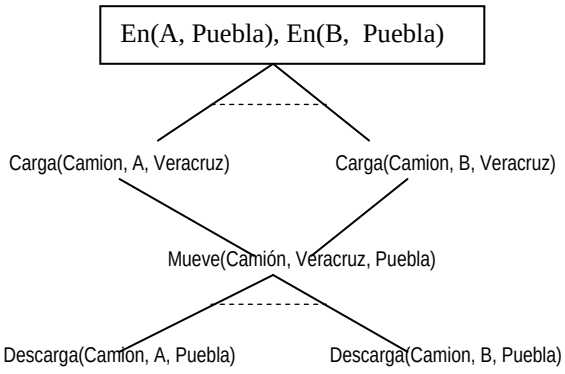


Figura 4.7 Gráfica (AND-OR) acíclica de los papeles a ejecutarse

4.3 CONCLUSIONES DE PLANEACIÓN

La planeación permite a los sistemas multiagente (con agentes multipropósito), establecer la secuencia de acciones para lograr sus propósitos. En nuestro caso, nuestra aportación consiste en aplicar la planeación para un agente multipropósitos en función de los recursos que posee y los que accede de su ambiente.

La intención de está forma de planificar es hacerlo para cada agente considerando únicamente la utilidad individual de cada uno. La consideración de obtener un plan adicionando la utilidad global de los agentes participantes es motivo de otra línea de investigación. Hasta el momento la planeación permite integrar los elementos del modelo (MIA) para elegir los papeles de un agente que le lleva de una situación inicial (propósitos sin alcanzar) a otra en donde alcanzan sus propósitos con un grado de satisfacción.

Es posible que para cada agente se obtengan varios árboles con los papeles para diferentes propósitos, en ese caso proponemos el uso de la factorización del árbol aunque habrá que verificar la pertinencia en algunos casos, dado que la factorización debe considerar el nivel del árbol en que esto se hace. La ventaja es que se evita la ejecución duplicada en algunas ramas de papeles que ejecuta un mismo agente para alcanzar varios propósitos, por ejemplo si para alcanzar dos propósitos diferentes el agente tiene que hacer un retiro de un banco entonces el papel clienteBanco para hacer el retiro se puede hacer una sola vez.

CAPÍTULO 5

EVENTOS INESPERADOS (EI)

En este capítulo se describen los problemas y las alternativas de solución en los sistemas de información ante eventos que se desconce con cuando ocurrirán (llamados por nosotros eventos inesperados). El problema específico que nos interesa en este trabajo consiste en saber cual de las acciones que un agente tiene ante eventos inesperados, que ocurren en un ambiente, es la más apropiada. La elección entre un conjunto limitado de acciones se debe a que no es posible dotar a un sistema con acciones para reaccionar ante cada imprevisto porque existe un número infinito de estos.

En los sistemas que interactuan con su ambiente, por ejemplo, una hormiga explorando un terreno en busca de fuentes de alimento(figura 5.1), o cuando alguien conduce un automóvil; las condiciones que se presentan en el ambiente tampoco son previsibles, por lo tanto tampoco las secuencias de las acciones se conocen de antemano. Cada acción que se realiza es una decisión (o reacción) que depende de las percepciones en cada momento, esto implica un tratamiento similar a como se atienden los eventos inesperados, en donde se elige la acción más apropiada ante las condiciones percibidas en un momento determinado; esto implica que un agente reacciona de una forma en un momento dado y en otra en otro momento.

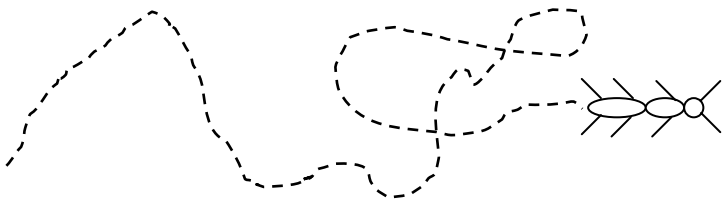


Figura 5.1 Trayectoria descrita por una hormiga buscando alimento

Para que un agente capte los eventos que ocurren en su entorno, requiere de mecanismos de percepción del ambiente que le proporcionen información. La información de las percepciones junto con las acciones que puede realizar le permite generar una decisión sobre la acción más pertinente dadas las condiciones en que se encuentra. En el ejemplo de la hormiga, esta utiliza sus antenas como sensores y las acciones que realiza son: avanza, retrocede, se detiene o da vuelta. En cada momento "decide" cual acción realizar en función de los estímulos y condiciones del ambiente. En un sistema de información las acciones son suspender el servicio y enviar un mensaje al cliente ante la caída de un servidor, o activar inmediatamente algún servidor que se tenga de respaldo.

El estudio de los elementos imprevistos se considera como una alternativa a la *Máquina de Turing* (MT) (la cual es apropiada para computar funciones pero no interacciones con entradas imprevistas). En la década de los 30s (antes del advenimiento de la computadora digital) varios matemáticos trabajaban sobre el significado de computar una función. Alonso Church y Alan Turing independientemente llegaron a conclusiones equivalentes. "Una función es computable si puede ser computada por una Máquina de Turing", este tipo de funciones se conocen como *recursivamente enumerables*. El hecho que no fuera el mecanismo más poderoso para representar computaciones era ya conocido por el mismo Turing quien mostró en 1939 que las MT con oráculo, por ejemplo, el oráculo de Delphi (que pueden predecir el futuro) eran más poderosas que las Máquinas de Turing.

La MT puede describirse como un dispositivo computacional que consiste de una cabeza de lectura/escritura que barre una cinta (posiblemente infinita) unidimensional (bidireccional) dividida en celdas, donde cada una contiene un elemento de un alfabeto finito (por ejemplo 0s, 1s) con la restricción que existe un número finito de símbolos no-blancos en las celdas de la cinta. La computación inicia con la maquina en un estado determinado, revisa el contenido de una celda, sustituye el contenido de la misma con lo que indica la transición, se mueve a la celda contigua y pasa a un estado nuevo. Las instrucciones tienen la forma:

```
(estado_actual, simbolo_actual, nuevo_estado, nuevo_simbolo, izquierda/derecha)
```

y se interpretan como sigue: al encontrarse la máquina en su `estado_actual`, con un `simbolo_actual` en la celda donde se encuentra la cabeza de lectura/escritura, cambiará su estado interno al `estado_nuevo`, reemplazando el símbolo en la cinta ubicado en la posición actual por el `nuevo_simbolo` y luego realizando un movimiento de la cabeza una celda en la dirección indicada (`izquierda` o `derecha`). Si se encuentra una condición (por ejemplo un símbolo de entrada desconocido) para el que no se tiene una instrucción, entonces se detiene la máquina, quedando la computación incompleta.

Un problema en la MT es que si cambian las entradas en forma imprevista el sistema queda estancado debido a que no cuenta con alguna forma de adicionar en su tabla de transiciones una nueva, ni tampoco un esquema que permita el tratamiento de estas entradas. Igualmente al modelo de computación dado por la MT, muchos sistemas de información al encontrarse con entradas para las cuales no tienen instrucciones simplemente se detienen. Al observar a las hormigas, personas u otros animales notamos que estos aun cuando las condiciones que les rodean cambian, son capaces de exhibir una conducta sobre el imprevisto, por ejemplo cuando una hormiga se encuentra buscando alimento y comienza a llover, reacciona dirigiéndose inmediatamente a su hormiguero; si una persona extravía su dinero, solicita ayuda a personas que se encuentren cerca o camina hacia su casa. El mismo criterio de capacidad de reacción ante imprevistos puede integrarse a los robots y a los agentes, así, un agente realizando transacciones comerciales al encontrar una oferta de productos

de los que tiene conocimiento que su usuario le interesa, puede localizarlo y avisarle para que decida si los adquiere o no, acción que el agente mismo se encargará de llevar a cabo.

Como alternativa al modelo de la MT, Wegner [Wegner 1995] ha propuesto las Máquinas de Interacción (MI) como una extensión a las primeras en el sentido de permitir manejar un alfabeto infinito, en este caso aunque existe un número finito de acciones se tiene una manera de asociar una entrada con una acción, entonces los comportamientos finitos se ejecutan en formas dictadas por los eventos externos. Las MT transforman cadenas de símbolos de entrada en cadenas de salida. En cada paso la MT lee un símbolo de entrada, hace una transición de estado y escribe un símbolo sobre una cinta de salida y vuelve a leer otro símbolo. Sin embargo, las MT no pueden aceptar símbolos externos diferentes a los que se encuentran en su alfabeto finito mientras realizan sus computaciones. En contraste las MI, funcionan, además de las reglas internas, con estímulos del ambiente lo cual, permite computaciones interactivas, como en el caso de la hormiga regida por la información del ambiente en que se encuentra.

Las MT y las MI pueden compararse mediante la métrica del comportamiento observable. Las máquinas de interacción tienen un comportamiento observable más amplio porque pueden consultar oráculos para obtener la respuesta a computaciones no algorítmicas y pueden reaccionar a secuencias generadas por los oráculos o procesos cuya naturaleza diferente a los recursivamente enumerables. De esta forma las MI secuenciales son transductores que transforman un flujo de entrada en un flujo de salida.

Wegner ha encontrado varios tipos de Máquinas de Interacción desde la más sencilla de estas que lee una secuencia de entradas y las transforma en salidas (considerada como un punto de origen en un espacio de cinco dimensiones, en cada dimensión se parte de componentes sencillos) hasta componentes más complejos:

a) *Flujos de entrada*, estos son sencillos o múltiples. La MI con entradas múltiples lee una entrada Xi de cada flujo de entrada y obtiene una Yi (respuesta) para cada una en cada paso de computación (realiza computaciones en paralelo).

b) *Sensibilidad al tiempo*, el tiempo exterior es un aspecto que las MT no pueden modelar, las MI superan esta limitación y les es posible interactuar con procesos externos en movimiento. La noción de tiempo relativo se expresa mediante secuencias de entradas ordenadas que incluyen un atributo de tiempo llamado traza.

c) *Sincronía*, si el tiempo en el que llegan los flujos de entrada se determinan externamente y no por el control de la MI se dice que es una MI asíncrona. Si la MI controla el tiempo de ejecución de las entradas entonces es una MI síncrona.

d) *Concurrencia*, las MI concurrentes pueden procesar entradas concurrentemente y dan lugar a comportamientos observables característicos que no pueden obtenerse mediante el procesamiento secuencial de entradas concurrentes. De hecho las interacciones concurrentes pueden realizarse en flujos de entrada múltiples o en un flujo de entrada único, en ambos casos la ejecución puede estar sobrelapada.

e) *Retroalimentación*, se adiciona a las MI añadiendo un estado local que transmite la historia de los pasos de computación anteriores a los más recientes.

Mediante las MI se amplia la jerarquía de Chomsky como se muestra en la figura 5.2, con lo cual se indica que las MI son un modelo que extiende los trabajos hasta ahora realizados sobre computabilidad y sistemas formales.

COMPORTAMIENTO OBSERVABLE	MECANISMOS DE COMPUTACIÓN
Conjuntos regulares	Autómata finito
Lenguajes libres de contexto	Autómata de pila
Lenguajes sensibles al contexto	Autómata lineal acotado
Funciones computables	Máquinas de Turing
Comportamiento de máquinas síncronas	Máquinas de interacción síncronas
Comportamiento de máquinas asíncronas	Máquinas de interacción asíncronas

Figura 5.2 Ampliación de la jerarquía de Chomsky con las Máquinas de Interacción (MI)

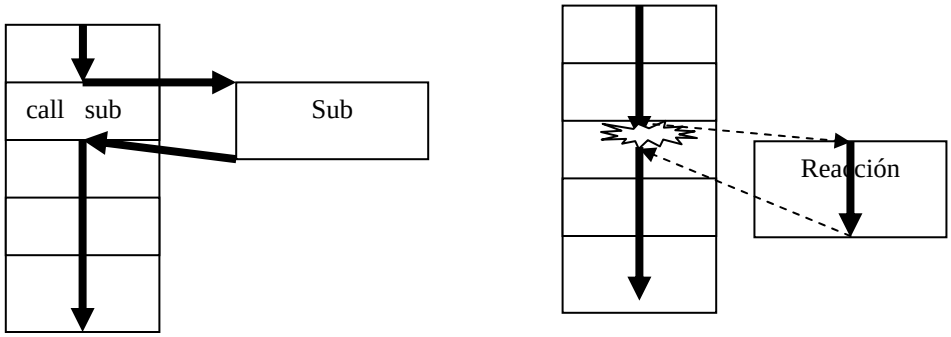
Las computaciones que toman en cuenta el exterior pueden dar origen a acciones más complejas que no pueden ser definidas por funciones computables dado que las transiciones entre estados no siguen una secuencia computable. En los sistemas reales considerar información del exterior permite tener más elementos para las decisiones, por ejemplo, los logros de los presidentes de grandes corporaciones o de países, dependen sobre el uso efectivo de la información de su ambiente. Los robots y los agentes tienen un modelo interactivo similar de computación dado el uso de la información que les proporcionan sus sensores para determinar sus acciones.

Hasta ahora muchos de los eventos que ocurren como excepciones al flujo normal de la ejecución de un proceso se han atendido como excepciones, por ejemplo el caso de una división por cero o la desconexión de un usuario de una base de datos al momento de realizar una transacción.

Cuando se realizan transacciones comerciales cuando ocurre un evento inesperado, se altera la secuencia de acciones que se está realizando y de las cuales ningún participante tiene el control. A diferencia de las excepciones, los eventos inesperados tienen diferentes acciones para su atención dependiendo de las capacidades y características del agente. Esto implica que algunos agentes continuan la ejecución normal de sus acciones mientras que otros las concluyen o activan una hebra de emergencia en lugar de las normales.

El estudio de los eventos inesperados tiene como objetivo modelar las situaciones del mundo real que afectan a los agentes y al ambiente en que estos se desenvuelven. En nuestro caso, como lo hemos visto en el capítulo 2, el ambiente se considera formado por recursos, características.

Para integrar las reacciones de emergencia a un agente que ya se encuentra realizando algunas acciones nos apoyamos en las ideas que proporciona la Programación Orientada a Aspectos [Ossher 2001] [Paniti 2001]. La cual considera a los sistemas formados por diferentes aspectos de interés programados por separado y cuya interacción se realizará en forma dinámica de acuerdo a como sean requeridos por las condiciones en que el sistema se va encontrando. Los aspectos se adicionan al sistema y se activan como consecuencia de los requerimientos de las situaciones en que opera el sistema. La integración de diferentes aspectos se hace en puntos de unión. Los puntos de unión no tienen un punto de llamado específico (figura 5.3) a diferencia de las subrutinas que tienen un punto de llamado y retorno específicos.



Llamado de una subrutina. Tiene un punto de llamado y otro de continuación fijos.

Reacción ante un evento inesperado. Se ejecuta en paralelo con otras hebras.Las hebras afectadas reinician su ejecución siempre al terminar la hebra de emergencia

Figura 5.3 Puntos de activación/continuación de una subrutinas y una reacción de emergencia. Las líneas oscuras continuas indican el flujo de control, las punteadas la activación/continuación de un proceso en paralelo

5.1 TIPOS DE EVENTOS INESPERADOS

Los *eventos inesperados que consideramos, son aquellas situaciones donde se desconoce el momento de su ocurrencia y la forma en que afectarán*, por esto algunos agente resultan afectados mientras que otros no. Durante la ocurrencia de un evento inesperado puede acontecer otro que afecte la atención al evento que ya estaba atendiéndose.

Para atender el problema en que no es posible prever todos los acontecimientos que ocurren como eventos inesperados, los organizamos en una taxonomía (Figura 5.4) formada por clases a diferentes niveles de abstracción. La raíz es el evento inesperado más general (`eventoGeneral`). Para cada nodo se indica el nombre del evento y las reacciones que sirven para que un agente lo afronte.

5.2 GENERACIÓN DE EVENTOS INESPERADOS

En el mundo real los eventos inesperados son imprevisibles, pero para efectos de nuestro modelo, los generamos de acuerdo a las características de su ocurrencia basándonos en la experiencia de su ocurrencia en lugar, tiempo y duración. Para esto se utiliza el generador de eventos inesperados, el cual es un mòdulo de SEA (véase el capítulo 2). Para cada evento generado en SEA, se indica la fecha y hora del evento, nombre y su duración. Estos es posible generarlos en forma automàtica utilizando distribuciones de probabilidad aunque por ahora se considera otra lìnea de investigación. Los eventos inesperados generados, al iniciar la ejecución de un sistema de agentes, se insertan en la cola de ejecución, que consta de un orden parcial en la línea del tiempo, los agentes no tienen acceso a la misma por lo que se considera que no pueden predecir el futuro (figura 5.5). Una vez que un evento ocurre se retira de la cola, por lo que el apuntador al inicio de la cola apunta al siguiente evento. El manejo de la afectación de los eventos inesperados en SEA se hace utilizando un módulo adicional llamado el Manejador de Eventos Inesperados (MEI), el cual se encarga de asignarle las

hebras de emergencia a los agentes que perciben el evento que ocurre en un momento determinado, también se encarga de reanudar la ejecución de los papeles suspendidos una vez que el evento inesperado ha ocurrido y que es posible continuarlos.

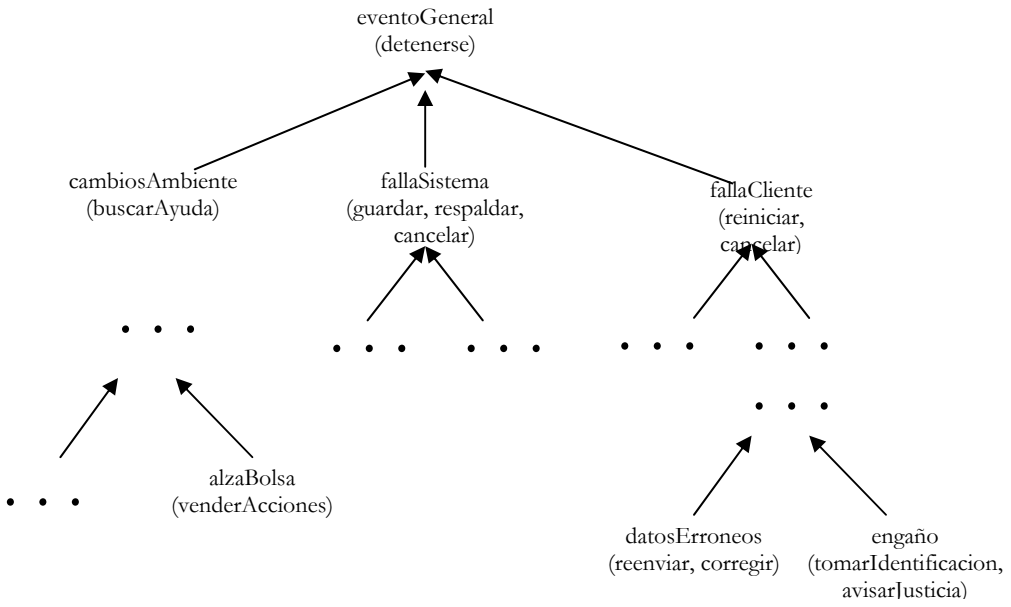


Figura 5.4 Los eventos se organizan en un árbol con su nombre y los papeles de emergencia útiles en su atención

Los eventos inesperados que se introducen a un modelo deben ser congruentes, por ejemplo, se pierde dinero cuando hay una baja en la bolsa y afecta a un agente que tiene acciones. Esto significa que los eventos inesperados guardan congruencia con el ambiente y se respeta su ocurrencia en espacio, tiempo y duración.

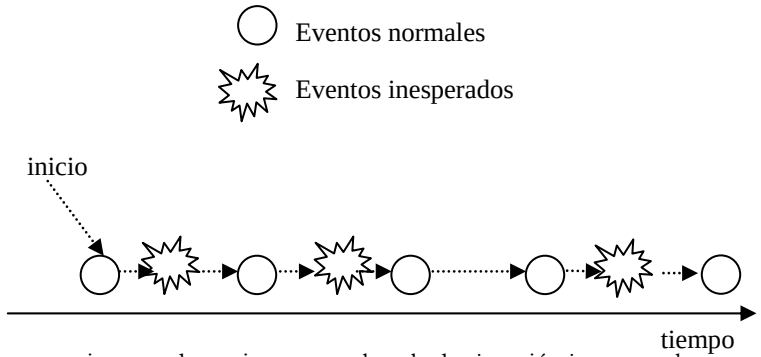


Figura 5.5 Los eventos inesperados se insertan en la cola de ejecución junto con los eventos normales de interacción entre agentes ordenados de acuerdo a su ocurrencia

En SEA los eventos inesperados que se generan se almacenan en un archivo, mismo que puede modificarse mediante las opciones utilizadas en el editor de eventos inesperados descrito en la sección 2.3. Otro elemento que forma parte de esto es el editor del árbol de eventos inesperados, en este caso y para propósitos de la simulación en el ambiente de SEA se utiliza un árbol finito pero en la realidad debería estar formado por un número infinito de eventos. En el árbol se registran los eventos y las reacciones que pueden utilizarse ante el mismo.

Las reacciones ante los eventos inesperados se almacenan en archivos diferentes expresadas en directivas de LIA que se integran al ambiente de ejecución de SEA en forma paralela y en ocasiones concurrente.

5.3 Ocurrencia de Eventos Inesperados

Dentro del sistema SEA de ejecución de agentes se integran los eventos inesperados en la cola de ejecución de hebras (véase la sección 2.3) a la cual no tienen acceso los agentes porque si esto fuera posible se consideraría que pueden visualizar el futuro y podrían entonces anticiparse a la ocurrencia de los mismos.

Una vez que el constructor ha creado las estructuras de datos del ambiente de ejecución y se tiene la cola de ejecución inicializada para ejecutar las operaciones, se cargan los eventos inesperados generando hebras que hacen referencia a los papeles que corresponden a los eventos inesperados. En la hebra del cada evento inesperado se tiene una instrucción de inicio del evento y otra con el fin del evento, estas al ejecutarse hacen llamados al módulo MEI.

Los eventos inesperados pueden favorecer o impedir que un agente alcance sus propósitos. Por lo tanto cuando ocurre un evento inesperado lo primero que se hace es identificar a los agentes que lo perciben. Para esto se revisa la lista de cada agente de eventos inesperados percibe, esto significa que algunos agentes perciben algunos eventos mientras que otros no, por ejemplo durante un temblor, un agente persona es posible que lo perciba si se encuentra quieto en una oficina, pero otro que se encuentre sobre un toro mecánico haciendo maniobras lo más probable es que pase desapercibido. Los agentes se modelan con los eventos que perciben y por lo tanto ser afectados por estos en forma positiva, negativa o nula.

Los agentes a los que afecta un evento inesperado se suspenden sus hebras de ejecución para dar lugar a la afectación del evento, suspención de las hebras que sean afectadas y la adición de hebras de emergencia, así como la alteración de sus recursos o características. Cada agente "nace" con un conjunto finito de comportamientos (papeles) que les permiten responder ante los eventos inesperados. Los eventos que el agente no percibe, no se toman en cuenta y para él son considerados como neutros.

Una vez detectados agentes afectados, se procede a buscar en el árbol el comportamiento más específico posible para el mismo. Esto significa que primero se ubica el evento inesperado en el árbol, utilizando el nombre del evento como índice. Luego se revisan los comportamientos dentro del nodo del evento para determinar cúal es el que realizará el agente. Si el agente tiene alguno de los comportamientos, se aplica ante el evento ocurrido, esto se hace generando una hebra de emergencia (este es el comportamiento más específico). Sino se encuentra la reacción se procede a subir al padre del evento que ocurrió y se toma su comportamiento, nuevamente si el agente lo tiene, se considera

como el más específico. Se continúa de esta forma hasta llegar a la raíz (eventoGeneral) el cual por definición un agente siempre tiene alguno de sus comportamientos para reaccionar.

Los eventos inesperados afectan a un agente en diferentes formas, la afectación es positiva cuando el evento lo hace adquirir más recursos o facilitarle el alcanzar un propósito; por otra parte, si se trata de un evento inesperado negativo para el agente, entonces pierde recursos o que algunas hebras en ejecución se hacen incompatibles con respecto a la de emergencia que se han tenido que activar. Esto es, la afectación de un agente por un evento insperado en nuestro modelo (MIA) y sistema de ejecución (SEA) es:

a) *Positiva*, hace que un agente de pronto cuente con recursos para intentar algunos papeles que hasta el momento dado no hubiera podido tomar, por ejemplo: ganar la lotería, encontrar dinero o a un amigo.

b) *Negativa*, provocan que alguna parte del plan de uno o varios agentes deban omitirla debido a la perdida de algunos requisitos necesarios para conservar sus papeles; por ejemplo, cuando un agente extravía dinero, la caída de un rayo en una casa.

c) *Neutra*, son aquellos que no afectan el desarrollo de un plan; por ejemplo, el agente se encuentra en México, D.F. y hay un fallo de energía en Wichita Kansas, EUA.

Durante la afectación de los eventos positivos y negativos es conveniente la activación de un módulo replanificador (o que realice ajustes al plan del agente considerando los avances que ya tenga); esté módulo se considera una línea de investigación adicional al trabajo realizado.

La afectación se refleja en los cambios de sus variables internas, regionales y globales que utiliza un agente. Los comportamientos que se integran a un agente se conocen como hebras de emergencia, y están sujetas al mismo tratamiento en caso de que ocurra otro evento inesperado. Como cada papel consta de un conjunto de requisitos, los papeles de las hebras de emergencia también los tienen y por ejemplo, no se podrá activar un comportamiento abrir paraguas si el agente no posee uno, la falta de activación de un evento inesperado por lo tanto provoca que se busque otro comportamiento para la atención del mismo.

Los agentes que se tratan en este trabajo son multihebra y por lo tanto, ejecutan varios papeles al mismo tiempo, pero estos son compatibles mediante la verificación en una tabla de compatibilidad. La forma de calcular la compatibilidad está sujeta a un trabajo adicional en otra línea de investigación.

Durante la duración del evento inesperado el agente puede ejecutar las hebras de emergencia concurrentemente con sus hebras normales. Al concluir el evento es las hebras de emergencia continúan si así lo determina el MEI, por ejemplo, cuando un servidor se daña por una descarga eléctrica es posible activar un servidor de respaldo pero el afectado continúa en la hebra reparación que le impide algunas otras acciones. En otros casos la hebra de emergencia concluye antes que termine el evento inesperado. La decisión de las hebras que terminan o siguen lo realiza el módulo manejador de eventos inesperados (MEI), este registra que una hebra fue afectada por un evento inesperado colocando un estado específico en la hebra.

Las hebras que se pueden suspender son las que se encuentran en el estado 0 (véase la sección 2.3) y pasan al estado 1. Las hebras de emergencia que se adicionan a un agente se marcan con estado 4. Si una hebra de emergencia la suspende otra hebra de emergencia se le asigna el estado 5.

Cuando se restaura la ejecución de una hebra se reasignan los estados 4 para una hebra de emergencia y 0 para una hebra normal. Si una hebra estaba suspendida esperando un mensaje entonces se asigna con el estado 3.

Las decisiones sobre la mejor forma de reactivar hebras suspendidas después de un evento inesperado es motivo de una línea de investigación dedicada a esto.

5.4 CONCLUSIONES DE EVENTOS INESPERADOS

Se ha presentado un componente del modelo presentado en el capítulo 2 relativo a la ocurrencia de eventos inesperados y la manera en que se da la reacción ante los mismos por parte de los agentes que se modelan. Esta propuesta es útil en el sentido que afronta situaciones potencialmente infinitas y que se requieren para evitar que los sistemas de información queden detenidos. El desarrollo de este tema está más detallado en la tesis de [Domínguez 2001], en nuestro caso se describe únicamente estos componentes como un complemento que forma parte del MEI, LIA y SEA.

Mediante el ambiente propuesto (SEA) se simula la ocurrencia y afectación de los agentes por eventos inesperados durante las interacciones, en algunos casos los agentes ya no alcanzan sus propósitos, otros se ven favorecidos por los eventos inesperados y para otros no hay cambio (afectación nula).

CAPÍTULO 6

EJEMPLOS

En este capitulo se presentan algunos ejemplos desarrollados como parte de esta tesis, la ejecución de los mismos se hace utilizando el modelo (MIA), el lenguaje (LIA) y el sistema (SEA) propuestos en esta tesis. Durante la ejecución de un sistema en SEA se invocan a los módulos de planeación, comparador de ontologías mixtas (COM) y el manejador de eventos inesperados (MEI). En la sección 6.1 se presenta un ejemplo en donde se tienen agentes compradores y vendedores, estos interactúan en una interacción (escenario) mercado. Para el problema del mercado se presentan tres ejecuciones, en la primera se muestra el problema utilizando ontologías donde hay equivalencia entre los conceptos que manejan los agentes y sin eventos inesperados, en la segunda se manejan dos ontologías diferentes y en la tercera se incluyen eventos inesperados. En la sección 6.2 se presenta un problema de agentes transportadores, donde cada mercancía se considera un agente y estas son capaces de moverse de un lugar de origen a uno de destino. En este caso se presenta el problema sin eventos inesperados y con ellos.

La especificación de un ambiente se hace mediante variables globales y regionales, de esta forma para un agricultor que vende su producción de maíz, es importante la producción lograda, información a la cual otros agentes compradores pueden tener acceso, el tipo de transporte disponible para llevar la mercancía a algún mercado, esto se describe en LIA como:

```
global
{
    int ProdMaiz ; // EXPRESADA EN TONELADAS
    float Precio ;  // PRECIO DE GARANTIA MINIMO (POR TONELADA DE MAIZ)
}

regional
{
     char transporte[30] ;
}
```

Los agentes se describen mediante sus variables regionales, sus recursos y características expresados en las variables internas, sus propósitos, los eventos inesperados que percibe y los papeles que le permiten reaccionar ante estos. Un agente persona por ejemplo se describe:

```
agent Persona
{
    regional {
      transporte ;
    }
    internal {
       char ubicacionBodega[30] ;
       int dinero ;
    }
    purpose { // PROPOSITOS DEL AGENTE
       comprar_auto  ;
     }
    percieve { // EVENTOS INESPERADOS QUE PERCIBE EL AGENTE
       lluvia, terremoto, plaga ;
    }
    action { // ACCIONES ANTE EVENTOS INESPERADOS
       fumigar, pedirAyuda  ;
    }
    initial { //  PAPELES INICIALES
       esperar  ;
    }
}
```

Las interacciones (escenarios) contienen los papeles que toman los agentes para alcanzar sus propósitos como se ha mencionando e ilustrado en el capítulo 2.

La sección principal de un sistema descrito en LIA generalmente contiene instrucciones para crear instancias de agentes y de interacciones. Aquí se asignan propósitos y recursos en forma individual a cada agente, en el ejemplo siguiente a dos agentes persona, uno quiere tortillas y otro ganar 10 pesos.

```
main()
{
    // CREACION DE INTERACCIONES
    addinteract(Mercado,CentralAbasto);

    // CREACION DE AGENTES
    addagent(Persona, Antonio,
         tener_TORTILLA, // PROPOSITOS DEL AGENTE
         ofrece := "dinero"; quiere := "TORTILLA"; // RECURSOS DEL AGENTE
         alimento := "TORTILLA"; dinero = 100;
         ontology := "comun");   // ONTOLOGÍA UTILIZADA POR EL AGENTE

    addagent(Persona, Rafael,
         gana_10,  // PROPOSITOS DEL AGENTE
         ofrece := "TORTILLA"; quiere := "10"; // RECURSOS DEL AGENTE
         recurso := "10"; dinero = 7;
         ontology := "tortillador" ); // ONTOLOGÍA UTILIZADA POR EL AGENTE
}
```

6.1 AGENTES COMPRADORES Y VENDEDORES

En este caso se tiene un (escenario) mercado en donde interactúan dos tipos de personas, cada una con diferentes propósitos, una tiene un producto y quiere venderlo para obtener dinero, mientras que otra tiene el propósito de adquirir el producto. En este caso se intercambiar mensajes entre los participantes en el mercado en donde uno es el comprador y otro el vendedor. El vendedor en este

caso toma la iniciativa y le envía un mensaje al vendedor sobre el producto que desea, el vendedor le responde el precio del artículo, el comprador verifica si el precio es aceptable para sus recursos económicos y en caso afirmativo realiza el pago y el vendedor le entrega la mercancía. Los principales mensajes intercambiados en este caso están en la figura 6.1.

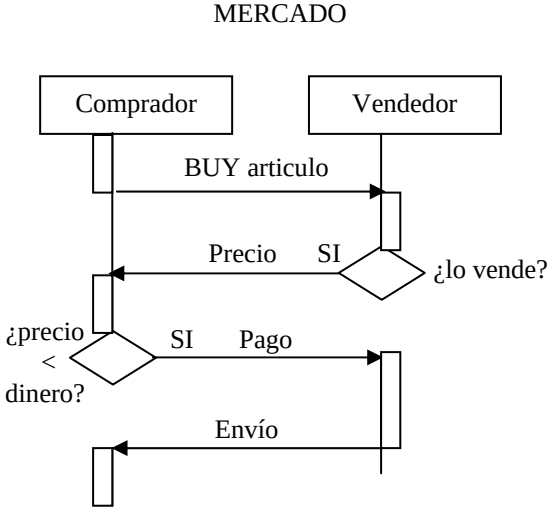


Figura 6.1 Principales mensajes que se intercambian entre un comprador y un vendedor en un mercado

El listado completo del ejemplo del mercado escrito en LIA donde se tienen un agente persona y los papeles de comprador y vendedor es el siguiente:

```
// EJEMPLO 1: Agentes compradores y vendedores
// Autor: Jesús Manuel Olivares Ceja mayo 2002
global {  // VARIABLES GLOBALES
    int Reloj ;
    port pcte ;
    port pprov ;
    char  mensaje[200],  tmp[100]  ,cual[100],
dato[50] ;
    int duracion, lectores ;
}
regional { // VARIABLES REGIONALES
     char localidad[30] ;
}
agent Persona {  // AGENTE PERSONA
    regional {
      localidad ;
    }
    internal {
       char quiere[30], ofrece[30],
         alimento[30], recurso[30] ;
       char ontology[50] ;
       float dinero ;
    }
    purpose { // PROPOSITOS DEL AGENTE
        vacaciones_cancun  ;
     }
    percieve { // EVENTOS INESP. QUE PERCIBE
        perderDinero ,encontrarDinero;
    }
    action { // ACCIONES ANTE EVENTOS INESP.
        pedirAyuda ,llorar  ;
    }
    initial { //  PAPELES INICIALES
       esperar  ;
    }
}
main()
{
    print("EJEMPLO 1: Agentes compradores y
      vendedores");

    // CREACION DE INTERACCIONES
    addinteract(Mercado,CentralAbasto);

    // CREACION DE AGENTES
    addagent(Persona, Antonio,
         tener_TORTILLA,
         ofrece := "dinero";
         quiere := "TORTILLA";
         alimento := "TORTILLA"; dinero = 100;
         ontology := "comun");

    addagent(Persona, Rafael,
         gana_10,
         ofrece := "TORTILLA"; quiere := "10";
         recurso := "10"; dinero = 7;
         ontology := "tortillador" );
}
interaction Mercado {
   role comprador(producto, recurso)
     [instances 1]
   {
```

```
 requisite
 {
     quiere_producto, ofrece_recurso,
     alimento_producto ;
 }
positive {
    tener_producto ;
}
negative {
    ofrece_recurso ;
}
local {
    char logro[100] ;
    int resultado ;
    char id_msg[30] ;
    float costo ;
    char cantidad[30] ;
    char mensaje[200] ;
    int estado ;
 }

 print("Papel: comprador");
 // SOLICITA UN PRODUCTO
 mensaje := "( BUY ARTICULO ";
 append(mensaje ,quiere);
 tmp := " )";
 append(mensaje ,tmp);
 id_msg := "solicitud";
 lectores = 2;
 duracion = 60; // EN SEGUNDOS
 out(pcte, id_msg, mensaje, lectores,
   duracion);
 // ACEPTA LA RESPUESTA
 id_msg := "respuesta";
 espera = 60;
 accept(pcte, id_msg, mensaje, espera);
 logro := "SI";
 dato := "RESPUESTA";
 extract(mensaje, dato, cual);
 compare(cual, logro, resultado);
 dato := "PRECIO";
 extract(mensaje, dato, cantidad);
 costo = cantidad ;
 estado = 0;  // SE PREPARA SI LA RESPUESTA
      // FUE 'NO' O NO LE ALCANZA EL DINERO
 // VERIFICA EL PRECIO Y DETERMINA SI SE
 // COMPRA
 if( resultado == 1 )
 {
     resultado = 0;
     if( costo <= dinero )
     {
            resultado = 1;
     }
     print("RESULTADO ");
     print(resultado);
     if( resultado == 1 )
     {
          // ACEPTA EL PRODUCTO Y CONCLUYE
          // LA COMPRA
          mensaje := "( PAY ";
          append(mensaje ,costo);
          tmp := " )";
          append(mensaje ,tmp);
          id_msg := "pago";
          lectores = 1;
          duracion = 60; // EN SEGUNDOS
          out(pcte, id_msg, mensaje,
            lectores, duracion);
          dinero = dinero - costo;
          estado = 1;
     }
     if( resultado == 0 )
     {
          // ACEPTA EL PRODUCTO Y CONCLUYE
          // LA COMPRA
          mensaje := "( PAY NO )";
          id_msg := "pago";
          lectores = 1;
          duracion = 90; // EN SEGUNDOS
          out(pcte, id_msg, mensaje,
            lectores, duracion);
      }
    }
    print(estado);
    if( estado == 1 )
    {
       // ACEPTA LA CONFIRMACION DE ENVIO
       id_msg := "confirmacion";
       espera = 100;
       accept(pcte, id_msg, mensaje, espera);
       dato := "SEND";
       extract(mensaje, dato, cual);
       tmp :=  "SI";
       compare(cual, tmp, resultado);
       if( resultado == 1 )
       {
            logro := "tener_";
            append(logro ,quiere);
            goal(logro);
            print("Dinero comprador");
            print(dinero);
       }
    }
    print(".");
}
role vendedor (bien, producto)[instances 1] {
    requisite {
        quiere_bien, ofrece_producto
          ,recurso_bien ;
    }
   positive {
       gana_bien ;
   }
   negative {
       ofrece_producto ;
   }
   local {
       char logro[100] ;
       char id_msg[30] ;
       char dato[30], cual[50] ;
       int resultado ;
    }
    print("Papel vendedor");
    // ACEPTA EL PRODUCTO A VENDER
    id_msg := "solicitud";
    espera = 100;
    accept(pcte, id_msg, mensaje, espera);
    dato:= "ARTICULO";
    extract(mensaje, dato, cual);
    compare(cual, ofrece, resultado);
    mensaje := "( SELL ARTICULO ";
    append(mensaje ,ofrece);
    tmp := " RESPUESTA NO )";
    if( resultado == 1 )
    {
          tmp := " RESPUESTA SI PRECIO ";
    }
    append(mensaje ,tmp);
    append(mensaje ,quiere);
    tmp := " )";
    append(mensaje ,tmp);
```

```
    id_msg := "respuesta";
    lectores = 1;
    duracion = 180; // EN SEGUNDOS
    print(mensaje);
    out(pcte, id_msg, mensaje, lectores,
      duracion);


    // ACEPTA EL PAGO O EL RECHAZO AL PRECIO
    id_msg := "pago";
    espera = 100;
    accept(pcte, id_msg, mensaje, espera);
    dato:= "PAY";
    extract(mensaje, dato, cual);
    dato := "NO";
    compare(cual, ofrece, resultado);
    if( resultado == 1 )
    {
        print("No se realizó la venta");
    }
    if( resultado == 0 )
    {
            // NOTIFICA EL ENVIO DEL PRODUCTO
            mensaje := "( SEND SI )";
            id_msg := "confirmacion";
            lectores = 1;
            duracion = 180; // EN SEGUNDOS
            out(pcte, id_msg, mensaje,
             lectores, duracion);
            print(
            "ya envie la confirmacion");
            print(mensaje);
            dinero = dinero + quiere;
            logro := "gana_10";
            goal(logro);
            print("Dinero vendedor");
            print(dinero);
    }
     print(".");
  }
}
```

En la figura 6.2 se muestran los eventos que se generan al ejecutar el sistema de compra-venta y donde hay agentes persona y el planificador le asigna el papel de comprador a uno y al otro le asigna el papel de vendedor. En este caso los agentes utilizan ontologías con equivalencia entre sus conceptos lo cual se nota al invocar al COM (comparador de ontologías mixtas). Es de notar que en este caso se está excento de eventos inesperados. Los mensajes que se intercambian indican el producto que desea el comprador y que en este caso el vendedor si puede surtir: `(BUY ARTICULO TORTILLA)` (se utilizan mayúsculas aquí para indicar conceptos). Luego se tiene la respuesta que da el vendedor, `( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO 10 )`, a lo que el comprador acepta el precio en este caso realizando el pago `( PAY 10.00 )` y finalmente el vendedor le notifica que ha enviado el producto `( SEND SI ).` Se observa que en este caso el comprador originalmente contaba con dinero = 100 y el vendedor dinero = 7, después de la interacción el primero quedó con 90, mientras que el segundo queda con 17. Ambos agentes alcanzan sus respectivos propósitos, aunque dado que cada uno tenía un propósito (`vacaciones_cancun`) que no cubren quedan con solamente 50% de satisfacción cada uno y por lo tanto el grado de satisfacción del sistema también es del mismo porcentaje como se indica en las estadísticas finales.

En este ejemplo los eventos inesperados ocurren en cualquier punto de la interacción entre agentes (figura 6.3), representados con una nube de estrella. En la figura 6.4 se observa la pantalla de ejecución con los eventos inesperados. En esos puntos se utilizan papeles de emergencia para que el agente afectado supere el evento inesperado o se cambien sus propósitos. Al ocurrir en el ejemplo el evento inesperado `perderDinero` se afecta al agente comprador y como se observa en la bitácora resultante (figura 6.5) al quedar sin dinero: `1/01/2000 9:04:30 Antonio: Al 'perder dinero' le queda: 0.000000` ya no es posible que alcance su propósito de comprar tortillas. El código del papel que afecta a los agentes del evento inesperado es el siguiente, donde notamos que solamente los agentes con más de 50 de dinero son los afectados:

```
role perderDinero()
{
  requisite
    {
      ninguno ;
    }
   local
    {
      int resulta ;
      int cantidad ;
    }
    print("Dinero inicial");
    print(dinero);
    cantidad = dinero;
    print("CANTIDAD");
    print(cantidad);
    resulta = 0;
    if( cantidad > 50 )
    {
         resulta = 1;
```

```
}
if( resulta == 1 )
{
    dinero = 0;
        print("Al 'perder dinero' le queda");
        print(dinero);
    }
}
```

```
1/01/2000 9:00:00 EJEMPLO 1: Agentes compradores y vendedores
1/01/2000 9:00:05 Interaction CentralAbasto from Mercado created
1/01/2000 9:00:10 Agent Antonio from Persona created
1/01/2000 9:00:55 Antonio starts esperar
1/01/2000 9:00:55 Plan for Antonio
        vacaciones_cancun
                Plan not found
        Tener_TORTILLA
              0 comprador from Mercado
1/01/2000 9:00:55 Antonio starts comprador from Mercado
1/01/2000 9:00:56 Antonio: Papel: comprador
1/01/2000 9:01:00 Agent Rafael from Persona created
1/01/2000 9:01:45 Rafael starts esperar
1/01/2000 9:01:45 Plan for Rafael
        vacaciones_cancun
                Plan not found
        Gana_10
              0 vendedor from Mercado
1/01/2000 9:01:45 Rafael starts vendedor from Mercado
1/01/2000 9:01:46 Rafael: Papel vendedor
1/01/2000 9:01:56 Antonio out solicitud ( BUY ARTICULO TORTILLA )
1/01/2000 9:02:11 COM from Antonio Rafael
1/01/2000 9:02:11 Rafael accept solicitud ( BUY ARTICULO TORTILLA )
1/01/2000 9:04:26 Rafael: ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO 10 )
1/01/2000 9:04:31 Rafael out respuesta ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO 10 )
1/01/2000 9:04:31 COM from Rafael Antonio
1/01/2000 9:04:31 Antonio accept respuesta ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO
10 )
1/01/2000 9:07:21 Antonio: RESULTADO
1/01/2000 9:09:06 Antonio out pago ( PAY 10.000000 )
1/01/2000 9:09:06 COM from Antonio Rafael
1/01/2000 9:09:06 Rafael accept pago ( PAY 10.000000 )
1/01/2000 9:11:21 Rafael out confirmacion ( SEND SI )
1/01/2000 9:11:26 Rafael: ya envie la confirmacion
1/01/2000 9:11:31 Rafael: ( SEND SI )
1/01/2000 9:11:46 COM from Rafael Antonio
1/01/2000 9:11:46 Antonio accept confirmacion ( SEND SI )
1/01/2000 9:12:21 Rafael reached gana_10
1/01/2000 9:12:26 Rafael: Dinero vendedor
1/01/2000 9:12:31 Rafael: 17.000000
1/01/2000 9:12:36 Rafael: .
1/01/2000 9:13:01 Antonio reached tener_TORTILLA
1/01/2000 9:13:06 Antonio: Dinero comprador
1/01/2000 9:13:11 Antonio: 90.000000
1/01/2000 9:13:16 Antonio: .


            GRADO DE SATISFACCIÓN DE LOS AGENTES CREADOS
            -------------------------------------------

AGENTE    INSTANCIA            PROPÓSITOS  ALCANZADOS  % SATISFACCIÓN

Persona
         Rafael                       2           1     50.00
         Antonio                      2           1     50.00


ESTADISTICA DEL SISTEMA               4           2     50.00
```

Figura 6.2 Principales mensajes que se intercambian entre un comprador y un vendedor en un mercado

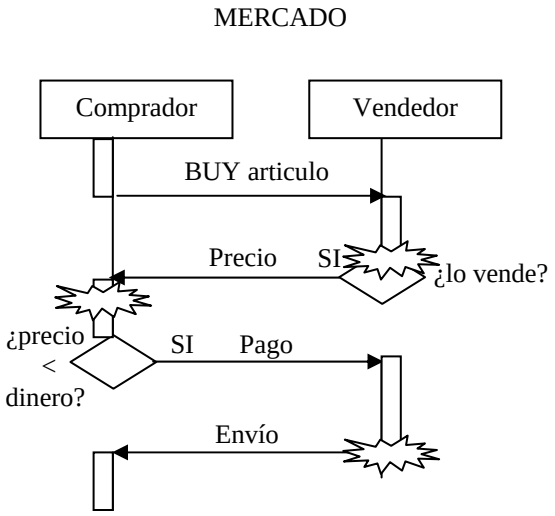


Figura 6.3 La ocurrencia de eventos inesperados sucede en cualquier punto del flujo de una interacción. La afectación es directa a un agente o indirecta cuando un agente se queda sin su interlocutor

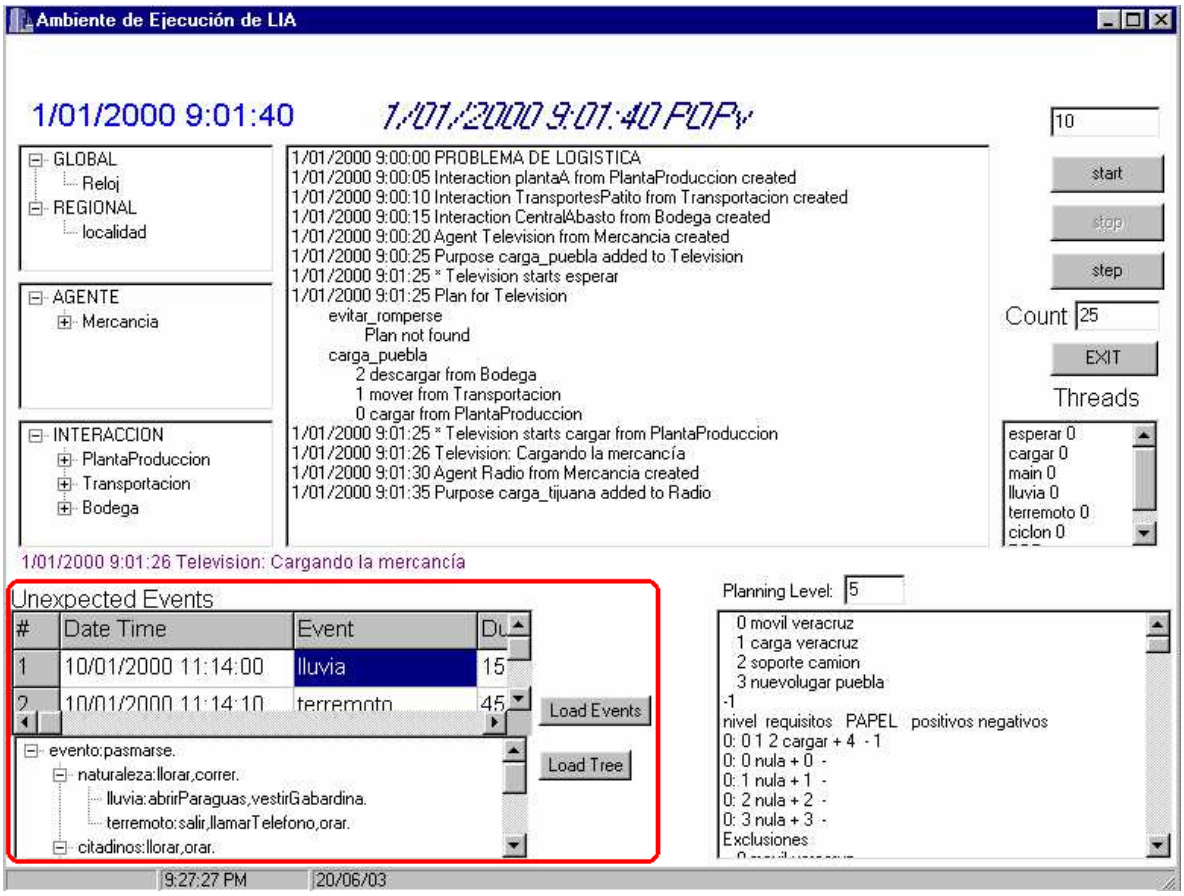


Figura 6.4 Pantalla de SEA con los eventos inesperados y el árbol de búsqueda del papel más específico

La atención al evento inesperado es el resultado de seleccionar un papel del árbol de eventos inesperados, en este ejemplo el papel hallado como el más específico para la atención del evento inesperado es llorar, su código LIA es:

```
role llorar()
{
  requisite
    {
      ninguno ;
    }
   local
    {
      int paso ;
    }
    print("llorando");
    paso = 0;
    while( paso < 3 )
   {
        paso = paso + 1;
        wait(1); // EN décimas de segundo
        print("... llora ");
  }
  print("Termino de llorar");
}
```

Cuando los agentes participantes utilizan ontologías diferentes por ejemplo entre un agente mecánico y un agente biologo y no cuentan con equivalencias entre sus conceptos entonces el módulo COM le es imposible encontrar la equivalencia entre los conceptos de los agentes, cuando ocurre esto, en los mensajes aparece la palabra reservada not_found en cada concepto en que no se encontró equivalencia, estos elementos los detecta entonces el agente y le permite conocer que no se logró entender con otro agente de donde resulta necesario tomar alguna acción o concluir la ejecución de su papel debido a que no logrará comunicarse, el fragmento de la ejecución en donde se observa la situación del mensaje cuando se manejan ontologías mixtas que no tienen equivalencia es:

```
1/01/2000 9:01:45 Rafael starts vendedor from Mercado
1/01/2000 9:01:46 Rafael: Papel vendedor
1/01/2000 9:01:56 Antonio out solicitud ( BUY ARTICULO TORTILLA )
1/01/2000 9:02:11 COM from Antonio Rafael
1/01/2000 9:02:11 Rafael accept solicitud ( BUY not_found not_found )
1/01/2000 9:04:21 Rafael: ( SELL ARTICULO TORTILLA RESPUESTA NO )10 )
1/01/2000 9:04:26 Rafael out respuesta ( SELL ARTICULO TORTILLA RESPUESTA NO )10 )
1/01/2000 9:04:26 COM from Rafael Antonio
1/01/2000 9:04:26 Antonio accept respuesta ( SELL not_found not_found not_found
not_found )10 )
1/01/2000 9:06:06 Antonio: 0.000000
1/01/2000 9:06:51 Antonio: .
```

En este caso los agentes participantes no alcanzan sus propósitos por no entenderse:

```
            GRADO DE SATISFACCIÓN DE LOS AGENTES CREADOS
            -------------------------------------------
AGENTE    INSTANCIA            PROPÓSITOS  ALCANZADOS  % SATISFACCIÓN
Persona
          Rafael                        2           0      0.00
          Antonio                       2           0      0.00

ESTADISTICA DEL SISTEMA                 4           0      0.00
```

```
1/01/2000 9:00:00 EJEMPLO 1: Agentes compradores y vendedores
1/01/2000 9:00:05 Interaction CentralAbasto from Mercado created
1/01/2000 9:00:10 Agent Antonio from Persona created
1/01/2000 9:00:55 Antonio starts esperar
1/01/2000 9:00:55 Plan for Antonio
        vacaciones_cancun
                Plan not found
        tener_TORTILLA
              0 comprador from Mercado
1/01/2000 9:00:55 Antonio starts comprador from Mercado
1/01/2000 9:00:56 Antonio: Papel: comprador
1/01/2000 9:01:00 Agent Rafael from Persona created
1/01/2000 9:01:45 Rafael starts esperar
1/01/2000 9:01:45 Plan for Rafael
        vacaciones_cancun
                Plan not found
        gana_10
              0 vendedor from Mercado
1/01/2000 9:01:45 Rafael starts vendedor from Mercado
1/01/2000 9:01:46 Rafael: Papel vendedor
1/01/2000 9:01:56 Antonio out solicitud ( BUY ARTICULO TORTILLA )
1/01/2000 9:02:11 COM from Antonio Rafael
1/01/2000 9:02:11 Rafael accept solicitud ( BUY ARTICULO TORTILLA )
1/01/2000 9:04:26 Rafael: ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO 10 )
1/01/2000 9:04:30 ***** perderDinero
1/01/2000 9:04:30 Antonio: Dinero inicial
1/01/2000 9:04:30 Antonio: 100.000000
1/01/2000 9:04:30 Antonio: CANTIDAD
1/01/2000 9:04:30 Antonio: 100.000000
1/01/2000 9:04:30 Antonio: Al 'perder dinero' le queda
1/01/2000 9:04:30 Antonio: 0.000000
1/01/2000 9:04:30 Rafael: Dinero inicial
1/01/2000 9:04:30 Rafael: 7.000000
1/01/2000 9:04:30 Rafael: CANTIDAD
1/01/2000 9:04:30 Rafael: 7.000000
1/01/2000 9:04:35 Rafael: llorando
1/01/2000 9:04:35 Antonio: llorando
1/01/2000 9:04:45 *end* perderDinero
1/01/2000 9:04:31 Rafael out respuesta ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO 10 )
1/01/2000 9:04:31 COM from Rafael Antonio
1/01/2000 9:04:31 Antonio accept respuesta ( SELL ARTICULO TORTILLA RESPUESTA SI PRECIO
10 )
1/01/2000 9:06:31 Antonio: ... llora
1/01/2000 9:06:31 Rafael: ... llora
1/01/2000 9:07:06 Antonio: 0.000000
1/01/2000 9:08:17 Rafael: ... llora
1/01/2000 9:08:17 Antonio: ... llora
1/01/2000 9:09:11 Antonio out pago ( PAY NO )
1/01/2000 9:09:11 COM from Antonio Rafael
1/01/2000 9:09:11 Rafael accept pago ( PAY NO )
1/01/2000 9:10:03 Antonio: ... llora
1/01/2000 9:10:03 Rafael: ... llora
1/01/2000 9:10:16 Rafael: No se realizó la venta
1/01/2000 9:11:03 Antonio: Termino de llorar
1/01/2000 9:11:03 Rafael: Termino de llorar
1/01/2000 9:11:15 Antonio: termina la espera
1/01/2000 9:12:05 Rafael: termina la espera

            GRADO DE SATISFACCIÓN DE LOS AGENTES CREADOS
            -------------------------------------------
AGENTE    INSTANCIA            PROPÓSITOS  ALCANZADOS  % SATISFACCIÓN

Persona
          Rafael                        2           0      0.00
          Antonio                       2           0      0.00

ESTADISTICA DEL SISTEMA                 4           0      0.00
```

Figura 6.5 Bitacora ante la ocurrencia del evento inesperado `perderDinero`

6.2 AGENTES TRANSPORTADORES

En esta sección se tiene mercancía que se espera llegue a un destino específico. Para evitar que un agente externo se encargue de estar validando el destino de cada producto, en este caso cada agente es una mercancía. Las mercancías participarán en diferentes interacciones para cargarse en el transporte adecuado, luego esperarán a llegar a su destino y ahí se descargarán.

En el proceso principal se crean tres agentes, dos con destino a Puebal y uno a Tijuana.

Las interacciones son `Almacén`, en donde se puede cargar mercancía. `Transportadora`, donde se realiza el movimiento de la mercancía y el `Destinatario` en donde se descarga la mercancía.

El listado en LIA del sistema en donde se tiene este problema se indica a continuación.

```
global
{
    int Reloj ;
}
regional
{
     float localidad ;
}
agent Mercancia
{
    regional
    {
      localidad ;
    }
    internal
    {
       char carga[30],
                movil[30],
                transporte[30] ,
               origen[30] ,
               destino[30],
               nuevolugar[30],
               soporte[30]  ;
    }
    purpose // PROPOSITOS DEL AGENTE
    {
        evitar_romperse  ;
     }
    percieve // EVENTOS INESP.
    {
       lluvia, terremoto, caidas  ;
    }
    action  // ACCIONES ANTE EVENTOS INESP.
    {
       pedirAyuda  ;
    }
    initial //  PAPELES INICIALES
    {
       esperar  ;
    }
}

main()
{
   print(
     "EJEMPLO 2: PROBLEMA DE TRANSPORTACION");
    addinteract(Almacen ,centralAbasto );
    addagent(Mercancia ,Television,
          carga_puebla,
          carga := "veracruz";
          movil := "veracruz";
          transporte := "camion";
          soporte := "camion";
          nuevolugar := "puebla";
          origen:="veracruz";
          destino := "puebla";
          money=1000;
          ontology := "electrodomestico" );
    addagent(Mercancia , Radio,
          carga_tijuana,
          carga := "veracruz";
          movil := "veracruz";
          transporte := "camion";
          soporte := "camion";
          nuevolugar := "tijuana";
          origen:="veracruz";
          destino := "tijuana";
          money=10000;
          ontology := "radios" );
    addagent(Mercancia , Refrigerador,
          carga_puebla,
          carga := "veracruz";
          movil := "veracruz";
          transporte := "camion";
          soporte := "camion";
          nuevolugar := "puebla";
          origen:="veracruz";
          destino := "puebla";
          money=1000;
          ontology := "blancos" );
}
interaction Almacen
{
  role cargar (lugar, transporte)[instances 1]
  {
    requisite
    {
      movil_lugar, carga_lugar,
      soporte_transporte ;
    }
```

```
   positive {
     carga_transporte ;
   }
   negative {
      carga_lugar ;
   }
   local
    {
       char logro[100] ;
    }
   print("Cargando la mercancía");
   logro := "carga_";
   append(logro, transporte);
   goal(logro);
  }
}
interaction Transportacion
{
role mover(origen, destino,
  transporte)[instances 1]
  {
    requisite
    {
        movil_origen ,nuevolugar_destino,
        carga_transporte ,soporte_transporte;
    }
   positive {
           movil_destino ;
   }
   negative {
          movil_origen ;
   }
   local
    {
       char logro[100] ;
       char mensaje[100] ;
       char hacia[20];
    }
    mensaje := "Moviendo desde ";
    hacia := " hacia ";
    append(mensaje ,origen);
    append(mensaje ,hacia);
    append(mensaje ,destino);
    print(mensaje);
    logro := "movil_";
    append(logro ,destino);
    goal(logro);
  }
}
interaction Destino
{
  role descargar(lugar, transporte)
    [instances 1]
  {
    requisite
    {
        movil_lugar, carga_transporte,
        soporte_transporte  ;
    }
   positive {
          carga_lugar ;
   }
   negative {
          carga_transporte ;
   }
   local
    {
       char logro[100] ;
    }
   print("Descargando el camión");
   logro := "carga_";
   append(logro, lugar);
   goal(logro);
  }
}
```

En este caso dos de las mercancías alcanzan su propósito de llegar y por lo tanto el grado de satisfacción del sistema es del 33.33 %, como se observa en la figura 6.6 que contiene la bitacora de la ejecución de este caso.

En este caso los agentes no intercambian mensajes durante su interacción sino que principalmente el problema radica en elegir la ruta apropiada para llegar a su destino (alcanzar su propósito), por lo que el proceso de planeación les determina a cada uno de estos agentes los papeles que los han de llevar de su origen a su destino.

En este caso tampoco se incluyen eventos inesperados por lo que las acciones de los agentes se hacen sin interrupción.

```
1/01/2000 9:00:00 EJEMPlO 2: PROBLEMA DE TRANSPORTACIÓN
1/01/2000 9:00:05 Interaction plantaA from PlantaProduccion created
1/01/2000 9:00:10 Interaction TransportesPatito from Transportacion created
1/01/2000 9:00:15 Interaction CentralAbasto from Bodega created
1/01/2000 9:00:20 Agent Television from Mercancia created
1/01/2000 9:01:25 Television starts esperar
1/01/2000 9:01:25 Plan for Television
        evitar_romperse
                Plan not found
        carga_puebla
              2 descargar from Bodega
              1 mover from Transportacion
              0 cargar from PlantaProduccion
1/01/2000 9:01:25 Television starts cargar from PlantaProduccion
1/01/2000 9:01:26 Television: Cargando la mercancía
1/01/2000 9:01:30 Agent Radio from Mercancia created
1/01/2000 9:01:41 Television reached carga_camion
1/01/2000 9:01:46 starts mover from Transportacion
1/01/2000 9:02:12 Television: Moviendo desde veracruz hacia puebla
1/01/2000 9:02:27 Television reached movil_puebla
1/01/2000 9:02:32 starts descargar from Bodega
1/01/2000 9:02:33 Television: Descargando el camión
1/01/2000 9:02:35 Radio starts esperar
1/01/2000 9:02:35 Plan for Radio
        evitar_romperse
                Plan not found
        carga_tijuana
              2 descargar from Bodega
              1 mover from Transportacion
              0 cargar from PlantaProduccion
1/01/2000 9:02:35 Radio starts cargar from PlantaProduccion
1/01/2000 9:02:36 Radio: Cargando la mercancía
1/01/2000 9:02:40 Agent Refrigerador from Mercancia created
1/01/2000 9:02:48 Television reached carga_puebla
1/01/2000 9:02:51 Radio reached carga_camion
1/01/2000 9:02:56 starts mover from Transportacion
1/01/2000 9:03:22 Radio: Moviendo desde veracruz hacia tijuana
1/01/2000 9:03:37 Radio reached movil_tijuana
1/01/2000 9:03:42 starts descargar from Bodega
1/01/2000 9:03:43 Radio: Descargando el camión
1/01/2000 9:03:45 Refrigerador starts esperar
1/01/2000 9:03:45 Plan for Refrigerador
        evitar_romperse
                Plan not found
        carga_puebla
              2 descargar from Bodega
              1 mover from Transportacion
              0 cargar from PlantaProduccion
1/01/2000 9:03:58 Radio reached carga_tijuana
1/01/2000 9:11:45 Television: termina la espera
1/01/2000 9:12:55 Radio: termina la espera
1/01/2000 9:14:05 Refrigerador: termina la espera

             GRADO DE SATISFACCIÓN DE LOS AGENTES CREADOS
             -------------------------------------------
AGENTE    INSTANCIA            PROPÓSITOS  ALCANZADOS  % SATISFACCIÓN

Mercancia
          Refrigerador                 2           0      0.00
          Radio                        2           1     50.00
          Television                   2           1     50.00

ESTADISTICA DEL SISTEMA                6           2     33.33
```

Figura 6.6 Bitacora del problema de transportación donde las mercancías son agentes

CONCLUSIONES

En este trabajo se presentó el desarrollo de un Modelo de Interacción entre Agentes (MIA), a partir del cual, se genera un lenguaje para definir agentes con propósitos y las interacciones (escenarios) en donde participan estos para alcanzar sus propósitos. El lenguaje le nombramos LIA (Lenguaje de Interacción entre Agentes) y el sistema donde se interpreta el código LIA traducido es llamado Sistema de Ejecución de Agentes (SEA), en este se lleva a cabo la ejecución de los ambientes y agentes descritos en LIA para lo cual primeramente se construyen las estructuras de datos que se necesitan y posteriormente se inicia la ejecución. Al inicio de la ejecución se cargan los eventos inesperados y el código de `main`, cada vez que se crea una instancia de un agente se activa un módulo de planeación para determinar los papeles que le permiten a este alcanzar sus propósito, durante la ejecución de estos papeles se activa el módulo comparador de ontologías mixtas cuando se envían mensajes entre agentes para establecer el mapeo entre conceptos de ontologías diferentes, también se invoca el manejador de eventos inesperados cuando ocurre y termina un evento inesperado.

El uso del módulo de planeación en función de los papeles disponibles en el ambiente y de los recursos y características de un agente hace diferente a nuestra propuesta de otros enfoques en donde los agentes cuentan con capacidades predefinidas y no pueden cambiar durante su ejecución. Otra de nuestras aportaciones es la capacidad de los agentes de ejecutar en paralelo los papeles de su plan cuando hay compatibilidad entre estos y que nombramos "agentes multihebra".

El enfoque que utilizamos para el mapeo de ontologías hace diferente a nuestra propuesta respecto a otras que establecen que para encontrar la equivalencia entre los conceptos que manejan dos agentes se recurre a una ontología de nivel superior (top ontology) o a la unión de dos ontologías para así referirse a conceptos comunes. El trabajo involucrado en generar una ontología o unir dos existentes hace que nuestra propuesta resulte más promisoria debido a que solamente se hace el mapeo entre los conceptos requeridos, evitando encontrar mapeos entre miles de conceptos que típicamente contiene una ontología.

En conclusión hemos propuesto una herramienta para modelar e implantar sistemas de agentes que contiene características que resultan útiles para problemas en donde se involucran agentes multipropósitos con capacidad de ejecución multihebra.

RECOMENDACIONES

A lo largo del texto hemos comentado algunas líneas de investigación que amplian el alcance del trabajo desarrollado y que al integrarlas brindarán una herramienta con mayores capacidades de modelación de los problemas relacionados con el uso de agentes, su entendimiento en el proceso de comunicación y su capacidad para reaccionar ante eventos inesperados. Sugerimos que las investigaciones sobre agentes y temas relacionados se realicen siguiendo en paralelo dos líneas: una desde el punto de vista experimental tomando casos de situaciones reales que permitan identificar los requerimientos y características de los problemas que se tienen en la realidad y la otra desde el punto de vista teórico para aprovechar los resultados de investigaciones que se realizan; esta sugerencia tiene como fundamento el constante desarrollo de estas áreas.

Resulta conveniente participar en foros de discusión virtuales y presenciales para fomentar el intercambio de conocimientos en cuanto a los resultados de investigación que se están realizando en diferentes partes del mundo.

Se sugiere la construcción de un repositorio de información sobre Agentes con conocimiento de dominio público en donde se cuente con artículos de investigación (como ocurre con ACM o Research Citation Index), software, hardware y los elementos que faciliten el conocimiento y aplicación de los resultados actuales de investigación en agentes, ontologías mixtas y eventos inesperados.

Se sugiere continuar profundizando en el tema del mapeo de ontologías mixtas para lograr una mejor comprensión tanto en los sistemas de agentes como en los sistemas de información sin agentes. Una de las ventajas que se obtiene al reducir la cantidad de información que requiere intercambiarse entre dos entidades (por ejemplo, dos empresas) es que facilita el proceso de interacambio de productos, favoreciendo los negocios entre ellas.

Con la experiencia obtenida en la realización de esta tesis y con base en las tendencias que hemos podido observar en el transcurso del desarrollo de la misma, estimamos conveniente generar varios trabajos como continuación de este proyecto. Entre ellos generar un traductor de LIA al lenguaje de programación Java, incluyendo la visualización gráfica de los resultados.

En cuanto al manejo de la satisfacción o insatisfacción de un agente es posible que durante la ejecución se pueda realizar algún proceso de replanificación o reajuste de los propósitos de un agente. La selección se continúa haciendo con base en los papeles disponibles en el ambiente y a los recursos y características de un agente. También es posible adecuar los propósitos de un agente a los recursos con que cuenta, por ejemplo, si su propósito es `comprar_autoNuevo` y tiene poco dinero, se cambiará su propósito a `comprar_autoUsado` obteniendo como consecuencia que el agente eleve su grado de satisfacción.

Hasta ahora se desarrollo y presentó el sistema para una computadora, pero es posible extender a SEA para funcionar en forma distribuida colocando en cada nodo . Un agente funcionando en un nodo puede acceder a otros papeles que se ejecutarán en el nodo en que se encuentran pero los cambios a los recursos del agente se reflejan en el nodo donde reside el mismo (figura A), los nodos se representan con cuadros con puntas redondeadas, las interacciones (escenarios) con rectángulos, los agentes con figuras de agente, lo ovalos son papeles y las flechas indican los papeles que tomó un agente.

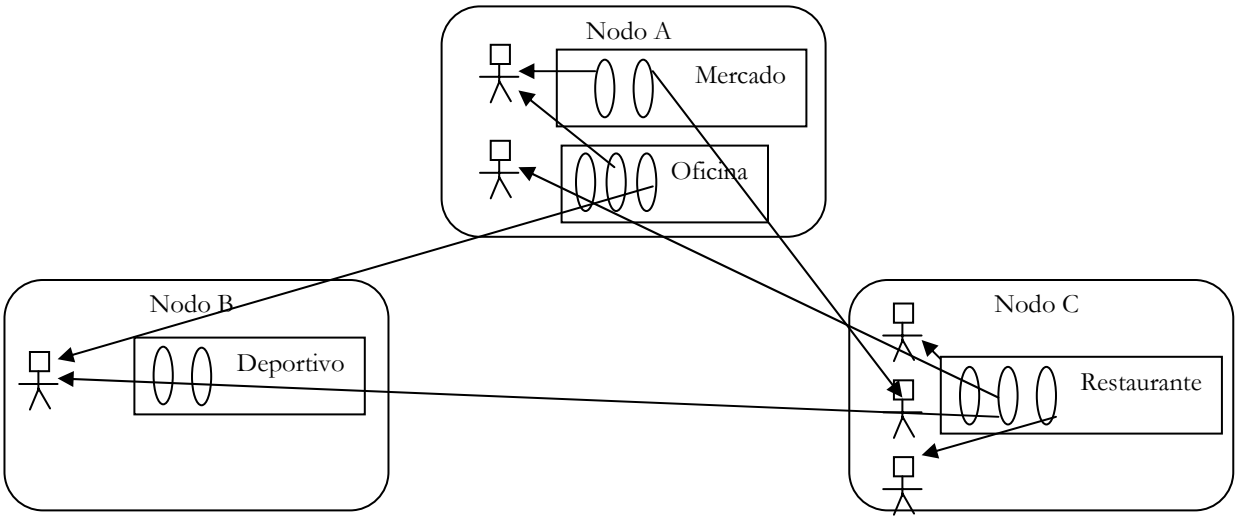


Figura A Ambiente SEA distribuido, un agente ubicado en un nodo ejecuta papeles en otro nodo pero los resultados se reflejan en el agente

ANEXO A PLATAFORMAS PARA DESARROLLAR AGENTES

Hasta ahora diversas empresas y universidades han desarrollado diferentes plataformas para implementar sistemas de agentes, algunas de propósito específico y otras de propósito general. Más de la mitad de proyectos utilizan el lenguaje de programación Java como base para implantar sistemas de agentes por su capacidad de funcionar sobre Internet y por su máquina virtual que le permite funcionar en diferentes plataformas de hardware.

Los desarrollos se han hecho desde el punto de vista comercial y como proyectos académicos y de investigación[16]

La tabla que sigue contiene algunos de los proyectos que proponen una plataformas para desarrollo de agentes, en cada una se indica su nombre, la empresa o universidad que la desarrolla, la página web donde hay información sobre la misma, un resumen de sus principales características, si considera el manejo de ontologías mixtas, eventos inesperados, planeación y definición separada de agentes e interacciones.

PLATAFORMAS COMERCIALES

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
AgenTalk *NTT, Ishida Lab y Univ. Kioto Japón*	http://www.kecl.ntt.co.jp/csl/msrg/topics/at/ Es un lenguaje para descripción de protocolos de sistemas multiagentes, por ejemplo, el protocolo Contract Net. Escrito en Common Lisp. Producto disponible sólo en Japón.	no	no	no	no
AgentBuilder *Reticular Systems, Inc. USD$999*	http://www.agentbuilder.com/ Es un conjunto de herramientas para construir agentes inteligentes de software. Consiste de dos componentes principales el Toolkit y el sistema de Run Time. Permite definir agentes en Java que se comunican en KQML. Permite construir agentes inteligentes y programas basados en Agentes. Soporta además CORBA y TCP/IP.	no	no	no	no
Agentx *International Knowledge Systems*	http://www.mobileagenttech.com/ Es una plataforma que aprovecha la máquina virtual de Java para ejecutar Agentes móviles que eficientemente pueden ejecutarse en un máquina, concurrentemente o migrar de un lugar a otro. Compatible 100% con JDK de Sun, además cuenta con funcionalidad de ORB. Permite que programas complejos se escriban como procesos pequeños que pueden distribuirse.	no	no	no	no
Aglets *IBM Japón*	http://www.trl.ibm.com/aglets/ Un aglet es un objeto Java que puede moverse de una computadora a otra en Internet llevándose sus datos y su estado de ejecución. Cuenta con un mecanismo de seguridad para proteger a los servidores de aglets intrusos.	no	no	no	no
Concordia *Mitsubishi Electric, Information Technology Center America*	http://www.merl.com/HSL/Projects/Concordia/ Es una plataforma de desarrollo de aplicaciones basadas en agentes móviles para accesar información anywhere, anytime y en dispositivos que soportan Java. Los usuarios pueden estar desconectados de la red durante algún tiempo. Los agentes pueden operar en LAN, Intranet o Internet. Puede usar comunicación wired o wireless. Soporta diversos dispositivos como computadoras de escritorio, PDAs, portátiles y teléfonos inteligentes.	no	no	no	no

[16] http://www.agentbuilder.com/AgentTools

PLATAFORMAS COMERCIALES

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
DirectIA SDK *MASA Group – Adaptive Objects*	http://www.animaths.com/ Es una plataforma que permite crear agentes que se adaptan a un ambiente dinámico para descubrir nuevos comportamientos y verificar su eficiencia en tiempo real. Se proporciona como bibliotecas en C++ para Windows 98 y NT. Comprende una aplicación de prueba, un tutorial, y herramientas para depurar y afinar los agentes.	no	no	no	no
Gossip *Tryllian*	http://www.tryllian.com/ Es una plataforma de agentes móviles que busca información en Internet para un usuario. Una vez que el agente conoce los intereses entra a un Marketplace en un servidor e intercambia información con otros agentes mediante una conversación. Los agentes pueden manejar hardware y software.	no	no	no	no
Grasshopper *IKV++*	http://www.grasshopper.de/index.html Es la primera plataforma de agentes móviles que cumple con el estándar MASIF construido sobre CORBA para proporcionar la integración de los sistemas cliente/servidor tradicionales con los agentes. Los agentes pueden aprovechar las líneas de alta velocidad de una red local y el acceso local a datos.	no	no	no	no
IGEN *CHI Systems*	http://www.cognitiveagent.com/ Es un toolkit y un ambiente de desarrollo para construir aplicaciones con agentes inteligentes.	no	no	no	no
Intelligent Agent *Factory Bits & Pixels USD$289 (incluyendo código fuente USD$479)*	http://www.bitpix.com/ Permite ahorrar tiempo en la construcción de soluciones basadas en agentes inteligentes. Los agentes se controlan mediante reglas escritas en CLIPS y se generan mediante especificaciones simples de workflows.	no	no	no	no
Intelligent Agent Library *Bits & Pixels USD$459 (incluyendo código fuente USD$1495)*	http://www.bitpix.com/ Es una biblioteca que proporciona un ambiente de trabajo que incluye facilidades para que los agentes se comuniquen y construir grandes conglomerados de agentes. Hay una plataforma basada en KQML que ilustra agentes que realizan actividades en la web. También soporta agentes móviles.	no	no	no	no
JACK *Intelligent Agents Agent Oriented Software Pty, Ltd*	http://www.agent-software.com.au/ Es un sistema de agentes que proporciona una arquitectura y capacidades para desarrollar agentes de software en aplicaciones distribuidas. Escrito en Java, usa una extensión de Java (JACK Agent Language) para proporcionar los beneficios de Java.	no	no	no	no
JAM *Intelligent Reasoning Systems*	http://members-http-2.rwc1.sfba.home.net/marcush/IRS/ Es una arquitectura basada en el modelo Belief-Desire-Intention sobre el Procedural Reasoning System (PRS) de Georgeoff, Rao, Lansky, et. al. JAM se puede aplicar a diferentes dominios. Soporta razonamiento top-down, goal-based y bottom-up data driven.	no	no	no	no
Jumping Beans *Ad Astra Engineering, Inc.*	http://www.jumpingbeans.com/ Es una plataforma que permite a los desarrolladores crear agentes móviles basados en Java. Aún en sitios donde la aplicación aún no estaba instalada.	no	no	no	no
LPA Agent Toolkit *Logic Programming Associates, Ltd.*	http://www.lpa.co.uk/atk.htm Es un toolkit que permite escribir programas orientados a agentes usando un enfoque basado en lógica. Este toolkit le proporciona al usuario un esqueleto de un agente el cual exhibe las principales características: autonomía, persistencia, cooperación y adaptabilidad. Los agentes se escriben en Prolog.	no	no	no	no

PLATAFORMAS COMERCIALES

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
MadKit *Madkit Development Group*	http://www.madkit.org/ MadKit Es una plataforma multiagente basada en Java construido sobre un modelo organizacional. Provee las facilidades generales de agente (gestión de su ciclo de vida, paso de mensajes, distribución, entre otras), y permite alta heterogeneidad en arquitectura se agentes y lenguajes de comunicación además de otras personalizaciones.	no	no	no	no
Microsoft Agent *Microsoft Corp.*	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msagent/agentstartpage_7gdh.asp El agente de Microsoft versión 2.0 proporciona una tecnología para crear una interfaz conversacionales para aplicaciones y páginas web. Proporciona capacidades de animación, interactividad y versatilidad con facilidades para su desarrollo.	no	no	no	no
Network Query Language *NQL Solutions*	http://www.nqli.com/ NQL Es un lenguaje de script para construir agentes y aplicaciones en la web. La comunicación es una parte importante de NQL. El acceso a Internet con protocolos comunes como http, ftp, nntp, y telnet estan interconstruidos en el lenguaje. Conexiones TCP/IP personalizadas pueden construirse usando sockets en NQL. Accesa e-mail, bases de datos y aplicaciones de escritorio.	no	no	no	no
Open Cybele *Intelligent Automation Inc.*	http://www.opencybele.org/view.asp?type=view&subID=16&topicID=1 Es un ambiente de ejecución construido sobre la plataforma de Java para el control de la ejecución de agentes. Los agentes y el manejo de eventos lo hace OpenCybele en vez de la MV de Java.	no	no	no	no
Pathwalker *Fujitsu*	http://www.labs.fujitsu.com/free/paw/ Pathwalker es una biblioteca de agentes orientada a procesos distribuidos escrita completamente en Java. Tiene algunas características que se usan para programación distribuida como mensajes asíncronos, identificador global único y procesos ligeros, manejo de código movil.	no	no	no	no
Swarm *Swarm Development Group*	http://www.swarm.org/ Es un paquete de software para simulación de sistemas multiagente. Se propone como una herramienta para investigadores de varias diciplinas, especialmente en vida artificial. La plataforma básica para este sistema es Solaris sobre Sparc Sun, GNU/Linux y Windows NT.	no	no	no	no
Topia Personal Agents *Topia Ventures*	http://www.topiaventures.com/index.asp Realizan tareas en nombre del usuario, como por ejemplo busqueda, adquisición, filtrado y distribución de información, negociación para servicios y productos, colaborando con otros agentes para realizar las tareas y monitorear las actividades de los agentes.	no	no	no	no
UMPRS *Intelligent Reasoning Systems*	http://members-http-1.rwc1.sfba.home.net/marcush/IRS/ Es una arquitectura basada en el modelo Belief-Desire-Intention (BDI) sobre el Procedural Reasoning System (PRS). UMPRS se aplica a diferentes dominios, está desarrollado en C++.	no	no	no	no
Voyager *Object Space*	http://www.objectspace.com/ Es un ORB que combina las capacidades de los agentes autonomos móviles y la invocación de métodos remotos con soporte de CORBA y tiene servicios de directorio, persistencia y mensajes multicast.	no	no	no	no

PLATAFORMAS ACADÉMICAS Y DE INVESTIGACIÓN

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
Agent Builder Shell (ABS) *Enterprise Integration Laboratory, University of Toronto*	http://www.kecl.ntt.co.jp/csl/msrg/topics/at/ Es un proyecto en donde se desarrolla un Kernel que proporciona capas reutilizables de lenguajes y servicios para construir sistemas de agentes: lenguajes de comunicación y coordinación, manejo de conocimiento basado en lógica, distribución de información cooperativa, administración de conflictos y modelos de organizaciones. El enfoque está siendo usado para desarrollar aplicaciones multiagente en el área de manufactura.	no	no	no	no
Agent TCL *Dartmouth University*	http://agent.cs.dartmouth.edu/general/agenttcl.html Agent TCL es una herramienta para desarrollar sistemas de agentes móviles. Los agentes se crean utilizando el lenguaje de comandos utiles (Tool Command Language, TCL). TCL es un lenguaje basado en scripts disponible libremente. Los agentes migran de un lugar a otro suspendiendo su ejecución y continuandola sín cambios en otro equipo.	no	no	no	no
Agent Tool *Kansas State University*	http://www.cis.ksu.edu/~sdeloach/ai/ Permite especificar formalmente la estructura y comportamiento de un sistema multiagente. Usa un mecanismo gráfico y una metodología de ingeniería de sistemas multiagente.	no	no	no	no
Architecture type-based Development Environment (ADE) *University of Postdam*	http://samuel.cs.uni-potsdam.de/soft/taxt/research/ade/ade.html Es una plataforma que modela explícitamente las interacciones entre agentes, a diferencia de otras plataformas, permitiendo mapear los sistemas y modelos en plataformas de agentes. Mediante ADE se pueden crear aplicaciones para varias plataformas.	no	no	no	no
Bee-gent Toshiba Corp., Systems and *Software Research Laboratories*	http://www2.toshiba.co.jp/beegent/index.htm Agentifica completamente la comunicación que se da entre aplicaciones de software. La plataforma consiste de dos tipos de agentes: envolventes, que se usan para agentificar las aplicaciones existentes y los mediadores, que soportan la cordinación interna de las aplicaciones y manejan todas las comunicaciones.	no	no	no	no
Bond Distributed Object System *Purdue University*	http://bond.cs.purdue.edu/ Proporciona un ambiente orientado al intercambio de mensajes desarrollado para realizar aplicaciones distribuidas.	no	no	no	no
Cable *Logica Corporation*	http://public.logica.com/~grace/Architecture/Cable/public/ Es una arquitectura genérica para desarrollar y ejecutar aplicaciones distribuidas que estan basadas en la metáfora de multiples agentes inteligentes cooperativos. Proporciona un lenguaje de definición de agentes (Agent Definition Language, ADL) y un traductor llamado Ascribe.	no	no	no	no
D'Agents *Dartmouth University*	http://agent.cs.dartmouth.edu/ El objetivo de este proyecto es porporcionar soporte a las aplicaciones que requieren de la recuperación, organización y presentación de información distribuida en redes arbitrarias.	no	no	no	no
DECAF Agent Framework *University of Delaware*	http://www.eecis.udel.edu/~decaf/ Proporciona una plataforma para desarrollar agentes, la arquitectura básica está desarrollada en Java por lo que permite aprovechar sus características	no	no	no	no

PLATAFORMAS ACADÉMICAS Y DE INVESTIGACIÓN

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
EXCALIBUR *GMD First, Technical University of Berlin*	http://www.ai-center.com/projects/excalibur/publications.html Es una arquitectura genérica para agentes autónomos utilizada en ambientes de videojuegos. Los agentes son capaces de encontrar las acciones correctas en base a los objetivos que se tienen y adaptan su comportamiento para nuevos oponentes o ambientes.	no	no	SI	no
FIPA-OS *FIPA*	http://fipa-os.sourceforge.net/ Es una plataforma basada en componentes que permiten desarrollar agentes que cumplen con la definición de FIPA.	no	no	no	no
Gypsy *Technical University of Viena*	http://www.infosys.tuwien.ac.at/Gypsy/ Es un proyecto que utiliza Java para la implementación de un ambiente flexible para experimentar con agentes móviles. Se pretende que sea útil para aplicaciones en Internet como recuperación de información, comercio y computación móvil.	no	no	no	no
Hive *The Media Lab. MIT*	http://www.hivecell.net/ Es una plataforma de software para crear aplicaciones distribuidas. Los programadores pueden crear sistemas que se conectan y usan los datos de cualquier parte de Internet. Es un ambiente para que puedan vivir agentes distribuidos y que se comunican para lograr los objetivos.	no	no	no	no
Infospiders *University of California San Diego Computer Science Department*	http://dollar.biz.uiowa.edu/~fil/IS/ Es una plataforma basada en el modelo de vida artificial para la recuperación de información en grandes bases de datos dinámicas, distribuidas y heterogéneas. La energía para sobrevivir cada uno de los agentes la toma del ambiente y del usuario en los intercambios de información relevante.	no	no	no	no
JADE *CSELT S.p.A., University of Parma*	http://sharon.cselt.it/projects/jade/ Es una plataforma para desarrollar aplicaciones basadas en agentes que cumplen con los estándares de FIPA. JADE se basa en un agente genérico que el usuario puede extender para dotarlo de la funcionalidad esperada.	no	no	no	no
JAFMAS *University of Cincinnati*	http://www.ececs.uc.edu/~abaker/JAFMAS/ Es una plataforma para guiar el desarrollo de sistemas multiagente junto con un conjunto de clases para el desarrollo de agentes en Java. Soporta comunicación a uno o múltiples agentes usando KQML.	no	no	no	no
JATLite *Stanford University*	http://java.stanford.edu/java_agent/html/ Es un conjunto de paquetes de Java que permiten construir sistemas multiagente usando Java.	no	no	no	no
JATLiteBean *University of Otago*	http://kmi.open.ac.uk/people/emanuela/JATLiteBean/ Es una extensión a JavaBean y JATLite adicionando la funcionalidad de KQML.	no	no	no	no
JIAC *Technische Universitat Berlin*	http://dai.cs.tu-berlin.de/english/forschung/projekte/JIAC/ Es una arquitectura de agentes abierta, escalable implementada en Java, ofrece agentes móviles basados en componentes, así como soporte para crear aplicaciones de comercio electrónico y servicios distribuidos de telecomunicaciones.	no	no	no	no
KLAIM *Universita'Di Frenze*	http://music.dsi.unifi.it/ Es una plataforma para soportar el paradigma de procesos junto con los datos pueden moverse de un equipo de computo a otro.	no	no	no	no

PLATAFORMAS ACADÉMICAS Y DE INVESTIGACIÓN

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
Knowbot System *Software CNRI*	http://www.cnri.reston.va.us/home/koe/ Es una infraestructura de investigación para agentes móviles, cuyo uso se pretende sea en sistemas altamente distribuidos tales como Internet. Está escrito en Python el cual es un lenguaje de programación orientado a objetos interpretado.	no	no	no	no
Mobiware Middleware Toolkit *Columbia University*	http://comet.columbia.edu/mobiware/ Es una tecnología de objetos distribuidos construida mediante Java y CORBA. Se ejecuta en dispositivos móviles, puntos de acceso inalámbricos y ruteadores con capacidad móvil proporcionando un conjunto de algoritmos e interfaces programables abierto para redes móviles adaptativas.	no	no	no	no
MOLE *University of Stuttgart*	http://mole.informatik.uni-stuttgart.de/ Es un prototipo que adiciona mecanismos de migración y comunicación usando el lenguaje de programación Java.	no	no	no	no
Multi-Agent Modeling Language (MAML) *Central European University*	http://www.maml.hu/ Es un lenguaje de programación para construir modelos basados en agentes. Es parte de investigaciones sobre modelación asistida por computadora. La disciplina de modelación soportada es basada en agentes.	no	no	no	no
MultiAgent System Tool (MAST) *Technical University of Madrid*	http://www.gsi.dit.upm.es/~mast/ Es una plataforma distribuida de propósito general para la cooperación de agentes heterogéneos. Consiste de dos entidades básicas: los agentes y la red a través de la cual interactuan ellos.	no	no	no	no
NOMADS *Institute for Human and Machine Cognition University of West Florida*	http://nomads.coginst.uwf.edu/ Es un sistema de agentes moviles que soporta la movilidad fuertemente y la ejecución segura en Java. Se compone de dos partes: un ambiente de ejecución llamado Oasis y una nueva Máquina Virtual compatible con Java llamada Aroma.	no	no	no	no
Open Agent Architecture *SRI International*	http://www.ai.sri.com/~oaa/ Es una plataforma para integrar una comunidad de agentes de software heterogeneos en un ambiente distribuido. En este contexto un agente se define como un proceso que registra sus servicios en una forma aceptable, utiliza el Lenguaje de Comunicación Interagente (ICL) y comparte funcionalidades comunes a todos los agentes.	no	no	no	no
ProcessLink *Stanford University*	http://www-cdr.stanford.edu/ProcessLink/ Es una plataforma que consiste de agentes genéricos y un protocolo de mensajes para integrar proyectos distribuidos y multidisciplinarios. Permite envolver software legado con código que le habilita para proporcionarle funciones de coordinación.	no	no	no	no
RETSINA *Carnagie Mellon University*	http://www-2.cs.cmu.edu/~softagents/ Tiene cuatro módulos para comunicación, planeación, calendarización y monitoreo de la ejecución de tareas y requerimientos de otros agentes.	no	no	no	no
Sensible Agents *The University of Texas at Austin, Laboratory for Intelligent Processes and Systems*	http://www.lips.utexas.edu/agents_one_pager.htm Representan una ayuda nueva para planeación, monitoreo y decisión. Los agentes en este ambiente son capaces de evaluar la propagación de las decisiones locales. Priorizan planes dinámicamente. Proporcionan sugerencias para las interacciones más eficientes y productivas con otros agentes. Pueden replanificar cuando ocurren cambios en el ambiente.	no	no	SI	no

PLATAFORMAS ACADÉMICAS Y DE INVESTIGACIÓN

OM: Soporta Ontologías Mixtas
EI: Soporta Eventos Inesperados
P: Planifica automáticamente
SAI: Implementa Agentes separados de las Interacciones (escenarios)

PROYECTO	DESCRIPCIÓN	OM	EI	P	SAI
Social Interaction Framework (SIF) *DFKI (German Research Institute forAI)*	http://www.lhbasif.com/agentworkshop/index.asp Es una herramienta para generar prototipos de simulaciones de en donde se involucran múltiples agentes.	no	no	no	no
Sodabot *MIT Artificial Intelligence Lab*	http://www.ai.mit.edu/people/sodabot/sodabot.html Es un ambiente de propósito general para desarrollo y ejecución de agentes de software. Este proyecto desarrolla un lenguaje de programación para programar a los agentes mediante primitivas que describen la actividad del agente.	no	no	no	no
SOMA (Secure and Open Mobile Agent) *University of Bologna*	http://lia.deis.unibo.it/Research/SOMA/ Es una plataforma basada en Java utilizada en Internet, diseñada para los objetivos de seguridad e interoperabilidad. Es compatible con CORBA y MASIF.	no	no	no	no
TeamBots *The Robotics Institute, Carnegie Mellon University*	http://www.teambots.org/ Es una colección de programas de aplicación basada en Java para investigar robots móviles multiagente. El ambiente de simulación está escrito en Java.	no	no	no	no
TuCSoN *Universita di Bologna*	http://www.lia.deis.unibo.it/Research/TuCSoN/ Es un modelo para la coordinación de agentes de Internet.	no	no	no	no
Zeus *British Telecommunications Labs*	http://193.113.209.147/projects/agents.htm Es un ambiente colaborativo y una biblioteca de componentes para la construcción de agentes escritos en Java. Consta de tres capas. La capa de definición representa las habilidades del agente, como razonamiento, aprendizaje, sus metas, recursos, habilidades, creencias, preferencias, entre otros. La capa de organización describe las relaciones de un agente con otros. La capa de coordinación describe la coordinación y técnicas de negociación del agente, en este se encuentra el módulo de comunicación.	no	SI	SI	no

ANEXO B GRAMÁTICA DE LIA

En este anexo se indica el archivo con la gramática de LIA como se utiliza en el traductor. Las palabras en mayúsculas son símbolos no terminales, las escritas con minúsculas son los símbolos terminales, los números indican rutinas semánticas y los símbolos que inician con el símbolo # son los indicadores de error en la sintaxis.

```
;           Instituto Politécnico Nacional
;      Centro de Investigación en Computación
;
; GRAMATICA DEL LENGUAJE DE INTERACCIONES
ENTRE AGENTES
;                6 diciembre 2000
;
; Es una Gramática Top-Down cuyos componentes
son:
;     . simbolos no terminales (MAYÚSCULAS)
;     . terminales (minúsculas)
;     . número (rutinas semánticas)
;     . #número (errores)
;=============================================
========================
; SECCION INICIAL
;=============================================
========================
INICIO -> GLOBAL REGIONAL AGENTES  MAIN
INTERACCIONES
;
;=============================================
========================
; VARIABLES GLOBALES
;=============================================
========================
GLOBAL -> global 1 s{ LISTA_VARSG X1
X1 -> s}
X1 -> #1
GLOBAL -> global s{ #2
GLOBAL -> global #3
GLOBAL -> #4
;
LISTA_VARSG -> VARSG LISTA_VARSG
LISTA_VARSG -> VARSG
;
VARSG -> TIPO LISTAVG
TIPO -> int 2
TIPO -> char 2
TIPO -> float 2
TIPO -> port 2
;
LISTAVG -> VARG s, LISTAVG
LISTAVG -> VARG s;
LISTAVG -> #5
;
VARG -> id  s[ numero  s]  s[ numero   s] 3
VARG -> id  s[ numero  s] 4
VARG -> id  5
VARG -> #6
;
;=============================================
========================
; VARIABLES REGIONALES
;=============================================
========================
REGIONAL -> regional 1 s{ LISTA_VARSR X2
X2 -> s}
X2 -> #8
REGIONAL -> regional  1 s{ #9
REGIONAL -> regional  #10
REGIONAL -> #11
;
LISTA_VARSR -> VARSR LISTA_VARSR
LISTA_VARSR -> VARSR
;
VARSR -> TIPO LISTAVR
;
LISTAVR -> VARR s, LISTAVR
LISTAVR -> VARR s;
LISTAVR -> #13
;
VARR -> id  s[ numero  s]  s[ numero   s] 3
VARR -> id  s[ numero  s] 4
VARR -> id  5
VARR -> #12
;
;=============================================
========================
; A G E N T E S
;=============================================
========================
AGENTES -> AGENTE AGENTES
AGENTES -> AGENTE
;
;----------------------------------
; Agente
;----------------------------------
AGENTE -> agent 1 id 6 s{ LREGIONAL INTERNAS
PROPOSITOS PERCIBE ACCIONES INICIALES s}
AGENTE -> agent id s{ LREGIONAL INTERNAS
PROPOSITOS #13
AGENTE -> agent id s{ LREGIONAL #14
AGENTE -> agent id s{ #15
AGENTE -> agent id #16
AGENTE -> agent #17
;
;----------------------------------
; Variables regionales del agente
;----------------------------------
LREGIONAL -> regional 1 s{ LISTAREG s}
LISTAREG -> id 6 s, LISTAREG
LISTAREG -> id 6 s;
;
;----------------------------------
; Variables internas
;----------------------------------
INTERNAS -> internal 1 s{ LISTA_VARSI X4
X4 -> s}
X4 -> #19
INTERNAS -> internal s{ #20
INTERNAS -> internal #21
INTERNAS -> #22
;
LISTA_VARSI -> VARSI LISTA_VARSI
LISTA_VARSI -> VARSI
;
VARSI -> TIPOI LISTAVI
TIPOI -> int 2
TIPOI -> char 2
TIPOI -> float 2
;
LISTAVI -> VARI s, LISTAVI
LISTAVI -> VARI s;
```

```
LISTAVI -> #23
;
VARI -> id  s[ numero  s]  s[ numero   s] 3
VARI -> id  s[ numero  s] 4
VARI -> id  5
VARI -> #24
;
;-----------------------------------
; Propósitos
;-----------------------------------
PROPOSITOS -> purpose 1 s{ LISTAPROP s}
LISTAPROP -> id 6 s, LISTAPROP
LISTAPROP -> id 6 s;
;
;-----------------------------------
; Eventos Inesperados percibidos
;-----------------------------------
PERCIBE -> percieve 1 s{ LISTAPERCI s}
LISTAPERCI -> id 6 s, LISTAPERCI
LISTAPERCI -> id 6 s;
;
;-----------------------------------
; Acciones para Eventos Inesperados
;-----------------------------------
ACCIONES -> action 1 s{ LISTAACCION s}
LISTAACCION -> id 6 s,  LISTAACCION
LISTAACCION -> id 6 s;
;
;-----------------------------------
; Papeles iniciales
;-----------------------------------
INICIALES -> initial 1 s{ LISTAINI s}
LISTAINI -> id 6 s,  LISTAINI
LISTAINI -> id 6 s;
;
;=========================================
========================
; INTERACCIONES
;=========================================
========================
INTERACCIONES -> INTERACCION INTERACCIONES
INTERACCIONES -> INTERACCION
;
;-----------------------------------
; Interacción
;-----------------------------------
INTERACCION -> interaction 1 id 6 s{ PAPELES
s}
;
PAPELES -> PAPEL PAPELES
PAPELES -> PAPEL
;
;-----------------------------------
; Papel
;-----------------------------------
PAPEL -> role 1 id 6 s( PARAMS s[ instances
numero 10 s] s{ REQ POSITIVO NEGATIVO LOCALES
8 L_INSTR s} 9
;
;-----------------------------------
; Parametros
;-----------------------------------
PARAMS -> id 60 s, PARAMS
PARAMS -> id 60 s)
;
;-----------------------------------
; Requisitos
;-----------------------------------
REQ -> requisite 1 s{ LISTA_REQ s}
LISTA_REQ -> id 6 s, LISTA_REQ
LISTA_REQ -> id 6 s;
;
;-----------------------------------
; Efectos Positivos
;-----------------------------------
POSITIVO -> positive 1 s{ LISTA_POS s}
LISTA_POS -> id 6 s, LISTA_POS
LISTA_POS -> id 6 s;
;
;-----------------------------------
; Efectos Negativos
;-----------------------------------
NEGATIVO -> negative 1 s{ LISTA_NEG s}
LISTA_NEG -> id 6 s, LISTA_NEG
LISTA_NEG -> id 6 s;
;
;-----------------------------------
; Variables Locales
;-----------------------------------
LOCALES -> local 1 s{ LISTALOC X5
X5 -> s}
X5 -> #30
LOCALES -> local s{ #31
LOCALES -> local #32
LOCALES -> #33
;
LISTALOC -> VARSLOC LISTALOC
LISTALOC -> VARSLOC
;
VARSLOC -> TIPOI LISTALO
;
LISTALO -> VARLO s, LISTALO
LISTALO -> VARLO s;
;
VARLO -> id  s[ numero  s]  s[ numero   s] 3
VARLO -> id  s[ numero  s] 4
VARLO -> id  5
VARLO -> #35
;
L_INSTR -> INSTR L_INSTR
L_INSTR -> INSTR
;
;-----------------------------------
; INSTRUCCIONES DE LIA
;-----------------------------------
INSTR -> print s( cadena 11 s) s;
INSTR -> print s( id 49 s) s;
INSTR -> if s( EXPR_L s) s{ L_INSTR s} else s{
L_INSTR s}
INSTR -> if s( EXPR_L s) 12 s{ L_INSTR s} 13
INSTR -> while 14 s( 15 16 EXPR_L s) 17 s{
L_INSTR s} 18
;INSTR -> optional s[ EXPR_ARITM s] s{ L_INSTR
s}
INSTR -> addagent s( id  s, id  s, 19 50
LISTA_PROP_AG LISTA_REC_AG s; 59
INSTR -> addinteract s( id  s, id  s) s; 20
INSTR -> addpurpose s( id  s, id  s) s; 21
INSTR -> delpurpose s( id  s, id  s) s; 22
INSTR -> out s( id s, id s, id  52 s, id s, id
53 s) s;
INSTR -> accept s( id s, id s, id 54 s, id 55
s) s;
INSTR -> append s( id s, cadena 46 s) s;
INSTR -> append s( id s, id 47 s) s;
INSTR -> extract s( id s, id s, id 56 s) s;
INSTR -> compare s( id s, id s, id 57 s) s;
INSTR -> delmessage s( id s, id 58 s) s;
INSTR -> wait s( numero s) s; 45
INSTR -> goal s( id 61 s) s;
;INSTR -> addevent agente evento
;INSTR -> delevent agente evento
```

```
;INSTR -> addaction agente action
;INSTR -> delaction agente action
;INSTR -> start papel tiempo prioridad
;INSTR -> kill papel tiempo prioridad
INSTR -> VAR_A_NUM s= 15 EXPR_ARITM s; 23
INSTR -> id s:= cadena s; 48
INSTR -> id s[ numero s] s:= cadena s;
INSTR -> id s:= id s[ id s] s;
;
;-----------------------------------
; Propósitos Nuevos de un Agente
;-----------------------------------
LISTA_PROP_AG -> id 6 s; LISTA_PROP_AG
LISTA_PROP_AG -> id 6 s,
;
;-----------------------------------
; Recursos Iniciales de un Agente
;-----------------------------------
LISTA_REC_AG -> ACCION_AG s; LISTA_REC_AG
LISTA_REC_AG -> ACCION_AG s)
;
ACCION_AG -> id s:= cadena 51
ACCION_AG -> VAR_A_NUM s= 15 EXPR_ARITM 23a
;
;-----------------------------------
; Expresion Lógica
;-----------------------------------
EXPR_L -> TERMINO_L s|| EXPR_L 32
EXPR_L -> TERMINO_L
TERMINO_L -> FACTOR_L s&& TERMINO_L 33
TERMINO_L -> FACTOR_L
FACTOR_L -> EXPR_ARITM OP_REL EXPR_ARITM 34
FACTOR_L -> s! FACTOR_L
FACTOR_L -> s( EXPR_L s)
;
OP_REL -> s>   35
OP_REL -> s<   36
OP_REL -> s==  37
OP_REL -> s>=  38
OP_REL -> s<=  39
OP_REL -> s!=  40
;
;-----------------------------------
; Expresión Aritmética
;-----------------------------------
EXPR_ARITM -> TERMINO s+ EXPR_ARITM  25
EXPR_ARITM -> TERMINO s- EXPR_ARITM  26
EXPR_ARITM -> TERMINO
TERMINO -> FACTOR s* TERMINO  27
TERMINO -> FACTOR s/ TERMINO  28
TERMINO -> FACTOR
FACTOR -> s( EXPR_ARITM s)
FACTOR -> numero  24
FACTOR -> VAR_NUM
;
VAR_NUM -> id s[ id s] s[ id s]  31
VAR_NUM -> id s[ id s]  30
VAR_NUM -> id  29
;
;-----------------------------------
; Otra cosa
;-----------------------------------
VAR_A_NUM -> id s[ id s] s[ id s]  41
VAR_A_NUM -> id s[ id s]  42
VAR_A_NUM -> id s[ numero s]  43
VAR_A_NUM -> id  44
;
VAR_PUERTO -> id s[ numero s] s[ numero s]
VAR_PUERTO -> id s[ id s] s[ id s]
VAR_PUERTO -> id s[ numero s]
VAR_PUERTO -> id s[ id s]
VAR_PUERTO -> id
;
EXPR_CADENA -> cadena
;
VALUE -> id
VALUE -> numero
;
TIEMPO -> id   65--
;
EMISOR -> self  60--
EMISOR -> id    59--
;
PUERTO -> VAR_PUERTO
;=========================================
=========================
; PAPEL PRINCIPAL
;=========================================
=========================
MAIN -> main s( s) 7 8 s{ L_INSTR s} 9
; fin gramatica
```

REFERENCIAS

Adelson 1992 — Beth Adelson, Evocative Agents and Multimedia Interface Design en *Conference Proceedings on Human Factors in Computing Systems*, 1992, pp 351-356

Aldunate 2000 — Felipe Aldunate Montes, Prueba Navideña en *América Economía*, 28 diciembre 2000

Altinkemer 2001 — Kemal Altinkemer, Bundling e-banking Services en *Communications of the ACM*, Junio 2001, vol 44, no. 6, pp 45 - 47

Alvarado 2002 — Matias Alvarado, Leonid Sheremetov, Erick Alva, Ernesto Germán, Towards a Temporal Modal Logic for Concurrent Interacting Actions, Center for Computing Research of the IPN, México, 2002

Alvarado 2002a — Matias Alvarado, Leonid Sheremetov, Ernesto Germán, Erik Alva, Logic of Interaction for Multiagent Systems en *Proc. MICAI 2002*, LNAI 2313, C.A. Coello Coello, et. al. (editores), Springer-Verlag, pp387-400, 2002

Amandi 1997 — Analia Amandi, Ana Price, Towards Object-Oriented Agent Programming: The Brainstorming Meta-Level Architecture en *Proceedings of Autonomous Agents 97*, Marina del Rey, CA, USA

Amor 2000 — Daniel Amor, *La (R)evolución e-business*, editorial Prentice Hall, Buenos Aires Argentina, 200O, ISBN 987-97892-5-3

Amori 1992 — Richard D. Amori, An adversarial plan recognition system for multi-agent airborne threats en *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing (vol. I): Technological Challenges of the 1990's*, 1992, pp 497-504

Anderson 1998 — Corin R. Anderson, David E. Smith, Daniel S. Weld, Conditional Effects in Grpahplan, en *Proc. Fourth International Conference on AI Planning Systems AIPS-98*, Pittsburgh, PA, pp. 44-53. 1998

Araya 1989 — Agustin Araya, Handling Contingencies in Planned Office Activities en *Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1989

Ayala 1998 — Gerardo Ayala y Yoneo Yano, A collaborative learning environment based on intelligent agents en *Expert Systems with Applications*, vol. 14, 1998, pp 129-137

Bakre 1995 — Ajay Bakre and B. R. Badrinath, M-RPC: a Remote Procedure Call Service for Mobile Clients en *Proceedings of the First Annual International Conference on Mobile Computing and Networking*, 1995, pp 97-110

Bakre 1996 — Ajay V. Bakre and B. R. Badrinath, Reworking the RPC Paradigm for Mobile Clients en *Mobility Network and Applications* (Jan. 1996), pp 371-385

Barret 1997 — Rob Barret, Paul P. Maglio, Daniel C. Kellem, WBI: A Confederation of Agents that Personalize the Web, en *Proceedings of Autonomous Agents 97*, Marina del Rey, CA, USA

Bates 1994 — Joshep Bates, The Role of Emotion in Believable Agents, en *Communications of the ACM*, Julio 1994, Vol. 37, No. 7

Becker 2001 — Ted Becker, Rating the impact of new technologies on democracy, en *Communications of the ACM*, vol. 44, no. 1, enero 2001, p39-43

Bergel 1998 — Hall Bergel, The New Push for push technology en *Network*, vol. 2.3, 1998

Bergel 1997a — Hal Bergel, Watermarking Cyberspace en *Communications of the ACM* vol. 40, no. 11 (Nov. 1997), pp 19-24

Bergel 1997b — Hal Bergel, Cyberspace 2000 Dealing with Information Overload en *Communications of the ACM* vol. 40, no. 2 (Feb. 1997), pp 19-24

Blum 1995 — A. L. Blum, M. L. Furst, Fast Planning through Planning Graph Analysis en *Artificial Intelligence* 90 (1-2) 279-298

Bhimani 1996 — Anish Bhimani, Securing The Commercial Internet en *Communications of the ACM*, vol. 39, no. 6, junio 1996

Boden 1994 — Margaret A. Boden, Agents and Creativity en *Communications of the ACM*, Julio 1994, Vol. 37, No. 7

Borenstein 1996 — Nathaniel S. Borenstein, Perils and Pitfalls of Practical Cybercommerce en *Communications of the ACM* vol. 39, no. 6 (Jun. 1996), pp 36 – 44

Cassell 1994 — Justine Cassell, Catherine Pelacahud, Norman Badler, Mark Steedman, et. al., Animated Conversation: Rule-based Generation of Facial Expression, Gesture & Spoken Intonation for Multiple Conversational Agents en *Proceedings of the 21st Annual Conference on Computer Graphics*, 1994

Chaib-draa 1997 — B. Chaib-draa, Connection between Micro and Macro aspects of Agent Modelling en *Proceedings of Autonomous Agents 97*, Marina del Rey, CA, USA, 1997

Collins 1998 — John Collins, Scott Jamison, Bamshad Mobasher, Maria Gini. A Market Architecture for MultiAgent Contracting en *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, May 1998

Contreras 2001 — Miguel Contreras Montoya, Desarrollo de una Infraestructura Multiagentes para Organizaciones Virturales (tesis de maestría), Centro de Investigación en Computación-IPN, México, 2001

Covington 1997 — Michael A. Covington, On Designing a Language for Electronic Commerce en *International Journal of Electronic Commerce*-Summer 1997, vol. 1, no. 4

Chan 1998 — Chan C., and Swatman, P.M.C., (1998) EDI Implementation: A Broader Perspective, "Bled'98" en *Eleventh Bled International Conference on Electronic Commerce*, Bled, Slovenia, 90-108.

Chavez 1997 Anthony Chavez, Pattie Maes, Kasbah: An Agent Marketplace for Buying and Selling Goods, MIT Media Lab, Cambridge, MA, USA, 1997

Do 2000 Minh Binh Do and S. Khambhampati. Solving Planning-Graph by Compiling it into CSP en *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

Domínguez 2001 Ma. del Carmen Domínguez Ayala, *Manejo de Infinitos Eventos Inesperados en Interacciones entre Agentes* (Tesis de Maestría en Ciencias) en el Centro de Investigación en Computación, México 2001

Durfee 1997 E. H. Durfee, Daniel L. Kiskis, William P. Birmingham, The Agent Architecture of the University of Michigan Digital Library, University of Michigan, abril, 15, 1997

Erceau 1993 Jean Erceau, Jacque Ferber, La Inteligencia Artificial Distribuida, en *Mundo Científico* 116 May Volume 11 1993

Farquhar 1996 Adam Farquhar, Richard Fikes, and James Rice, The Ontolingua Server: a Tool for Collaborative Ontology Construction, en *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW '96)*, 1996

Finin 1995 Tim Finin, Yannis Labrou, James Mayfield, KQML as an Agent Communication Language, CS Electrical Engineering, Unversity of Mariland Baltimore County, Baltimore MD USA, September 1995

Fipa 2001a FIPA Ontology Service Specification XC00086C www.fipa.org (2001)

Fipa 2001b FIPA SL Content Language Specification XC00008F www.fipa.org (2001)

Fipa 2001c FIPA English Auction Interaction Protocol Specification XC00031E www.fipa.org (2001)

Fipa 2001d FIPA Dutch Auction Interaction Protocol Specification XC00032E www.fipa.org (2001)

Fisher 1994 Michael Fisher, A Survey of Concurrent METATEM- The Language and its Application en D.M. Gabbay and H. J. Ohlbach editores en *Proc. 1st International Conference on Temporal Logic, Lecture Notes on AI* 827, pp. 480-505, 1994

Flores 1999 Roberto A. Flores-Mendez, Towards a Standardization of Multi-Agent System Framework en *Crossroads* vol. 5, no. 4 (Jun. 1999), pp 18-24

Genesereth 1994 Michael R. Genesereth, Steven P. Ketchel , Software Agents en *Communications of the ACM*, Julio 1994, Vol. 37, No. 7

Giacomo 2000 Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque, ConGolog, A Concurrent Programming Language based on the Situational Calculus en *Artificial Intelligence* 121, 2000, pp. 109-169

Goh 1998 Khim-Yong Goh, Hock-Hai Teo and Kwok-Kee Wei, Electronic Markets and Intelligent Agents: an Experimental Study of the Economics of Electronic Commerce *en Proceedings of the international Conference on Information Systems*, 1998, pp 293 - 295

Goldin 1998 Dina Goldin and Peter Wegner, Mathematical Models of Interactive Computing, Draft on Observability and Empiricism, 1998a

González 2001 Óscar González, *XML*, editorial Anaya (Multimedia) Madrid, España, ISBN 84-415-1108-X (2001)

Gruber 1993 Thomas R. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing *en Formal Ontology in Conceptual Analysis and Knowledge Representation*, edited by Nicola Guarino and Roberto Poli, Kluwer Academic Publishers, Italy 1993

Grönlund 2001 Åke Grönlund, Democracy in an IT-framed society: introduction en *Communications of the ACM*, vol. 44, no. 1, enero 2001, p22-26

Guha 1994 R. V. Guha and Douglas B. Lenat Enabling Agents to Work Together en *Communications of the ACM* 37, 7 (Jul. 1994), Pages 126 - 142

Guzmán 1997 Adolfo Guzmán A., Hallando los Temas Principales en un Artículo en Español. (1997) *en Soluciones Avanzadas* Vol. 5, núm. 45, pág. 58 también en *Memorias del Simposium Internacional de Computación*, 36-51.Centro de Investigación en Computación. Instituto Politécnico Nacional, Noviembre 12-14. México, D.F.1999

Guzmán 1998 Adolfo Guzmán A., Finding the Main Themes in a Spanish Document *en Journal Expert Systems with Applications*, Vol. 14, No. 1/2, 139-148, Jan./Feb. 1998

Guzmán 1998 a Adolfo Guzmán and Gustavo Núñez. Virtual Learning Spaces in Distance Education; Tools for the EVA Project. (1998) en *Journal of Expert Systems with Applications*, 15, 34, 205-210. Oct. Elsevier.

Guzmán 2000 Adolfo Guzman, Jesús Olivares, Araceli D., Carmen D., *Interaction of Purposeful Agents that use Different Ontologies*, Informe Técnico No. 46, enero 2000, ISSN 970-18-4132-8

Guzmán 2000 a Adolfo Guzmán, Jesús Olivares, Araceli Demetrio and Carmen Domínguez, Interaction of purposeful agents that use different ontologies. (2000a) en *Lecture Notes in Artificial Intelligence* 1793, MICAI 2000: Advances in A. I. Osvaldo Cairo, Enrique Sucar, Francisco J. Cantu (eds). Springer Verlag. Pages 557-573

Guzmán 2002 Guzmán, A., Domínguez, C., and Olivares, J. Reacting to Unexpected Events and Communicating in spite of Mixed Ontologies,en Proceedings MICAI 2002, Mérida, Mexico, 2002

Huhns 1999 Michael N. Huhns and Larry M Stephens, Multiagent Systems and Societies of Agents *en Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Gerhard Weiss (Eds.), MIT Press, Cambridge, Massachusetts,

	USA, 1999
Huhns 2001	Michael N. Huhns and Larry M Stephens, Effective Management of Supply Chain Management (SCM) Standards using New Coordination Policy Models en (presentation to the MEL Management Council), *Computer Science and Engineering*, University of South Carolina, USA, March 2001
Iglesias 1998	C. A. Iglesias, M. Garijo, and J. C. Gonzalez. A survey of agent-oriented methodologies. In J. P. Muller, M. P. Singh, and A. S. Rao, editors, Intelligent Agents V, volume 1555 of *Lecture Notes in Artificial Intelligence*, pp 317-330. Springer, 1998
Imielinsky 1994	Tomasz Imielinski, B. R. Badrinath, Mobile Wireless Computing: Challenges in Data Management en *Communications of the ACM* vol. 37, no. 10 (Oct. 1994), pp 18 - 28
Inverno 1998	Mark D'Inverno y Michael Luck, Engineering AgentSpeak(L): A Formal Computational Model en Journal of Logic Computing, Vol. 8, No. 3, pp 1-27, 1998
Inverno 2000	Mark D'Inverno, Koen Hidrinks, Michael Luck, A Formal Architecture for the 3APL Agent Programming Language en *Proceedings of the First International Conference of B and Z Users*, Springer 2000
Kaindl 1999	Hermann Kaindl, John M. Carroll (guest editors) Symbolic Modelling in Practice *en Communications of the ACM*, vol. 42, no. 1, enero 1999 pp 28-30
Kambil 1997	Ajit Kambil, Doing Bussiness in the Wired World en *Computer IEEE*, vol. 30, no. 5, mayo 1997
Kashyap 1998	V Kashyap, A.Sheth, Semantic Heterogeneity in Global Information Systems *en Cooperative Information Systems*, ed. Academic Press (editores Michael P. Papazoglou y gunter Schalageter), India, 1998
King 1996	William Josseph King, Jun Ohya, The Representation of Agents: Antropomorphism, Agency and Intelligence en *Proceedings of the CHI '96*, Companion, Vancouver, BC, Canada
La Porta 1996	Thomas F. La Porta, Krishan K. Sabnani and Richard D. Gitlin, Challenges for nomadic computing mobility management and wireless communications en *Mobility Network Applications* vol. 1, no.1 (Aug. 1996), pp 3 - 16
Leiner 1997	Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Lawrence G. Roberts and Stephen S. Wolff; The Past and Future History of the Internet en *Communications of the ACM* 40, 2 (Feb. 1997), pp 102 - 108
Lenat 1990	Douglas B. Lenat, R. V. Guha, Karen Pittman, Dexter Pratt and Mary Shepherd Cyc: Toward Programs with Common Sense en *Communications of the ACM* 33, 9 (Sep. 1990), Pages 30 - 49
Lenat 1995a	Douglas B. Lenat, CYC: a Large-Scale Investment in Knowledge Infrastructure *en Communication of the ACM*, vol. 38, no. 11 (Nov. 1995), pp 33-38
Lenat 1995b	Doug Lenat, George Miller and Toshio Yokoi, CYC, WordNet, and EDR critiques and responses en *Communications of the ACM* vol. 38, no. 11 (Nov. 1995), pp 45-48
Leonid 1999	Multi-Stage Cooperation Algorithm and Tools for Agent-Based Planning and Scheduling in Virtual Learning Environment, Centro de Investigación en Computación, México, 1999
Lester 1997	James C. Lester, Brian A. Stone, Increasing Believability in Animated Pedagogical Agents *en Proceedings of Autonomous Agents 97*, Marina del Rey, CA, USA
Leyton 2000	Kevin Leyton-Brown, Yoav Shoham, Moshe Tennenholtz, Bidding clubs: Institutionalized Collusion in Auctions en *Proceedings of the 2nd ACM Conference on Electronic Commerce*, octubre 2000
Levesque 1997	Hector J. Levesque, Raymond Reiter, et. al. GOLOG: A Logic Programming Language for Dynamic Domains en *Journal of Logic Programming* 31, April-June 1997, pp. 59-83
Lieberman 1998	Henri Lieberman, Integrating User Interface Agents with Conventional Applications en *Proceedings of the IUI* 98, USA
Lomuscio 2000	A. R. Lomuscio, M. Wooldridge and N. R. Jennings, A Classification Scheme for Negotiation in Electronic Commerce en *Agent-Mediated Electronic Commerce: A European Perspective* (eds. F. Dignum and C. Sierra), Springer Verlag, 19-33, 2000
Maes 1994	Pattie Maes, Agents that Reduce Work and Information Overload, en *Communications of the ACM*, Julio 1994, Vol. 37, No. 7 pp
Maes 1995	Pattie Maes, Artificial Life Meets Entertainment: Lifelike Autonomous Agent *en Communications of the ACM*, November 1995, Vol. 38, No. 11
Maes 1996	Pattie Maes, Alan Wexelblat, Interface Agents en *CHI '96* April 13-16, 1996
Mass 1999	Yosi Mass, Amir Herzberg, VRCommerce - Electronic Commerce in Virtual Reality *en Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999
Mohen 2001	Joe Mohen and Julia Glidden, The Case for Internet Voting en *Communications of the ACM*, vol. 44, no. 1, enero 2001, p 72
Moon 1996	Youngme Moon, Clifford Nass, Adaptive Agents and Personality Change: Complementary versus Similarity as Forms of Adaption, en *Proceedings of CHI '96*, Companion, Vancouver, BC, Canada
Morris 2000	Joan Morris, Peter Ree, Pattie Maes, Sardine: Dynamic Seller Strategioes in an Auction Marketplace, en

Proceedings of the 2nd ACM Conference on Electronic Commerce, octubre 2000

Mylopoulos 1999 John Mylopoulos, Lawrence Chung, Erick Yu, From Object-Oriented to Goal-Oriented Requirements Analysis en en *Communications of the ACM*, vol. 42, no. 1, enero 1999 pp 31-37

Naur 1960 Peter Naur (editor), Revised Report on the Algorithmic Language ALGOL 60 en *Communications of the ACM*, Vol. 3 No.5, pp. 299-314, Mayo 1960

Nonell 1999 Pedro Nonell, Conferencias On-Line de Comercio Electrónico en *http://www.reingex.com*, agosto 1999

Noriega 1997 Pablo C. Noriega Blanco V., *Agent Mediated Auctions: The Fishmarket Metaphor* (Tesis doctoral) Memory to obtain his Ph.D., Universitat Autònoma de Barcelona, Bellaterra, December 12th, 1997

Noriega 2001 Sergio Noriega, Intercambio de e-documentos para transacciones de comercio electrónico (comunicación personal), Centro de Investigación en Computación, Ciudad de México, octubre 2001

Nwana 1996 Hyacinth S. Nwana, Software Agents: An Overview en Knowledge Engineering Review, vol. 11, no. 3, pp 1-40, sept. 1996

Nwana 1997 Nwana, H.S. & Wooldridge, M., Software Agent Technologies en Nwana, H.S. & Azarmi, N. (eds.) Software Agents and Soft Computing: Concepts and Applications en *Lecture Notes in Artificial Intelligence Series*, Berlin 1997, Springer-Verlag, pp 59-78

Nwana 1999 Nwana, H., Ndumu, D., Lee, L., and Collis, J., ZEUS: A Tool-kit for Building Distributed Multiagent Systems en *Applied Artifical Intelligence Journal*, 13(1):129-186. http://www.labs.bt.com/ projects/agents/zeus/index.htm, 1999

Olivares 1991 Olivares C. Jesús Manuel, *Sistema Evolutivo para Representación del Conocimiento* (tesis de licenciatura), IPN-UPIICSA, clasif. 7.152, Ciudad de México, abril 1991

Olivares 2000 Jesús M. Olivares C., et. al., *Un Modelo de Agentes en Espacios de Interacción y la Especificación de su Interprete*, (Informe Técnico) Serie Roja No. 77, ISBN 970-18-4222-7, CIC-IPN, México, marzo 2000

Ossher 2001 Harold Ossher y Peri Tarr, Using Multidimensional Separation of Concerns to (Re)shape evolving Software en *Communications of the ACM*, Vol 44, No. 10, Octubre 2001, pp 43-50

Paniti 2001 Paniti Netinant, et. al. A Layered Approach to Building Open Aspect-Oriented Systems en *Communications of the ACM*, Vol 44, No. 10, Octubre 2001, pp 83-85

Panurach 1996 Patiwat Panurach, Money in Electronic Commerce: Digital Cash, Electronic Fund Transfer, and Ecash en Communications of the ACM, vol. 39, no. 6, 1996, pp 45-50

Parkes 1999 Parkes, D. C. Optimal auction design for agents with hard valuation problems en *Proc. 2nd Workshop on Agent Mediated Electronic Commerce (AmEC-99)* (July 1999). Stockholm.

Phillips 2001 Deborah M. Phillips and Hans A. von Spakovsky, Gauging the Risks of Internet Elections *en Communications of the ACM*, vol. 44, no. 1, enero 2001, pp 73-85

Quillian 1968 M Ross Quillian, Semantic Memory en *Semantic Information Processing*, Marvin Misky (editor), The MIT Press, USA, 1968, (sexta impresión 1988)

Rao 1997 Anad S. Rao, Michael P. Georgeoff, Modelling Rational Agents within a BDI Architecture *en Readings in Agents*, Editores Michael N. Huhns y Munindar P. Singh, Ed. Morgan Kauffman, EUA 1997, pp 317-328

Reeves 1999 Daniel M. Reeves, Benjamin N. Grosof, Michael P. Wellman, and Hoi Y. Chan. Toward a Declarative Language for Negotiating Executable Contracts *en AAAI-99 Workshop on Artificial Intelligence in Electronic Commerce (AIEC-99)*, Menlo Park, CA, USA, 1999.

Rist 1998 Thomas Rist, Elizabeth André, Jochen Müller, Adding Animated Presentation Agents to the Interface en *Proceedings of the IUI 98*, USA

Rus 1997a Daniela Rus, Robert Gray, David Kotz, Transportable Information Agent, en *Proceedings of Autonomous Agents 97*, Marina del Rey, CA, USA, 1997

Rus 1997b Rus Daniela, Subramanian Devika, Customizing Information Capture and Access *en ACM Transactions on Information System*, Vol. 15, No. 1, January 1997b

Ruthfield 1995 Scott Ruthfield, The Internet's History and Development: from Wartime Tool to Fish-cam en *Crossroads of the ACM* 2, 1 (Sep. 1995), pp 2-4

Saatcioglu 2001 Kemal Saatcioglu, Jan Stallaert and Andrew B. Whinston, Design of a financial portal *en Communications of the ACM*, Junio 2001, vol 44, no. 6, pp 33-38

Sánchez 1997 J Alfredo Sánchez, John J. Leggett, John L. Schnase, AGS: Introducing Agents as Services Provided by Digital Libraries en *Proceedings of the 2nd ACM International Conference on Digital Libraries*, Philadelphia, PA, USA, 1997

Sánchez 1999 Gildardo Sánchez-Ante, Fernando Ramos, Juan Frausto, *Survey on Planning* (Informe Técnico), Computer Science Department, ITESM Campus Morelos, México, April 30, 1999

Sandholm 1993 Tuomas Sandholm. An implementation of the Contract Net Protocol Based on Marginal Cost Calculations en *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp 256--262, Washington, Julio 1993.

Sandholm 1995 Tuomas W. Sandholm and Victor R. Lesser, Issues in Automated Negotiation and Electronic Commerce:

	Extending the contract Net Framework en *First International Conference on Multiagent Systems*, pp 328-335, Menlo Park, CA, 1995
Schafer 1999	J Ben Scahfer, Josepth Konstan, John Riedi, Recommender systems in e-commerce en *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999
Selker 1994	Selker Ted, COACH: A Teaching Agent that Learn en *Communications of the ACM*, Julio 1994, Vol. 37, No. 7
Shoham 1997	Yoav Shoham, Agent Oriented Programming en *Readings in Agents*, Editores Michael N. Huhns y Munindar P. Singh, Ed. Morgan Kauffman, EUA 1997, pp 329-349
Sivori 1996	John R. Sivori, Evaluated Receipts and Settlement at Bell Atlantic en *Communications of the ACM*, vol. 39 no. 6 (Jun. 1996) pp 24 - 28
Snellen 2001	Ignace Snellen, ICTs, Bureaucracies and the Future of Democracy en *Communications of the ACM*, vol. 44, no. 1, enero 2001, pp 45-48
Stok 2001	Gustavo Stok, Osito en Línea en *América Economía*, 3 may 2001, pp 46-47
Strom 2000	Stephen Strom, Building a Large-Scale Generic Object Model: Applying the CYC Upper Ontology to Object Database Development in Java en *Conference on Object-Oriented Programming, Systems, Languages, and Applications on addendum to the 2000 Proceedings*, 2000, Pages 37 - 38
Tewari 2000	Guarav Tewari, Pattie Maes, Design and Implementation of an Agent-based Intermediary, *en Proceedings of the 2nd ACM Conference on Electronic Commerce*, octubre 2000
Thomas 1997	Thomas Christoph, Fischer Gerhard, Using Agents to Personalize the Web en *IUI 97* Orlando, Florida, USA
Tian 1999	Zhong Tian and Jen-Yao Chung, Business-to-Business e-Commerce with Open Buying on the Internet en *Report of the IBM Institute of Advanced Commerce*, May 1999, disponible también en http://www.ibm.com/iac/tech-paper.html
Towns 1998	Stuart G. Towns, Charles B. Callaway, Jennifer L. Voermann, James C. Lester, Coherent Gestures, Locomotion and Speech in Life-Like Pedagogical Agents, en *Proceedings of the IUI 98*, USA
Tygar 1996	J. D. Tygar, Atomicity in Electronic Commerce *en Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8-26, May 1996
Wachsmuth 1997	Ipke Wachsmuth, Britta Lenzmann, Tanja Jörding, et. al., A Virtual Interface Agent and its Agency en *Proceedings of the Autonomous Agents 97*, Marina del Rey, CA, USA, 1997
Watson 2001	Richard T. Watson and Bryan Mundy ,A Strategic Perspective of Electronic Democracy *en Communications of the ACM*, vol. 44, no. 1, enero 2001, pp 27-35
Wegner 1995	Peter Wegner, *Tutorial Notes: Models and Paradigms of Interaction* (Informe Técnico), Department of Computer Science, Brown University, USA, Septiembre 1995
Wegner 1996	Peter Wegner, *The Paradigm Shift from Algorithms to Interaction* (Informe Técnico), Department of Computer Science, Brown University, USA, October 14th, 1996
Wegner 1998	Peter Wegner, Interactive Foundations of Computing en *Theoretical Computer Science* no. 192 (feb. 1998), pp. 315-351
Weinstein 1997	P. Weinstein, G. Alloway, G. Seed Ontologies: Growing Digital Libraries as Distributed Intelligent Systems en *Proceedings of the Second ACM Digital Library Conference*, Philadelphia, PA, USA (July), 1997, ACM Press
Wooldridge 1998	Michael Wooldridge. Agents and Software Engineering en *AI*IA* Notizie, XI(3):31--37, September 1998
Wooldridge 1999	Michael Wooldridge, Intelligent Agents en *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Gerhard Weiss (Eds.), MIT Press, Cambridge, Massachusetts, USA, 1999
Wooldridge 2000	Michael Wooldridge, *Reasoning about Rational Agents*, The MIT Press, Cambridge Massachusets, London, England, 2000
Wooldridge 2000 a	M. Wooldridge, N. R. Jennings, D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design, en *Journal of Autonomous Agents and MultiAgent Systems*, 3(3):285-312, 2000
Zerda 2001	Guillermo Zerda Santiago, Impuestos en Línea en *América Economía*, 5 abr 2001, p 34
Zunino 2000	Alejandro Zunino, Brainstorm/J: A Framework for Intelligent Agents en *Proceedings of the 2nd Argentinian Symposium on Artificial Intelligence (ASAI'2000 JAIIO),* 2000
Zwingle 1999	Erla Zwingle, Los bienes circulan. [...] Las Ideas Circulan y las Culturas También en *National Geographic*, agosto 1999, pp12-33