Pontificia Universidad Católica de Valparaíso

Facultad de Ingeniería

Escuela de Ingeniería Informática

Ingeniería Civil en Informática

RESOLUCION DE PROBLEMAS COMBINATORIALES DIFICILES UTILIZANDO AGENTES INTENCIONALES

Autor:

Jimena Alejandra Ahumada Martinez

Informe final del Proyecto para optar al Título profesional de

Ingeniero Civil en Informática

Profesor guía:

José Miguel Rubio León

Profesor Co-referente:

Broderick Crawford Labrin

Diciembre 2008

Dedicado a mis padres por su apoyo e incentivo al crecimiento constante, y fundado en valores, tanto en personal como profesional.

Agradecimientos

A Dios por darme la oportunidad de estudiar y desarrollarme profesionalmente.

A mi familia, en especial a mis padres y hermano, por su confianza, paciencia y apoyo incondicional. A mi pololo por su constante ayuda y ánimos para seguir adelante.

A mis profesores por su orientación y consejo a lo largo de todo el proceso.

Lista de Abreviaturas

AAII: Australian Artificial Intelligence Institute

ABT: Asynchronous Backtracking

BDI: Beliefs, Desires, Intentions

BT: Backtracking

CSP: Constraint Satisfaction Problem

DCSP: Distributed Constraint Satisfaction Problem

dMARS: distributed Multi-Agent Reasoning System

FIPA: Foundation for Intelligent Physical Agents

GAIA: Group of Artificial Intelligence Group

IRMA: Intelligent Resource-bounded Machine

JADE: Java Agent DEvelopment

JADEX: Java Agent DEvelopment extension

KIF: Knowledge Interchange Format

KQML: Knowledge Query and Manipulation Language

MaSE: Multiagent Systems Software Engineering

OCML: Operational Conceptual Modelling Language

OWL: Web Ontology Language

PASSI: Process for Agents Societies Specification and Implementation

PRS: Procedural Reasoning System

PRS-CL: Procedural Reasoning System (Common Lisp version)

SHOE: Simple HTML Ontology Extensions

SMA: Sistema Multi-Agentes

UML: Unified Modeling Language

UMPRS: University of Michigan Procedural Reasoning System

XML: Extensible Markup Language

XOL: Ontology Exchange Language

Resumen

Los problemas combinatoriales difíciles representan un gran desafío para las ciencias de la computación y las matemáticas en su conjunto. Se desarrolla, en este trabajo de título, un sistema multiagentes, basado en agentes intencionales para la resolución de problemas combinatoriales difíciles, como son los problemas de las n-reinas, o el sudoku, los cuales han sido modelados como problemas de satisfacción de restricciones distribuidos DCSP.

En una primera etapa se diseñaron los agentes utilizando la metodología de desarrollo orientada a agentes AAII, para posteriormente ser implementado en JADEX, siguiendo el modelo de agentes intencionales BDI, el cual trabaja en función de conjuntos de creencias (Beliefs), deseos (Desires) e intenciones (Intentions), obteniendo resultados satisfactorios dentro de un rango acotado de agentes.

Palabras clave: Sistemas Multiagentes, Agentes Intencionales, Agentes BDI, Problemas Combinatoriales, Problemas de Satisfacción de Restricciones Distribuidos.

Abstract

Difficult combinatorial problems represents a challenge for computer science and mathematics as a whole. It is developed in this work, a multiagent system based on intentional agents for solving difficult combinatorial problems, such as n-queens, or sudoku problems, which has been modelled as distributed constraint satisfaction problems DCSP.

In the first stage agents were designed using the agent oriented methodology AAII, to be implemented after in JADEX, following the model of intentional BDI agents, which works in terms of sets of Beliefs, Desires and Intentions, obtaining satisfactory results within a bounded range of agents.

Key words: Multiagent Systems, Intentional Agents, BDI Agents, Combinatorial Problems, Distributed Constraint Satisfaction Problems.

Capítulo 1
Introducción

Cuando se habla de problemas combinatoriales difíciles, puede referirse a la agrupación, ordenamiento o asignación de un conjunto discreto y finito de objetos que satisfacen ciertas condiciones. Estos representan un gran desafío para las ciencias de la computación y las matemáticas en su conjunto, pues el tiempo que demora encontrar la solución a ellos se incrementa a medida que aumenta su complejidad.

Existe una amplia variedad de problemas combinatoriales que han sido estudiados y para cuya resolución se han desarrollado numerosas técnicas y combinaciones entre ellas, entre las que se encuentran la fuerza bruta, técnicas de Investigación de Operaciones, Inteligencia Artificial, Simulaciones, etc. Como ejemplo a este tipo de problemas, es posible mencionar problemas de satisfactibilidad, como la coloración de grafos, resolución de puzzles, asignación de tareas, de turnos, problemas de ruteo, entre otros.

Se ha propuesto en este trabajo de título, para resolver problemas combinatoriales difíciles, el estudio y utilización de la tecnología multiagentes, un tipo de sistemas que surge de la fusión de conceptos de inteligencia artificial y computación distribuida. Esto a través del modelado y resolución de los problemas en forma de Problemas Distribuidos de Satisfacción de Restricciones [Yokoo et. al, 1998], en que las restricciones y las variables que componen al problema son distribuidas en varios agentes.

Se plantea en particular, el uso de un tipo de agentes denominados como intencionales, tomando como base la arquitectura BDI inspirada por Michael Bratman [Bratman, 1987], y formalizada por David Kinny, Annand Rao y Michael Georgeff [Kinny et al., 1996], quienes definen a un agente BDI como un tipo de agente racional que presenta actitudes mentales: Creencias (Beliefs), Deseos (Desires) e Intenciones (Intentions), emulando las decisiones que probablemente tomaría un ser humano, en respuesta a la información que éste tiene de su entorno en forma dinámica.

Un agente intencional trabaja colaborativamente con otros en búsqueda de la solución, negociando con ellos, y elaborando una serie de planes que le permitan cumplir su objetivo final, en este caso la resolución de problemas combinatoriales difíciles que demuestren características suficientes para ser implementados como un sistema multiagente intencional de modo de aprovechar al máximo sus capacidades, y contrastar los resultados con los que se han conseguido utilizando otras técnicas.

1.1 Definición de Objetivos

1.1.1 Objetivo General

- Resolver problemas combinatoriales típicos difíciles utilizando la arquitectura de agentes intencionales (BDI).

1.1.2 Objetivos Específicos

- Estudiar y analizar los sistemas multi-agente en general, profundizando en la arquitectura de agentes intencionales BDI.

- Estudiar y analizar problemas combinatoriales difíciles típicos.

- Diseñar y documentar la solución de los problemas seleccionados utilizando agentes intencionales.

- Implementar la solución bajo la plataforma de desarrollo JADEX.

- Realizar pruebas, validar y contrastar con resultados obtenidos con otras técnicas.

Capítulo 2

Tecnología Multiagentes

2.1 Definición Agentes Inteligentes y Sistemas Multiagentes

Los Sistemas Multiagentes (SMA) han ido abarcando cada día más dominios de aplicación, proporcionando excelentes resultados en el desarrollo de sistemas distribuidos.

El concepto de agente no tiene una definición formal, una de las más aceptadas es la publicada por Michael Wooldridge en [Wooldridge, 1995], que dice que un agente es "un sistema informático, situado en un entorno, que es capaz de realizar acciones flexibles y autónomas para alcanzar sus objetivos".

Un agente inteligente además de tener autonomía debe cumplir otras características:

• Reactividad: capacidad de percibir cambios en el ambiente y responder a ellos. Un agente debe tener un conjunto de sensores que le permitan saber de su entorno.

• Pro-actividad: es decir, la iniciativa que debe tener el agente para cumplir sus metas. Debe tener un carácter emprendedor.

• Habilidad Social: capacidad de interacción con otros agentes y seres humanos para cumplir sus metas.

Lo anterior constituye la noción débil de agente. Una noción fuerte incluye las características recientemente mencionadas, agregándole otras cuatro más que son:

• Movilidad: capacidad del agente de trasladarse a través de la red.

• Veracidad: un agente no comunicará información que considere falsa.

• Benevolencia: el agente siempre estará dispuesto a ayudar a otros agentes.

• Racionalidad: el agente actúa en forma racional en consecuencia con sus metas.

Una clasificación interesante de agentes, propuesta en [Nwana, 1996], se observa en la Figura 2.1, en esta publicación se divide a los agentes en:

• Agentes de Interfaz: asisten al usuario en la ejecución de tareas.

• Agentes Colaborativos: cooperan entre sí para conseguir ciertos objetivos.

• Agentes Inteligentes: agentes con una estructura general que tienen: capacidad de percibir y representar el ambiente, un conjunto de acciones que puede llevar a cabo, poder de decisión sobre las acciones, un historial de los estados del ambiente y un comportamiento no determinístico.

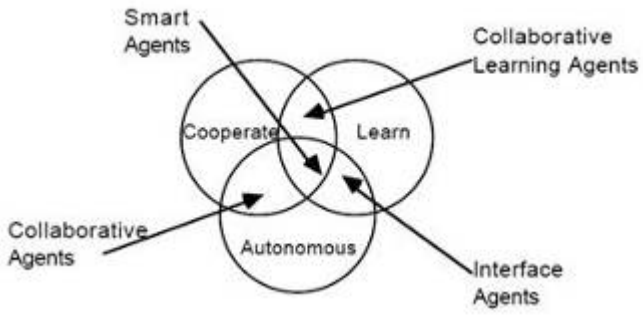
• Agentes Móviles: agentes capaces de moverse a través de redes.

• Agentes de Información: su función principal es la manipulación de información.

• Agentes Reactivos: reaccionan ante estímulos externos.

• Agentes Híbridos: combinan dos o más tipologías de agentes.

Figura 2. 1 Clasificación de agentes



Los Sistemas Multiagentes son aquellos compuestos por varios agentes distribuidos que se comunican y negocian entre sí para proveer una cierta funcionalidad requerida.

Las características de un SMA son:

• Los agentes por sí solos tienen información incompleta o tienen capacidades limitadas para resolver el problema.

• No existe un control centralizado.

• Los datos están dispersos en el sistema.

• La comunicación es asíncrona.

2.2 Infraestructura para agentes

Los agentes necesitan comunicarse entre sí para poder cumplir sus objetivos. Para ello es necesario definir la infraestructura que tendrán los agentes, de modo que puedan entenderse, compartir conocimientos e interactuar.

La infraestructura para agentes se compone de ontologías, lenguajes de comunicación y protocolos de interacción.

2.2.1 Ontologías

Las ontologías definen los conceptos, y sus relaciones, de un dominio determinado. El objetivo es tener un consenso en la especificación del conocimiento para poder ser compartido.

La formalización de la ontología suele incluir:

• Definición de clases que representan los elementos del dominio.

• Definición de los atributos o roles de las clases y sus valores permitidos.

• Relación y jerarquización de las clases mediante enlaces.

Un lenguaje ontológico permite la descripción de los conceptos del dominio de una forma no ambigua. Los primeros lenguajes se basaban en lógicas de primer orden y lógicas descriptivas. Ejemplos de ellos son KIF, OCML, entre otros. Los lenguajes modernos son del tipo markup. Ejemplo: XML, SHOE, XOL, OWL, etc.

2.2.2 Lenguajes de Comunicación

Los agentes envían y reciben mensajes, por ello es necesaria la definición de un lenguaje que otorgue una sintaxis (simbología y su estructuración), una semántica (el significado de los símbolos) y una pragmática (la interpretación).

La teoría del acto del habla (Speech Act Theory) se usa como modelo de comunicación. Los agentes se comunican para mostrar a otros agentes su estado mental y/o intentar modificar el estado mental de otros agentes. Los actos del habla hacen referencia a las acciones intencionales dentro de una conversación.

Quien habla no declara solamente sentencias ciertas o falsas, también realiza actos de habla: peticiones, sugerencias, promesas, amenazas, etc. Cada declaración es un acto de habla.

El acto del habla incluye tres aspectos:

- Locución: el sonido o texto emitido.
- Ilocución: el significado de los sonidos.
- Perlocución: la acción que resulta de la locución.

Los performatives se utilizan para identificar la fuerza ilocutoria: pregunta, petición, etc.

2.2.3 Protocolos de Interacción

Para la comunicación de los agentes, estos se coordinan generando secuencias de mensajes que siguen ciertos patrones a los cuales se les denomina protocolos de interacción. Se describen en ellos el tipo y orden de los mensajes, las secuencias permitidas y las restricciones.

La interacción puede ser directa, a través de mensajes/diálogos, o indirecta, por ejemplo a través del modelo de pizarra. Esta última representa un lugar común al que todos los agentes tienen acceso y de la cual pueden “recoger” o “agregar” información.

Existen muchas formas de protocolos de interacción, pueden ser por ejemplo, en los protocolos de negociación en que los agentes "negocian" entre sí para intentar satisfacer sus objetivos. También existen protocolos de subastas donde los agentes hacen sus "ofertas" a un "subastador".

FIPA ha definido algunos protocolos de interacción estándar, definidos en una Librería, entre los que se encuentran:

- Request: el emisor desea que el receptor realice cierta acción.
- Request When: el emisor desea que el receptor realice cierta acción cuando sea cumplida una condición.
- Query: se consulta por la veracidad de cierta proposición.
- Contract Net: el agente envía propuestas a un potencial agente contratista.
- Subscribe: subscripción a notificaciones de cambios.
- Propose: se envía una propuesta a realizar bajo ciertas condiciones.

2.3 Arquitecturas de Agentes

Para modelar agentes existen tres tipos de arquitecturas principales: reactivas, deliberativas y mixtas. El trabajo a realizar en este trabajo de título se basa en una arquitectura deliberativa. A continuación se exponen las características fundamentales de estas tres arquitecturas.

2.3.1 Arquitecturas Reactivas

Como el nombre lo sugiere, es una arquitectura de agentes reactivos, aquellos que responden con acciones a los estímulos que perciben desde el entorno, pues no tienen una representación del ambiente o de otros agentes, tampoco tienen historia ni planificación.

Un ejemplo de arquitectura reactiva es la de subsunción, donde los agentes tienen dos características primordiales:

i. La toma de decisiones es realizada por un conjunto de comportamientos que realizan tareas. El comportamiento es implementado como una máquina de estados finito, donde situación → acción.

ii. Se pueden activar varios comportamientos a la vez. Los comportamientos están agrupados en capas, las cuales tienen prioridades predefinidas.

Entre sus ventajas se encuentran la simplicidad, velocidad y seguridad en caso de fallas. Sin embargo la falta de información sobre el ambiente, la inexistencia de planificación, la poca capacidad de aprendizaje, son vistos como desventajas de este tipo de arquitecturas.

2.3.2 Arquitecturas Deliberativas

Al contrario de las reactivas, las arquitecturas deliberativas sí planifican las acciones a seguir en base a un conjunto de objetivos, por lo que la toma de decisiones se torna un proceso constante.

Utilizan una representación simbólica lógica del ambiente y son capaces de hacer inferencias sobre el conocimiento adquirido.

Los agentes intencionales son un ejemplo típico de este tipo de arquitectura y son especificados con mayor detalle en el apartado 2.4.

2.3.3 Arquitecturas Mixtas

Las arquitecturas mixtas, o híbridas, combinan elementos de los dos tipos anteriores.

Están basadas en capas, conteniendo por lo menos dos capas: una reactiva y una deliberativa. Las capas pueden ser de dos tipos:

- Horizontales: si todas las capas están conectadas a la entrada y salida del agente, es decir, actúan como si fueran un agente.

- Verticales: si la entrada y la salida están conectadas a una única capa del agente.

Las capas pueden tener comportamientos deliberativos o reactivos.

2.4 Agentes Intencionales: Arquitectura BDI

2.4.1 Agentes BDI

El concepto de intencionalidad ha sido estudiado extensamente en el ámbito filosófico. La teoría propuesta por el filósofo Daniel Dennett [Dennett, 1987] otorga una aproximación a la aplicación de este término en los agentes, al describir un sistema intencional como aquél cuyo comportamiento puede predecirse atribuyéndole creencias, deseos y perspicacia racional.

La arquitectura BDI es del tipo deliberativa, y está basada en el modelo cognitivo propuesto por Bratman [Bratman, 1987]. Se describe a los agentes en esta arquitectura en términos de tres estados mentales particulares: creencias, deseos e intenciones, de los cuales toma las siglas BDI (Beliefs, Desires, Intentions).

Por otro lado, el estudio publicado por Wooldridge y Jennings [Wooldridge y Jennings, 1995] divide estos estados, a los cuales se refiere como actitudes proposicionales, en dos categorías:

• Actitudes de información, que es la información que tiene el agente sobre el mundo que lo rodea: creencias, conocimiento.

• Pro-actitudes, que son las razones que guían las acciones del agente: deseos, intenciones, obligaciones, etc.

Apuntando, a partir de esto, que parece razonable sugerir que un agente debe ser representado en términos de al menos una actitud de información y al menos una pro-actitud, pues un agente intencional tendrá determinados deseos o intenciones en base a lo que éste sabe de su entorno.

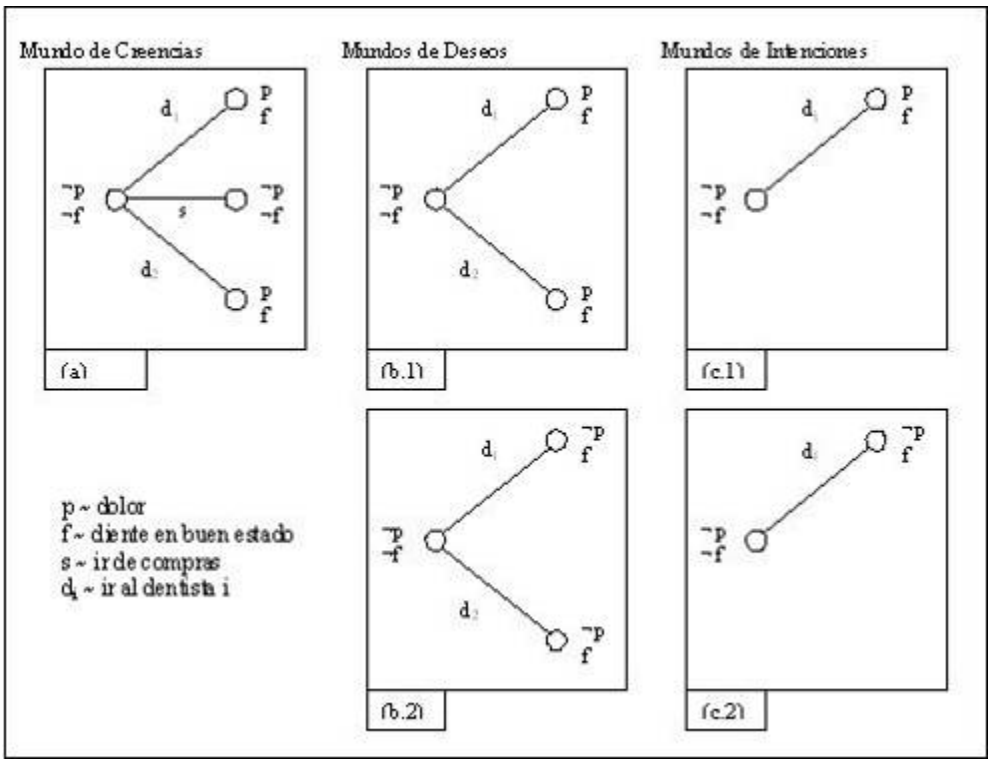
2.4.1.1 Lógica BDI

Varios autores han aportado en la formalización de una lógica BDI, entre ellos se encuentran Cohen y Levesque [Cohen y Levesque, 1990], Rao y Georgeff [Rao y Georgeff, 1991] [Rao y Georgeff, 1993] [Rao y Georgeff, 1995a] y Michael Wooldridge

[Wooldridge, 2000]. Se explica típicamente la lógica BDI como una combinación de las lógicas computacionales de árbol CTL* y lógicas modales.

La lógica CTL* (extended Computation Tree Logic) pretende representar las situaciones o "mundos posibles" en que podría encontrarse el sistema en un punto del tiempo mediante árboles de decisión, donde los nodos representan los estados y los arcos, los eventos. Así, cada árbol va a simbolizar un mundo posible de creencias, deseos o intenciones.

Figura 2. 2 Ejemplo de mundos posibles



La Figura 2.2 es un ejemplo de lo anterior. Suponiendo que un agente tiene un diente en mal estado (¬f), pero no siente dolor (¬p) aún, el estado actual del agente está representado por el nodo que está más a la izquierda. El agente sabe que tiene tres opciones: la primera es ir a un dentista 1, que le producirá dolor, pero le reparará el diente, la segunda es ir de compras, que no le ocasiona dolor, pero tampoco le soluciona el problema, y la tercera que es ir a un dentista 2, con quien tendrá el mismo resultado que con el 1. Todo esto está representado en el árbol de mundo de creencias posibles (a). En los dos siguientes árboles (b.1 y b.2) se ilustran los mundos de deseos posibles, el agente desea arreglar su diente, sea con dolor o sin dolor. Y en los últimos dos cuadros (c.1 y c.2) se muestran los mundos de

intenciones posibles, donde se observan las intenciones que debería tener el agente para conseguir sus deseos. Se puede ver que el mundo de deseos posibles (b.2) no es compatible con las creencias del agente, por lo tanto, éstos no deberían intentar cumplirse, por lo que tampoco se llevarán a la acción las intenciones que se relacionan con éste deseo (c.2). Cabe mencionar que el estado actual (el nodo raíz) es el mismo en todos los árboles de mundos posibles.

En tanto que la lógica modal, proporciona una forma de representar el conocimiento y razonar sobre lo que es o podría ser (mundos posibles) a través de operadores modales.

Se presentan a continuación dos de las lógicas más conocidas. La propuesta de A. Rao y M. Georgeff, es la que ha sido estudiada más ampliamente en la literatura asociada, esta lógica se denomina BDI_{CTL} y BDI_{CTL*} (una extensión de la primera), y está influenciada por la lógica publicada anteriormente por P. Cohen y H. Levesque.

2.4.1.2 Cohen y Levesque

Philip Cohen y Héctor Levesque propusieron en [Cohen y Levesque, 1990], una lógica basada en creencias y deseos. Definieron un lenguaje que contiene los siguientes operadores primitivos:

- HAPPENS α: una acción α sucederá luego.
- DONE α: una acción α ha ocurrido.
- AGT i a: el agente i es el único agente de una acción atómica a.
- BEL i φ: la fórmula φ proviene de las creencias de i.
- GOAL i φ: la fórmula φ proviene de los deseos de i.
- $a \leq b$: la acción a es una sub-secuencia de un b inicial.

Se considera una desventaja de esta lógica el hecho de no considerar las intenciones de los agentes.

2.4.1.3 Rao y Georgeff: BDICTL

- **Sintaxis:**

La lógica BDI_{CTL} distingue entre fórmulas de estado (aquellas que son evaluadas en una situación o mundo posible) y fórmulas de camino (evaluadas en una ruta formada por transiciones entre mundos posibles).

Los operadores temporales CTL* son representados por:

F : eventualmente en el futuro. También simbolizado por "◊"

G : siempre (globally), también simbolizado por "□"

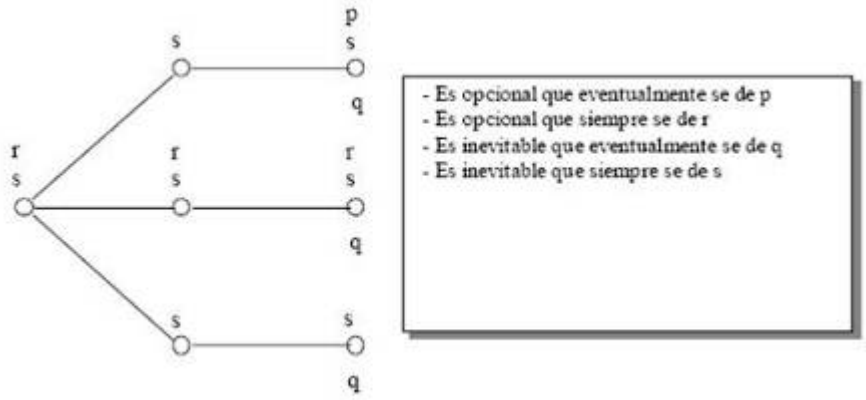
U : hasta (until)

X : siguiente (next), también simbolizado por "O"

E : algún camino, opcionalmente.

A : todos los caminos, inevitablemente.

Figura 2. 3 Lógica BDI



Las modalidades agregadas en la lógica BDI_{CTL} incluyen además los operadores modales:

BEL : creencias (beliefs)

DES : deseos (desires)

INTEND : intenciones (intentions)

2.4.1.4 Arquitectura BDI

Una arquitectura abstracta clásica de agentes BDI, basada en el modelo IRMA [Bratman *et al.*, 1988] contiene:

- Creencias: es la información que tiene el agente sobre su entorno.

- Deseos: son motivaciones del agente, cosas o eventos que el agente quiere conseguir. Un subconjunto de deseos relacionados entre sí conforman un objetivo que el agente persigue. También se les denomina "metas".

- Intenciones: son los objetivos elegidos por el agente. Aquello que se compromete a conseguir.

Además de esto, el agente llevará a cabo algunas funciones:

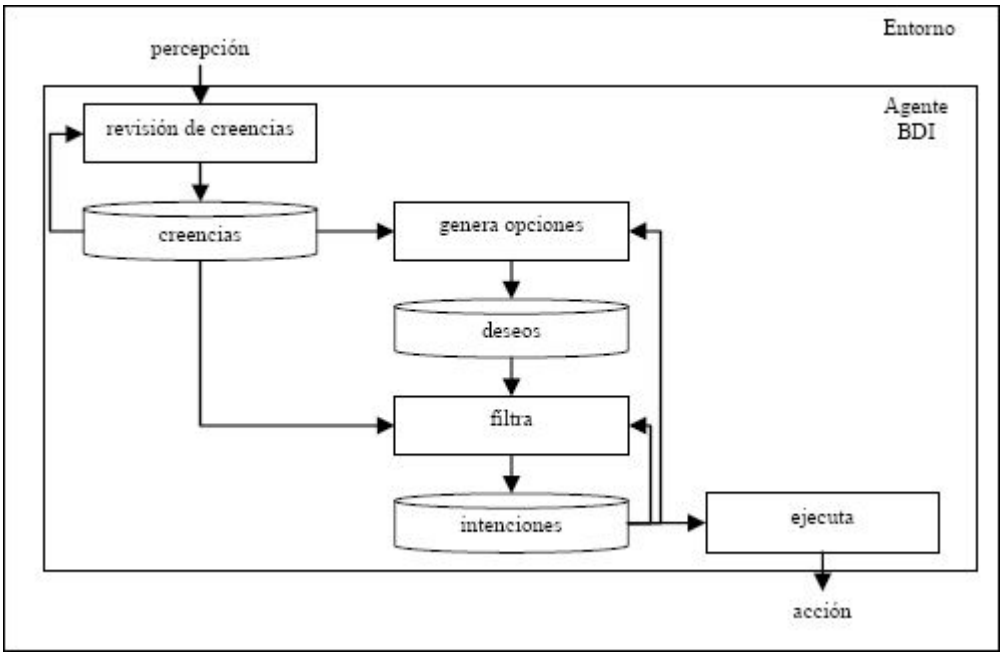
- Revisión de creencias: función que actualiza las creencias del agente en base a las percepciones.

- Generación de opciones: que genera los deseos a partir de las creencias del agente.

- Filtro: que realiza el proceso de deliberación, determinando las nuevas intenciones en función de las creencias, deseos e intenciones.

- Ejecución: determina las acciones a seguir.

Figura 2. 4 Arquitectura de agentes BDI basada en IRMA



Se puede apreciar claramente en la Figura 2.4 la forma en que el agente BDI se comporta a nivel interno y cómo interactúa con su entorno. El agente percibe a través de sensores lo que ocurre en su entorno, procesa la información, creando en base a ella un conjunto de creencias y a partir de ellas genera un conjunto de deseos, los cuales son posteriormente filtrados para crear las intenciones que darán origen a las acciones que el agente realiza, las creencias también son utilizadas al momento de filtrar los deseos e intenciones para descartar aquellos que son incompatibles con la realidad que conoce el agente.

2.4.1.5 Arquitectura BDI Extendida

La arquitectura anterior comprende los componentes básicos de un agente BDI. Sin embargo, esta arquitectura ha sido extendida mediante la inclusión de Objetivos y Planes.

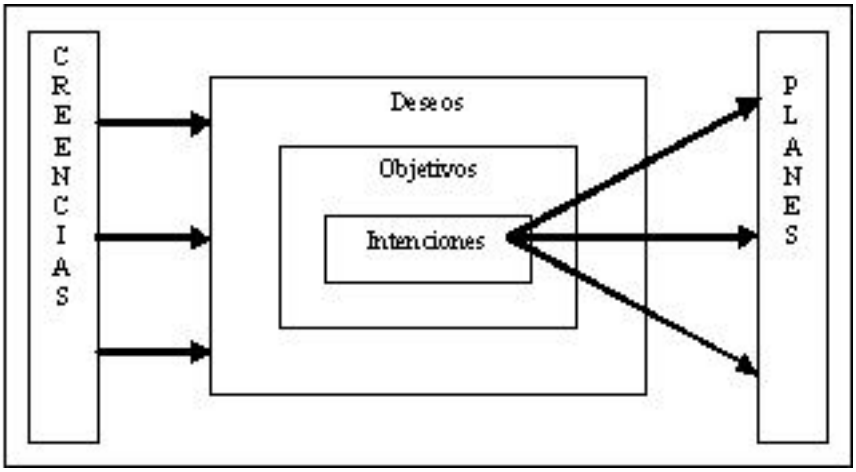
Los objetivos son conjuntos realistas de deseos relacionados entre sí a los cuales podría dedicarse el agente.

Los planes son la información tanto de los medios para el logro de determinados deseos como de las opciones de las que dispone el agente [Rao y Georgeff, 1995]. En otras palabras, son las acciones que realizará el agente para conseguir sus intenciones. En [Guerra, 2006] se explicitan dos características principales de los planes:

1. Los planes son típicamente parciales, es decir que no son completos, sino que se van refinando a medida que se requiere.

2. Los planes tienen típicamente una estructura jerárquica, conllevan otros planes. Usualmente deben cumplirse ciertos medios y pasos preliminares necesarios para conseguir los fines.

Figura 2. 5 Arquitectura extendida de agentes BDI



La Figura 2.5 es una representación de la arquitectura extendida de agentes BDI, que es la comúnmente adoptada por los desarrolladores de agentes BDI.

Cabe mencionar que en [Rao y Georgeff, 1995] se propone una arquitectura muy similar. Agregando a los componentes de la arquitectura una cola de eventos. Los eventos pueden ser externos (que pueden ser estímulos obtenidos desde el entorno del agente) o internos (respuestas a eventos externos, acciones, etc.). Los eventos incluyen la adquisición o eliminación de una creencia, la recepción de mensajes y la adquisición de una nueva meta [Guerra, 2006].

Al interior del agente, la creación de intenciones y su ejecución es realizada por un intérprete. Rao y Georgeff [Rao y Georgeff, 1995] plantean para éste el ciclo mostrado en la Figura 2.6

Figura 2. 6 Intérprete BDI

```
Interprete-BDI
inicializar-estado();
REPEAT
      opciones := generador-de-opciones(cola-de-eventos);
      opciones-seleccionadas := deliberar(opciones);
      actualizar-intenciones(opciones-seleccionadas);
      ejecutar();
      obtener-nuevos-eventos-externos();
      eliminar-actitudes-exitosas();
      eliminar-actitudes-imposibles();
END REPEAT
```

Al inicio de cada ciclo, la función generadora de opciones lee la cola de eventos y retorna una lista de opciones. Luego, el deliberador selecciona un subconjunto de acciones para ser adoptadas y las agrega a la estructura de intenciones. Si existe la intención de realizar una acción atómica en este punto en el tiempo, el agente la ejecuta. Cualquier acontecimiento externo que se haya producido durante el ciclo de intérprete, se añade a la cola de eventos. Los eventos internos son añadidos a medida que éstos ocurren. Posteriormente, el agente modifica las estructuras de intenciones y deseos, abandonando los deseos exitosos y las intenciones satisfechas, así como también los deseos imposibles y las intenciones irrealizables [Rao y Georgeff, 1995].

2.4.2 Implementaciones BDI

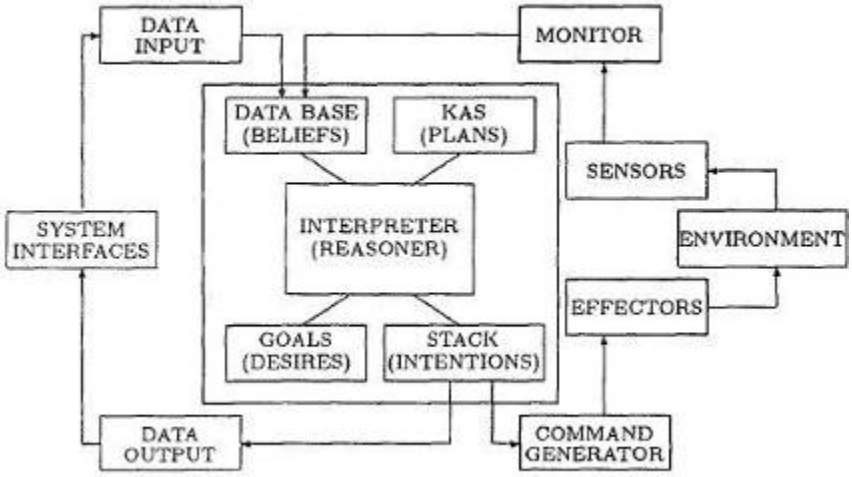
Se han propuesto e implementado varias arquitecturas, lenguajes y plataformas para el desarrollo de sistemas de agentes BDI. La arquitectura PRS ha dado pie a la creación de múltiples implementaciones, y es por esta razón que se expone en este apartado, además de otras relevantes como dMARS de forma más superflua por tratarse de una extensión a PRS, el lenguaje para la arquitectura BDI: AgentSpeak(L) y la arquitectura también descendiente de PRS: JADEX.

2.4.2.1 PRS

El Sistema de Razonamiento Procedural PRS, denominado así por sus siglas en inglés, nace de una propuesta de Michael Georgeff y Amy Lansky en [Georgeff y Lansky, 1987] como un sistema para el razonamiento y realización de tareas complejas en entornos dinámicos utilizando los conceptos de creencias, deseos e intenciones. Fue desarrollada en Lisp por el Instituto Australiano de Inteligencia Artificial y es la primera implementación de agentes BDI.

La Figura 2.7 representa la estructura básica de un sistema PRS.

Figura 2. 7 Estructura del Sistema PRS



El interior del cuadro central corresponde al sistema. Dentro de él se encuentran cinco componentes principales:

• Base de datos (creencias): La base de datos consta de un conjunto de descripciones de estado que describen lo que se considera cierto en el instante de tiempo actual [Georgeff y Lansky, 1987].

• Objetivos (deseos): Representan los comportamientos (behaviors) deseados del sistema. Existen objetivos para la satisfacción de determinadas condiciones, restringiendo o no los medios usados para lograrlos. Existen objetivos que permiten comprobar la veracidad o falsedad de cierta condición, esperar hasta que se produzca una situación, reservar recursos, asegurar que durante la consecución de un estado se ha respetado una condición importante o simplemente objetivos para modificar creencias [García, 2007].

• Áreas de conocimiento (planes): Son conjuntos de planes. Describiendo cómo pueden ser llevadas a cabo las secuencias de pruebas condicionales y acciones para lograr ciertos objetivos o para reaccionar ante determinadas situaciones [Mascardi *et al.*, 2005].

• Intérprete: Encargado de mantener la consistencia entre los demás componentes, desechando aquellos que no son compatibles entre sí, de modo que los planes a ejecutar sean acordes a la realidad. La idea del intérprete es análoga a la expuesta en la Figura 2.6, explicada en el apartado 2.4.1.5.

• Múltiples PRSs asíncronos: En algunas aplicaciones es necesario monitorear y procesar muchos recursos de información al mismo tiempo. Debido a esto, PRS fue diseñado para permitir que varias instancias del sistema básico puedan correr en paralelo [Georgeff y Lansky, 1987].

Los agentes PRS son capaces de comunicarse entre sí y con otras aplicaciones a través del paso de mensajes. Para ello se usa un predicado *send-message* dentro de un objetivo *ACHIEVE*, concretamente *(ACHIEVE (send-message receptor mensaje)*, donde *receptor* es el nombre del agente o proceso receptor y *mensaje* es el contenido del propio mensaje [García, 2007]. La información recibida por el agente pasa por un proceso de deliberación para decidir si es o no compatible con sus creencias antes de incorporarla como una más de ellas.

Las implementaciones más conocidas de PRS son PRS-CL y UMPRS, desarrollados por el SRI Internacional y la Universidad de Michigan respectivamente.

2.4.2.2 dMARS

dMARS son las siglas de distributed Multi-Agent Reasoning System. Esta implementación (en C++) es un sucesor de PRS y fue realizada en el Instituto Australiano de IA.

Cada agente tiene una biblioteca de planes que pueden ser llevados a cabo para conseguir sus intenciones y almacena los eventos que genera el mismo y su entorno en una cola.

Cada plan contiene una condición de invocación (o trigger) que indica cuándo debe ser "disparado", un contexto opcional, para especificar posibles precondiciones que deben cumplirse para la ejecución del plan, un cuerpo, que corresponde a un árbol que representa los flujos de acción donde los arcos son acciones y los nodos, estados, una condición de mantenimiento, que debe cumplirse a lo largo de la ejecución del plan, de no ser así, será

descartado, y un conjunto de acciones internas (que se realizan si el plan falla) y externas (si el plan se concreta).

Para tratar los eventos, el intérprete de PRS escogía un evento a atender de los eventos del sistema. Dicho evento activaba unos planes, algunos de los cuales formaban el conjunto de planes aplicables. De dicho conjunto el intérprete escogía un plan para su instanciación, pasando a formar parte de las intenciones, y finalmente, de todas las intenciones escogía una para su ejecución parcial. Estos tres pasos de selección ahora son tratados como tres funciones de selección que definen el comportamiento del agente [García, 2007].

Las tres funciones de selección que realizan el trabajo realizado por el intérprete PRS son:

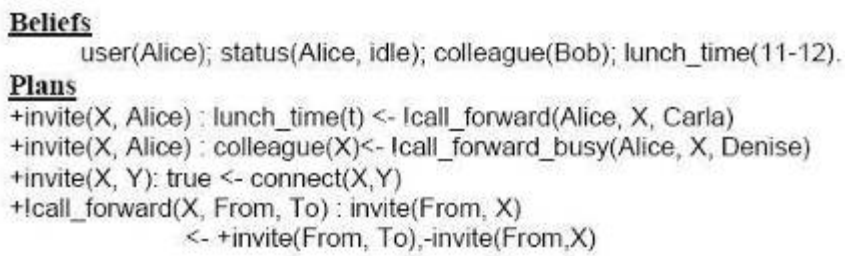
- Función de selección de planes.
- Función de selección de eventos.
- Función de selección de sustituciones.

2.4.2.3 AgentSpeak(L)

AgentSpeak(L) es un lenguaje que formaliza la semántica operacional de PRS y dMARS. Está basado en un lenguaje de primer orden restringido. Las creencias, deseos e intenciones de los agentes no son representados como fórmulas modales, sino que son atribuidos a los agentes, en forma implícita, en tiempo de diseño [Mascardi *et al.,* 2005].

La Figura 2.8 es un ejemplo de la sintaxis de AgentSpeak(L) utilizado en [Plesa, 2006].

Figura 2. 8 Ejemplo de sintaxis de AgentSpeak(L)



Las creencias y los planes son representados utilizando lógica de predicados. Los planes constan de una cabeza (consistente de un evento de disparo que pueden ser de adición (+) o de borrado (-) y un contexto) y un cuerpo (una secuencia de acciones o planes a realizar). Así, para el ejemplo de la Figura 2.8, Alice planea pasar las llamadas entrantes a Carla durante la hora de almuerzo. Para este plan el evento de disparo es una llamada entrante a Alice denotada por el evento invite. El contexto en el cual debe realizarse el plan es que sea la hora de almuerzo, esto es lunch_time(t), siendo t la hora actual. Y el cuerpo del plan, la acción a realizar en este caso es desviar la llamada a Carla, expresada por !call_forward(Alice, X, Carla), donde ! representa un objetivo a realizar, en tanto que ? representa un objetivo a testear.

Al igual que en PRS y dMARS, en cada ciclo de interpretación de un agente, AgentSpeak (L) actualiza una lista de eventos, que pueden ser generados a partir de la percepción del entorno, o de la ejecución de intenciones (cuando los sub-objetivos son especificados en el cuerpo del plan) [Mascardi et al., 2005].

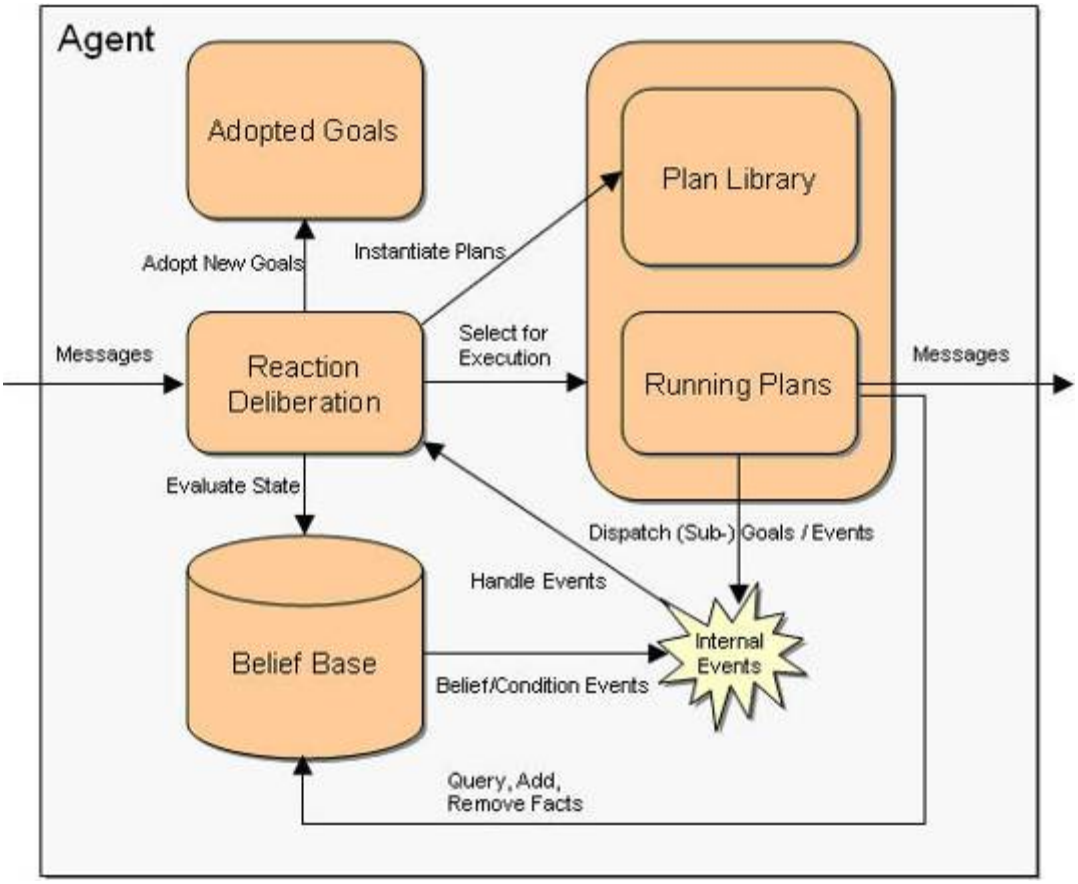
Entre las implementaciones de AgentSpeak(L) se encuentran SIM_Speak, Jason y AgentTalk.

2.4.2.4 JADEX

Se puede decir que JADEX es un sucesor de PRS, está basado en la plataforma JADE e integra las teorías de agentes con la orientación a objetos y descripciones XML. El proyecto está en manos del Grupo de Sistemas Distribuidos y Sistemas de Información de la Universidad de Hamburgo.

El agente JADEX contiene un conjunto de creencias, de planes y de objetivos. La arquitectura JADEX está representada en la Figura 2.9

Figura 2. 9 Arquitectura JADEX



En Jadex una creencia puede ser cualquier objeto Java. Las creencias se almacenan en una base de creencias y pueden ser referenciadas, accedidas y modificadas mediante una interfaz [García, 2007]. La representación de creencias en JADEX es muy simple y no soporta mecanismos de inferencia.

Los objetivos (goals) son deseos concretos y momentáneos de un agente y los intentará satisfacer hasta que los considere logrados, inalcanzables o si deja de ser un deseo. Jadex identifica cuatro tipos de objetivos:

- Perform: El agente quiere realizar una acción, independientemente del resultado.
- Achieve: Especifica un estado del mundo deseado.
- Query: Información que el agente quiere tener.
- Mantain: Cuando el agente quiere mantenerse en un estado.

Los planes representan el conocimiento procedural. Se componen de una cabeza y un cuerpo, siendo análogos a los conceptos explicados en el apartado anterior.

Los eventos pueden ser de tres tipos: de mensajes (envío y recepción), de objetivos (nuevos o cambios en los existentes) e internos (time-outs, ejecución de planes, condiciones de trigger).

2.5 Metodologías de Desarrollo Orientadas a Agentes

En el desarrollo de sistemas complejos se vuelve necesaria la adopción de una metodología que permita obtener una visión clara del problema a resolver y llevar a cado los pasos y procedimientos necesarios para conseguirlo. En este apartado se exponen metodologías de desarrollo que han sido propuestas específicamente para agentes, debido a la naturaleza del trabajo de título, se pone énfasis en las metodologías más utilizadas para el desarrollo de sistemas de agentes BDI, como AAII y Prometheus, y se entrega una referencia de otras también relevantes como GAIA, MaSE y PASSI.

2.5.1 AAII

La metodología de desarrollo AAII considera dos puntos de vista, externo e interno.

Desde el punto de vista externo, el sistema es descompuesto en agentes, modelado como objetos complejos caracterizados por sus propósitos, responsabilidades, los servicios que otorgan, la información que requieren y mantienen y sus interacciones externas [Kinny *et al*, 1996].

Desde el punto de vista interno, los elementos requeridos por una arquitectura de agentes en particular deben ser modelados para cada agente [Kinny *et al*, 1996].

2.5.1.1 Punto de Vista Externo

Desde este punto de vista se identifican dos modelos:

• Modelo de Agentes: que describe la relación jerárquica entre las clases de agentes y las instancias de agentes que pueden existir en el sistema. Para este modelo se utiliza notación de diagramas de clase UML.

• Modelo de Interacción: que describe las responsabilidades de los agentes, los servicios que proporcionan, interacciones asociadas, relaciones de control entre agentes. No hay una notación estándar.

La elaboración de estos modelos está basada en análisis de roles, los pasos a seguir para ello son cuatro:

• Identificar los roles del dominio de aplicación y sus ciclos de vida. Definición inicial de las jerarquías de clases de los agentes.

• Para cada rol, identificar las responsabilidades asociadas y los servicios provistos para llevarlas a cabo. Se descomponen las clases de agentes hasta el nivel de servicios.

• Para cada servicio se debe identificar las interacciones asociadas, los performatives requeridos y el contenido de su información. Se determinan las relaciones de control entre agentes y se realiza un modelo interno de cada clase de agente.

• Refinar la jerarquía de agentes. Definir superclases, herencias, agregaciones y refinar las relaciones de control. Introducir clases concretas de agentes considerando aspectos específicos de implementación.

Los roles pueden ser vistos como un conjunto de responsabilidades, las que a su vez son un conjunto de servicios.

2.5.1.2 Punto de Vista Interno

Este punto de vista se basa en la arquitectura de agentes BDI. Los eventos, creencias, objetivos y planes son capturados por los siguientes modelos:

• Modelo de Creencias: describe la información sobre el entorno, los estados internos del agente y las acciones que puede realizar. Se definen las creencias de los agentes y sus propiedades.

• Modelo de Objetivos: describe los objetivos que los agentes podrían tener y su estado, así como los eventos a los cuales debe responder y su dominio. Esto se realiza utilizando lógica de primer orden con operadores modales.

• Modelo de Planes: se definen las propiedades y estructuras de control de los planes para conseguir los objetivos. Se utilizan para esto diagramas de estado UML.

La consecución de estos modelos es lograda a partir de dos pasos:

• Analizar los medios para conseguir los objetivos. Descomponer los objetivos en actividades (subobjetivos) y acciones. Analizar el orden y las condiciones para que las actividades y acciones sean realizadas, generando planes para ello.

• Construir las creencias del sistema. Descomponer en creencias las actividades y acciones considerando sus contextos y condiciones de control. Analizar las entradas y salidas para cada subobjetivo de los planes estén disponibles como creencias o como subobjetivos previos del plan.

El refinamiento de los modelos internos alimenta a los modelos externos.

2.5.2 Prometheus

La metodología Prometheus [Padgham y Winikoff, 2002] define un lenguaje genérico a cualquier arquitectura SMA. Sin embargo, ha sido mayormente utilizado en el diseño de sistemas de agentes BDI.

Prometheus consta de tres fases:

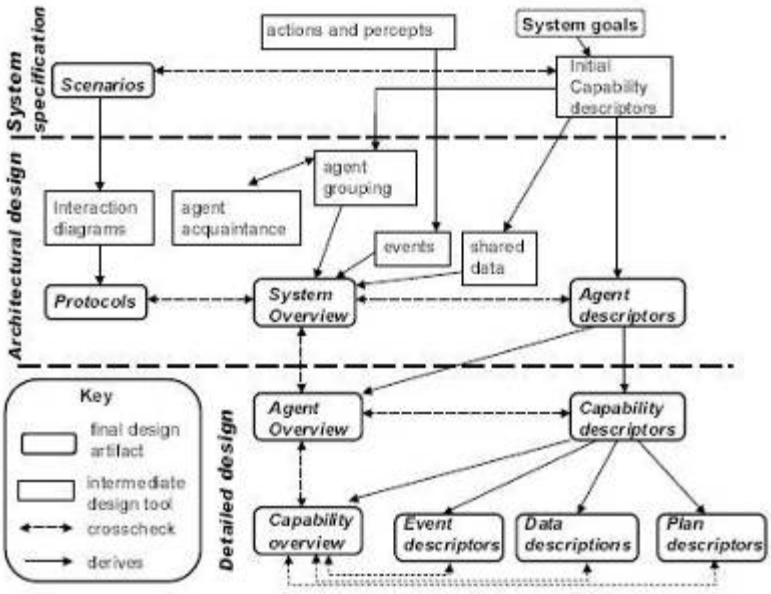
• Fase de especificación del sistema, se centra en la identificación de sus funcionalidades básicas, en conjunto con las entradas (percepciones), las salidas (acciones) y cualquier fuente de datos compartida [Padgham y Winikoff, 2002].

• Fase de diseño arquitectónico, utiliza las salidas de la fase anterior para determinar los agentes que contendrá el sistema y la forma en que van a interactuar [Padgham y Winikoff, 2002].

• Fase de diseño detallado mira el funcionamiento interno de cada agente y la forma en que cumplirá sus tareas dentro del sistema en general [Padgham y Winikoff, 2002].

La Figura 2.10 ilustra las fases, artefactos y relaciones en el proceso de diseño.

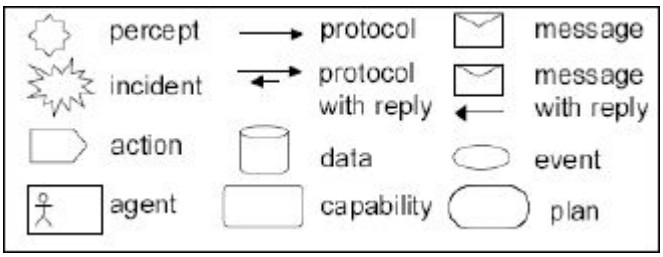
Figura 2. 10 Prometheus [Padgham y Winikoff, 2002].



Para la primera fase, los objetivos se representan mediante un diagrama de objetivos, donde éstos son representados por óvalos relacionados entre sí por flechas dirigidas desde los objetivos padre a los subobjetivos. En tanto que para los escenarios se utilizan escenarios de casos de uso.

En la fase de diseño arquitectónico, los agentes son agrupados en base a sus especificaciones y descritos utilizando descriptores de agentes. Los protocolos son representados mediante diagramas de interacción derivados de los escenarios de casos de uso. La visión general del sistema (system overview) es representada mediante un diagrama que captura los tipos de agentes en el sistema, sus límites e interfaces, utilizando la notación de la Figura 2.11

Figura 2. 11 Notación para la visión general del sistema [Padgham y Winikoff, 2002]



Luego en el diseño detallado, se diagraman las funcionalidades de los agentes agrupadas en capacidades. También se incluye un diagrama general de agentes que represente la estructura interna de los agentes, con una notación análoga a la de la Figura 2.11

2.5.3 GAIA

La metodología GAIA abarca el análisis y diseño de SMA, partiendo con conceptos abstractos en la fase de análisis que se van haciendo cada vez más concretos en la fase de diseño. Se dice que es un proceso de diseño organizacional, pues existe una organización definida por colecciones de roles que interactúan entre sí.

El proceso de análisis considera dos modelos:

• Modelo de Roles: se identifican los roles y sus atributos (responsabilidades, permisos, actividades y protocolos).

• Modelo de Interacciones: donde se definen los protocolos entre roles.

El proceso de diseño agrupa los roles en agentes, generando tres modelos:

• Modelo de Agentes: se modelan los agentes y sus instancias.

• Modelo de Servicios: donde se identifican las funciones (servicios) de cada rol.

• Modelo de Conocidos: que define con quienes se puede comunicar.

2.5.4 MaSE

En MaSE (Multiagent Systems Software Engineering) los agentes son una abstracción conveniente, pueden tener o no inteligencia, e interactúan entre sí para conseguir una meta, asumiéndolos como especializaciones de objetos.

Al igual que en GAIA se identifican una fase de análisis y otra de diseño.

El análisis incluye diagramas de objetivos para representar los requerimientos funcionales del sistema, diagramas de roles, para representar los roles, sus tareas y comunicaciones, y diagramas de secuencia de los casos de uso considerados.

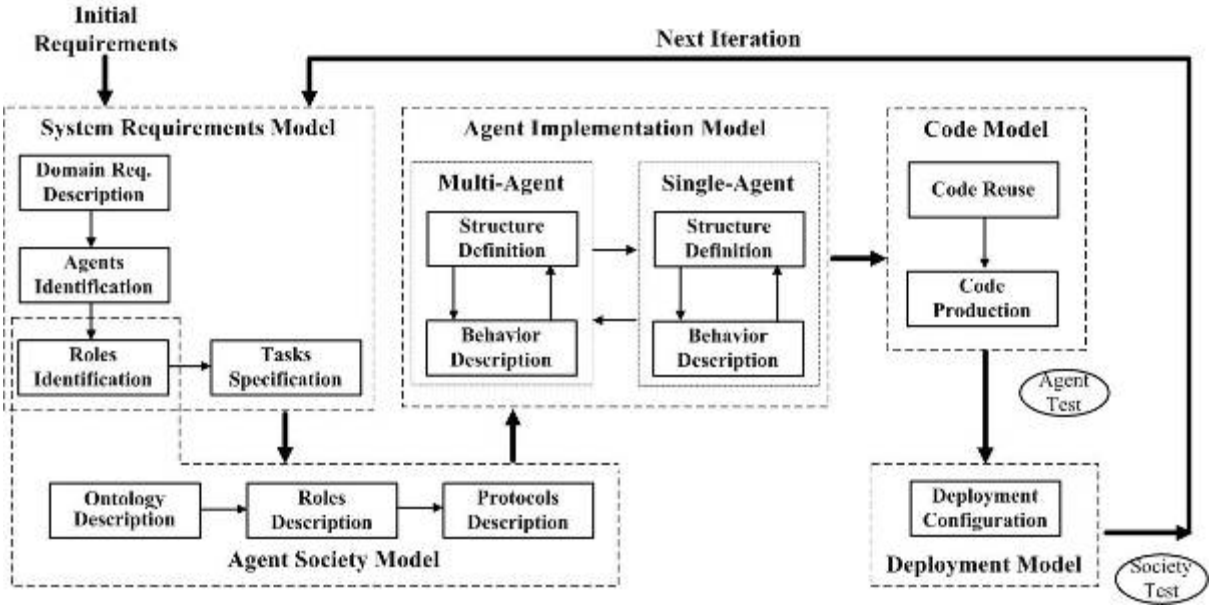
El diseño es un proceso de cuatro pasos: la creación de diagramas de clases de los agentes, diagramas de conversación para representar la comunicación entre los agentes, el ensamble de las clases de agentes mediante una arquitectura y el diseño del sistema a través de diagramas de despliegue.

2.5.5 PASSI

PASSI corresponde a las siglas de Process for Agents Societies Specification and Implementation, es una metodología de cinco pasos que integra conceptos de Ingeniería de Sotware Orientada a Objetos y de Inteligencia Artificial, que adopta UML como lenguaje de modelamiento.

La metodología de PASSI contiene las fases y los pasos a seguir de la Figura 2.12

Figura 2. 12 Metodología PASSI



Capítulo 3

Problemas Combinatoriales Difíciles

3.1 Definición

Un problema combinatorial puede referirse, según [Hoos y Stützle, 2004], a la agrupación, ordenamiento o asignación de conjunto discreto y finito de objetos que satisfacen ciertas condiciones.

En un problema combinatorial se encontrarán los conceptos de:

- **Candidatos a solución:** que son combinaciones de componentes de solución que pueden ser encontradas durante un intento de resolución, pero no necesariamente cumple con todas las condiciones dadas.

- **Soluciones:** que son candidatos a solución que satisfacen todas las condiciones dadas.

Los problemas combinatorios difíciles son aquellos para los cuales, debido a su complejidad, existe una gran cantidad de posibles soluciones y no se conocen algoritmos eficientes de resolución (un algoritmo es eficiente si su tiempo de ejecución está acotado polinomialmente en función del tamaño de los datos que describen el problema), y por ende, no se puede garantizar la mejor solución en un tiempo razonable.

La programación por restricciones es una metodología para la resolución de ciertos problemas combinatoriales difíciles, que trabaja con problemas modelados en forma de problemas de satisfacción de restricciones. Se utiliza como base en este trabajo de título para definir, modelar y resolver algunos problemas combinatoriales difíciles, en conjunto con los conceptos de agentes BDI.

3.2 Problemas de Satisfacción de Restricciones

Para los efectos del presente trabajo de título, se modelarán tres problemas combinatoriales difíciles típicos modelados como Problemas de Satisfacción de Restricciones, concepto a describir en este apartado.

La literatura referente a CSP (Constraint Satisfaction Problems) ha formalizado de forma muy similar en la mayoría de las publicaciones este tipo de formulación de problemas, en este caso recurriremos a la definición publicada en [Rossi et. al, 2006], que dice que un CSP P es una tripleta $P=\langle X,D,C\rangle$ donde X es una n-tupla de variables $X=\langle x_1,x_2,...,x_n\rangle$, D es una n-tupla de los dominios correspondientes $D=\langle D_1,D_2,...,D_n\rangle$ de modo que $x_i \in D_i$, C es una t-tupla de restricciones $C=\langle C_1,C_2,...,C_t\rangle$. Una restricción C_j es un par $\langle R_{S_j},S_j\rangle$ donde R_{S_j} es una relación de las variables en $S_i=scope(C_i)$. En otras palabras, R_i es un subconjunto del producto cartesiano de los dominios de las variables en S_i.

Como ejemplos de representaciones clásicas de CSP es posible mencionar el problema de las n-reinas y coloraciones de grafos.

3.2.1 Backtracking (BT)

Backtracking (o Vuelta Atrás) es un algoritmo de búsqueda completo y uno de los más utilizados en la resolución de problemas combinatoriales con restricciones.

Realiza una búsqueda exhaustiva y sistemática en el espacio de soluciones, generando los nodos del árbol de búsqueda primero en profundidad.

El algoritmo es recursivo y es como se muestra en la Figura 3.1 [Guerequeta y Vallecillo, 1999]. En cada paso, encuentra un valor válido para asignarlo a la variable actual. Si se encuentra un valor válido, se asigna a la variable actual y se avanza al siguiente paso. Si no existe un valor válido, “vuelve atrás” a la última variable para asignarle otro valor, esperando que el último valor de la última variable pueda conducir a encontrar exitosamente un valor válido para la variable actual. Un valor válido para la variable actual es aquel que no tiene conflictos con ninguna variable asignada.

Figura 3. 1 Algoritmo Backtracking

```
PROCEDURE VueltaAtras(etapa);
BEGIN
       IniciarOpciones;
       REPEAT
              SeleccionarNuevaOpcion;
              IF Aceptable THEN
                     AnotarOpcion;
                     IF SolucionIncompleta THEN
                            VueltaAtras(etapa_siguiente);
                     IF NOT exito THEN
                            Cancelar Anotacion
                     END
              ELSE (* solucion completa *)
                     exito := TRUE
              END
       END
       UNTIL (exito) OR (UltimaOpcion)
END VueltaAtras;
```

3.3 Problemas Distribuidos de Satisfacción de Restricciones

La formalización de un CSP distribuido es la misma que para un CSP, la diferencia radica en la forma en que se resuelve el problema. Un Problema Distribuido de Satisfacción de Restricciones (o DCSP – Distributed Constraint Satisfaction Problem) distribuye las variables y restricciones en varios agentes que negocian y colaboran entre sí para resolver el problema en conjunto.

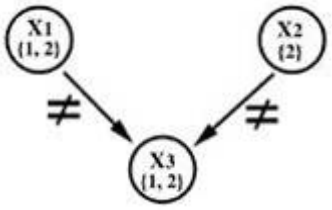
Formalmente, según [Yokoo et. al, 1998], existen m agentes 1, 2,…, m. Cada variable χ_j pertenece a un agente i (esta relación es presentada como $belongs(\chi_j,i)$). Las restricciones también están distribuidas entre los agentes. El hecho de que un agente l conoce un predicado de restricción p_k es representado como known(p_k,l).

Los mismos problemas clásicos identificados como CSP pueden ser modelados como DCSP.

Para ilustrar el problema a modelar, se considera en este apartado, a modo de ejemplo, un CSP distribuido en tres agentes. Al igual que en [Yokoo et. al, 1998], supondremos que cada agente tiene exactamente una variable, todas las restricciones son binarias y cada agente conoce todos los predicados de restricciones relevantes a su variable. La Figura 3.2 ilustra lo anterior, normalmente se modelará un DCSP como un grafo donde cada nodo

representa a un agente que maneja una única variable dentro de un dominio, y cada arista representa las restricciones entre cada par de variables. El enlace dirigido indica, como se detalla más adelante, que el agente con enlace entrante es el que maneja la restricción entre él y el agente desde donde sale dicho enlace.

Figura 3. 2 Ejemplo de DCSP



3.3.1 Backtracking Asíncrono (ABT)

El algoritmo Backtracking Asíncrono (Figura 3.3) fue publicado por primera vez en [Yokoo et. al, 1998] como una adaptación del backtracking tradicional a un entorno distribuido.

Como paso inicial, cada agente instancia su variable concurrentemente y envía su valor a los agentes a los cuales está conectado por enlaces salientes (Figura 3.3-a).

Los mensajes que intercambian los agentes pueden ser dos: uno es el mensaje ok?, que recibe un agente evaluador de un agente emisor preguntando si el valor es o no aceptable, y el segundo es un mensaje nogood que un agente emisor recibe, indicando que el agente evaluador ha encontrado una violación a la restricción y que no es capaz de resolver sus restricciones por si solo.

Cada agente tiene un conjunto de valores de los agentes que están conectados mediante enlaces entrantes. Estos valores que va conociendo a medida que recibe mensajes ok?, constituyen la visión del agente (Figura 3.3-a), el agente intentará encontrar un valor que satisfaga estas restricciones en base a su visión. Si el agente no es capaz de encontrar un valor, entonces “volverá atrás”, enviando un mensaje nogood a un vecino que se conecta a él mediante un enlace entrante, para que este modifique su valor (Figura 3.3-b).

Se establece un orden de prioridad entre los agentes, éste es definido por el orden alfabético o secuencial de los identificadores de los agentes, esto para evitar que se produzcan ciclos. Así, por cada restricción, el agente de menor prioridad será un evaluador y el agente de mayor prioridad enviará un mensaje ok? al evaluador. Además, si se encuentra un nogood, un mensaje nogood será enviado al agente de menor prioridad en el nogood.

Un mensaje nogood puede ser relativo a un valor antiguo si es que el agente evaluador no ha actualizado su visión con el valor actual de sus emisores antes de enviar el mensaje. Es por esto que un mensaje nogood lleva información adjunta, esta es el subconjunto de la vision del agente que gatilló el nogood. De este modo, al recibir un mensaje de nogood, el receptor sólo cambia su valor si el nogood es compatible con su visión y su propia asignación.

Un nogood puede ser visto como una nueva restricción derivada de las restricciones originales. Mediante la incorporación de una nueva restricción, los agentes pueden evitar repetir el mismo error. Para ello es posible que durante el proceso se requiera agregar nuevos enlaces entre los agentes (Figura 3.3-c).

Figura 3. 3 Ejemplo Algoritmo Backtracking Asíncrono.

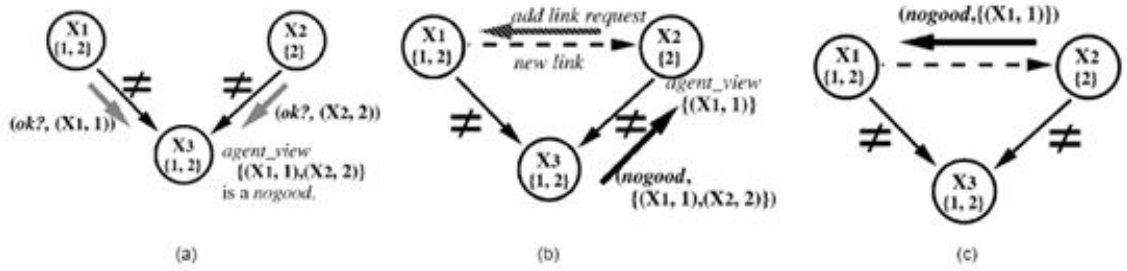


Figura 3. 4 Algoritmo Backtracking Asíncrono.

```
WHEN recibido (ok?, (x_j, d_j)) DO
      agregar(x_j, d_j) a vision_agente;
      revisar_vision_agente;
END DO;

WHEN recibido (nogood, x_j, nogood) DO
      agregar nogood a listado_nogood;
      WHEN (x_k, d_k) donde x_k no esta conectado y esta en el contenido del nogood
      DO
            solicitar a x_k agregar un enlace desde x_k a x_i;
            agregar(x_k, d_k) a vision_agente;
      END DO;
      valor_antiguo ← valor_actual; revisar_vision_agente;
      WHEN valor_antiguo = valor_actual DO
            enviar(ok?, (x_i, valor_actual)) a x_j;
      END DO;
END DO;

PROCEDURE revisar_vision_agente
      WHEN vision_agente y valor_actual no son consistentes DO
            IF ningun valor en D_i es consistente con vision_agente THEN
                  backtrack;
            ELSE seleccionar d∈D_i donde vision_agente y d son consistentes;
                  valor_actual ← d;
                  enviar(ok?, (x_i, d)) a enlaces salientes; END IF; END DO;

PROCEDURE backtrack
      nogoods ← (V | V = subconjunto inconsistente de vision_agente);
      WHEN un conjunto vacio es un elemento de nogoods DO
            Emitir a otros agentes que no existe solucion,
                  Terminar este algoritmo; END DO;
      FOREACH V ∈ nogoods DO;
            seleccionar(x_j, d_j) donde x_j tiene la menor prioridad en V;
            enviar (nogood, x_i, V) a x_j;
            remover(x_j, d_j) de vision_agente;
      END DO;
```

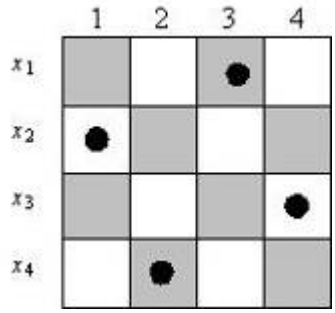
3.4 Ejemplos de DCSP

Existen numerosos problemas que pueden ser representados como DCSP, en el presente trabajo de título se abordarán los tres problemas que se describen brevemente a continuación: N-Reinas, Sudoku y Cuadrado Mágico. Estos problemas son modelados con mayor detalle en el capítulo 4 (Modelado del Sistema).

3.4.1 N-Reinas

El problema de las n-reinas consiste de un tablero de n x n en el cual se deben ubicar n reinas en posiciones tales que estas no se ataquen entre sí, considerando que éstas se mueven como lo harían en un tablero de ajedrez. Un ejemplo de solución se puede ver en la Figura 3.5.

Figura 3. 5 Tablero de 4-Reinas



3.4.2 Sudoku

Sudoku es un juego que consiste en un cuadrado de N x N celdas como el de la Figura 3.6, donde N son números cuadrados n, es decir, $n^2 = N$, dividido en seis pequeñas cajas de n x n. La idea del juego es completar el cuadro con números que van de 1 a N, sin que éstos se repitan en las filas y columnas del cuadro principal, ni tampoco dentro de las cajas de n x n.

Figura 3. 6 Tablero de Sudoku de orden 9

	1		6		7			4
	4	2						
8	7		3			6		
	8			7			2	
			8	9	3			
	3			6			1	
		8			6		4	5
						1	7	
4			9		8		6	

3.4.3 Cuadrado Mágico

El cuadrado mágico consiste en disponer en una matriz de n x n, una serie de números enteros (que generalmente van de 1 a n^2), de modo que la suma de las filas, columnas y las diagonales sea la misma.

La fórmula de Gauss dice que para una matriz de n x n, la suma debe ser igual a:

$$m = 1 + 2 + 3 + \ldots + n^2 = \frac{n^2(n^2+1)}{2} \qquad (3.1)$$

Un ejemplo de solución, para n = 3, se puede ver en la siguiente figura (Figura 3.7):

Figura 3. 7 Tablero de Cuadrado Mágico

4	9	2
3	5	7
8	1	6

3.5 Binarización de CSP's

Se ha mencionado en el apartado 3.3, que para poder modelar el un CSP como DCSP es necesario que todas las restricciones del problema sean binarias, es decir, que incluyan solamente un par de variables por cada una.

El modelado común del problema del cuadrado mágico es n-ario, pues sus restricciones comprenden a más de 2 variables cada una, y por ello, se explican en este apartado dos técnicas de binarización de restricciones, que son capaces de convertir un problema n-ario en uno binario, mayor información sobre estas técnicas se puede encontrar en [Rossi et al, 1990] y [Bacchus y van Beek, 1998]. Cabe destacar que todo problema n-ario puede transformarse en uno binario mediante estas técnicas [Rossi et al, 1990].

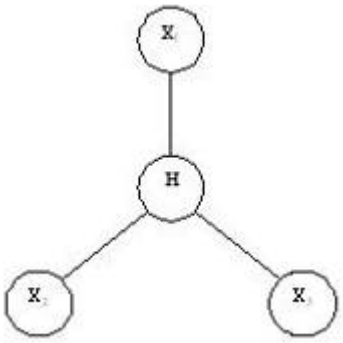
Para conseguir esto se describen a continuación: el método de la variable oculta y el método de representación dual.

3.5.1 Método de la Variable Oculta

Esta representación utiliza las mismas variables del CSP original, agregando otras Hi variables ocultas (hidden) por cada restricción Ci. Así, para cada variable Hi se agregará una restricción binaria entre ésta y cada una de las variables contenidas en la restricción Ci, las cuales conforman una tupla manipulada por H_i.

Por ejemplo, si se tiene originalmente tres variables (X_1, X_2 y X_3) incluidas en una sola restricción, la binarización de ello daría origen a la variable oculta H que manipula estos tres valores, como se muestra en la Figura 3.8, en que las restricciones binarias están dadas por los enlaces, y que indican que cada variable Xi es igual al valor correspondiente manipulado por la variable oculta H.

Figura 3. 8 Método de la Variable Oculta

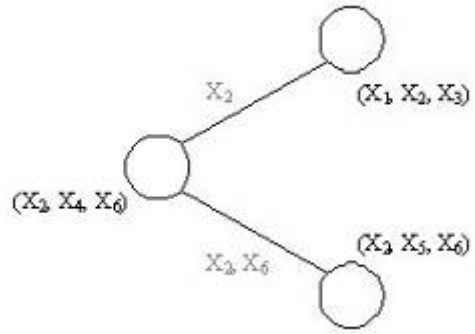


3.5.2 Método de Representación Dual

En este método las restricciones también son transformadas en variables. Sin embargo, las variables originales son representadas solamente en las tuplas. Va a existir una restricción entre dos variables si es que éstas comparten algún valor en sus tuplas, representando la igualdad entre ellas.

Por ejemplo, si se tiene originalmente seis variables (X_1, X_2, X_3, X_4, X_5 y X_6) y existen restricciones entre (a) X_1, X_2 y X_3, (b) entre X_2, X_4 y X_6, y (c) entre X_2, X_5 y X_6, la representación binaria sería la representada en la Figura 3.9.

Figura 3. 9 Método de Representación Dual



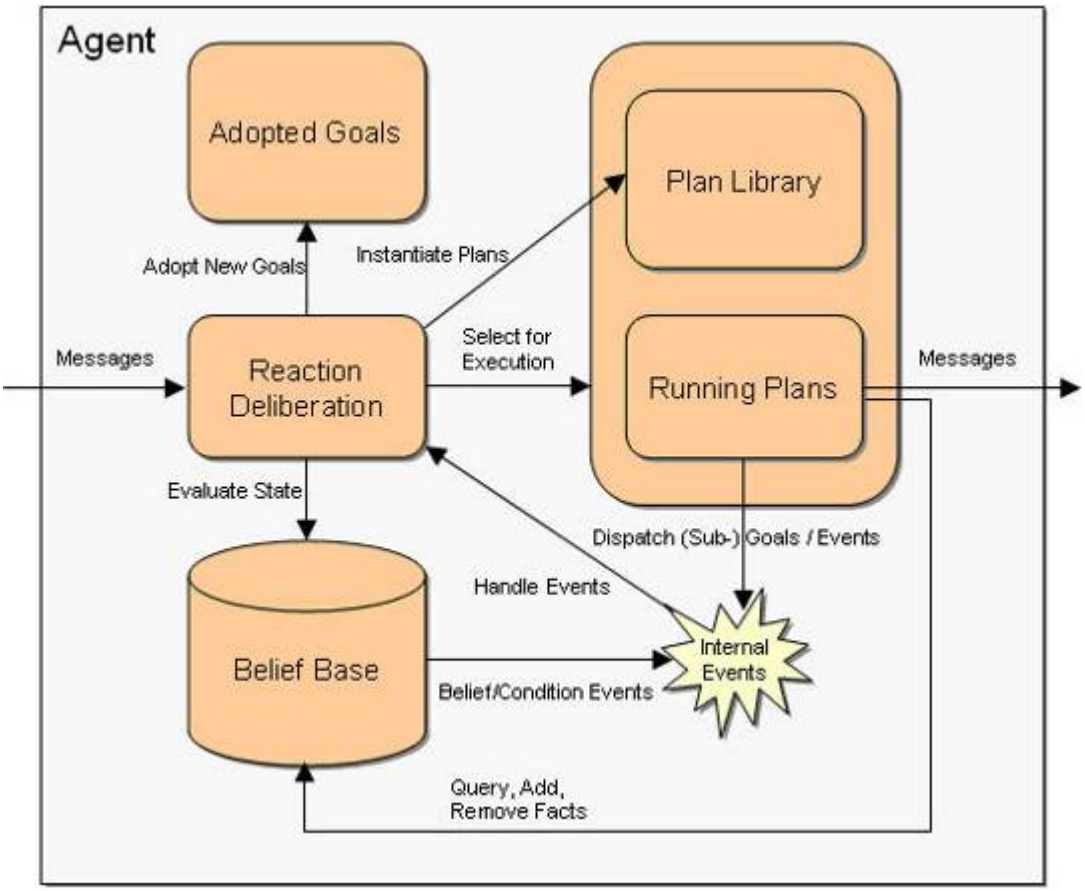
Capítulo 4

Desarrollo del Sistema

En el presente trabajo, la arquitectura del agente DCSP será la de JADEX, descrita con mayor detalle en el capítulo 2, puesto que el sistema será implementado en esta plataforma. En la Figura 4.1 se puede observar que trabaja con tres conceptos: Creencias (Belief Base), Objetivos (Adopted Goals) y Planes (Plan Library y Running Plans).

El agente recibe mensajes desde el exterior, alimentando la base de creencias, activando objetivos a realizar y gatillando la ejecución de planes.

Figura 4. 1 Arquitectura JADEX



4.1 Modelo del Sistema

Se aplicará como estrategia de resolución al problema el algoritmo backtracking asíncrono, de la forma propuesta en [Le Dinh y Seaw, 2007], obteniendo a partir de ello, el modelo del sistema, utilizando la metodología de desarrollo orientada a agentes BDI: AAII, se recurrirá a esta metodología debido a que sus modelos representan claramente a cada uno de los componentes principales de un agente JADEX: creencias, objetivos y planes.

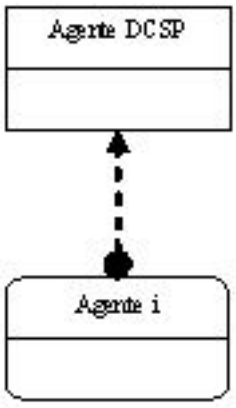
4.1.1 Punto de Vista Externo

Desde el punto de vista externo, el sistema es descompuesto en agentes que deben ser modelados, así como también sus instancias y las interacciones entre ellos [Kinny *et al*, 1996], para ello se genera el Modelo de Agentes y el Modelo de Interacción del Sistema.

4.1.1.1 Modelo de Agentes

El modelo general del agente DCSP será como se muestra en la Figura 4.2, en este caso, el sistema trabaja con múltiples agentes, todos de la misma clase. Esto está representado por la clase Agente DCSP, el círculo en el origen de la flecha punteada indica que pueden existir múltiples instancias del agente.

Figura 4. 2 Modelo general del agente DCSP



4.1.1.2 Modelo de Interacción

Los agentes necesitarán comunicarse para:

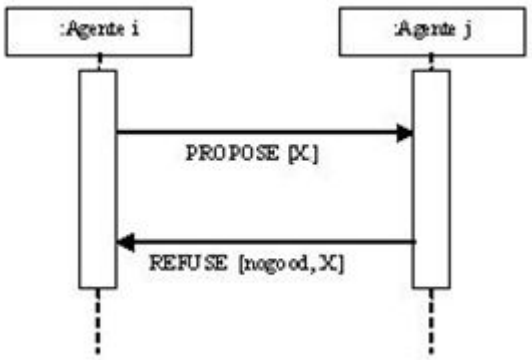
- Proponer el valor de la variable que maneja el propio agente.

• Si el agente receptor, dada la propuesta del agente emisor, no es capaz de encontrar una solución, envía en respuesta un mensaje nogood.

• Solicitar ser agregado como vecino.

No existe una notación estándar para este modelo. Mediante un diagrama de secuencia se muestra en la Figura 4.3 el modelo de interacción que deberán seguir los agentes, para la primera y segunda interacción:

Figura 4. 3 Modelo de Interacción Agentes DCSP: Propuesta y Rechazo de valor



A continuación se detalla el contenido de los mensajes que emiten los agentes, a través de lenguaje KQML, correspondiente a los tres mensajes que aparecen en el diagrama, PROPOSE, AGREE y REFUSE respectivamente.

En el primer mensaje se propone al agente vecino j un valor para la variable X_i perteneciente al agente i:

Informar valor

(propose

:sender Agente i

:receiver Agente j

:content X_i

)

En el segundo mensaje, el agente j puede rechazar el valor de la variable propuesta por i. De no cumplir con las restricciones, el agente j rechaza el valor de X_i:

Rechazar valor de X_i – noood

(refuse

:sender Agente j

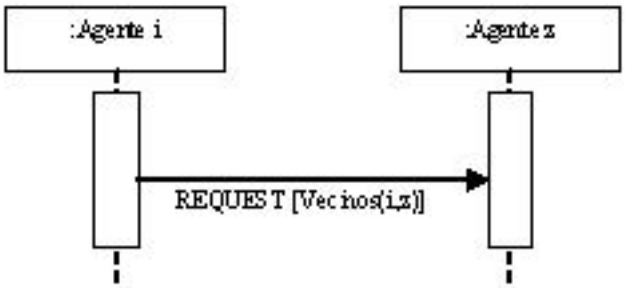
:receiver Agente i

:content X_i

)

La Figura 4.4 representa a la tercera interacción mencionada: Solicitar ser agregado como vecino.

Figura 4. 4 Modelo de Interacción Agentes DCSP: Solicitud de vecindad



En Lenguaje KQML:

Solicitar ser agregado

(request

:sender Agente i

:receiver Agente z

:content vecinos(i,z)

)

4.1.2 Punto de Vista Interno

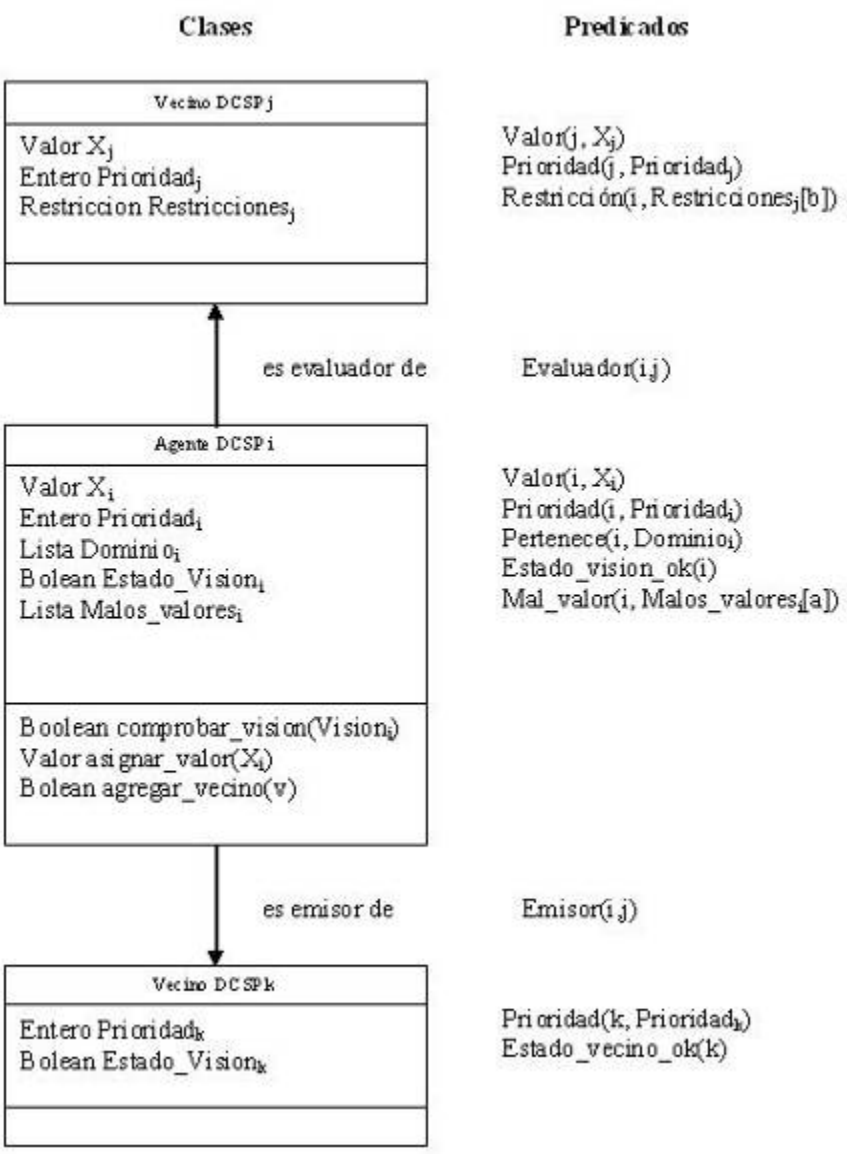
El punto de vista interno, los elementos requeridos por una arquitectura de agentes en particular deben ser modelados para cada agente [Kinny *et al*, 1996]. A continuación, se describen el Modelo de Creencias, el Modelo de Objetivos y el Modelo de Planes del sistema.

4.1.2.1 Modelo de Creencias

Para confeccionar el modelo de creencias, algunos parámetros que serían de utilidad para el agente son: (i) el valor actual de su variable, (ii) la prioridad que tiene el agente frente a sus vecinos (iii) el dominio de la variable asignada, (iv) el listado de los agentes vecinos (v) las restricciones de los vecinos con prioridad mayor o igual a la de él mismo y (vi) la visión que el agente de la situación actual en que se encuentra la resolución del problema.

Las creencias se modelan en un diagrama de clases y se especifican en forma de predicados (Figura 4.5).

Figura 4. 5 Modelo de creencias del agente



En base a esto, el sistema podrá encontrarse en una de las tres siguientes situaciones [Le Dinh y Seaw, 2007]:

(i) es posible cambiar la asignación de su variable para mejorar la situación actual.

(ii) no es posible cambiar la asignación de su variable, puesto que algunas violaciones a las restricciones no pueden ser resueltas.

(iii) no necesita cambiar la asignación de su variable para que las restricciones sean satisfechas.

4.1.2.2 Modelo de Objetivos

Los objetivos son conjuntos realistas de deseos relacionados entre sí a los cuales podría dedicarse el agente. El Modelo de Objetivos de la Figura 4.6 muestra los objetivos del agente DCSP. Se identifican dos objetivos.

Figura 4. 6 Modelo de Objetivos

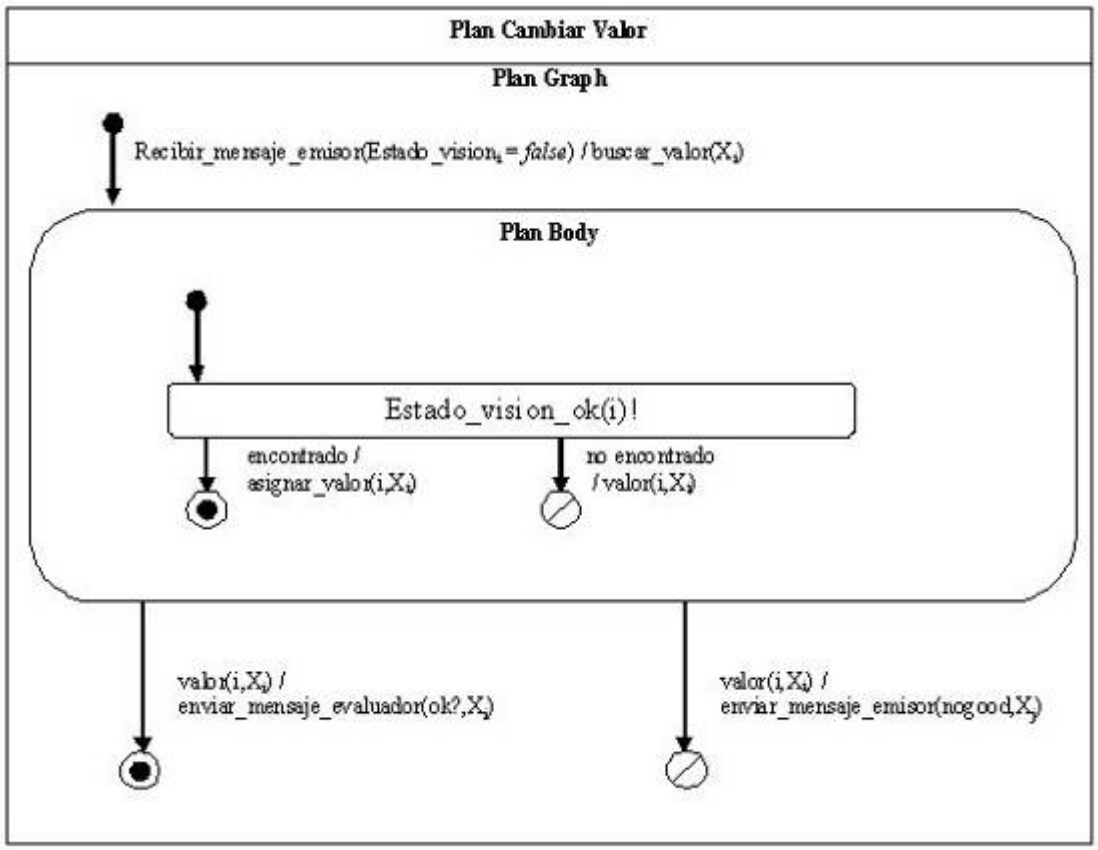
El primer objetivo (Figura 4.6-i), consiste en conseguir que el valor asignado a la variable que manipula el agente i, es decir X_i, cumpla todas las restricciones de los agentes j que tienen prioridad mayor o igual que la de él, y no forme parte de una visión calificada como nogood, es decir, el atributo Estado_vision del agente i debe ser igual a *true*. Este objetivo se activa una vez que se recibe un mensaje de algún vecino emisor del agente i, con un valor que no cumple las restricciones que maneja el agente i. El símbolo (!) denota que es un objetivo a conseguir.

Un segundo objetivo (Figura 4.6-ii), que se activa en el mismo momento que el objetivo anterior, es Evalua(i,z) el cual indica que el agente deseará convertirse en evaluador de un agente z, emisor de j, que no se encuentra en su lista de vecinos.

4.1.2.3 Modelo de Planes

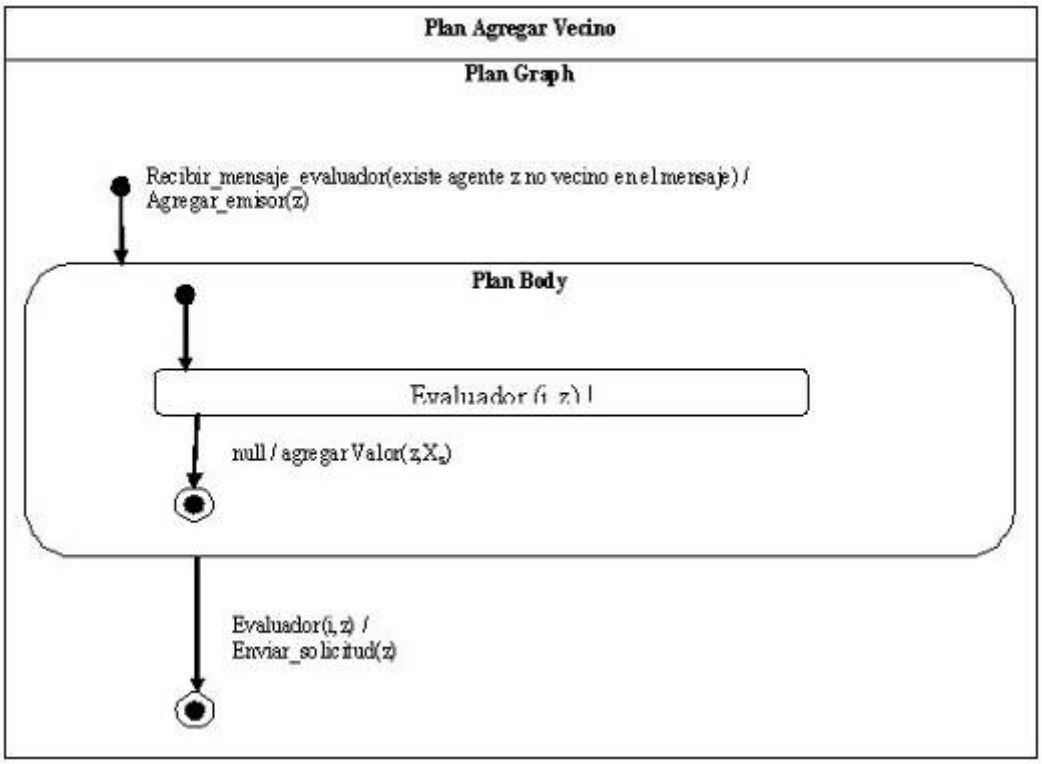
Para finalizar, se define un plan como la información tanto de los medios para el logro de determinados objetivos como de las opciones de las que dispone el agente. El Modelo de Planes de la Figura 4.7 representa al plan que intentará conseguir el objetivo Estado_vision(i, *true*)!, plan que se activará cuando se percate de que su visión se encuentra en un estado inconsistente, pues la creencia Estado_vision es *false*, es decir, no se cumplen todas las restricciones con ellos, o se ha recibido una visión "nogood" que concuerda con la visión actual del agente. Posterior a ello, si es posible encontrar un valor, entonces realizará el cambio y lo informará a sus vecinos, sino hará "backtrack", enviando un mensaje "nogood" a su vecino emisor de menor prioridad.

Figura 4. 7 Modelo de Plan: Cambiar Valor



El plan de la Figura 4.8 corresponde a Agregar Vecino. Cuando se recibe un nogood desde un agente k, el agente i recibe un listado con agentes cuyos valores conoce el agente k. El agente revisa quien no está en su lista de vecinos (un agente z) y agrega el valor de z a sus creencias, convirtiéndose en evaluador de éste. El agente i envía una solicitud al agente z para que este también lo agregue.

Figura 4. 8 Modelo de Plan: Agregar Vecino



4.2 Modelo de DCSPs

En este apartado se modelan los tres problemas seleccionados para la implementación del sistema: N-Reinas, Sudoku y Cuadrado Mágico, determinando para cada uno de ellos las variables, sus dominios y restricciones, así como también el grafo que los representa.

4.2.1 N-Reinas

El modelo que se utilizará para este problema es el estándar. Considera *n* variables (una reina por fila), y por ende, n agentes, donde el dominio de cada una de ellas es D={1,…,*n*} (columna en que se posiciona la reina) y cuya formulación matemática es:

Variables:

$$x_i,\ \forall i \in \{1,\ldots,n\}$$

Dominios:

$$x_i \in \{1,\ldots,n\},\ \forall i \in \{1,\ldots,n\}$$

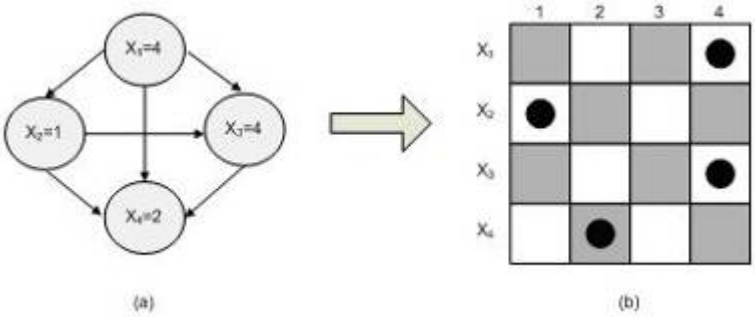
Restricciones:

$$x_i \neq x_j,\ \forall i,j \in \{1,\ldots,n\} \wedge i \neq j$$

$$|x_i - x_j| \neq |i-j|,\ \forall i,j \in \{1,\ldots,n\} \wedge i \neq j$$

En la Figura 4.9 aparece una representación del problema con n = 4, distribuido en cuatro agentes, en donde el grafo (a) muestra la disposición de los agentes los agentes, los valores de sus variables, y la dirección en que se envían los mensajes que está dada por los enlaces dirigidos del grafo, y el es esquema (b) es el tablero representado por el grafo.

Figura 4. 9 Modelo DCSP: 4-Reinas



4.2.2 Sudoku

Sudoku puede ser visto como un problema de coloración de grafos. Sin embargo, su formulación se vuelve mucho más compleja, no por su dificultad, sino por el número sustantivamente mayor de restricciones que este posee, creando múltiples conexiones en el grafo que lo representa.

Este problema contiene NxN variables, donde cada una representa una celda del tablero. El modelado del problema sería el siguiente [Barber y Salido, 2008], para N=9:

Variables:

$$x_{ij},\ \forall i,j \in \{1,2,\ldots,9\}$$

Dominios:

$$x_{ij} \in \{1,2,\ldots,9\},\ \forall i,j \in \{1,2,\ldots,9\}$$

Restricciones: existe una relación binaria de desigualdad entre:

- Todos los pares de variables en cada fila:

  $\neq(x_{11},x_{21},x_{31}\ldots,x_{91}),\ \neq(x_{12},x_{22},x_{32}\ldots,x_{92}),\ \ldots,$
  $\neq(x_{19},x_{29},x_{39}\ldots,x_{99})$
- Todos los pares de variables en cada columna:

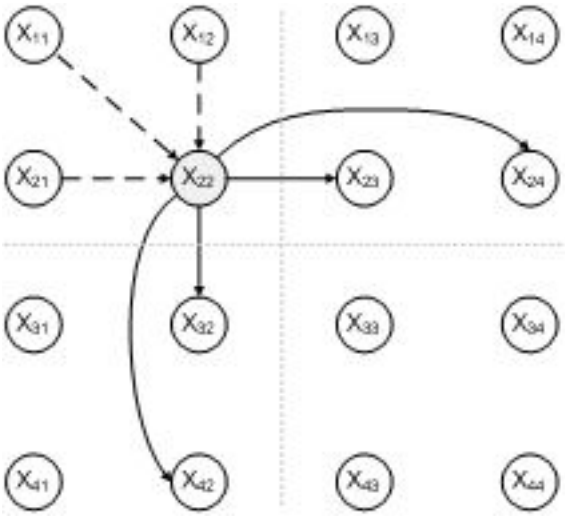
  $\neq(x_{11},x_{12},x_{13}\ldots,x_{19}),\ \neq(x_{21},x_{22},x_{23}\ldots,x_{29}),\ \ldots,$
  $\neq(x_{91},x_{92},x_{93}\ldots,x_{99})$
- Todos los pares de variables en cada submatriz:

  $\neq(x_{11},x_{12},x_{13},x_{21},x_{22},x_{12},x_{31},x_{32},x_{33}),\ldots$ idem para las restantes submatrices.

De este modo, se obtiene un sistema de NxN agentes, uno por cada celda del tablero. El modo en que estos se comportan es el mismo que para el problema de las n-reinas. Los vecinos de un agente serán aquellos que se encuentran en su misma fila, más los que se encuentran en su misma columna, y más los que se encuentran en su misma submatriz.

El grafo generado por el tablero, debido a su gran cantidad de restricciones, y por ende, de enlaces, torna complejo el modelado del mismo. En la Figura 4.10 se presenta el modelo para la aplicación del backtracking asíncrono, donde las flechas punteadas representan los agentes que le envían valores al agente x_{22} para que éste los evalúe, y las flechas salientes indican los agentes a los cuales x_{22} les envía su propio valor, es decir, sus evaluadores.

Figura 4. 10 Modelo DCSP: Sudoku de orden 4



4.2.3 Cuadrado Mágico

Una formulación estándar para este modelo, siendo n x n el tamaño de la matriz y m determinado por la ecuación (3.1), sería la siguiente:

Variables:

$$x_{ij} \quad \forall i,j \in \{1,2,...,n\}$$

Dominios:

$$x_{ij} \in \{1,2,...,n^2\}, \quad \forall i,j \in \{1,2,...,n\}$$

Restricciones:

$$\sum_{j=1}^{n} x_{ij} = m, \quad \forall i \in \{1,2,...,n\}$$

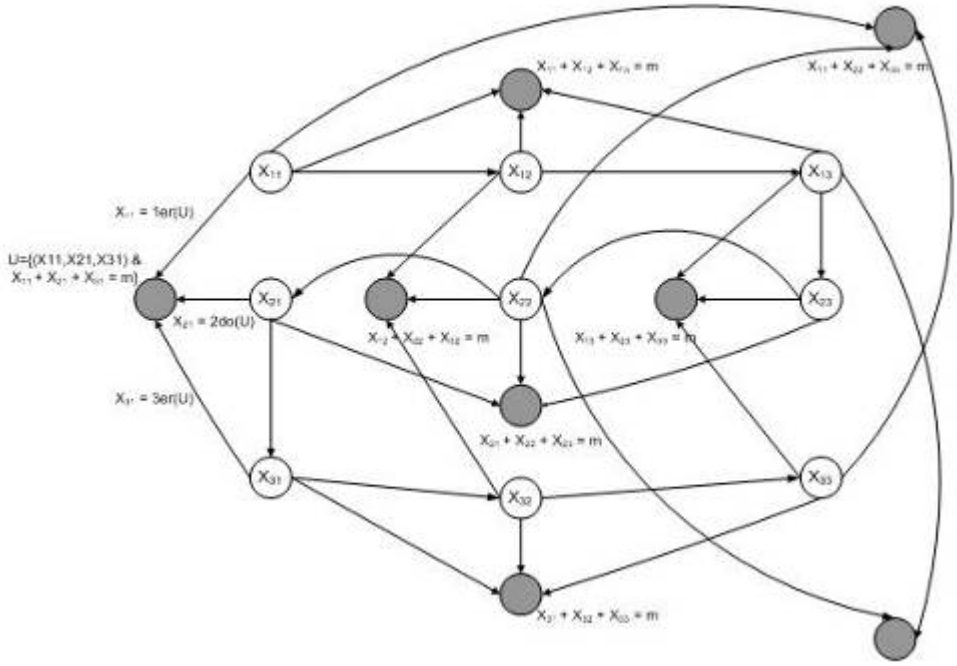
$$\sum_{i=1}^{n} x_{ij} = m, \quad \forall j \in \{1,2,...,n\}$$

$$\sum_{i=1}^{n} x_{ii} = m$$

$$\sum_{i=1}^{n} x_{i(n+1-i)} = m$$

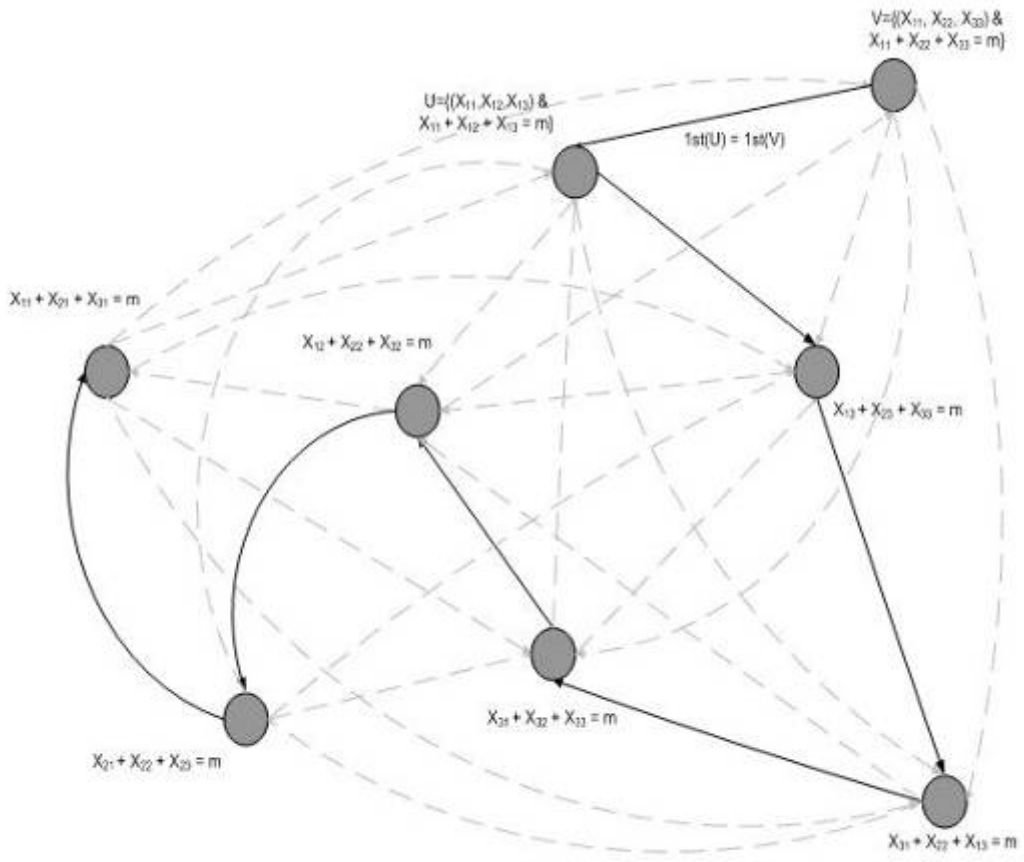
Sin embargo, el modelado del problema del Cuadrado Mágico ha debido ser binarizado para poder ser modelado como DCSP. En las Figuras 4.11 y 4.12 se muestran los dos modelos generados en este proceso (descrito en el apartado 3.5).

Figura 4. 11 Modelo DCSP: Cuadrado Mágico (Método de la Variable Oculta)



En la Figura 4.11 se despliega el modelo obtenido a través del método de la Variable Oculta, éstas variables han sido representadas por los círculos en gris. Se muestra la tupla U que maneja una de las variables ocultas y las restricciones que tiene con los tres agentes que le envían valores, en este caso, X_{11} debe ser igual al primer elemento de la tupla de U, X_{21} debe ser igual al segundo elemento de la tupla de U y X_{31} debe ser igual al tercer elemento de la tupla de U.

Figura 4. 12 Modelo DCSP: Cuadrado Mágico (Método de Representación Dual)



En la Figura 4.12 se despliega el modelo obtenido a partir del método de Representación Dual. Se muestran las tuplas que manejan las variables U y V, la restricción en este caso consiste en que el tercer elemento de U debe ser igual al tercer elemento de V.

Los enlaces en línea punteada representan todas las restricciones que posee el grafo, los enlaces marcados representan el intercambio de mensajes del agente "V".

A pesar de que el segundo modelo contiene menos agentes y su representación es similar al de las N-Reinas, su representación se vuelve compleja, puesto que se amplía el dominio de las variables (ya que cada una contiene una tupla de n elementos), **debido a esto se ha seleccionado el modelo obtenido a partir del método de la variable oculta**, pues su representación sigue siendo sencilla y debería generar una menor cantidad de nogoods debido a varias de las variables sólo manejan un único valor, siendo las variables ocultas sólo encargadas de evaluar valores y no de proponerlos.

Capítulo 5

Implementación del Sistema

5.1 Componentes

Se ha implementado a la fecha una primera aproximación a la resolución del problema de las N-Reinas. Esto ha sido realizado con la ayuda de la plataforma JADEX, y se han creado los siguientes elementos:

Agentes:

• **AgenteDCSP.agent.xml:** agente base del sistema. Este archivo estructura las creencias, objetivos y planes del agente, el cual es instanciado tantas veces como el problema en particular lo requiera, así por ejemplo, para resolver el problema de 4-Reinas, se necesitarán cuatro agentesDCSP, uno por cada reina, en tanto que para un Sudoku de orden 4 se necesitarán 16 agentesDCSP, uno por cada celda del tablero.

• **Manager.agent.xml:** agente que inicializa el sistema, creando a los agentes DCSP, y maneja el tablero, mostrando la solución una vez que ésta ha sido encontrada.

5.1.1 Agente DCSP: Descripción

Creencias

• **Prioridad:** la prioridad del agente.

• **Asignación:** el valor actual del agente + su dominio (min, max).

• **Prioridad_menor:** el agente emisor con menor prioridad al cual se enviarán los nogood.

• **Evaluador:** el listado de agentes “evaluadores” a los cuales el agente envía su valor.

• **Visión:** la visión del agente, es decir, los valores de los agentes emisores.

• **Visión_ok:** el estado de la visión del agente. Verdadero si es consistente (si se cumplen todas las restricciones) o Falso en caso contrario.

• **Vision_nogood:** si no es posible encontrar un valor con la visión actual.

• **Mal_valor:** listado de las visiones que no deben ser generadas, dado que no conducen a posibles resultados.

• **Restricción:** listado de los agentes de menor prioridad con los cuales posee restricciones.

• **Enviado:** último valor enviado por el agente.

Objetivos

• **Obtener_visionOK:** Cuando el agente recibe un valor o un "nogood", éste actualiza sus creencias, si producto de ello resulta que el estado del agente es inconsistente, entonces se activa el objetivo Obtener_visionOK, el cual a su vez, activará los planes que permitan conseguir una visión consistente, esto es encontrar un nuevo valor que cumpla las restricciones de sus vecinos, y en caso de no encontrarlo, enviar un nogood a su agente emisor de menor prioridad.

• Debido a la simetría de los problemas seleccionados no fue necesario incluir el objetivo "Evaluador" incluído en el Modelo de Objetivos, puesto que no se requiere la adición de nuevos enlaces.

Planes:

• **IniciarPlan.java:** Plan que inicializa a cada agente, configurando las creencias que le permitirán saber quienes son sus agentes evaluadores, y con cuáles agentes tiene restricciones. Cada agente, al ser creado, envía un mensaje con su valor a sus agentes evaluadores, dando inicio a la búsqueda.

• **EvaluarValorPlan.java:** Plan que se activa al recibir un mensaje con el valor de un agente emisor, evalúa si éste cumple con las restricciones y actualiza la base de creencias.

• **EvaluarNogoodPlan.java:** Plan que se activa al recibir un mensaje “nogood”, que contiene la visión que lo generó, evalúa si la visión recibida corresponde con su visión actual y actualiza la base de creencia agregando esta visión a sus “malos valores”.

• **BuscarPlan.java:** Plan que busca un valor que cumpla con las restricciones que maneja y que no se encuentre en la visión, este plan es activado al detectar que la visión actual es inconsistente. Si el agente es capaz de encontrar un valor, entonces lo envía a sus evaluadores, en caso contrario, envía un “nogood” a su emisor de menor prioridad.

5.2 Agente DCSP: Arquitectura

La Figura 5.1 se muestra al agente DCSP, en base a la arquitectura JADEX utilizada en la implementación del sistema.

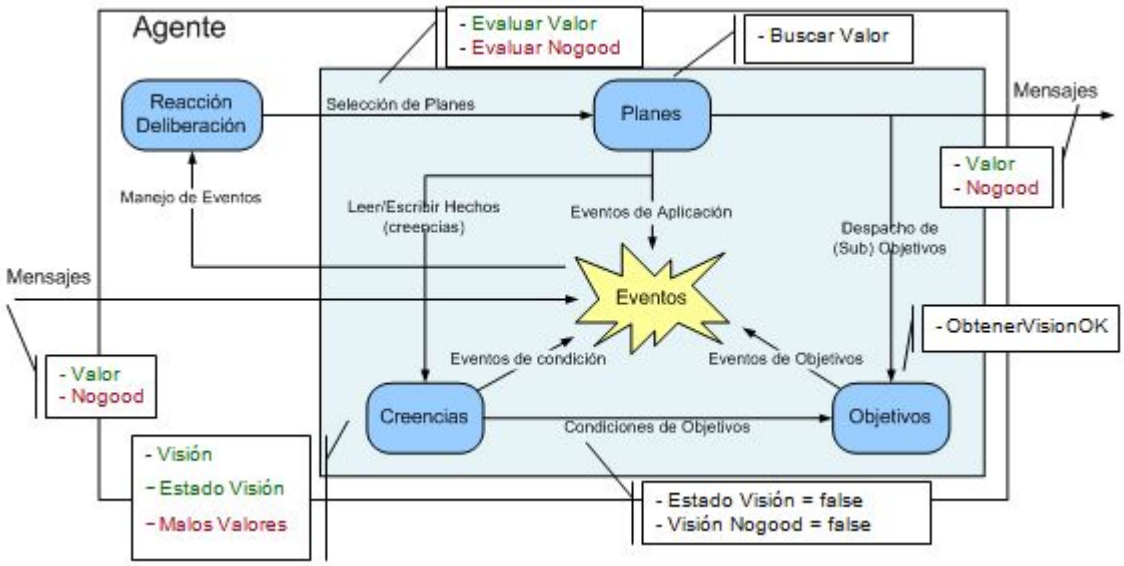
Los mensajes son recibidos y colocados en una cola de Eventos, y cuyo contenido (un valor asignado o un nogood) es utilizado para la actualización de la base de creencias.

El agente delibera qué plan se debe llevar a cabo dependiendo del tipo de mensaje recibido, si se trata de un valor entonces activará el plan Evaluar Valor, en caso contrario, de ser un nogood, el plan seleccionado es Evaluar Nogood.

El plan activado lee el conjunto de creencias y analiza el estado en que se encuentra, actualizando el estado del agente si se trata de un valor, y la lista de malos valores si es un noood.

Cuando se detecta en las creencias que el estado de la visión es inconsistente, pero no se puede determinar si se trata de un nogood, se cumplen las condiciones que activan el objetivo ObtenerVisionOk, el cual, nuevamente tras el proceso de deliberación, esta vez va a activar el plan Buscar Valor. Si se encuentra un valor, actualiza su base de creencias con el nuevo valor del agente, y lo envía a sus evaluadores, en caso contrario envía un nogood a su vecino de menor prioridad.

Figura 5. 1 Arquitectura del agente DCSP


Agente
- Evaluar Valor
- Evaluar Nogood
- Buscar Valor
Reacción Deliberación
Selección de Planes
Planes
Mensajes
- Valor
- Nogood
Manejo de Eventos
Leer/Escribir Hechos (creencias)
Eventos de Aplicación
Despacho de (Sub) Objetivos
Mensajes
Eventos
- ObtenerVisionOK
- Valor
- Nogood
Eventos de condición
Eventos de Objetivos
Creencias
Condiciones de Objetivos
Objetivos
- Visión
- Estado Visión
- Malos Valores
- Estado Visión = false
- Visión Nogood = false

Capítulo 6

Resultados Obtenidos

6.1 Parámetros de Prueba

Con el fin de probar el rendimiento del sistema se han seleccionado ciertos parámetros a considerar en la generación de soluciones, y poder así, realizar comparaciones con otras técnicas de resolución de CSP's.

• Tiempo: El tiempo que demora el sistema en encontrar la primera solución. Para ello, se tomará en cuenta desde el momento en que se da inicio a la búsqueda con el mensaje enviado por el primer agente hasta que es encontrada la primera solución. Se espera, a futuro, obtener además el tiempo que demora en encontrar todas las soluciones al problema.

• Número de Backtracks: El número de backtracks en el algoritmo ABT corresponde al número de "nogoods" que se envían en la ejecución del sistema, en otras palabras, el número de veces que se "vuelve atrás" en búsqueda de la solución. Es interesante comprobar su variación con respecto a otras técnicas de resolución similares, puesto que podrían influir en el rendimiento del sistema.

• Número de Mensajes: Se contará el número de mensajes que se envían los agentes entre sí, este parámetro puede ser comparado con otras técnicas de resolución de DCSP's.

Para medir el rendimiento del sistema, se ha tomado registros del rendimiento que actualmente ofrece el sistema, y se ha comparado con el algoritmo Backtracking, implementado también en Java.

El sistema ha generado las siguientes estadísticas promedio de tiempo, número de backtracks y número de mensajes enviados:

Número de Reinas	Tiempo (ms.)	# Backtracks	# Mensajes

4	637	8	20
6	1.106	66	210
8	1.692	200	779
10	1.856	211	1.085
12	2.703	312	1.944
14	9.582	1.645	8.907
16	7.844	1.160	7.339
18	17.563	7.433	22.448
20	43.851	5.051	39.516
22	133.234	25.392	98.547
24	270.219	75.881	234.781
25	5.344	25	3.423

Tabla 6. 1 Rendimiento N-Reinas (ABT intencional)

Orden (N)	Tiempo (ms.)	# Backtracks	# Mensajes
4	637	401	1.838

Tabla 6. 2 Rendimiento Sudoku (ABT intencional)

Los resultados de la ejecución del Sudoku se muestran sólo para N=4, dado que para valores de N mayores (que pueden ser números cuadrados: 9, 16, 25, etc.) no se consiguieron resultados en tiempos razonables.

La implementación de Backtracking centralizada, utilizada para la comparación de resultados, fue seleccionada por su similitud en la forma de elección de valores, de manera de poder contrastar las diferencias generadas principalmente debido a la distribución del problema en agentes.

La comparación con los resultados obtenidos del algoritmo "Backtracking" (BT) centralizado se ha realizado en función del tiempo, con un timeout de 5 minutos (300.000 ms.), y se puede observar el resumen en la tabla 6.2:

Número de Reinas	ABT Intencional Tiempo (ms.)	BT Centralizado Tiempo (ms,)
4	637	4
5	578	4
6	1.106	13
7	785	3
8	1.692	109
9	1.012	32
10	1.856	459
11	1.076	188
12	2.703	7.868
13	1.336	1.943
14	9.582	18.529
15	6.332	14.911

16	7.844	Timeout
17	1.887	Timeout
18	17.563	Timeout
19	4.813	Timeout
20	43.851	Timeout

Tabla 6. 3 Comparación para N-Reinas de BT centralizado v/s ABT Intencional

Orden (N)	ABT Intencional Tiempo (ms.)	BT Centralizado Tiempo (ms,)
4	637	82

Tabla 6. 4 Comparación para Sudoku de BT centralizado v/s ABT Intencional

De la Tabla 6.2 se puede notar que los algoritmos se comportan de manera similar, es decir, el tiempo varía dependiendo del número de reinas que implica el problema, y de la disposición que tienen las reinas en el tablero solución. Sin embargo, en el sistema implementado es menos eficiente cuando se trabaja con pocas reinas, obteniendo mejores tiempos de búsqueda en comparación al algoritmo backtracking implementado, a medida que aumenta el número de reinas. Esta situación se refleja en el Gráfico 6.1, donde se observa que desde N = 12, los tiempos de búsqueda disminuyen considerablemente.

Figura 6. 1 Comparación BT centralizado v/s ABT Intencional (N-Reinas)

El intercambio de mensajes implica un mayor tiempo de búsqueda en las primeras pruebas, dado que el problema se resuelve con pocas iteraciones, un algoritmo centralizado es capaz de obtener una solución sin el retardo que los mensajes provocan. Sin embargo, al aumentar el número de reinas, un algoritmo centralizado tiene muchas más variables que manejar y por ende, demora más en encontrar una solución que un algoritmo distribuido en el cual varios agentes colaboran en búsqueda de la solución, siendo el tiempo que se utiliza en el intercambio de mensajes, en este caso, menos relevante.

Para el problema de las N-Reinas se lograron resultados hasta N=25, en tanto que para el problema del Sudoku, sólo se tuvo éxito para el caso N=4, se observa entonces que el algoritmo se comporta de forma satisfactoria y en mejores tiempos en cierto rango que para efectos de este trabajo de título se encuentra entre 12 y 25, para valores de N muy pequeños el sistema es más lento que un algoritmo centralizado, en tanto que para N grandes la búsqueda toma tiempos extremadamente altos.

Capítulo 7
Conclusiones y trabajos futuros

El ámbito de los agentes intencionales es un tema relativamente nuevo, ha tomado mayor importancia en los últimos años y aún existen muchas propuestas en cuánto a la formalización de los mismos. Sin embargo, existen algunas herramientas y metodologías que hacen posible el desarrollo de un sistema de este tipo y poder seguir investigando en este tema, ejemplos de ello son la metodología de desarrollo orientada a agentes utilizada en la primera parte del trabajo de título: AAII, y la plataforma JADE y en particular su extensión JADEX, han permitido crear los agentes con actitudes mentales BDI.

Por otro lado, existen estudios que hablan de la aplicación de agentes en problemas combinatoriales difíciles, principalmente modelados a modo de DCSP, pero pocos que incorporan la arquitectura BDI en ellos. En este caso, se ha utilizado un algoritmo distribuido denominado como Backtracking Asíncrono, una de las primeras propuestas orientadas en este camino. Así, el sistema mantiene una base de creencias actualizada de su entorno en función de la información que recibe del mismo, la que utiliza para la activación y ejecución de los objetivos y planes que conducirán a cada agente a la resolución parcializada del sistema, y al conjunto de agentes a la consecución de la solución final.

Se ha conseguido implementar entonces, un sistema multiagentes intencional que es capaz de resolver problemas combinatoriales difíciles como son N-Reinas y Sudoku, cumpliendo así con los objetivos del trabajo de título. Se han realizado evaluaciones sobre la ejecución del sistema y se obtuvieron resultados satisfactorios para ello cuando el número de agentes no es muy pequeño ni muy grande, esto es, para efectos de este trabajo de título, cuando el número de agentes se encuentra en este caso entre 12 y 25, es por tanto en este caso, donde se aprovecha de mejor manera la distribución de la tarea en agentes.

La implementación llevada a cabo corresponde a la versión inicialmente propuesta en [Yokoo et. al, 1998] del algoritmo Backtracking Asíncrono, con la inclusión de conceptos

de intencionalidad. Sin embargo, es posible introducir mejoras en el algoritmo que permitan perfeccionar el rendimiento del mismo, y ampliar el rango de agentes para los cuales se obtienen tiempos satisfactorios. Por lo demás, existen en la literatura, algunas variaciones a este algoritmo que pueden ser estudiadas y posteriormente aplicadas.

La implementación de problemas de satisfacción de restricciones no binarios es otro desafío, se ha modelado en este trabajo de título uno de los problemas que cumple con estas características, y se ha propuesto, como técnica de binarización [Rossi et al, 1990] a utilizar, el método de la Variable Oculta frente al de Representación Dual, debido a que en el grafo generado cada agente maneja una sola variable, simplificando el proceso de la búsqueda de un valor, al buscar dentro de un dominio más reducido.

Se han generado los resultados para la comparación de los mismos con otras técnicas. Como trabajo futuro, se propone la realización de comparaciones con resultados de otros sistemas ya sean centralizados o distribuidos.

Capítulo 8
Referencias

[Bacchus y van Beek, 1998] F. Bacchus, P. van Beek, *On the conversion between Non-Binary and Binary Constraint Satisfaction Problems*, In Proc. National Conference on Artificial Intelligence (AAAI-98), Madison, Wisconsin, 1998.

[Barber y Salido, 2008] F. Barber, M. A. Salido, *Inteligencia Artificial: Técnicas, métodos y aplicaciones*, (Ed. McGraw-Hill), pp:385-432, 2008.

[Bratman *et al.*, 1988] Michael E. Bratman, David J. Israel y Martha E. Pollack, *Plans and Resource-Bounded Practical Reasoning*, Computational Intelligence, 4(4):349-355, 1988.

[Bratman, 1987] Michael E. Bratman, *Intention, Plans, and Practical Reasoning*. Harvard University Press, Cambridge MA., USA. 1987.

[Cohen y Levesque, 1990] Philip R. Cohen y Hector J. Levesque, *Intention is Choice with Commitment*, Artificial Intelligence, 42(3):213–261, 1990.

[Dennett, 1987] Daniel C. Dennett, *The Intentional Stance*, MIT Press, Cambridge, MA., USA, 1987.

[Garamendi, 2004] Juan Garamendi. *Agentes Inteligentes: JADE.* Programa de Doctorado: Informática y Modelización Matemática Universidad Rey Juan Carlos. Abril 2004

[Garcia, 2007] Carlos García-Montoro*, Análisis y clasificación de lenguajes de programación orientados a agentes*. Informe Técnico. Universidad Politécnica de Valencia. 2007

[Georgeff y Lansky, 1987] M. Georgeff y A. L. Lansky, *Reactive reasoning and planning*. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pages 677-682, Seattle, WA. 1987

[Guerequeta y Vallecillo, 1999] Rosa Guerequeta, Antonio Vallecillo. *Técnicas de Diseño de Algoritmos*. Servicio de Publicaciones de la Universidad de Málaga, 1999.

[Guerra, 2006] Alejandro Guerra-Hernández, *Agentes Intencionales*, Universidad Veracruzana, México, Abril 2006.

[Hoos y Stützle, 2004] Holger H. Hoos y Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[Kinny *et al.*, 1996] David Kinny, Michael P. Georgeff, y Anand S. Rao, *A methodology and modelling tecnique for system of BDI agents*, Proc. of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Vol. 1038): 56-71, Springer, 1996.

[Le Dinh y Seaw, 2007] Bao Chau Le Dinh, Kiam Tian Seow: *Unifying Distributed Constraint Algorithms in a BDI Negotiation Framework*. AAMAS 2007.

[Mascardi et al., 2005] Viviana Mascardi, Daniela Demergasso, Davide Ancona. *Languages for Programming BDI-style Agents: an Overview.* WOA 2005: 9-15. 2005

[Nwana, 1996] Hyacinth S. Nwana, H. *Software Agents: An Overview*. Knowledge Engineering Review. Cambridge University Press. v.3, p.1-40. 1996.

[Padgham y Winikoff, 2002] Lin Padgham and Michael Winikoff, *Prometheus: A Methodology for Developing Intelligent Agents*. AAMAS'02 Workshop on Agent Oriented Software and Engineering (AOSE-2002), 135-145, 2002.

[Plesa y Logrippo, 2006] Romelia Plesa, Luigi Logrippo. *Enhanced Communication Services through Context Integration*. 2006

[Pokahr et al, 2005] Alexander Pokahr, Lars Braubach y Winfried Lamersdorf. *JADEX: A BDI Reasoning Engine*. University of Hamburg. 2005

[Rao y Georgeff, 1991] Anand S. Rao y Michael P. Georgeff, *Modeling Rational Agents within a BDI Architecture*, Technical Note 14, Australian Artificial Intelligence Institute, Carlton, Victoria, February 1991.

[Rao y Georgeff, 1993] Anand S. Rao y Michael P. Georgeff, *Decision Procedures for Propositional Belief – Desire – Intentions Logic*, Technical Note 44, Australian Artificial Intelligence Institute, September 1993.

[Rao y Georgeff, 1995] Anand S. Rao y Michael P. Georgeff, *BDI Agents: From Theory to Practice*, Technical Note 56, Australian Artificial Intelligence Institute, Carlton, Victoria, April 1995.

[Rao y Georgeff, 1995a] Anand S. Rao y Michael P. Georgeff, *Formal Models and Decision Procedures for Multi-Agents Systems*, Technical Note 61, Australian Artificial Intelligence Institute, Carlton, Victoria, June 1995.

[Rossi et al, 1990] F. Rossi, C. Petrie, y V. Dhar. *On the equivalence of constraint satisfaction problems*. In Proc. 9th European Conf. Artificial Intelligence ECAI'90, pages 550--556, 1990.

[Rossi et. al, 2006] F. Rossi, P. Van Beek, T. Walsh (eds.), *Handbook of Constraint Programming*. Elsevier, Elsevier, 2006.

[Seow et al, 2007] Kiam Tian Seow, Kwang Mong Sim y Yuan Chia Kwek, *Collaborative Assignment Using BDI Multiagent Negotiation*, Technical Report No. TR-IIS-06-013. Institute of Information Science, Academia Sinica. Taipei, Taiwan, ROC. 2007

[Wooldridge y Jennings, 1995] Michael Wooldridge y Nicholas R. Jennings, *Intelligent Agents: Theory and Practice*, The Knowledge Engineering Review, vol. 10(2) pp. 115-152, 1995.

[Wooldridge, 2000] Michael Wooldridge, *Reasoning about Rational Agents*, MIT Press, Cambridge, MA., USA, 2000.

[Yokoo et. al, 1998] Makoto Yokoo, Edmund H. Durfee, Toru Ishida y Kazuhiro Kuwabara, *The Distributed Constraint Satisfaction Problem: Formalization and Algorithms*, IEEE Transactions on Knowledge and Data Engineering,Vol. 10, No. 5, pp. 673-685,1998.

APÉNDICE A – CÓDIGO FUENTE

Texto completo de Apéndice A, en :