Aprendizaje Incremental en Sistemas Multi-Agente BDI

Gustavo Ortiz Hernández

Director de Tesis:

Alejandro Guerra Hernández

Tesis presentada para obtener el grado de

Maestro en Inteligencia Artificial

Departamento de Inteligencia Artificial
Fac. de Física e Inteligencia Artificial
Universidad Veracruzana
México

Septiembre 2007

Dedicatoria

Con el más profundo amor para: Gustavo, Lidia y Citlalic...

Aprendizaje Incremental en Sistemas Multi-Agente BDI

Gustavo Ortiz Hernández

Presentada para obtener el grado de Maestro en Inteligencia Artificial

Septiembre 2007

APROBADA:

Dr. Alejandro Guerra Hernández

Dr. Nicandro Cruz Ramírez

Dra. Cora Beatriz Excelente Toledo

Agradecimientos

Al Dr. Alejandro Guerra Hernández, por su impecable dirección, por transmitir su gusto y pasión a la investigación... además del innumerable material de apoyo brindado, sin el cual no hubiera sido posible esta tesis.

A la Dra. Cora Beatriz Excelente Toledo, por su profesionalismo y acertados comentarios: permitieron un trabajo de mayor calidad.

A el Dr. Nicandro Cruz Ramírez, por sus puntuales observaciones: con las que se logró una mayor claridad.

A la M.C. Viginia Angélica García Vega y al Dr. José Negrete Martínez, por su ejemplo, enseñanzas de vida, su paciencia, y compartirnos en cada momento sus experiencias de una manera amena y admirable.

Al Dr. Fernando Martín Montes González, el Dr. Héctor Gabriel Acosta Mesa, el Dr. Guillermo de Jesús Hoyos Rivera, el M.C. Rubén de la Mora Basañez, el Dr. Miguel Angel Jiménez Montaño, el Dr. Antonio Marín Hernández y el Dr. Manuel Martínez Morales, por sus excepcionales clases, disponibilidad para atender nuestro mar de dudas y a todo el Depto. de Inteligencia Artificial, por tener siempre las puertas abiertas, y la encantadora humildad de todos sus miembros.

A Wulfrano, Carlos, Rosy, Daniel y Karina, mis compañeros de desvelos... de batalla.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Los agentes racionales pueden ser caracterizados en términos de sus creencias, deseos e intenciones, así como de las relaciones existentes entre ellos; el modelo *Belief-Desire-Intention* (*BDI*) ha surgido como el paradigma dominante dentro del campo de investigación y desarrollo de agentes [69,78] principalmente por sus bases filosóficas sólidas sobre intencionalidad y razonamiento práctico, la elegante lógica de su semántica abstracta y algunas implementaciones exitosas [35].

Sin embargo, el modelo *BDI* hasta ahora presenta una limitación bien conocida: la falta de un mecanismo formal de aprendizaje [35]. Creemos fuertemente que el aprendizaje, sobre todo aquel basado en la experiencia (incremental), es un aspecto que debe ser seriamente considerado en el desarrollo de agentes inteligentes. Parece ser que la clave de la inteligencia en el ser humano recae en gran medida sobre su capacidad para inferir relaciones *efecto-causa* [1] [31]. El aprendizaje incremental da a los sistemas robustez y adaptabilidad, provocando comportamientos que exhiben mayor grado de inteligencia. Por ejemplo, un agente que implemente aprendizaje por inducción de manera incremental puede modificar su estado de cognición, y con ello su comportamiento, para permanecer consistente con su ambiente en tiempo real.

El tema de aprendizaje ha sido abordado por un número considerable de investigadores, y es un campo extenso dentro de la Inteligencia Artificial (*IA*). Dentro del *Aprendizaje*

[1] Conectamos la idea de *efecto-causa* con la relación de función directa y función inversa propuesta por el Paradigma de Sapiencia Computacional [59]. Por ejemplo, en el área de la lógica, la entrada de la función directa son los hechos conocidos acerca de una situación, el programa realiza las acciones convenientes y el logro de una meta es producido como salida; mientras que la función inversa consiste en inferir los hechos que permitieron establecer una meta a partir de la experiencia

Automático (AA) se han desarrollado diversas técnicas de aprendizaje denominadas de *atributo-valor*. Sin embargo, es necesario considerar un gran número de detalles antes de pensar en la incorporación de dichas técnicas de aprendizaje dentro del modelo BDI. Por ejemplo, las técnicas de AA son computacionalmente eficientes, pero no es posible emplearlas bajo el modelo de agentes BDI; principalmente porque estos últimos, en su mayoría, se encuentran expresados en lógica de primer orden, $\mathcal{FOL}$ (*First Order Logic*), o en su defecto, extenciones de la misma. Tal es el caso de $\mathcal{LORA}$ [2] (*Logic-of-Rational-Agents*) [78].

Lo anterior nos lleva a recurrir, en primera instancia a la Programación Lógica Inductiva (PLI), para contender con aprendizaje en un enfoque de agentes. La PLI proporciona una mayor expresividad, pero resulta computacionalmente pesada, ya que difícilmente alcanza los estándares manejados por la comunidad de minería de datos para aprendizaje. La ineficiencia de la PLI recae principalmente en la suposición de que varios ejemplos se encuentran estrechamente correlacionados, y por lo tanto no pueden ser manejados de forma independiente. Existe un método llamado aprendizaje por interpretaciones que logra extender la expresividad del aprendizaje proposicional al mismo tiempo que mantiene su eficiencia [6].

Otras dos de las problemáticas bien conocidas en métodos como los señalados anteriormente son: i) requieren de la presencia de un mecanismo de arbitraje entre dos fases: ejecución normal y aprendizaje. Este mecanismo es generalmente presentado como un estado condicional que determina si es necesario aprender o no; por ejemplo, cuando un plan determinado falla (condición), el mecanismo de aprendizaje es ejecutado [35]. ii) también es necesario contar con un número suficiente de ejemplos para aprender, rompiendo así la forma más natural de aprendizaje, caracterizado como aprendizaje por experiencia o *aprendizaje incremental*.

En este trabajo adoptamos un protocolo sólido [3] de aprendizaje incremental multiagente llamado *SMILE* (*Sound Multi-agent Incremental LEarning*) [11]. En términos generales el protocolo de *SMILE* supone que cada agente puede aprender de manera incremental en función de la información que va obteniendo. Cada agente debe contar con un mecanismo de revisión de creencias que mantenga la consistencia en toda actualización

[2] $\mathcal{LORA}$ es una lógica multimodal basada en $\mathcal{FOL}$ desarrollada para agentes propuesta por originalmente por Michael Wooldridge {Wooldridge-Reasoning-About-Rational

[3] De la palabra en inglés *sound*

realizada a la base de creencias. El aprendizaje surge mientras se llevan a cabo una serie de interacciones entre agentes del Sistema Multi-Agente, *SMA*, (con ciertas creencias en común) y el propósito de mantener la consistencia global.

El protocolo descrito por *SMILE* permite el aprendizaje incremental de forma distribuida, evitando así, la necesidad de incorporar un mecanismo de arbitraje entre la función de aprendizaje y la ejecución normal del agente. Tampoco se impone la limitante de tener un mínimo número de ejemplos para garantizar aprendizaje, debido a que el aprendizaje es visto como la restauración de la consistencia del conocimiento de un agente con relación a su ambiente.

En este trabajo buscamos definir *SMILE* en términos del modelo *BDI*; por medio de extender la semántica operacional de *AgentSpeak(L)* [68], logrando así presentar una aproximación formal para aprendizaje incremental. Posteriormente implementamos dicho protocolo utilizando Jason, intérprete de *AgentSpeak(L)* [61].

A continuación definimos la estructura de esta tesis. En el capítulo dos introduciremos la noción de agente, y expondremos la problemática en su definición, así como las diferentes categorizaciones que se han hecho al respecto. Presentaremos una formalización abstracta para definir de una manera más precisa algunos tipos de agentes que han sido implementados con éxito. Posteriormente, en el tercer capítulo adoptamos el concepto de noción fuerte de agencia y profundizamos en el modelo *BDI*, detallando el aspecto filosófico e introduciendo una arquitectura de software. Dejaremos de lado las lógicas multimodales para enfocarnos en la arquitectura *BDI* como tal, y nos inclinaremos por la nueva corriente en la cual una representación de arquitectura se logra mediante la definición de la semántica operacional de un lenguaje formal, como es el caso de *AgentSpeak(L)*[4], que representa tanto la arquitectura de implementación, como el aspecto formal. Luego introducimos *Jason*, intérprete de una versión extendida de *AgentSpeak(L)*, en la cuarta parte de este documento.

En el capítulo cinco hablaremos de aprendizaje y árboles de decisión, tanto en una representación proposicional (*ID3*), como en primer orden (*TILDE*)[5]. Explicaremos el enfoque de aprendizaje incremental adoptado y expondremos los argumentos para llevar a cabo la metodología. Esta última será dividida, en el sexto capítulo, en tres partes: i)

[4] *AgentSpeak(L)* cambia la aproximación a la ingeniería *BDI*. Éste se presenta como un lenguaje formal orientado a agentes y no como una arquitectura.

[5] Se les llama árboles lógicos de decisión a los que presentan una representación de primer orden.

la implementación de nuestro antecedente principal [6]; ii) el escalamiento del protocolo *SMILE* [11] a $\mathcal{FOL}$, mediante la definición del mismo en términos de planes de AgentSpeak(L); y iii) la extensión a la semántica operacional de *AgentSpeak(L), para presentar una aproximación formal del modelo de aprendizaje intencional manejado en esta tesis.*

Finalmente, logramos una versión incremental del protocolo presentado originalmente en [35] basada en un escalamiento de *SMILE* para usarlo bajo el modelo *BDI*: proponemos una extensión a la semántica operacional de *AgentSpeak(L)*. Y discutiremos la implementación usando un algoritmo basado en *TILDE* para aprendizaje incremental que promete ser *localmente eficiente*[7], llamado *ILDT* (*Incremental Induction of Logical Decision Trees*) [41].

1.1. Antecedentes

El modelo *Belief-Desire-Intention* ha surgido como el paradigma dominante dentro del campo de investigación y desarrollo de agentes [69,78] principalmente por sus bases filosóficas sólidas sobre intencionalidad y razonamiento práctico, su elegante lógica semántica abstracta y algunas implementaciones exitosas [35].

El Aprendizaje Automático es el campo central de la *IA* que estudia los aspectos computacionales del aprendizaje. Aquí nos referimos particularmente a la interacción entre *AA* y el concepto de agencia, incluyendo Sistemas Multi-Agente (*SMA*) e Inteligencia Artificial Distribuida. Esta perspectiva es comúnmente conocida como aprendizaje *SMA*, y es caracterizada como la intersección entre Sistemas Mult-Agente y Aprendizaje Automático. Existe un interés creciente en la investigación de métodos de aprendizaje *SMA*; el comportamiento flexible remarcado por el concepto de agente en la *IA* y la inherente complejidad de los sistemas multi-agentes son dos de las principales razones.

Stuart Russell y Peter Norvig [71] han propuesto una arquitectura genérica de agente para aprendizaje compuesta de cuatro elementos:

[6] La tesis doctoral de Alejandro Guerra-Henández [35], una implementación de aprendizaje en el *nivel 1* y *nivel 2* programada en *LISP* basada en *IRMA* [13], en la cual propone emplear el fallo del los planes como evento de disparo para lanzar el mecanismo de aprendizaje, y justifica el aprendizaje sobre el contexto de los planes, dirigido hacia el *qué hacer*

[7] Como es descrito en [11], el mecanismo de aprendizaje es *localmente eficiente* si para todo caso logra restablecer la consistencia del agente con su medio, es decir, aprende una hipótesis consistente. Existe una *sma-consistencia*, para definir una consistencia global en el sistema.

- El elemento a ser mejorado, el cual describe en este caso a un agente que no incorpora aprendizaje.
- Un elemento que juega el rol de crítico, encargado de retroalimentar el elemento de aprendizaje, i.e., señalar qué tan bien el agente está siendo mejorado.
- Generador de problemas, responsable de sugerir acciones que conducirán a nuevas experiencias con información valiosa para el aprendizaje.
- Elemento de aprendizaje, que tiene el papel de hacer las mejoras en el agente.

Guiándose por la arquitectura genérica expuesta anteriormente, Alejandro Guerra-Hernández propone una arquitectura *BDI* extendida para incluir mecanismos de aprendizaje [35], logrando que los agentes definidos por esta arquitectura sean capaces de reconsiderar después de su experiencia (y posiblemente después de la experiencia de otros agentes en el sistema) la *razones prácticas* que tienen para adoptar un plan como parte de sus intenciones, i.e., las razones que se tienen para comprometerse en actuar de cierta forma y no de otra.

El uso de inducción de árboles lógicos de decisión, mediante TILDE (*Top-down Induction Logical Decision Trees*) [5], como método de aprendizaje empleado en [35], fue considerado debido a la representación disyuntiva de conjunciones en lógica de primer orden, que determina el contexto de un plan; elemento que se consideró como objetivo para las tareas de aprendizaje *BDI*. Lo anterior hizo posible expresar el problema de aprendizaje en términos de una arquitectura *BDI* de la siguiente manera: el aprendizaje comienza cuando la ejecución de un plan instanciado falla, los ejemplos de entrenamiento son colectados después de la experiencia de cada agente, y éstos son caracterizados como el conjunto de creencias que hicieron a un plan relevante o aplicable; los ejemplos son etiquetados después con 'éxito' o 'fallo' de acuerdo a la ejecución del plan correspondiente, y el proceso de aprendizaje en si mismo está representado en términos de planes y eventos. Aquí, *TILDE* no permite lograr un verdadero aprendizaje incremental: por lo general éste requiere de un mínimo número de ejemplos (es necesaria información suficiente para discriminar empates de hipótesis generadas internamente por el algoritmo) para garantizar el aprendizaje. Por otro lado el protocolo SMILE acomete lograr un aprendizaje incremental distribuido real, por medio del supuesto de un mecanismo de aprendizaje *localmente eficiente*.

1.2. Propuesta

Este trabajo se presentará primero como una extención a [35]. Adoptaremos *SMILE*, un protocolo que acomete resolver el problema de aprendizaje colectivo en un *SMA*, para definirlo en términos del modelo *BDI*; y buscaremos probar que es posible lograr un *aprendizaje incremental* multi-agente con base en las suposiciones de dicho protocolo. Se busca un aprendizaje *incremental* de forma distribuida que no presente las problemáticas mencionadas anteriormente para los métodos de *AA*. Ubicando el aprendizaje logrado en el *nivel 1* y *nivel 2* de *circunspección* definidos también en [35].

El mecanismo de aprendizaje empleado para restaurar la consistencia en cada agente que adoptamos es un algoritmo, inspirado en *TILDE*, que acomete con el aprendizaje incremental y promete ser localmente eficiente. Éste es *ILDT* (Incremental Induction of Logical Decision Trees) [41]. Trabajaremos aquí bajo la suposición de que se puede lograr inducir las creencias que hacen inconsistente el estado actual del agente con la información recibida, generando así una hipótesis que restablezca dicha consistencia.

1.3. Justificación

La carencia de una propuesta formal de aprendizaje [35] en agentes especificados por el modelo *BDI*. El paradigma de agentes sapientes propuesto en [59] presenta razones fuertes para incorporar aprendizaje basado en la experiencia en los agentes inteligentes. El aprendizaje incremental es un aspecto que debe ser seriamente considerado en el desarrollo de agentes inteligentes. El protocolo SMILE reduce el envío de mensajes en la comunicación, y envía aquella evidencia que aporta información relevante para el aprendizaje, lo anterior se traduce a un menor uso de recursos computacionales, y en una convergencia más rápida de la hipótesis aprendida. Además, para el caso de los ejemplos vistos en esta tesis, no siempre resultó posible aprender localmente, debido principalmente a la carencia de informaciónTowards-BDI-SA, Jason-Smiles.

En esta tesis atacamos:

1. **Teoría de Aprendizaje Formal**: extendemos la semántica operacional de *AgentSpeak(L)*, presentando así, una primera aproximación formal del proceso de aprendizaje.

2. **Aprendizaje Distribuido Incremental**: el aprendizaje por *SMILE* se categoriza

en el *nivel 1* y *nivel 2* de *circunspección* para agentes, bajo la suposición de que el conocimiento se encuentra distribuido en el *SMA*.

Capítulo 2

Agencia

2.1. Definición de Agente

Antes de que incursionemos en la búsqueda de una definición *satisfactoria* sobre el término *agente*, una aclaración previa es necesaria: tal definición no existe. Dentro del área de la IA, no se ha logrado conciliar un acuerdo general sobre el concepto de agente (mucho menos a un nivel interdisciplinario). Sin embargo, tenemos a nuestra disposición una gran cantidad de conceptualizaciones, y contamos con algunas definiciones consensuadas, que si bien podrían resultar muy generales para ciertos casos, han sido ampliamente aceptadas por la comunidad de investigadores de la IA. En este capítulo pretendemos familiarizar al lector con la *noción de agencia*. También es preciso poder diferenciar entre programación orientada a agentes y un enfoque de programación clásico. Confiamos en que desarrollo de los dos puntos anteriores resulta más sustancial y atinado, que optar por intentar alcanzar una definición única y completa para el término de *agente*. El uso de los términos debe ser visto desde un enfoque más pragmático y debería de respetarse cierta libertad entre las definiciones de agente según diferentes contextos, después de todo es una herramienta más que una problemática real. Los autores Stuart Russell y Peter Norvig [71] argumentan al respecto: "La noción de *agente* está destinada a ser una herramienta para analizar sistemas, no una caracterización absoluta que divida el mundo en aquello que es un agente y lo que no podemos nombrar un agente", *pp.* 33.

Históricamente la palabra agente, como tal, ha sido empleada en dos sentidos. En un ámbito filosófico, con Aristóteles; y en uno legal, dentro del derecho romano. El sentido legal conceptualiza a los agentes con un matiz que dicta, cierta autonomía concedida o delegación, en dos enunciados: i) el que actúa, o quien puede actuar; ii) el que actúa en

lugar de alguien más con una autorización. Los filósofos por su parte, aluden la búsqueda de la meta, lo situan y determinan que no están aislados: una entidad que actúa con propósito dentro de un contexto social. Obsérvese que la segunda concepción de agente implica en algún sentido a la primera.

Se hará evidente en la marcha que algunas de las definiciones de agentes son verdaderamente tan generales como para incluir dentro de su descripción cosas tan diversas, que pueden partir desde un simple reloj, hasta un ser humano, e.g. una definición común de agente: "un agente es un sistema capaz de percibir y actuar en un medio ambiente." [47,57]. A partir de lo anterior surge una de las principales problemáticas en el área de las Ciencias de la Computación, con respecto a la teoría de agentes. Comúnmente se confunde lo que es un enfoque de agentes y uno de programación modular. Por ejemplo, no sería complicado hacer la analogía de una subrutina de cualquier tipo de programa, con la de un agente que se encuentra situado en el sistema operativo de una computadora (ambiente), y que percibe una serie de eventos a los cuales reacciona (e.g. *agente reactivo*) ejecutando una línea de comandos, cualquiera que éstos sean.

A continuación se presentan algunas definiciones enmarcadas dentro del contexto de la IA, las cuales se han considerado más representativas. Y que han sido extraídas de un trabajo hecho por Stan Franklin y Art Graesser [29, 30]. En él se presenta un análisis profundo de los diferentes enfoques que caracterizan la noción de agente.

- **Stuart Roussell y Peter Norvig:** "Un agente es cualquier cosa capaz de percibir su ambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores." [71].

- **Pattie Maes:** "Un agente autónomo es un sistema computacional situado en algún ambiente complejo, capaz de sensar y actuar autónomamente en su ambiente, con el propósito de realizar un conjunto de metas o tareas para las cuales ha sido diseñado." [55].

- **David Canfield Smith, Allen Cypher y Jim Spohrer:** "Definamos a un agente como una entidad de software persistente dedicada hacia un propósito específico. La característica de *persistencia* distingue a los agentes de las sub-rutinas; los agentes tienen sus propias ideas sobre como llevar a cabo sus tareas (agendas propias). El ser de *propósito específico* los distingue de la gama de aplicaciones multifuncionales; los agentes son comunmente más pequeños." [17].

- **Barbara. Hayes Roth:** "Los agentes inteligentes efectuan de forma continua tres funciones: percepción de condiciones dinámicas dentro de un ambiente; acciones que afectan tales condiciones en el ambiente; y, razonamiento para interpretar las percepciones, resolver problemas, generar inferencias y determinar acciones." [42].

- **Michael Wooldridge:** "Un agente puede ser algún *hardware*, o más comunmente, un sistema basado en *software* computacional, que reune las siguientes propiedades: autonomía, actividad social, reactividad e *iniciativa*." [79].

Es posible alcanzar una definición consensuada con base en lo anterior: "*agente autónomo* es un sistema situado en un ambiente, capaz de sensar dicho ambiente y de actuar en éste dentro de un tiempo determinado, con el propósito de satisfacer su propia agenda y así cambiar lo que pueda sensar en el futuro". Es posible afinar un poco más la definición anterior; Alejandro Guerra-Hernández concreta una noción de agente [35], utilizando el análisis hecho por Stan Franklin y Art Graesser [29, 30], y dos de las definiciones más aceptadas por la comunidad de la IA [71, 79]: "Agente es un sistema computacional persistente en el tiempo, situado dentro de un ambiente, capaz de actuar autónomamente para alcanzar sus objetivos y metas". Posteriormente, señala algunas ventajas en adoptar esta noción originalmente expuestas por los trabajos de Stuart Russell [70] y, Zohar Manna y Amir Pnueli [56]: i) presenta una perspectiva abstracta de alto nivel para ver a los agentes basados en su *situacionalidad* (figura 2.1); ii) permite ver las facultades cognitivas de los agentes, empleadas en la tarea de cómo *hacer lo correcto*; ii) es posible considerar diferentes tipos de agentes, incluso aquellos que no están diseñados con tales facultades cognitivas; iii) nos permite más libertad para considerar varias especificaciones, límites e interconexiones de los subsistemas que componen a los agentes; y iv) dentro del contexto de las Ciencias de la Computación, los agentes pueden ser vistos como *sistemas reactivos*.

Es necesario tomar en cuenta las siguientes claves, para entender bien la noción de agente y la profundidad que existe en este término, para no caer en el error de utilizarlo a la ligera.

- Autonomía: Arie A. Covrigaru y Robert K. Lindsay [22] argumentan que ser autónomo, depende no sólo de la habilidad para seleccionar metas u objetivos de entre un conjunto de ellos, ni de la habilidad de formularse nuevas metas; sino de tener el tipo adecuado de metas. Aquellas que no atenten contra su autonomía. Y que no contradigan a otras metas.

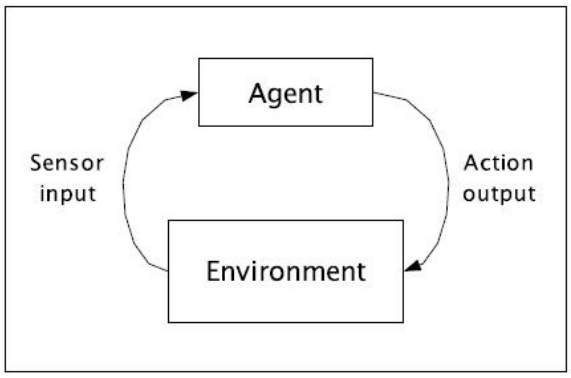

Figura 2.1: *Perspectiva abstracta de alto nivel. El agente tiene como entrada el sensado de su ambiente donde está situado, y produce acciones como salida en respuesta a sus percepciones*

- Inteligencia: Otro aspecto importante dentro de la teoría de agentes, y por supuesto en el campo de la IA, es el de *inteligencia*. Lenny Foner [28] argumenta que la inteligencia en un agente se percibe como una clase particular de comportamiento, identificado por Michael Wooldridge y Nick Jennings como *comportamiento flexible y autónomo*.

Recordemos que los agentes artificiales son usualmente diseñados para llevar a cabo tareas por nosotros, de forma que debemos comunicarles qué es lo que esperamos que hagan. En un sistema computacional tradicional esto se reduce a escribir el programa adecuado y ejecutarlo. Podría pensarse que un agente puede ser instruido sobre qué hacer usando un programa, con la ventaja colateral de que su comportamiento estará libre de incertidumbre, sin embargo, lo anterior atenta contra su autonomía, teniendo como otro efecto colateral la incapacidad del agente para enfrentar situaciones imprevistas mientras ejecuta su programa. Dicho en otras palabras, indicarle al agente lo que tiene que hacer, sin decirle cómo hacerlo (inteligente y autónomo, pero además "'obediente"' y predecible).

En las siguientes páginas presentaremos una aproximación formal del concepto de agente. Pero antes es necesario ahondar un poco más en la conceptualización de un ambiente, dentro del ámbito de agencia.

2.2. Ambiente

Análogamente a que el agente por excelencia es el ser humano, nuestro ambiente por excelencia es el mundo real [14]. Y entendemos por *ambiente*: el espacio donde un agente, o un grupo de ellos, se encuentra situado. Rodney Brooks en su propuesta argumenta que todo agente toma una forma robótica [14]. Por el contrario, Oren Etzioni [26], considera que no es necesario que los agentes tengan implementaciones robóticas porque los ambientes virtuales, como los sistemas operativos y el web, son igualmente válidos que el mundo real. En este trabajo asumimos la posición de Etzioni, resaltando que lo importante es que la interacción del agente con su ambiente se dé en forma autónoma y bajo persistencia temporal. Nuevamente Stuart Russell y Peter Norvig [71] desde una perspectiva más funcional señalan que, más allá de la controversia, es importante identificar que existen diferentes tipos de ambientes:

- **Accesible vs. inaccesible.** Si un agente puede percibir a través de sus sensores, los estados completos del ambiente donde se encuentra, se dice que el ambiente es *accesible*. Un ambiente es *efectivamente accesible* si el agente puede percibir todos aquellos elementos en el ambiente, que son relevantes para su toma de decisiones. De lo otra forma se considera inaccesible.

- **Determinista vs. no determinista.** Si el próximo estado del ambiente está determinado por la acción que ejecuta el agente, se dice que el ambiente es *determinista*. Si otros factores influyen en el próximo estado del ambiente, éste es *no determinista*.

- **Estático vs. dinámico.** Si el ambiente puede cambiar mientras el agente se encuentra deliberando, se dice que es dinámico; de otra forma, se dice estático.

- **Discreto vs. continuo.** Si hay un número limitado de posibles estados del ambiente, diferenciables y claramente definidos, se dice que el ambiente es *discreto*; de otra forma se dice que es *continuo*.

También nos presentan algunos ejemplos de ambientes bien estudiados dentro de la IA y sus propiedades (cuadro 2.1). Cada ambiente, o clase de ambiente, requiere de alguna forma agentes diferentes para que éstos tengan éxito. La clase más compleja de ambientes corresponde a aquellos que son inaccesibles, no episódicos, no deterministas, dinámicos y continuos, lo que desgraciadamente corresponde, casi siempre, a nuestro ambiente cotidiano.

Ambiente	Accesible	Determinista	Episódico	Estático	Discreto
Ajedrez sin reloj	si	si	no	si	si
Ajedrez con reloj	si	si	no	semi	si
Análisis imágenes	si	si	si	semi	no
Backgammon	si	no	no	si	si
Poker	no	no	no	si	si
Tutor inglés	no	no	no	no	si
Robot toma piezas	no	no	si	no	no
Controlador refinería	no	no	no	no	no
Robot navegador	no	no	no	no	no
Conductor de autos	no	no	no	no	no
Diagnóstico médico	no	no	no	no	no

Cuadro 2.1: *Ejemplos de ambientes estudiados en IA y sus propiedades (tomado de Stuart Russell y Peter Norvig [71]).*

2.3. Clases de agentes

Ahora introduciremos una notación usada por Michael Wooldridge para describir formalmente un agente abstracto [78].

Primero se asume la existencia de una ambiente. Éste se representa, en la práctica, mediante un estado finito de estados de la manera $E = \{e, e', \dots\}$. A partir de un estado inicial del ambiente e_0 y de cualquier estado $e \in E$ el agente deberá elegir una acción a llevar a cabo. Todo agente cuenta con un conjunto de acciones básicas que puede llevar a cabo bajo ciertas circunstancias: $Ac = \{\alpha, \alpha', \dots\}$. Estas acciones son la única forma que el agente tiene para repercutir en su ambiente[1]

Ahora definamos una *corrida* como una serie de ejecuciones de acciones, formalmente decimos $r = e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$.

Los agentes se modelan como funciones que mapean corridas, que terminan en un estado del ambiente, a acciones: $Ag : R^E \to Ac$. La forma más sencilla de implementar un programa de agente es a través de su mapeo ideal. El algoritmo 1 muestra la función que implementa un agente de este tipo.

Es posible clasificar a los agentes de acuerdo a la estrategia con la que éstos contienden con su medio para lograr resolver sus *metas* e intentar alcanzar exitosamente los fines

[1] Las acciones cambian el estado del ambiente de la forma $e_i \xrightarrow{\alpha_i} e_{i+1}$.

Algoritmo 1 Agente basado en mapeo ideal

```
1: function AGENTE-MAPEO-IDEAL(p)          ▷ p es una percepción del ambiente.
2:     percepciones ← percepciones ∪ p
3:     acción ← busca(percepciones, mapeo)          ▷ mapeo predefinido.
4:     return acción
5: end function
```

para los cuales fueron diseñados. Lo anterior incluye implícitamente la forma en la que estructuran su conocimiento y evalúan el ambiente para decidir sus respectivos cursos de acción (figura 2.2).

Figura 2.2: *Clasificación de agentes según Jose C. Brustoloni [16]. Las clases no son excluyentes, así que, por ejemplo, podemos hablar de agentes adaptativos planificadores.*

Una aproximación más moderna y basada en las diferentes implementaciones hechas hasta la década pasada (figura 2.3). Se identifican básicamente cuatro tipos de agentes: *agentes de reflejo simples, agentes que mantienen un estado del mundo, agentes basados en metas y agentes basados en funciones de utilidad.*

Figura 2.3: *Clasificación de agentes por Stuart Russell y Peter Norvig [71].*

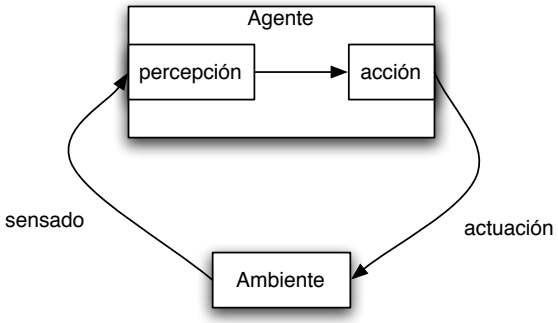

Figura 2.4: *Abstracción de un agente reactivo.*

2.3.1. Agentes reactivos

Los *agentes reactivos* [15], o de reflejo simple [70], seleccionan sus acciones basados en su percepción actual del ambiente. La figura 2.4 ilustra la interacción de estos agentes con su ambiente.

Estos agentes se implementan generalmente mediante reglas, las cuales mapean un estado del ambiente percibido, que juega el papel de condición, a una acción ($e_i \rightarrow a_i$). El algoritmo 2 muestra la forma general de un programa de agente reactivo.

Algoritmo 2 Agente reactivo o reflex

1: **function** AGENTE-REACTIVO(e)
2: $estado \leftarrow percibir(e)$
3: $regla \leftarrow selecciónAcción(estado, reglas)$ ▷ $reglas$ predefinidas.
4: $acción \leftarrow Acción - Regla(regla)$
5: **return** $acción$
6: **end function**

Existen autores como Rodney Brooks [14] que argumentan que no es necesario incorporar un proceso deliberativo a los agentes para que éstos logren exhibir un comportamiento inteligente. Aquellos autores comparten una noción débil del modelo de agencia. Para nosotros todos los agentes de tipo meramente reactivo comparten una limitación, sin importar la manera en que éstos sean implementados: para que el comportamiento sea racional (en este caso particular, óptimo), es necesario conocer la acción adecuada a ejecutar para cada estado posible del ambiente, y que este último sea por lo menos, efectivamente accesible.

2.3.2. Agentes con estado

Este tipo de agentes es interesante desde la perspectiva de la *noción fuerte* de agencia [2], porque como veremos más adelante, en el modelo BDI, podemos identificar a los agentes de acuerdo a su estado actual interno. Dicho estado cambia conforme el agente percibe su ambiente($siguiente : I \times Per \rightarrow I$). En las implementaciones más generales de este tipo de agentes, se mapea un conjunto de estados internos I a acciones posibles ($acción : I \rightarrow Ac$).

El algoritmo 3 nos muestra el programa de un agente con estado. La diferencia con el agente puramente reactivo, es que la función $siguiente/2$ (línea 3, algoritmo 3) modifica el estado interno del agente interpretando su entrada perceptual, usando el coocimiento que el agente tenía sobre el ambiente.

Algoritmo 3 Un programa de agente con estado

```
1: function AGENTE-CON-ESTADO(e)                          ▷ e ∈ E
2:     p ← percibir(e)
3:     estado ← siguiente(estado, p)
4:     regla ← selecciónAcción(estado, reglas)            ▷ reglas predefinidas.
5:     acción ← AcciónRegla(regla)
6:     return acción
7: end function
```

2.3.3. Agentes lógicos

Los agentes lógicos se basan en el conocimiento que éstos comprenden. La forma más simple de hacer un agente basado en conocimiento es diciéndole a éste qué es lo que necesita saber. Los agentes lógicos incorporan una representación basada en lógica para representar su conocimiento, a esta representación se le nombra generalmente base de conocimiento (BC), y podemos decir que a grandes rasgos esta es una serie de enunciados (lógicas). El agente tiene la capacidad de manipular dicho conocimiento mediante dos operaciones básicas, *tell* y *ask*. La primera es un mecanismo para añadir sentencias a la BC, y la segunda para preguntar qué se sabe en la BC. Ambas operaciones requieren realizar inferencia.

[2] La explicación de esta noción de agencia la explicaremos en el capítulo 3. Aquí destacamos la importancia de la descripción de los agentes de estado.

La posibilidad de los agentes de manipular su BC con la finalidad de generar nuevo conocimiento que le permita adaptarse a las diferentes situaciones que se le presenten, es uno de los principales aspectos que hacen de estos agentes, un enfoque muy atractivo. Además, debido al empleo de la lógica, éstos resultan mucho más expresivos y elegantes.

En la propuesta de un enfoque moderno de la IA, hecha por Stuart Russell y Peter Norvig en [71], se argumenta que la construcción de agentes racionales[3] surge como un concepto unificador de la IA. Estos autores se inclinan por un enfoque basado en agentes lógicos.

El comportamiento racional puede obtenerse a partir de una representación simbólica del ambiente y el comportamiento deseado. El agente manipulará sintácticamente esta representación para actuar. La aproximación anterior nos lleva a formular el estado de un agente como un conjunto de fórmulas lógicas y la selección de acción como demostración de teoremas o deducción lógica.

Un algoritmo genérico de un agente lógico puede ser como se muestra en el algoritmo 4. Cada vez que el programa del agente es invocado, realiza dos cosas. Primero, dice a la base de conocimiento lo que ha percibido. Luego, pregunta a la base de conocimiento qué acción debe de ejecutar. El proceso de selección de acción implica un proceso de razonamiento extensivo acerca del estado actual del mundo, de los efectos de las posibles acciones, etc.

Algoritmo 4 Un agente lógico

```
1: function AGENTE-LOGICO(Percepción : Ac)          ▷ Ac Acciones.
2:     static BC                                    ▷ BC Base de Conocimiento.
3:     tell(BC, Percepción)                         ▷ Lógicos: BC ⊨ Percepción
4:     Ac ← ask(BC, pedirAcción())
5:     return Ac
6: end function
```

2.3.4. Agentes basados en metas

En la tradición de IA, las *metas* describen situaciones que son deseables para un agente, y son definidas como cuerpos de conocimiento acerca de los estados del medio ambiente que el agente desea ver realizados [64]. Este método de diseñar los agentes es inherente

[3] A la manera correcta de actuar se le llama comportamiento racional. En términos generales, lo que conduce al agente a obtener mejores resultados, con base en una medida de desempeño.

a los conceptos de *planificación* y *búsqueda*. La decisión correcta de un agente depende de su meta, esto es un estado del ambiente deseable. Ahora, supongamos cada acción perteneciente al conjunto de acciones disponibles por un agente como una operación, es posible expandir un espacio de búsqueda. Y el plan, el cual determina el curso de acciones a seguir es la ruta al estado deseable (meta).

Implícita en la arquitectura del agente, está su "intención" de ejecutar las acciones que él "cree" le garantizan satisfacer cualquiera de sus metas. Esto se conoce en filosofía como *silogismo práctico* y constituye la definición de *racionalidad* usada en IA [60, 64].

Una vez que un agente alcanza un estado que satisface la meta, se dice que ésta ha sido satisfecha. Las metas definidas de esta forma, constituyen un subconjunto de las metas que un agente puede enfrentar. Arie A. Covrigaru y Robert K. Lindsay [22] concluyen que un sistema se percibe más fácilmente como autónomo si tiene un conjunto de metas de alto nivel (no son submetas de otra meta) y algunas de ellas son homeostáticas (la satisfacción de la meta no termina al alcanzar el estado que la define, el sistema monitorea continuamente si ese estado ha sido abandonado, para buscar satisfacerlo nuevamente).

Esta forma de ver a los agentes permite, en sacrificio de un poco de eficiencia, una mayor flexibilidad. En ambientes más dinámicos y complejos resultan, incluso más eficientes que un enfoque reactivo, para el cual el número de reglas condición-acción crece demasiado.

2.3.5. Agentes basados en funciones de utilidad

Una *utilidad* es un valor numérico que denota la bondad de un estado del ambiente. El agente busca alcanzar aquellos estados que maximizan su utilidad a largo término. Por lo tanto, la especificación de una tarea en este enfoque corresponde simplemente a una función $u : E \rightarrow \Re$ (la función mapea una recompensa $\Re$ para cada Estado E del agente) la cual asocia valores reales a cada estado del ambiente. Para poder especificar una visión a largo plazo del desempeño del agente, algunas veces las utilidades no se asignan a los estados del ambiente, sino a las corridas del agente.

Un agente basado en metas podría tener diversas soluciones para resolver un mismo objetivo. La función de utilidad determinaría cuál sería la solución correcta. El concepto de utilidad está estrechamente relacionado con el de racionalidad: i) en caso de objetivos conflictivos (e.g. velocidad y seguridad), la función de utilidad debería determinar el equilibrio adecuado; y ii) tratándose de varios objetivos que presentan incertidumbre en su realización, es posible ponderar, mediante una función de utilidad la probabilidad de

éxito.

2.3.6. Agentes que incorporan aprendizaje

Stuart Russel y Peter Norvig han propuesto una arquitectura genérica de agente para aprendizaje compuesta de cuatro elementos [71]:

- El elemento a ser mejorado, el cual describe en este caso a un agente que no incorpora aprendizaje.
- Un elemento que juega el rol de crítico, encargado de retroalimentar el elemento de aprendizaje, i.e., señalar qué tan bien el agente está siendo mejorado.
- Generador de problemas, responsable de sugerir acciones que conducirán a nuevas experiencias con información valiosa para el aprendizaje.
- Elemento de aprendizaje, que tiene el papel de hacer las mejoras en el agente.

2.4. Sistemas Multi-Agentes: Comunicación

La interacción social en los agentes puede llevarse a cabo en distintos niveles y de diversas formas. Por ejemplo, en un grupo de agentes trabajando de forma cooperativa para resolver una misma meta, podríamos especificar diferentes protocolos de comunicación, interacción, circunspección, etc. También nos resulta imposible hablar de aprendizaje incremental de forma colectiva, sin tratar antes el tema de la comunicación. En esta sección, no pretendemos profundizar en los aspectos sociales de agencia; la presentamos, más bien, como una introducción a la comunicación en términos de actos de habla. Entonces queremos hacer conciencia de que dejaremos de lado algunos aspectos que creemos no reelevantes para esta tesis, sobre actividad social y cooperación en sistemas multiagentes, pero no por eso menos importantes. Michael Wooldridge presenta una revisión detallada, enfocada al modelo BDI, sobre estos temas en [78]. La aproximación de actividad social que utilizamos en este trabajo, es la misma que emplea en su trabajo [35]: i)*interacción* como lo define Jacques Ferber [27] y; ii)*acción social* como la caracteriza Cristiano Castelfranchi [18]. También nos adjudicamos la nesecidad de comunicación entre agentes, como un accesorio sustancial para el problema de aprendizaje en un SMA.

En términos del modelo de agencia, podemos encontrar ciertos fines para los cuales resulte no tan práctica dicha perspectiva. Imaginemos la siguiente situación por un momento:

tengo el deseo de que un agente robot ejecute una tarea que pertenece a su conjunto de habilidades. Ahora pensemos en la autonomía que caracteriza a nuestra noción de agencia. Debido a que los agentes exhiben cierta autonomía, lograr que nuestro robot lleve a cabo nuestro deseo, debería ser consecuentemente, una decisión tomada por el propio agente robot. Es decir, el control de ejecutar o no dicha tarea (que pertenece a su conjunto de habilidades y que será realizada por el agente robot) debe encontrarse finalmente en el propio agente, para evitar romper el modelo de agencia. Lo anterior es algo muy intuitivo si hablaramos de nosotros, seres humanos, que nos caracterizamos por ser agentes con características fuertes de autonomía, en sentido de decidir sobre el control de nuestras acciones. Pero para un programador que está familiarizado con un enfoque de programación clásica, lo anterior puede resultar engorroso al momento de la implementación. Ejemplo, un objeto programado en java, puede ejecutar una función miembro de otro objeto; a partir del código que define al objeto o_1, puedo hacer una invocación como $o_2.function(arg1)$. Lo anterior, como hemos mencionado, no podría ser cierto sí o_1 y o_2 fueran, verdaderamente agentes. La forma en que un agente puede influir en otro, de una forma similar a la del ejemplo anterior es por medio de la comunicación.

Entendemos por comunicación a un intercambio intencional de información efectuado mediante la producción y percepción de signos pertenecientes a un sistema compartido de signos convencionales.

Los agentes emplean lenguajes de comunicación basados en actos de habla (ver sección 2.4.1). Esta forma de caracterizar el lenguaje como acciones se le conoce como teoría de actos de habla y tiene sus orígenes en los trabajos de John R. Searle [73]. Posteriormente, John L. Austin extiende esta teoría y acuña el término por el cual hoy en día se le conoce [3]. A continuación introduciremos los puntos clave de la teoría de actos de habla.

2.4.1. Actos de habla

John L. Austin se plantea el lenguaje desde un punto de vista funcional. Es decir, antepone la función principal de encausar a la cooperación de aquellos agentes jugando el rol de receptores en el acto de la comunicación, es decir, el mover a la acción a los demás agentes por medio del habla.

Austin utiliza como ejemplos para su tesis actos como declarar la guerra, o pronunciar frases como “los declaro marido y mujer”. Los enunciados anteriores tienen un efecto

en el mundo físico de forma irrevocable[4]. La teoría de actos de habla tiene su principio fundamental en que el lenguaje es acción, es decir, un agente emite con el propósito de cambiar el estado de el mundo, en el mismo modo que este lleva a cabo acciones físicas. La distinción entre los actos de habla y el resto de las acciones, radica básicamente en que la parte que el agente, llevando acabo el acto de habla, desea modificar es, de hecho el estado mental del agente receptor. Existen tres aspectos presentes en los actos de habla:

- **Locutiva:** es el enunciado propiamente dicho. Es decir, una serie de sonidos con significado. Es realizado por el emisor.
- **Ilocutiva:** es lo que se pretende hacer al enunciar algo (interpretado por el receptor): ordenar, afirmar, prometer, preguntar, etc. Aquí aparecen las cualidades de entonación e intensidad que dan al enunciado su *fuerza ilocutiva*. "Cierra aquella puerta" y "¿sería mejor cerrar la puerta?" son modalidades del mismo enunciado (con formas proposicionales difererentes) que poseen distinta fuerza ilocutiva.
- **Perlocutiva:** es lo que se consigue como resultado de la locución a causa de los efectos de la ilocución. Por ejemplo "Los declaro marido y mujer" trae como resultado, casar. "no tardaré" induce una creencia. Etcétera.

En su trabajo, John Searle [72] identifica varias condiciones que deben ser verdaderas para que la acción de habla pueda llevarse acabo, por ejemplo: i) el receptor debe ser capaz de escuchar correctamente el mensaje del emisor; ii) una condición inicial que lleva al emisor a considerar que el receptor está en condición de ejecutar la petición hecha mediante el acto de habla por el emisor; iii) el acto de habla debe existir bajo un contexto de sinceridad, es decir, si se requiere al agente receptor para llevar a cabo una acción, se espera que realmente el agente emisor desee que su receptor efectue tal acción. También existe una clasificación sistemática de los tipos de actos de habla, que comúnmente se hace como sigue:

- **Representativas**: el emisor de un acto de habla de este tipo, se compromete a que la proposición expresada es verdadera (*informar*).
- **Directivas**: es un intento del emisor de que el receptor de la proposición haga algo (*pedir*).

[4]Consideremos que el efecto conseguido es resultado de una compleja red de estados mentales entre los agentes involucrados, es decir, existen convenciones previas y se respetan en cierto grado.

- **Comisivas**: Compromete al que ejecuta el acto de habla a un cierto curso de acción (*prometer*).
- **Expresivas**: Estos tipos de acto de habla expresan estados psicológicos (*agradecer*).
- **Declaraciones**: Dictamina un cambio en el estado de las cosas (*Declarar la guerra*).

Otras aspectos relacionados al problema de comunicación, al momento de la implementación, son las convenciones y los roles sociales que son discutidos en [52, 76]. Algunas de las fuerzas ilocutorias descritas anteriormente, pueden requerir además, de una relación (subordinación o confianza) entre el agente emisor y el receptor, para que estás puedan llevarse a cabo correctamente; por ejemplo, un comando (directiva), requiere una relación de subordinación, y esta última no sería necesaria en el caso de una fuerza ilocutiva expresiva. Stephen C. Levinson [52] propone que es necesaria cierta clasificación de las relaciones entre los actos de habla, tales como que, a una petición debe sucederle siempre una respuesta correspondiente; o una declaración tiene más fuerza que un acto informativo.

Con el objetivo de que los agentes basados en planes pudieran incluir actos de habla, C. Cohen y G. Perrault [21] empiezan a crear lo que posteriormente comenzaría a ser la teoría de actos de habla. Principalmente el trabajo de C. Cohen y G. Perrault fue demostrar como la precondicions y poscondiciones de los actos de habla podían ser expresadas en lógica multimodal incorporando operadores para describir las creencias, habilidades y deseos de los participante en los actos de habla. Posteriormente, Cohen y Levesque [19] desarrollan una teoría en la cual los actos de habla son modelados como acciones ejecutados por agentes racionales como parte de sus intenciones. Esta última basada en la teoría de intencionalidad descrita en [20].

2.4.2. KQML y FIPA: lenguajes para la comunicación de agentes

El lenguaje de comunicación KQML (*Knowledge Query Meta-Language*, por sus siglas en inglés) es el primer intento por definir un lenguaje práctico de comunicación de agentes basado en actos de habla. KQML es básicamente un lenguaje para el encapsulamiento de mensajes, al estilo de la sintaxis de LISP. Este lenguaje se conforma por 41 palabras claves (*performativas*), tales como *ask-if* y *tell*. Una descripción detallada puede encontrarse en [58, 66]. Por ejemplo, la performativa *tell* es empleada con la "intención" de cambiar las creencias del receptor, mientras que *achieve* se utiliza para expresar la intención de cambiar las metas del agente receptor.

El estándar FIPA[5] para la comunicación de agentes fue liberado en el 2002, pero éste se encuentra estrechamente relacionado con KQML. El concepto y la sintaxis son básicamente idénticas, y difieren tanto en el conjunto de performativas como en algunos detalles en la semántica [50].

Para entender mejor KQML, veamos un ejemplo:

```
(ask-one
    :content sample(planStack,State,Label)
    :receiver ag1
    :language AgentSpeak(L)
    :ontology HELP
)
```

La interpretación del mensaje anterior es como sigue: el emisor requiere del $ag1$ un ejemplo relevante para el *planStack*. La fuerza ilocutoria del mensaje es *ask-one*, la cual es empleada para efectuar una pregunta al agente receptor. Los atributos de este mensaje son: i) *:content*, especifica el contenido del mensaje; ii) *:receiver*, en este campo se especifica a quien se desea enviar el mensaje; iii) *:language*, aquí se especifica el lenguaje en el cual está expresado el contenido; y iv) *:ontology*, define los términos usados en el mensaje.

[5] http://www.fipa.org/specs/fipa00037/SC00037J.html

Capítulo 3

El Modelo BDI

Las definiciones que hemos expuesto en el capítulo 2 coinciden en la mayoría de los aspectos con un enfoque de agente descrito por Michael Wooldridge en [78], llamado la *noción débil* de agente. Sin embargo, actualmente dentro del campo de la IA, el término agente tiene un significado mucho más preciso y fuerte; que se identifica como la *noción fuerte* de agente. Y es esta última, por la que nos inclinamos para desarrollar este trabajo. La noción fuerte de agencia, en adición a las características de autonomía y reactividad explicitadas en la mayoría de las definiciones ya mencionadas, argumenta que los agentes deberían presentar ciertas facultades que conciernen a los seres humanos de una forma natural; tales como creencias, deseos, intenciones, emociones, etc. Incluso hay quienes defienden la necesidad de incorporar atributos tales como *emociones* [4], principalmente en aquellos agentes que tienen que interactuar de forma directa con seres humanos.

Particularmente nos enfocaremos a una clase de agentes conocida como *agentes intencionales*; los cuales pertenecen a la noción fuerte de agencia, y tienen un fundamento filosófico sólido, acerca de como los seres humanos tomamos decisiones racionales sobre las acciones. Los aspectos relevantes de esta teoría han sido rigurosamente formalizados en una familia de *lógicas BDI* [78] por sus siglas en inglés (*Belief-Desire-Intention*). Esta familia de lógicas representa el componente lógico del modelo BDI, el cual captura los aspectos claves del modelo como un conjunto de axiomas lógicos. Hemos mencionamos que el modelo BDI tiene un fundamento filosófico basado en razonamiento práctico desarrollado por el filósofo Michael Bratman [12]. A continuación introduciremos detalladamente este aspecto y posteriormente, al final de este capítulo, presentaremos una arquitectura abstracta inspirada en IRMA [35], la cual hace evidente el tercer componente señalando en [78]: la arquitectura de software.

3.1. Intencionalidad

El tema de intencionalidad se remonta desde Aristóteles, y algunos filósofos medievales (aunque en otro sentido teórico) como San Agustín, Santo Tomás, Duns Escoto y Avicena. En la filosofía moderna es introducido por Franz Brentano, en su obra *Psychologie vom empirischen standpunkt* en 1874, y retomada posteriormente por Chisholm, quien ofrece una versión metalingüística de la tesis de Brentano [43]. De entre las diferentes teorías de intencionalidad [53], un enfoque funcionalista propuesto por Daniel Dennett [23] ha sido utilizado ampliamente como el fundamento filosófico para describir a los agentes como entidades que son sujeto de creencias, deseos y otras actitudes proposicionales, es decir, como *sistemas intencionales*.

Dennet examina el fenómeno de la creencia, y sostiene que la creencia sólo puede ser percibida si se adopta una cierta estrategia predictiva, a la que él denomina *estrategia intencional*, y que es la que posee quien se encuentra en lo que él llama *actitud intencional* (*intentional stance*). La estrategia intencional consiste en tratar el objeto cuya conducta se quiere predecir como un agente racional, con creencias, deseos y otros estados mentales que presenten lo que Brentano y otros han venido llamando intencionalidad. La tesis de Dennett es que cualquier sistema (o cualquier objeto) cuyo comportamiento sea bien predicho por medio de tal estrategia es un sujeto de creencias, o dicho con otras palabras, un sistema intencional.

3.1.1. Los Sistemas Intencionales

El filósofo Daniel Dennett acuñó el término *sistemas intencionales*. Él argumenta en su trabajo que el comportamiento humano puede ser predicho y explicado a través de la atribución de *actitudes* [23] a los que llama estados intencionales, por el hecho de que parecen tener una estructura o prototipo que consiste en una *actitud* (e.g, creer, desear, querer). A este estado mental intencional se le conoce también como *actitud proposicional*.

A la hora de predecir el comportamiento, Daniel Dennett identifica otras estrategias; entre ellas habla de dos aproximaciones; la física y la de diseño; la primera, es de la que hacen uso las ciencias físicas, como la biología. Lo que es igual a predecir un sistema en fase a su constitución física. Al respecto, vale la pena pensar en lo siguiente: no sería nada práctico explicar el procesador de una computadora en términos físicos. Más efectivo resultará hacerlo en términos de diseño, es decir, bajo el supuesto de que el sistema se comportará tal y como ha sido diseñado para comportarse. Por ejemplo, no es necesario

conocer la constitución física de una calculadora para predecir con éxito su comportamiento.

Una vez expuesto lo anterior, es evidente el sentido funcional y práctico de emplear una estrategia intencional. Ahora recordemos que el funcionamiento de esta estrategia requiere la suposición de que el agente es racional, y por tanto que se comporta como debería comportarse en función de sus creencias, sus deseos y de sus propósitos. También presupone que atribuimos al agente como creencias todas aquellas verdades que son *relevantes* para sus deseos y que le han proporcionado su experiencia, así como la atribución de todos aquellos deseos que son buenos para él. Ambos supuestos se reducen a lo que llama Dennett regla fundamental, y que el enuncia así: atribuyamos al sistema aquellas creencias y aquellos deseos que debería tener (esto es lo necesario para cumplir sus propósitos, comportándose de una manera racional). En su supuesto el filósofo parte de un ideal de racionalidad perfecta, asumiendo que el sistema cree todas las implicaciones de sus creencias, y que no acepta ningún par de creencias que sean contradictorias entre si; pero las implicaciones pueden ser infinitas. Rápidamente se intuye la problemática en la implementación de una estrategia que asume todo de manera tan ideal, e.g., revisión de consistencia en las creencias. Sin embargo, argumenta que para que la estrategia de intencionalidad trabaje adecuadamente, basta considerar aquellas implicaciones que son *relevantes* (relacionadas con la circunstancia, e.g., no voy a considerar cuanto cuesta el boleto para ir a francia, si lo que quiero es ir a la fiesta de Lulú, y ella es mi vecina) para la situación en que se encuentra el sistema en ese momento desde el punto de vista de su comportamiento. Lo cual la hace más atractiva para una posible implementación.

Finalmente comentaremos, lo que en [23] identifican como diferentes grados de intencionalidad; en primer grado podemos hablar de actitudes como creencias y deseos, pero no creencias sobre creencias y deseos (que se clasifican en un segundo orden); y así de manera incremental. Este orden provee una *escala de inteligencia*.

La manera en que un sistema intencional produce una acción racional es el tema del trabajo de Michael Bratman [12] sobre razonamiento práctico, el cual introducimos a continuación.

3.2. Razonamiento Práctico

La parte de la teoría BDI que trata de modelar la racionalidad de aquellas acciones tomadas por los seres humanos en determinadas circunstancias fue desarrollada por el

filósofo Michael Bratman [12]. Esto tiene sus orígenes en una tradición filosófica que busca comprender lo que llamamos *razonamiento práctico*, es decir, razonamiento dirigido hacia las acciones: hacia el proceso de decidir qué hacer. A diferencia del razonamiento teórico que está dirigido hacia las creencias. Por ejemplo, si yo creo que todos los hombres son mortales, y creo que Sócrates es hombre, normalmente llegaré a la conclusión de que Sócrates es mortal. El proceso de concluir que Sócrates es mortal se le llama razonamiento teórico, debido a que éste afecta solamente mis creencias acerca del mundo. Ahora pensemos en el proceso de decidir tomar un tren en lugar de un autobús, esto último se le conoce como razonamiento práctico, porque como se ha mencionado, se encuentra dirigido hacia las acciones.

Este tipo de razonamiento comprende dos actividades: i) deliberación, es decir, decidir cuáles son las *metas* a satisfacer; y ii) un análisis medios-fines, es decir, decidir cómo es que el agente va a lograr satisfacer esas metas. Ambas actividades pueden verse como procesos computacionales ejecutados por agentes con *racionalidad acotada*. La racionalidad acotada tiene dos implicaciones importantes [78]: i) Puesto que una computación es un recurso valioso para los agentes situados en ambientes de tiempo real, un agente debe controlar su razonamiento eficientemente para tener un buen desempeño; y ii) los agentes no pueden deliberar indefinidamente, deben detener su deliberación en algún momento, elegir los asuntos a atender, y comprometerse a satisfacerlos.

Las intenciones vistas desde este punto de vista, son asuntos que el agente ha elegido y se ha comprometido a satisfacer. Intencional se usa aquí para calificar tanto las acciones, por ejemplo, hacer algo intencionalmente; como estados mentales, por ejemplo, tener la intención de hacer algo (intenciones dirigidas al futuro).

Al hacer la distinción de dos tipos de intenciones, con respecto a su proyección debemos tomar en cuenta lo siguiente: i) la intención presente nos sugiere qué hacer en el ahora, y que la acción llevada a cabo es intencional; ii) la intención futura puede ser vista más bien como un compromiso, el cual dirige el curso de acción, y está estrechamente relacionada con lo que llamamos planes.

Las intenciones dirigidas a futuro tienden a encausar los cursos de acción con mayor fuerza que los deseos. Las intenciones deben tener un grado de persistencia y consistencia, no tiene sentido adoptar una intención determinada para abandonarla posteriormente; las intenciones no se manejan así en la vida real, por ejemplo, si Pedro tiene la intención de hacer un *PhD*, no se rendirá al primer intento (el caso de pedro es ser un agente

consistente). Tampoco suena conveniente adoptar una segunda intención, que resulte inconsistente con la primera; la consistencia aquí funciona como un filtro de admisión, que restringe las posibles condiciones que un agente debe considerar. Esto último acota el razonamiento y resulta útil en la práctica donde los sistemas tienen recursos limitados y se manejan en tiempo real.

Mencionamos que las intenciones futuras están estrechamente relacionadas con la planeación. Planear es una actividad central en el razonamiento práctico. Los agentes racionales acotados tienen dos necesidades generales para usar planes. Primero, puesto que tienen recursos limitados para deliberar al momento de actuar, la influencia de su deliberación sería mínima si no se extendiera sobre el tiempo. Los planes son formas de influir la acción más allá del presente. Segundo, los agentes usan planes para relajar las demandas de coordinación individual (actividades presentes y futuras) y con los demás (mis actividades y las actividades de los demás). Debido a la racionalidad acotada, los planes totales están más allá de los límites de un agente. Por ello, los agentes usan *planes parciales*. Esta "incompletez" crea la necesidad de un análisis medios-fines para ir ajustando los planes parciales con medios apropiados y cursos de acción más específicos. Lo anterior establece una planeación jerárquica. Y la acotación surge en la consistencia obligada que debe existir en la adopción de sub-planes, conforme los planes se van haciendo más específicos.

Michael Bratman y Martha E. Pollack establece una distinión entre los planes: como estructuras abstractas y los planes vistos como estados mentales [13, 67]. Los primeros pueden ser vistos como una receta o procedimiento para alcanza cierto fin; mientras que los planes como estados mentales son equivalentes a las intenciones. Presentan las mismas propiedades, tales como inercia, control conductal y también son tomados en cuenta para la decisión de procesos posteriores de razonamiento práctico y planeación. Los planes debido a su complejidad, y el hecho de que los seres humanos somos agentes racionales acotados, deben ser instanciados de forma parcial (generales), para posteriormente ir afinando los detalles; si Pedro planea conocer a Lulú, empezará por definir un plan general, invitar a Lulú; más tarde podrá pensar en como hacerle llegar la invitación o como se le informará. Los planes también presentan una estructura jerárquica, esto es, tienen otros planes empotrados que conciernen a los medios y pasos preliminares para conseguir su cometido. Por ejemplo, el plan de conocer a Lulú puede tener empotrado el plan de hacer una fiesta en casa de Juan. Esto mueve a los agentes a deliberar sobre ciertas partes de

sus planes, mientras otras partes se mantienen fijas, es decir, deben mantener los fines intentados mientras deliberan sobre los medios y pasos preliminares.

Es evidente cómo los planes presentados con las características de parcialidad y jerarquía permiten a los agentes coordinar sus actividades sociales e individuales, a un nivel parejo con sus limitadas capacidades para deliberar y procesar información. Porque permiten a las intenciones previas moldear y ajustar la conducta posterior, al mantener la consistencia. A diferencia de los planes completos, los cuales resultar ineficientes o inútiles en un medio dinámico, los planes parciales y jerárquicos proveen a mucha intenciones y acciones de un carácter híbrido: una acción puede ser deliberativa en un sentido y no deliberativa en otro. Por ejemplo, Pedro puede deliberar si hacer la fiesta en casa de Juan o de José, mientras que la intención superior de hacer una fiesta no es reconsiderada.

Es por medio de estos planes parciales, jerárquicos resistentes a la reconsideración, y eventualmente moderadores de conducta, que la conexión entre nuestra deliberación y nuestra acción se extienden sobre el tiempo.

En cuanto a las creencias, Bratman asume que estas son de tipo booleano, y que no admiten grados de verdad, ni probabilidades subjetivas. Si tengo el plan de ir a Veracruz, mi creencia booleana, de que no tengo auto es importante para procesar el estándar de relevancia y el filtro de admisibilidad. Si sólo asigno probabilidades altas al hecho de que no tengo auto, entonces el plan que necesita de un auto confronta problemas de inconsistencia.

Las intenciones y los planes soportan la coordinación en parte porque proveen un sustento a nuestras expectativas de que serán correctamente ejecutados, por ejemplo, la intención de Lulú de ir a la fiesta provee soporte para la coordinación, porque provee sustento para la expectativa de Pedro de que Lulú estará en la fiesta. Las intenciones deben ser consistentes con las creencias. Violaciones a esta restricción constituyen una forma criticable de irracionalidad. Por otro lado , la intención de Pedro en darle un beso a Lulú, no requiere en efecto ni de la creencia por parte de Pedro en que tendrá éxito, ni tampoco de que fallará. Los casos donde un agente intenta A sin creer que logrará A se conocen como intención-creencia incompletos. Casos donde un agente intenta A creyendo que no logrará A se conocen como intención-creencia inconsistentes y están más cercanos a la irracionalidad, es decir, el hecho de que Pedro intenta besar a Lulú, cuando cree que no lo logrará suena más irracional que el hecho de intentar lo mismo creyendo que lo logrará. Es decir, el intentar algo que se cree posible suena más racional que intentar algo que no se cree posible.

3.3. Agentes BDI

Entre los modelos de agencia racional propuestos en IA, el modelo de agentes intencionales BDI (*Belief-Desire-Intention*) ha resultado de gran relevancia [35]. Esto obedece a que el modelo cuenta con sólidos presupuestos filosóficos, basados en la postura intencional de Daniel Dennett [23] y la teoría del planes, intenciones y razonamiento práctico de Michael Bratman [12]. Estas dos nociones de intencionalidad nos proveen con las herramientas para describir los agentes a un nivel adecuado de abstracción, al adoptar la postura intencional y definir funcionalmente a estos agentes de manera compatible con tal postura, como sistemas de razonamiento práctico. Ambas formas de intencionalidad han sido sido formalmente expresadas y estudiadas bajo diferentes lógicas multimodales de elegante semántica[1] [68,76,78]. Existen además diferentes implementaciones del modelo BDI, como IRMA [13], y los sistemas basados en PRS [34]: dMARS [25], Jam! [45] y Nuin [24], aplicados en tareas de diagnóstico para el transbordador espacial, control de procesos de manufactura y administrativos, control de aeropuertos y explotación de la información en el web (*semantic web*). En la figura 3.2 se muestra una arquitectura genérica BDI.

Una arquitectira desarrollada siguiendo el modelo BDI comúnmente consta de un conjunto de estructuras de datos, las cuales representan cada una a las creencias, deseos, eventos[2], planes, intenciones.

Michael Wooldridge [78] presenta diferentes algoritmos para el intérprete BDI, que corresponden a diferentes estrategias de compromiso del agente, por ej., agentes que nunca reconsideran sus intenciones (single-minded), agentes que reconsideran sus intenciones cuando creen que estas no son realizables (open-minded), etc. David Kinny and Michael Georgeff [48] presentan un estudio comparativo de dos estrategias identificadas como firme (single-minded) y precavida (open-minded). Ambas estrategias funcionan bien en ambientes que no son demasiado dinámicos, pero en caso contrario, es mejor ser precavido.

Un intérprete BDI opera en dos fases: una de creación de intenciones y otra de ejecución de las mismas. Y es necesario mencionar que los agentes BDI llevan a cabo dos tipos de razonamiento[3]. El razonamiento práctico que los lleva a formar intenciones y ejecutarlas.

[1] Los antecedentes de estos estudios sobre la utilización de operadores modales para representar verdad y creencia, pueden encontrarse en los estudios sobre la verdad de Saul Kripke [1, 49].

[2] La percepción de los agentes se mapea a eventos discretos, almacenados temporalmente en una cola de eventos (e.g., actualizaciones en la base de datos)

[3] En el capítulo siguiente se describe a detalle como se llevan a cabo estos tipos de razonamiento, por

Y el razonamiento epistémico, típico de la IA, al momento de validar si el contexto de un plan es consecuencia lógica de sus creencias. La técnica de razonamiento epistémico puede variar, por ej., unificación en demostración de teoremas en primer orden [32].

A continuación explicaremos a grandes rasgos la arquitectura de un agente BDI, inspirado en IRMA [4] [13]. También se adoptan algunas ideas fundamentales de Michael Wooldridge [78]. Este tipo de arquitectura puede verse como un agente tropístico (ver capítulo 2) en el cual el estado interno consiste en cuatro estructuras de datos:

- **Creencias.** Es el conjunto de la creencias actuales del agente, denotado por Bel. Las creencias se representan simbólicamente, como hechos en Prolog [77], es decir, literales cerradas de la lógica de primer orden. Para efectos de esta arquitectura abstracta no es necesario especificar la representación exacta de las creencias.

- **Deseos.** Es el conjunto de los deseos actuales del agente, denotad por Des. Al igual que con las creencias, los detalles de representación serán por el momento omitidos.

- **Biblioteca de planes.** Es el conjunto de planes que el agente tiene como recetas o procedimientos, denotado por $Plans$. Para denotar planes individuales usaremos π. La función $execute(\pi)$ toma un plan como argumento y lo ejecuta. La biblioteca de planes está incluida en $Bels$, puesto que los agentes creen sus planes como procedimientos, mientras que, siguiendo la tesis de asimetría, las intenciones no están incluidas en $Bels$.

- **Intenciones.** Es el conjunto de las intenciones actuales del agente, denotado por Int. Las intenciones son planes como estados mentales, esto es, planes tomados de la biblioteca e instanciados por contexto.

El ciclo de interacción de un agente basado en una arquitectura BDI es aproximadamente como sigue:

---

medio de lo que se le conoce como cabeza de planes, las cuales se integran por evento de disparo y un contexto.

[4] La arquitectura IRMA, diseñada por Martha E. Pollak, David Israel and Michael Bratman [13], fue la primer implementación de un sistema computacional basado en razonamiento práctico. De cualquier forma, la implementación mejor conocida de un sistema intencional es PRS, desarrollado por Michael Georgeff y Amy Lansky [33, 34] y sus diferentes re-implementaciones como UM-PRS [51], dMARS [25], or Jam! [45].

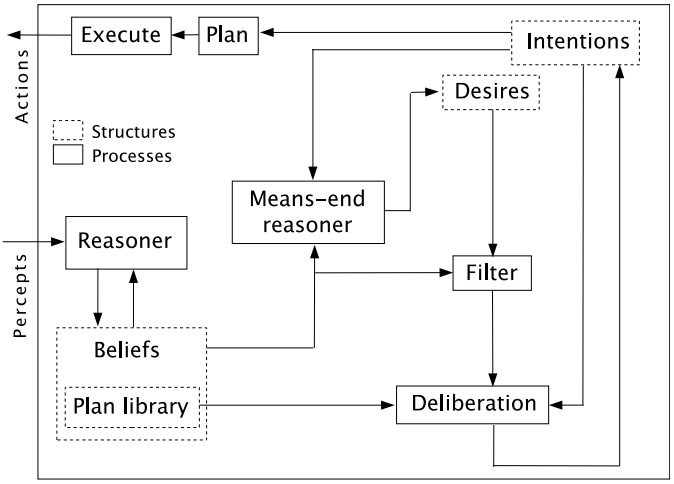

Figura 3.1: *Una arquitectura para agentes racionales acotados basada en IRMA [13].*

1. Aunque la percepción es normalmente empaquetada en paquetes discretos llamados perceptos. Por simplicidad, sólo se muestra en la figura 3.1 como una entrada al agente ligada directamente a las creencias.

2. Un agente actualiza sus creencias con una función de revisión de creencias basada en su percepción y sus creencias actuales. Algunas veces esta función se conoce como razonador (*reasoner*).

3. Un agente selecciona los planes relevantes usando un razonador de medios-fines (*means-end reasoner*), basado en las creencias actuales del agente y sus intenciones previas. Los planes seleccionados por esta función se consideran como los deseos del agente.

4. Los planes generados por el razonador de medios-fines son filtrados por el agente, basado en sus creencias actuales, sus deseos y sus intenciones previas. La función filtro (*filter*) mantiene la consistencia y restringe las opciones que serán consideradas en la deliberación. El filtrado debe ser computacionalmente acotado (restringido en tiempo).

5. Finalmente una función de *deliberación* selecciona las opciones que serán incorporadas como intenciones, basada en razones creencia-deseo y la biblioteca de planes.

Debido a que las intenciones están estructuradas como planes, una función *plan* es utilizada para seleccionar la intención que será ejecutada.

3.4. Lenguajes computables orientados a agentes BDI

Para comprender una de las principales problemáticas en torno al modelo BDI, primero es necesario hacer una distinción entre dos aspectos del mismo: el teórico y el práctico. Aunque la investigación en el campo de agentes se ha ocupado por abordar ambas perspectivas por igual, la existencia de un abismo entre el aspecto práctico y el teórico ha representado un escollo para el modelo BDI y otras nociones fuertes de agencia, desde su comienzo.

La separación entre teoría y práctica se debe, principalmente, a la complejidad computacional de los algoritmos solucionadores de teoremas y de verificación de modelos, los cuales resultan necesarios para implementar la expresividad de las lógicas usadas para modelar los agentes BDI.

Al respecto Anand S. Rao señala [68], que a pesar de existir un número considerable de implementaciones éxitosas usando agentes BDI, dentro de diversos dominios de aplicación, y para los cuales la reactividad y consistencia resultan críticas; la complejidad del código escrito para estos sistemas y ciertos supuestos [5] hechos para simplificar el modelo, hacen evidente que estos sistemas carecen de bases sólidas o de un fundamento teórico.

Las especificaciones lógicas formales han ofrecido poca luz en el aspecto pragmático. Los diseñadores de software optan por resolver los problemas que trae consigo la aplicación desde un enfoque totalmente de ingeniería. Dejando así, de lado la simplicidad y expresividad que trae consigo la abstracción inherente del modelo BDI [6]. En consecuencia, pareciera que ambos aspectos (el teórico y el práctico) divergieran al momento de desarrollarse dentro de la investigación.

La programación e implementación de agentes BDI toma un nuevo giro, a partir del trabajo de Anand S. Rao [68]. Él propone un lenguaje computable para agentes BDI, inspirado en la arquitectura genérica BDI que se muestra en la figura 3.2 (implementación de dMARS [25]), y desarrolla un lenguaje, cuya semántica operacional resulta ser una gran contribución para unificar los aspectos teórico y práctico, en lo que se refiere al modelo

[5] Los diseños de implementación hacen uso de un gran número de simplificaciones y supuestos, con la finalidad de modelar las actitudes de creencias, deseos e intenciones como estructuras de datos. También existen otras alternativas, como la escritura de planes y programas por el usuario, que pueden acelerar el proceso de computación en estos sistemas.

[6] Pasa del uso de lógicas multimodales para modelar agentes BDI, a un conjunto de reglas que definen un sistema de transiciones, el cual a diferencia de las primeras, es directamente computable.

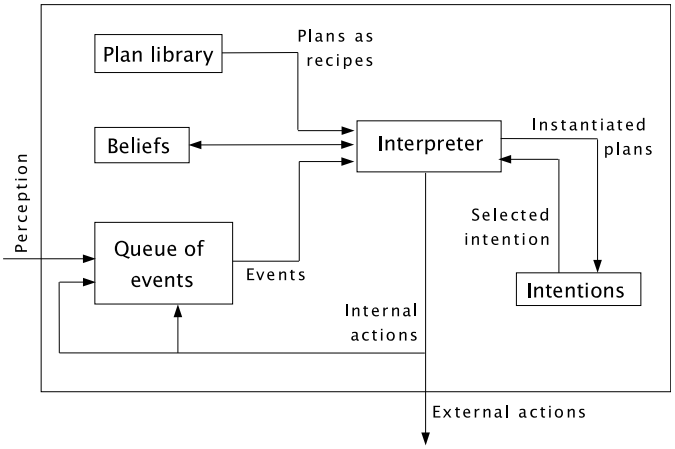

Figura 3.2: *Arquitectura genérica BDI basada en la implementación de dMARS [25].*

BDI. El lenguaje propuesto hace posible la escritura e interpretación de los programas de agente de forma similar a la lógica de programa de claúsulas de Horn.

Aquí encontramos una nueva tendencia, de unificar los aspectos teórico y práctico, mediante el desarrollo de lenguajes de progamación que sean computables, y directamente implementables. Esta última aproximación es la que usaremos para nuestro trabajo, al utilizar un lenguaje de programación orientado a agentes para poner en marcha la teoría de aprendizaje intencional en [36–39] y de aprendizaje multi-agente [11]. A continuación describiremos de forma detallada el lenguaje abstracto que hemos mencionado en esta sección.

3.5. AgentSpeak(L)

El lenguaje abstracto de programación AgentSpeak(L) [68] presenta un marco de trabajo elegante, basado en lógica de primer orden, para programar agentes BDI. Pero además proporciona una semántica operacional computable, la cual logra acercar considerablemente el aspecto teórico con el pragmático.

Un agente en AgentSpeak(L) se compone básicamente por:

- Un conjunto de creencias (*Belief Base*).
- Un conjunto de planes (*Plan Library*).

El lenguaje originalmente propuesto por Anand S. Rao en [68], se basa en los siguientes constructores:

- **Creencias:** Literales de base de la lógica de primer-orden. Constituyen la representación que cada agente tiene de su entorno.

- **Metas:** Se reducen a preguntarse (?) si una literal es verdadera (*test goal*) dadas las creencias del agente; y a lograr (!) que una literal se vuelva verdadera (*achieve goal*) a través de la actuación de los agentes.

- **Eventos disparadores:** El propósito de un agente es observar su medio ambiente, y con base en sus observaciones y sus metas, ejecutar ciertas acciones. Los agentes en AgentSpeak(L) son especialmente sensibles a los llamados eventos disparadores (*trigger events*), que cubren alteraciones en las creencias y metas del agente. Agregar (+) o eliminar (−) creencias y metas, constituyen las observaciones que mueven a un agente a elegir sus cursos de acción.

- **Acciones:** Las acciones son procedimientos que al ejecutarse cambian el estado del medio ambiente donde se encuentra el agente. Se representan por un nombre y un conjunto de parámetros que toman la forma de términos de la lógica de primer orden, pero se representan mediante símbolos para distinguirlos de los demás predicados.

- **Planes:** Toman la forma siguiente: *evento-disparador : contexto* ← *cuerpo*. El contexto es una conjunción de literales, que debe observarse como verdadera para que el plan pueda ser *aplicable*. El cuerpo del plan es una secuencia de metas o acciones.

Un agente en AgentSpeak(L) puede ser considerado como un sistema reactivo basado en planes. Los planes son disparados por la adición (+) o sustracción (−) de creencias debido a la percepción del ambiente, o de la adición o sustracción de metas como resultado de la ejecución de los planes ejecutados en eventos anteriores.

El mecanismo de funcionamiento de los agentes en AgentSpeak(L) está construido con base en sus creencias, deseos e intenciones: i) un agente tiene creencias, deseos (en términos de metas) e intenciones como planes adoptados; ii) los agentes mantienen un repositorio de los planes disponibles, conocido como biblioteca de planes; iii) un agente responde a cambios en sus metas o creencias, que resultan de la percepción, las cuales están empaquetadas dentro de estructuras de datos llamadas eventos, representando tanto nuevas creencias como nuevas metas; iv) los agentes responden a estos eventos por medio de seleccionar planes de la librería de planes para cada evento y posteriormente instanciando uno de esos planes como una intención; v) las intenciones comprenden a su vez acciones

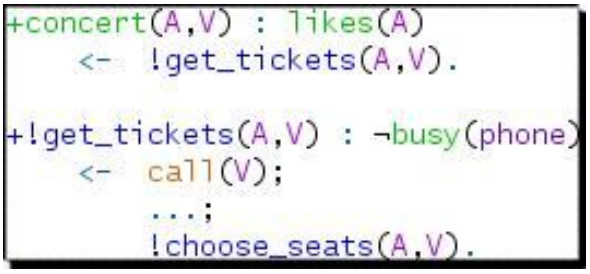

Figura 3.3: *Ejemplo de planes en AgentSpeak(L).*

y metas a ser alcanzadas, con la posibilidad de añadir posteriormente nuevos planes a esa intención.

En la figura 3.3 se ejemplifican dos planes formulados usando AgentSpeak(L) cuando un concierto por el artista A es anunciado en la locación V, es decir, desde la percepción del ambiente, por el agente, la creencia *concert(A, V)* es agregada, entonces el agente procede a ejecutar el cuerpo de este plan, el cual incluye la adición de la meta *get_tickets(A, V)* (conseguir los boletos para tal concierto); claro, siempre y cuando a nuestro agente le guste el artista A, *likes(A)*, es decir el contexto sea una consecuencia lógica de las creencias actuales del agente..

Un agente BDI consiste en una tupla $\langle E, B, P, I, A, S\rangle$ de conjuntos de eventos, creencias, planes, intenciones, acciones y funciones de selección, respectivamente. Donde las intenciones, son pilas de planes parcialmente definidos, cuya adopción responde a la percepción y metas del agente. AgentSpeak(L) define una teoría de prueba basada en un sistema de transición $\langle \Gamma \vdash\rangle$ donde Γ es un conjunto de configuraciones BDI y $\vdash$ una relación binaria de entre ellas.

3.5.1. Sintaxis

En AgentSpeak(L), como ya mencionamos, un agente *ag* se encuentra especificado simplemente por un conjunto de creencias *bs* (base de creencias) y un conjunto de planes *ps* (librería de planes); a continuación definimos la sintaxis de estos componentes.

Creencias

Cuando se especifica un conjunto de creencias en un programa de AgentSpeak(L), definimos aquello que el agente va a creer inicialmente acerca del ambiente donde éste ha sido situado. Posteriormente, conforme el ambiente cambia y el agente actúa, las creencias

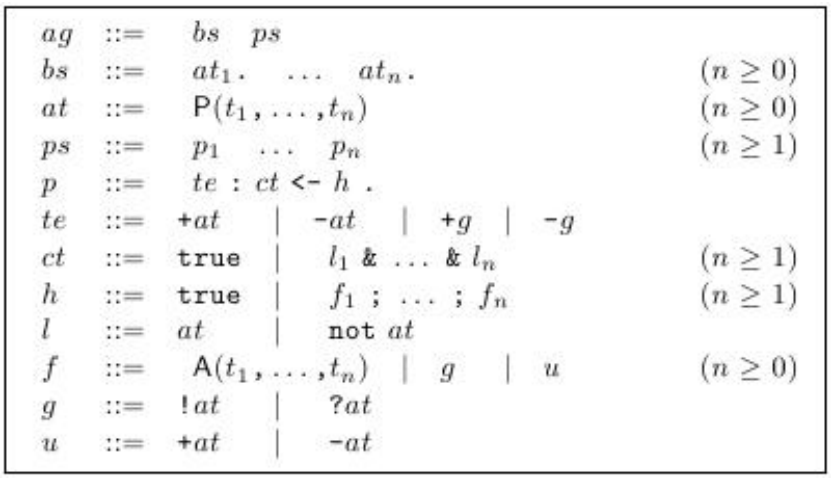

Figura 3.4: *La sintaxis de AgentSpeak(L) [8].*

del agente se irán actualizando de acuerdo a la percepción. A continuación definiremos sintácticamente un conjunto de creencias.

Las fórmulas atómicas del lenguaje son los predicados at definidos como $at \leftarrow P(t_1, \ldots, t_n)$ para $(n \geq 0)$; donde P es un símbolo de predicado, a es un símbolo de acción, y $t_1, \ldots, t_n$ son términos estándares de lógica de primer orden. El subconjunto de fórmulas de lógica de predicados de primer orden que se usa en los programas AgentSpeak(L) esta dado por:

$$\varphi \leftarrow at \mid \neg at \mid \varphi \wedge \varphi'.$$

Es importante señalar que las creencias (*beliefs*) son un conjunto de fórmulas atómicas, at, que no contienen variables (*grounded literals*); sintácticamente décimo denotamos $beliefs \leftarrow b_1 \cdots_n$ para $(n \geq 0)$.

Planes

El conjunto de planes ps denotado por ps en la figura 3.4 es una lista de planes p donde te es un evento de disparo, ct el contexto del plan y finalmente h que representa una secuencia de acciones, metas, y para el caso de la extensión realizada al lenguaje originalmente propuesto por Anand S. Rao hecha por Rafael Bordini y Jomi Fred Hübner, actualizaciones a la base de creencias. Comunmente $te : ct$ es referido como la cabeza del plan, y h como el cuerpo.

El cuerpo h del plan puede estar conformado por acciones, metas o actualizaciones a la base de creencias del agente ($A(t_1, \ldots, t_n)$, g, u respectivamente). Las acciones están conformadas por un símbolo de acción A y una lista de términos, si son necesarios. Las

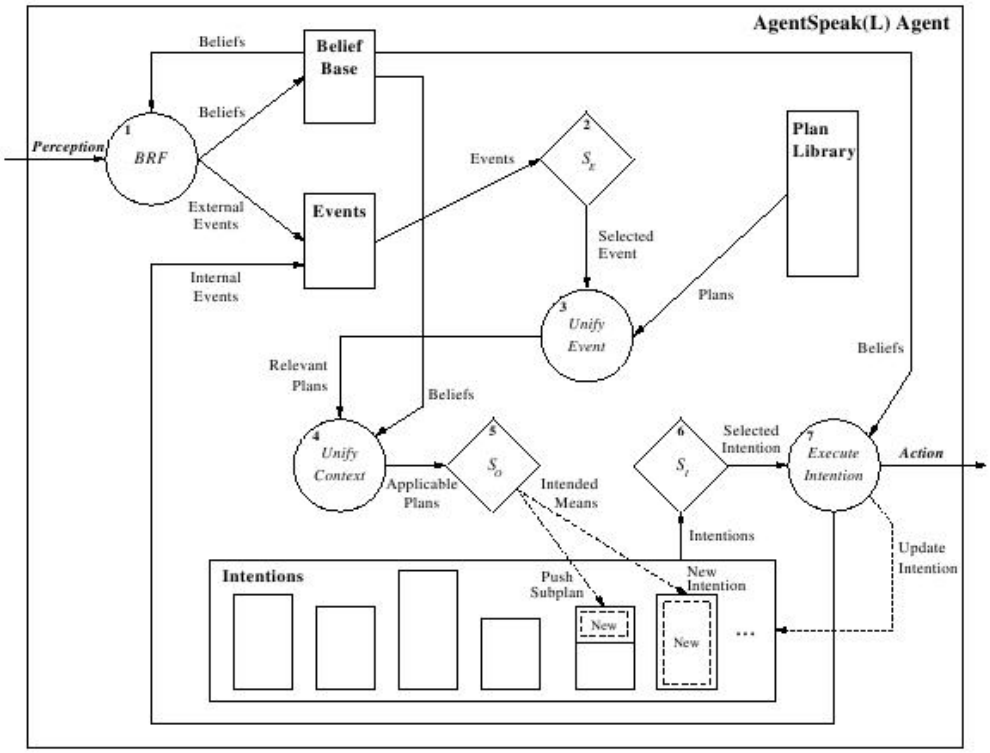

Figura 3.5: *Un ciclo de interpretación de un agente AgentSpeak(L) [54].*

metas pueden ser $!g$ o $?g$, éstas últimas son pruebas que el agente realiza en su base de creencias. Finalmente, las actualizaciones u se definen como $u \leftarrow +at \mid -at$.

3.5.2. Semántica

El intérprete de AgentSpeak(L) requiere, además, de tres funciones de selección[7]:

1. La función de selección de evento S_ε, la cual simplemente selecciona un evento del conjunto de eventos.

2. Una función S_O, la cual selecciona una opción del conjunto de planes aplicables.

3. S_I, función de selección de una intención en particular de un conjunto de intenciones.

Un algoritmo simplificado de la operación de un agente BDI, sería como sigue (las funciones de *selAplicable* y *selRelevante* son vistas en el algoritmo 5 como sub-funciones de S_O):

[7]Existe una extensión hecha por Bordini *et al.* reportada en [10]; AgentSpeak(XL), que incorpora la generación automática de funciones de selección eficientes.

Algoritmo 5 Un programa de agente con estado

```
1: while true do
2:     E ← Percepción()
3:     while E ≠null do
4:         e ← SelEvento(E)
5:         e ← SelAplicable(SelRelevante(B, P, e))
6:         e ← Actualiza(I, p)
7:     end while
8:     ejecuta(SelIntención(I)
9: end while
```

Las *intenciones* son cursos particulares de acción, con los que el agente se ha "comprometido" con la finalidad de manejar ciertos eventos. Cada inteción es una pila de planes parcialmente instanciados.

Los eventos que unifican con los eventos de disparo, te de los planes pueden ser de dos tipos:

- Externos, cuando éstos son originados por la percepción del agente sobre su ambiente
- Internos, si son generados por la ejecución de un plan perteneciente al agente (i.e., $+!g$).

Un ciclo del intérprete de AgentSpeak(L) inicia con la selección de un evento, luego selecciona los planes relevantes (aquellos planes incluidos en la librería para los cuales el evento unifica con el evento de disparo señalado en la cabeza del plan). A partir del conjunto anterior de planes se obtienen lo planes aplicables, que son aquellos para cuales el contexto es una consecuencia lógica de la base de creencias del agente[8].

La función de selección S_O, elige una opción del conjunto de planes aplicables, instancia como una intención más si el evento fue externo, o si éste fue interno empuja la pila de la intención que lo generó.

Finalmente, S_I selecciona una de las intenciones del conjunto de intenciones instanciadas y se ejecuta, realizando las actualizaciones necesarias tanto a la base de creencias como a la pila de la intención seleccionada (i.e., extraer la acción ejecutada).

[8]Note que la unificación que se lleva acabo para identificar los cursos de acción se hace mediante un proceso de razonamiento teórico, sin embargo, la postura de seleccionar aquellos planes que son relevantes y aplicables sigue un enfoque de razonamiento práctico [12], fundamento filosófico del modelo BDI.

En el siguiente capítulo se introducirá Jason: intérprete de una versión extendida del lenguaje AgentSpeak(L) implementado en Java. Presentaremos su semántica operacional como la arquitectura del componente BDI.

Capítulo 4

Jason: un intérprete de AgentSpeak(L)

Existen diversos sistemas que implementan agentes basados en el modelo BDI, sin embargo uno de los principales puntos a favor de Jason es el fundamento teórico que emplea. Con la implementación de Jason se busca probar ciertas características de los agentes BDI que sirvan para trabajar con la verificación formal de los agentes programados utilizando AgentSpeak(L). En la práctica, otra característica que tiene el intérprete Jason a su favor, que lo hace muy versátil, es el hecho de que está implementado en el lenguaje de propósito general Java, lo que le atribuye propiedades tales como ser ejecutado en diferentes plataformas y una fácil expansión, principalmente. Es importante mencionar que esta plataforma y todos sus componentes están distribuidos bajo licencia libre, en GNU LGPL.

En estricta comparación, por ejemplo con JADE (tenemos que éste último requiere ser programado directamente en Java, siendo más bien un paquete de clases y funciones de utilería para el desarrollo de sistemas basados en agentes BDI empleando el lenguaje abstracto AgentSpeak(L), mientras que Jason es una plataforma completa que permite interpretar directamente dicho lenguaje abstracto. A diferencia de los diversos sistemas que implementan agentes BDI, Jason es un lenguaje de más alto nivel.

El intérprete Jason implementa una semántica operacional extendida de AgentSpeak(L) [61], fue desarrollado en el lenguaje de propósito general Java, y su IDE soporta el desarrollo y la ejecución de sistemas multi-agentes distribuidos. Algunas características de Jason son:

- Actos de habla basados en comunicación entre agentes, incluyendo anotaciones en las creencias con información de las fuentes [**?**].
- Anotaciones sobre las etiquetas de los planes; las cuales pueden ser empleadas para indicar funciones de selección sofisticadas [10].
- La posibilidad de correr un SMA distibuido sobre una red, utilizando SACI o algún middleware [2].
- Funciones de selección totalmente configurables mediante Java.
- Posibilidad de extender el reportorio de acciones internas, programándolas directamente en código Java.
- Una clara noción de ambiente de SMA, que permite situar el sistema en cualquier ambiente, el cual puede ser implementado en Java.
- Incorpora un editor gráfico, jEdit que facilita el desarrollo de sistemas en Jason.

4.1. Sintaxis

La gramática presentada en la figura 4.1 detalla la sintaxis aceptada por Jason. $<ATOM>$ es un identificador que comienza con una letra minúscula ó el caracter '.'. $<VAR>$ es una variable, y ésta debe comenzar con una letra mayúscula. $<NUMBER>$ es cualquier entero o número con punto flotante. Y $<STRING>$ es una cadena de caracteres delimitada por comillas dobles.

Algunas de las principales diferencias de la sintaxis de Jason con la propuesta originalmente por Rao, para AgentSpeak(L) son:

- Las fórmulas atómica se extienden y se llaman literales; esta nueva definición permite, principalmente, anidar en los términos, por ejemplo de una acción, otro átomo.
- La sintaxis de Jason, a diferencia de AgentSpeak(L), permite el uso de anotaciones para las fórmulas atómicas, a pesar de hacer un poco más complicada la unificación, es posible indicar cosas como el origen de las percepciones.
- Es posible etiquetar los planes empleando '@'.
- La definición de anotaciones puede ser usada también en combinación con las etiquetas de los planes, es posible crear funciones de selección de acción más sofisticadas

```
agent             → ( init_bels | init_goals )* plans
init_bels         → beliefs rules
beliefs           → ( literal "." )*
rules             → ( literal ":-" log_expr ".")*
init_goals        → ( "!" literal "." )*
plans             → ( plan )*
plan              → [ "@" atomic_formula ] triggering_event [ ":" context ] [ "<-" body ] "."
triggering_event  → ( "+" | "-" ) [ "!" | "?" ] literal
literal           → [ "-" ] atomic_formula
context           → log_expr  |  "true"
log_expr          → simple_log_expr
                  |  "not" log_expr
                  |  log_expr "&" log_expr
                  |  log_expr "|" log_expr
                  |  "(" log_expr ")"
simple_log_expr   → ( literal | rel_expr | <VAR> )
body              → body_formula ( ";" body_forumula )*
                  |  "true"
body_formula      → ( "!" | "?" | "+" | "-" | "-+" ) literal
                  |  atomic_formula
                  |  <VAR>
                  |  rel_expr
atomic_formula    → ( <ATOM> | <VAR> ) [ "(" list_of_terms ")" ] [ "[" list_of_terms "]" ]
list_of_terms     → term ( "," term )*
term              → literal
                  |  list
                  |  arithm_expr
                  |  <VAR>
                  |  <STRING>
list              → "[" [ term ( "," term )* [ "|" ( list | <VAR> ) ] ] "]"

rel_expr          → rel_term ("<"|"<="|">"|">="|"=="|"\\=="|"=") rel_term
rel_term          → (literal|arithm_expr)
arithm_expr       → arithm_term [ ( "+" | "-" ) arithm_expr ]
arithm_term       → arithm_factor [ ( "*" | "/" | "div" | "mod" ) arithm_term ]
arithm_factor     → arithm_simple [ "**" arithm_factor ]
arithm_simple     → <NUMBER>
                  |  <VAR>
                  |  "-" arithm_simple
                  |  "(" arithm_expr ")"
```

Figura 4.1: *Sintaxis utilizada por el intérprete de Jason. [8].*

(i.e., *label[chanceSuccess(0.7),expectedPayoff(0.9)]*) note que las anotaciones también pueden ser pasadas al unificador.

Existen otras extensiones que resultan dirigidas en un sentido más práctico (e.g. el operador $-+$, el cual agrega una creencia inmediatamente despúes de remover la primera ocurrencia existente de esa creencia en la base de creencias del agente).

4.2. Semántica Operacional

En AgentSpeak(L) un estado particular de configuración C se define como una tupla $C =< I, E, A, R, Ap, \iota, \rho, \varepsilon >$ donde:

- I es un conjunto de *intenciones* $\{\imath, \imath', \dots\}$. Y cada *intención* $\imath$ es una pila de planes parcialmente instanciados $[p_1 \ddagger p_2, \dots, p_n]$. Se usa $\ddagger$ para separar los planes dentro de una intención.
- E es un conjunto de eventos $(te, i), (te', i'), \dots$. Cada evento es un par (te, i), donde te es un evento de disparo y la intención asociada a éste se denota con i.
- A es un conjunto de acciones.
- R es un conjunto de *planes relevantes*.
- Ap es un conjunto de *planes aplicables*.
- Cada configuración C tiene además tres componentes denotados por ι, ρ, ε. Estas variables mantienen un registro de la intención, evento y el plan aplicable, respectivamente, que son considerados a lo largo de la ejecución del agente.

Para presentar la semántica operacional de Jason es necesario antes definir una serie de funciones que facilitarán la descripción formal, éstas se detallan a continuación.

Lo primero que el intérprete de AgentSpeak(L) hace después de seleccionar un evento, es obtener una lista de los planes que resultan relevantes. Un plan se considera relevante en relación a un evento de disparo te si éste ha sido escrito para contender con ese evento. En la práctica, esto se verifica tratando de unificar la parte del evento de disparo del plan, con el evento de disparo que ha sido seleccionado del conjunto E para "resolverlo". En la definición siguiente se utiliza *mgu* para denotar el proceso de unificación que computa la mayoría de las sustituciones de dos eventos de disparo.

Definición 1 *Dados los planes de un agente (plans), y un evento de disparo (te), el conjunto* $RelPlans(plans, te)$ *de planes relevantes se obtiene como sigue:*

$$RelPlans(plans, te) = \{p\sigma \mid p \in plans \wedge \sigma = mgu(te, TrEv(p))\}$$

La selección de los planes aplicables, se realiza a partir del conjunto anterior (R), del cual se obtienen sólo aquéllos para los cuales su contexto resulta ser una consecuencia lógica de las creencias actuales del agente.

Definición 2 *Dado un conjunto* R *de planes relevantes y otro de creencias de un agente llamado* $beliefs$*, la lista de planes aplicables* $AppPlans(R, beliefs)$ *se encuentra definido como sigue:*

$$AppPlans(R, beliefs) = \{p\theta \mid R \in \wedge\theta ess.t.beliefs \models Ctxt(p)\theta\}$$

Recordemos que un agente en AgentSpeak(L) puede realizar pruebas a la base de creencias. En la práctica esto consiste en verificar si $?\varphi$ es una consecuencia lógica del conjunto de creencias del agente. Uno de los efectos de este tipo de *meta* es la producción de un conjunto de sustituciones a causa de la unificación.

Definición 3 *Dado el conjunto* $beliefs$ *representando las creencias de un agente, y una fórmula* φ *el conjunto de sustituciones* $Test(beliefs, \varphi)$ *producido por la prueba de* φ *en* $beliefs$ *es como sigue:*

$$Test(beliefs, \varphi) = \{\theta \mid beliefs \models \varphi\theta\}$$

Para efectos de describir las reglas semánticas para AgentSpeak(L), se adopta la siguiente notación:

- Se señala C como una configuración de AgentSpeak(L); para hacer referencia al componente E de C escribimos C_E. De manera similar para todos los demás componentes de C.
- Se emplea $C_\imath = \emptyset$ para indicar que no hay ninguna intención siendo considerada en la ejecución del agente. De forma similar para C_p y C_ε.
- Se usa $i[p]$ para hacer referencia a la intención que tiene al plan p en la cima de la pila que la contiene.

Debido a que los planes comúnmente se integran por cabeza y cuerpo, y se hace referencia a estos dos componentes por separado, es necesario definir formalmente dos auxiliares sintácticos, asumiendo que p es un plan de la forma $t : \varphi \leftarrow h$, podemos decir que:

- $TrEv(p) = t$ Devuelve la parte del plan p que corresponde al evento de disparo en t.
- $Ctxt(p) = \varphi$ Devuelve la parte del plan p que corresponde al contexto en φ.

En su extensión al lenguaje AgentSpeak(L), A. Moreira y R. Bordini [61] presentan la semántica de Jason organizada en una serie de reglas.

Selección de Evento: La primera regla asume la existencia de una función de selección de evento S_E que selecciona un evento del conjunto E.

$$\textbf{SelEnv}\ \frac{S_E(C_E)=(te,i)}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon = \emptyset, C_{Ap} = C_R = \{\}$$

A continuación, el evento seleccionado es removido de E y es asignado a el componente ε de la configuración.

$$C'_E = C_E - (te, i)$$
$$C'_\varepsilon = (te, i)$$

Planes Relevantes: La regla $\textbf{Rel}_1$ inicializa el componente R con el conjunto de planes relevantes. Si no existe ningún plan que sea relevante, el evento es descartado del componente ε por la regla $\textbf{Rel}_2$.

$$\textbf{Rel}_1\ \frac{RelPlans(plans,te)\neq\{\}}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon = (te, i), C_{Ap} = C_R = \{\}$$

$$C'_R = RelPlans(plans, te)$$

$$\textbf{Rel}_2\ \frac{RelPlans(plans,te)=\{\}}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon = (te, i), C_{Ap} = C_R = \{\}$$

$$C'_R = \emptyset$$

Planes Aplicables: La primera regla en este caso incializa el componente Ap con el conjunto de planes aplicables. En caso de no existir plan aplicable, el evento es descartado de φ por la segunda regla de **Appl**. Note que en ambos casos el componente R es inicializado.

$$\textbf{Appl}_1\ \frac{AppPLans(C_R,beliefs)\neq\{\}}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon \not=\emptyset, C_{Ap} = \{\} =, C_R \not=\{\}$$

$$C'_R = \{\}$$
$$C'_A p = AppPlans(C_R, beliefs)$$

$$\textbf{Appl}_2\ \frac{AppPLans(C_R,beliefs)=\{\}}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon \not=\emptyset, C_{Ap} = \{\} =, C_R \not=\{\}$$

$$C'_R = \{\}$$
$$C'_\varepsilon = \emptyset$$

Selección del Plan Aplicable: Esta regla asume la existencia de una función de selección S_{Ap}, la cual selecciona un plan a partir del conjunto Ap de planes aplicables.

$$\textbf{SelApplic}\ \frac{S_{Ap}(C_{Ap})=p}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon \not= \emptyset, C_{Ap} \not= \{\}$$

Después de que ha sido seleccionado uno de los planes aplicables, éste se asigna al componente p de la configuración y el conjunto de planes aplicables es descartado.

$$C'_p = p$$
$$C'_{Ap} = \{\}$$

Preparando el conjunto de Intenciones: Recordemos que Jason se distingue dos tipos de eventos, *internos* y *externos*. Los primeros se originan cuando éstos han sido generados por un plan que se encontraba previamente en ejecución, mientras que los últimos son generados a partir de la percepción del agente. La regla **ExtEv** contiende con los eventos externos, y dice que si el evento ε es externo (indicado por T en el argumento que señala la intención asociada a ε), una nueva intención es creada conformada por un sólo plan (p) anotado en C_p

$$\textbf{ExtEv}\ \frac{}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon = (te, T), C_p = p$$

$$C'_I = C_I \cup \{[p]\}$$
$$C'_\varepsilon = \emptyset$$
$$C'_p = \emptyset$$

Por otro lado, si el evento que se está manejando resulta ser interno, la regla **IntEv** manda que el plan en p debe ser colocado en la cima de la intención asociada con el evento (i). En ambos casos, tanto el evento como el plan pueden ser descartados de los componentes ε y $\imath$, respectivamente.

$$\textbf{IntEv}\ \frac{}{C,beliefs \rightarrow C',beliefs} \quad C_\varepsilon = (te, i), C_p = p$$

Note que en esta regla, la intención i completa que generó el evento interno necesita ser insertada nuevamente en C_I, con p como su cima.

$$C'_I = C_I \cup \{i[p]\}$$
$$C'_\varepsilon = \emptyset$$
$$C'_p = \emptyset$$

Selección de Intención: Esta regla hace uso de una función de selección, que debería estar previamente definida, para seleccionar una intencón a partir del componente I de la configuración.

$$\textbf{SelInt}\ \frac{S_I(C_I)=i)}{C,beliefs \rightarrow C',beliefs} \quad C_\imath = \emptyset$$

$$C'_\imath = i$$

Ejecutando el Cuerpo de los Planes: El grupo que se describirá a continuación, expresa el manejo de la ejecución de los planes. El plan siendo ejecutado es siempre aquel que se encuentra en la cima de la intención (las intenciones están representadas por una estructura de datos, una una pila de planes) que ha sido previamente seleccionada. Todas las reglas en este grupo terminan descartando el componente $\imath$. Es decir, después de aplicar cualquiera de estas reglas, otra intención puede ser seleccionada eventualmente.

El cuerpo de los planes puede estar conformado por tres tipos diferentes de fórmulas (acciones, metas y actualizaciones a la base de creencias). En el caso de que la primera fórmula del cuerpo fuera una acción, se aplica la siguiente regla (**Action**), la cual simplemente agrega la acción a en el conjunto de acciones, A. Inmediatamente después a es removida del cuerpo del plan, y la intención es actualizada para reflejar este cambio.

$$\textbf{Action}\ \frac{}{C,beliefs \rightarrow C',beliefs} \quad C_\imath = i[head \leftarrow a; h']$$

$$C'_\imath = \emptyset$$
$$C'_A = C_A \cup \{a\}$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i[head \leftarrow h']\}$$

La siguiente regla registra un evento interno nuevo, en el conjunto eventos E, el cual también podrá ser seleccionado a causa de la regla **SelEv**.

$$\textbf{Achieve}\ \frac{}{C,beliefs \rightarrow C',beliefs} \quad C_\imath = i[head \leftarrow !at; h]$$

$$C'_\imath = \emptyset$$
$$C'_E = C_E \cup \{(+!at, C_\imath\}$$
$$C'_I = (C_I - \{C_\imath\})$$

Observe como la intención que generó el evento interno es removida del conjunto de intenciones C_I Esto denota la idea de *suspender* intenciones. Si la fórmula siendo ejecutada actualmente es una meta del tipo $!g$, entonces un plan (puede ser una intención completa), hecha para manejar el evento interno generado por la regla **Achievement**, debe empujar a la intención actual. Y sólo cuando el curso de acción definido para dicho evento ha sido terminado se puede continuar con la ejecución de la intención que había sido *suspendida*, a partir de la siguiente fórmula del cuerpo del plan.[1].

Cuando la fórmula al principio del plan actualmente siendo ejecutado es del tipo $?\varphi$, aplican dos reglas. Ambas intentarán, a partir de la función predifinida $Test(beleifs, \varphi))$, producir un conjunto de sustituciones que pueda hacer a φ una consecuencia lógica de $beliefs$. La regla $\mathbf{Test}_1$ determina, básicamente que nada se hace si ninguna sustitución para lo anterior es encontrada.

$$\mathbf{Test}_1 \ \frac{Test(beliefs,\varphi)=\{\}}{C,beliefs \rightarrow C',beliefs} \qquad C_\imath = i[head \leftarrow ?at\varphi; h]$$

$$C'_\imath = \emptyset$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i[head \leftarrow h]\}$$

La siguente regla se aplica cuando $Test(beleifs, \varphi))$ no está vacío. Indica la aplicación de una de las sustituciones al plan.

$$\mathbf{Test}_2 \ \frac{Test(beliefs,\varphi)\neq\{\}}{C,beliefs \rightarrow C',beliefs} \qquad C_\imath = i[head \leftarrow ?at\varphi; h]$$

$$C'_\imath = \emptyset$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i[(head \leftarrow h)\theta]\}\theta \in Test(belief s, \varphi)$$

Una de las extensiones hechas a lenguaje propuesto originalmente por A. S. Rao en [68] es la siguiente regla, la cual permite, dentro del cuerpo de los planes realizar actualizaciones a la base de creencias. Descritas en la sintaxis por $+at$ y $-at$.[2]

La primera de las siguientes reglas simplemente agrega un nuevo evento a el conjunto de eventos E. Y la meta $+at$ es removido del cuerpo del plan y como en los demás casos, la pila que conforma la intención es actualizada apropiadamente.

[1] A partir de la versión 0.9.4 se implementa un nuevo operador para indicar sub-meta !!. Éste es un nuevo tipo de fórmula, que no suspende las intenciones, si no crea una nueva pila.

[2] Con esta extensión se le permite al agente actualizar internamente (no por medio de la percepción) su base de creencias.

$$\textbf{AddBel}\ \frac{}{C,beliefs \rightarrow C',beliefs'} \quad C_\imath = i[head \leftarrow +at; h]$$

$$C'_\imath = \emptyset$$
$$beliefs \models at$$
$$C_E \cup \{(+at, C_\imath)\}$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i[head \leftarrow h']\}$$

Para eliminar una creencia del la base de creencias del agente, se procede de manera similar. Note la diferencia radica en que ahora *at* no debe ser una consecuencia lógica de $beliefs$. [3]

$$\textbf{DelBel}\ \frac{}{C,beliefs \rightarrow C',beliefs'} \quad C_\imath = i[head \leftarrow -at; h]$$

$$C'_\imath = \emptyset$$
$$beliefs \not\models at$$
$$C_E \cup \{(-at, C_\imath)\}$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i[head \leftarrow h']\}$$

Para concluir con la serie de reglas, la semántica operacional de Jason define dos reglas más, las cuales llaman en su trabajo, *clearing house rules*. $\textbf{ClearInt}_1$ simplemente remueve una intención del conjunto de intenciones de un agente cuando no hay más que hacer al respecto, es decir, ya no quedan más fórmulas (acciones o metas) que ejecutar en el cuerpo del plan.

$$\textbf{ClearInt}_1\ \frac{}{C,beliefs \rightarrow C',beliefs'} \quad C_\imath = [head \leftarrow]$$

$$C'_\imath = \emptyset$$
$$C'_I = C_I - \{C_\imath\}$$

Cuando una intención es *suspendida*, recuerde que es necesario al final remover la intención que ha empujado a la primera. La siguiente regla se indica lo anterior.

$$\textbf{ClearInt}_2\ \frac{}{C,beliefs \rightarrow C',beliefs'} \quad C_\imath = i'[head' \leftarrow !at; h1 \ddagger head \leftarrow]$$

$$C'_\imath = \emptyset$$
$$C'_I = (C_I - \{C_\imath\}) \cup \{i'[head' \leftarrow h']\}$$

[3] Lo anterior indica de esta manera, porque se asume en la arquitectura que existe un verificador de consistencia de creencias, observe que no es tan simple como simplemente retirar *at* del conjunto $beliefs$, aunque ésta es la forma en la que lo hace en la práctica el intérprete Jason.

4.3. Semántica de comunicación

El mecanismo de comunicación que permite el intercambio de mensajes en Jason, se incorpora como parte de la arquitectura del agente y no como un componente de su razonamiento práctico. El envío de un mensaje corresponde a ejecutar la acción interna predefinida *.send* que aparece en el cuerpo de algún plan. La arquitectura del agente debe asegurar, en todo caso, que los mensajes serán correctamente entregados a sus destinatarios correspondientes.

Con el propósito de incorporar la capacidad de procesar comunicación en los agentes AgentSpeak(L), es necesario extender la semántica operacional original propuesta para Jason. Primero es necesario definir una estructura más dentro de la tupla de configuración C, la cual servirá como buzón de mensajes, denotado como M. Se asume el que la arquitectura de agente prevee un mecanismo para recibir y enviar mensajes de forma asíncrona, y que los mensajes son guardados en el buzón de mensajes M para su procesamiento al inicio de cada ciclo de razonamiento. Es necesaria además la existencia de una función de selección de mensaje S_M.

El formato de los mensajes es:

$$\langle Ilf, id, content \rangle$$

dónde

$$Ilf \in \{Tell; Untell; Achieve; Unachieve, TellHow, UntellHow\}$$

es la fuerza ilocutiva asociada al mensaje; id identifica al agente que ha enviado el mensaje (en la práctica esto es indicado por medio de anotaciones a los predicados); y $content$, el cual puede ser una proposición atómica at o un plan p.

Finalmente, para poder definir las reglas semánticas de comunicación, serán necesarias dos funciones más. $Trust(id)$, la cual regresa verdadero si id es identificado como una fuente confiable de información, es decir, el agente receptor confia en quien envía el mensaje. Se emplea para decidir si los mensajes $Tell$ serán procesados. $Power(id)$, es verdadero si el agente presenta una relación de subordinación con el agente emisor (id). Para todo caso en que $Power(id)$ sea verdadero, aquellos mensajes $Achieve$ deberían respetarse. La noción de emplear estás funciones se explica detalladamente en [9].

A continuación se presentan las reglas semánticas que extienden AgentSpeak(L) para incorporar el procesamiento de la recepción de mensajes expresando actos de habla por un agente AgentSpeak(L) [62].

Recepción de un mensaje de tipo $Tell$**:**

$$\textbf{TellRec} \qquad \frac{S_M(C_M) = \langle Tell, id, at\rangle \ Trust(id)}{\langle ag, C\rangle \rightarrow \langle ag, C'\rangle}$$

$$\begin{aligned} donde: \qquad & C'_M = C_M \backslash \{\langle Tell, id, at\rangle\} \\ & creencias' \models \begin{cases} at[sources \cup \{id\}], & si\ creencias \models at[sources] \\ at[id], & en\ otro\ caso \end{cases} \\ & C'_E = C_E \cup \{\langle +at[id], T\rangle\} \end{aligned}$$

El contenido del mensaje es agregado a la base de creencias del agente receptor. Note que si la información ya estaba en la base de creencias, el agente que envía el mensaje es agregado a la lista de *sources* (como anotación) para dar crédito a esa creencia.

Recepción de un mensaje de tipo $UnTell$

$$\textbf{UnTellRec} \quad \frac{S_M(C_M) = \langle Tell, id, at\rangle \ Trust(id)}{\langle ag, C\rangle \rightarrow \langle ag', C'\rangle}$$

$$\begin{aligned} donde: \qquad & C'_M = C_M \backslash \{\langle UnTell, id, at\rangle\} \\ & bs' \not\models at[id], \qquad ifbs \models at[id] \\ & bs' \models at[sources \backslash \{id\}], \qquad ifbs \models at[sources] \\ & C'_E = C_E \cup \{\langle -at[id], T\rangle\} \end{aligned}$$

La regla anterior simplemente elimina de la lista de *sources* el nombre del agente que envía el mensaje, para restar acreditación a la creencia. Si el emisor es el único en la lista, entonces la creencia se remueve de la base de creencias del agente receptor.

Recepción de un mensaje de tipo $AchieveRec$

$$\textbf{AchieveRec} \quad \frac{S_M(C_M) = \langle Achieve, id, at\rangle \ Power(id)}{\langle ag, C\rangle \rightarrow \langle ag, C'\rangle}$$

$$\begin{aligned} donde: \qquad & C'_M = C_M \backslash \{\langle Achieve, id, at\rangle\} \\ & C'_E = C_E \cup \{\langle +!at[id], T\rangle\} \end{aligned}$$

En la regla anterior, el agente agregará como un evento externo a la pila C_E de eventos una meta del tipo $+!at$, dónde at es el contenido proposicional asociado al mensaje en *content*. Lo anterior sólo en caso de que $Power(id)$ sea verdadera.

Recepción de un mensaje de tipo $UnAchieveRec$

$$\textbf{UnAchieveRec} \quad \frac{S_M(C_M) = \langle UnAchieve, id, at\rangle \ Power(id)}{\langle ag, C\rangle \rightarrow \langle ag, C'\rangle}$$

$$donde: \qquad C'_M = C_M \backslash \{\langle UnAchieve, id, at\rangle\}$$
$$C'_E = C_E \cup \{\langle -!at[id], T\rangle\}$$

De la misma manera, la regla anterior dice al agente que elimine de su intención la meta $!at$, y se logra agregando el elemento $-!at$ al componente C_E del agente receptor.

Recepción de un mensaje de tipo $TellHow$

$$\textbf{TellHowRec} \quad \frac{S_M(C_M) = \langle TellHow, id, p\rangle \ Trust(id)}{\langle ag, C\rangle \rightarrow \langle ag', C'\rangle}$$

$$donde: \qquad C'_M = C_M \backslash \{\langle TellHow, id, p\rangle\}$$
$$ps' = ps \cup \{p\}$$

El fuerza de ilocución $TellHow$ tiene su fundamento en los sistemas de planeación reactiva y el concepto de planes de forma equivalente a la noción de *know-how* de Singh [75]. Los mensajes de tipo $TellHow$ son empleados cuando un agente quiere informar a otro el procedimiento para manejar cierto tipo de eventos. Si la fuente es de confianza para el agente receptor, el plan (*content*) simplemente es agregado a la biblioteca de planes por el agente. También existe la posibilidad de que una fuente externa encuentre que un plan ha dejado de ser válido o eficiente para manejar el evento que se supone debería manejar. Para informar a un agente sobre lo anterior se tiene el tipo de mensaje $UnTellHow$, que simplemente borra de la librería de planes el plan dentro de *content* sólo si $Trust(id)$ es verdadero.

Recepción de un mensaje de tipo $UnTellHow$

$$\textbf{UnTellHowRec} \quad \frac{S_M(C_M) = \langle UnTellHow, id, p\rangle \ Trust(id)}{\langle ag, C\rangle \rightarrow \langle ag', C'\rangle}$$

$$donde: \qquad C'_M = C_M \backslash \{\langle UnTellHow, id, p\rangle\}$$

$$ps' = ps \backslash \{p\}$$

Capítulo 5

Aprendizaje y árboles de decisión

El aprendizaje puede ser visto como una actualización en el comportamiento, habilidades o conocimiento en general con la finalidad de mejorar el desempeño. El aprendizaje más común es el que hacemos nosotros los humanos a través de las observaciones o la experiencia. A este tipo de aprendizaje se le llama aprendizaje inductivo.

Existe una clasificación general en el campo de la IA basada en el tipo de información disponible para aprender:

- **Supervisado**: consiste en aprender una función a partir de ejemplos de sus entradas y sus salidas.
- **No supervisado**: aquí se aprende a partir de patrones de entradas para los que no se especifican valores de sus salidas.
- **Por refuerzo**: también llamado aprendizaje por recompensa. Es aquél en el que aquellos cursos de acción que resultan exitosos o en beneficio para el agente se ven favorecidos mediante una premiación. Posteriormente el agente optará por aquellos cursos de acción que presenten una mayor esperanza de beneficio.

Para explicar mejor en que consiste el aprendizaje inductivo dentro del área de Aprendizaje Máquina denotamos un ejemplo como un par $(x, f(x))$, donde x es la entrada, y $f(x)$ es la salida de la función f aplicada a x. Se le llama inducción al proceso de encontrar una función h que aproxime a f, dado un conjunto ejemplos de f.

La hipótesis aprendida h, en lógica proposicional, es comunmente expresada mediante una disyunción de conjunciones proposicionales a las que se les llama árboles de decisión. Por otro lado, los árboles lógicos, un conjunto de disyunciones de conjunciones, son emplea-

dos para representar h en lógica de primer orden. A continuación explicaremos brevemente estas dos formas de representación.

5.1. Árboles de Decisión

La representación por medio de árboles de decisión resulta muy natural para los seres humanos. Realizar inducción por medio de árboles de decisión es uno de los métodos más sencillos y exitosos para construir algoritmos de aprendizaje. Un árbol de decisión toma como entrada una situación descrita a través de un conjunto de atributos y devuelve una decisión: el valor previsto de la salida dada la entrada. Es decir, hace una clasificación para cada ejemplo. Un árbol de decisión evalúa cada entrada (ejemplo) con una serie de pruebas para poder alcanzar una clasificación (decisión de acuerdo a su etiqueta). Cada nodo interno del árbol corresponde con una prueba sobre el valor de una de las propiedades, y las ramas que salen del nodo están etiquetadas con los posibles valores de dicha propiedad. Cada nodo hoja del árbol, indica un valor de etiqueta que ha de ser asignado al ejemplo que está siendo evalúando en caso de que ese nodo hoja sea alcanzado.

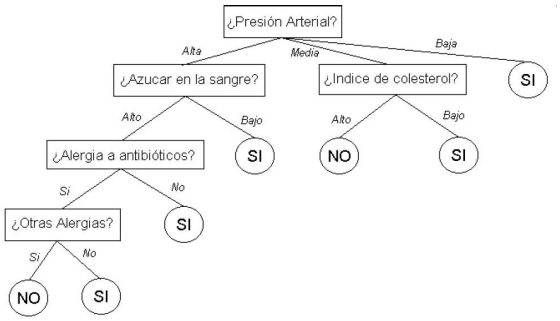

Figura 5.1: *Un árbol de decisión para hacer un diagnóstico.*

Desde un punto de vista lógico, un árbol de decisión puede verse como un conjunto de disyunciones de conjunciones, donde cada elemento de la lista, si es verdadero, clasifica los ejemplos con una etiqueta del conjunto de los nodos hoja. Por ejemplo, el árbol anterior se puede expresar como:

$pres(baja) \vee$
$pres(media) \wedge col(bajo) \vee$
$pres(alta) \wedge azuc(baja) \vee$
$pres(alta) \wedge azuc(alta) \wedge aler\_ant(no) \vee$

$pres(alta) \wedge azuc(alta) \wedge aler\_ant(si) \wedge aler\_otra(no) \vee$
$\leftarrow label(si)$

$pres(media) \wedge col(alto)$
$\vee pres(alta) \wedge azuc(alta) \wedge aler\_ant(si)$
$\leftarrow label(no)$

Existe una extensión a los árboles de decisión para su representación en lógica de primer orden: los árboles lógicos de decisión.

Un árbol lógico de decisión es un árbol binario de decisión donde:

- Los nodos del árbol contienen una conjunción de literales en lógica de primer orden.
- Nodos diferentes pueden compartir variables, bajo las siguientes restricciones: no es posible que una variable introducida en algún nodo ocurra en el lado derecho de ese mismo nodo, debido a que el lado derecho del sub-árbol es relevante sólo cuando la conjunción falla, y así referencias futuras a x en ese árbol no tienen sentido.

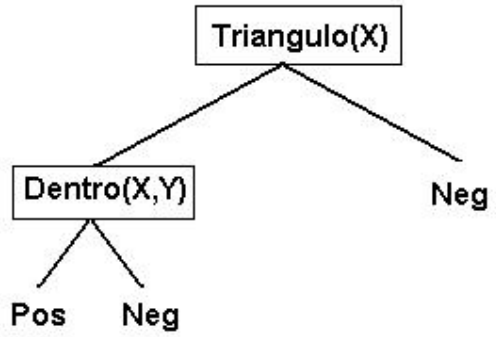

Figura 5.2: *Un árbol lógico de decisión.*

La versión lógica del árbol anterior (ver figura 5.2 sería:
$Triangulo(X) \wedge Dentro(X, Y) \leftarrow label(Pos)$

Observemos que esta última representación de árboles de decisión en primer orden resulta ser más intuitiva (expresiva) al momento de pensar en la hipótesis que podría hacer a un plan ejecutable (contexto).

Un algoritmo para crear este tipo de árboles es TILDE (*Top-down Induction of Logical Decision* [5]: un sistema de inducción *Top-Down* de árboles lógicos de decisión, basado en el método PLI de aprendizaje por interpretaciones [6]. Éste escala el algoritmo de

ID3 a una representación de primer orden. El aprendizaje por interpretaciones explota la suposición local. Toda la información que es relevante para un ejemplo es localizada en dos formas: i) la información contenida en los ejemplos es separada de la información contenida en el conocimiento *background*; ii) la información en un ejemplo es separada de la información de otros ejemplos. Con lo anterior se logra extender la expresividad del aprendizaje proposicional al mismo tiempo que mantiene su eficiencia.

5.2. Aprendizaje en Agentes

El Aprendizaje Máquina (AM) es el campo central de la IA que estudia los aspectos computacionales del aprendizaje. Aquí nos referimos particularmente a la interacción entre AM y el concepto de agencia, incluyendo SMA e Inteligencia Artificial Distribuida. Esta perspectiva es conocida en la literatura como aprendizaje SMA. Existe un interés creciente en la investigación de metodos de aprendizaje SMA; el comportamiento flexible remarcado por el concepto de agente en la IA y la inherente complejidad de los sistemas multi-agentes son dos de las principales razones.

Stuart Russell y Peter Norvin han propuesto una arquitectura genérica de agente para aprendizaje compuesta de cuatro elementos [71]:

- El elemento a ser mejorado, el cual describe en este caso a un agente que no incorpora aprendizaje.
- Un elemento que juega el rol de crítico, encargado de retroalimentar el elemento de aprendizaje, i.e., señalar qué tan bien el agente está siendo mejorado.
- Generador de problemas, responsable de sugerir acciones que conducirán a nuevas experiencias con información valiosa para el aprendizaje.
- Elemento de aprendizaje, que tiene el papel de hacer las mejoras en el agente.

Entendemos aquí por aprendizaje, la adquisición de una definición general a partir de una serie de ejemplos de entrenamiento etiquetados como negativos o positivos, es decir, inducir una función a partir de ejemplos de entrenamientos. El concepto de aprendizaje es central dentro del campo de la inteligencia artificial: un sistema que aprende es usualmente considerado como inteligente y vice-versa, un sistema considerado como inteligente, es normalmente considerado capaz de aprender [35]. Dentro del contexto de agencia se entiende por aprendizaje la adquisición de conocimientos y habilidades por un agente, al cual

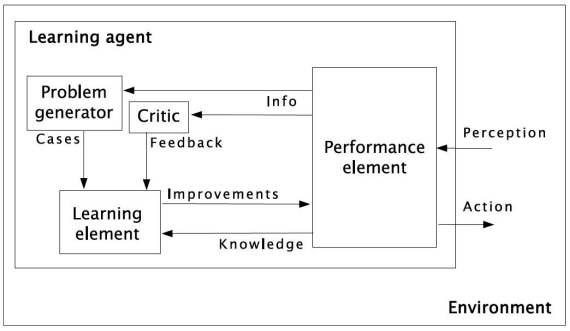

Figura 5.3: *Arquitectura abstracta de aprendizaje [71]*

llamamos *agente aprendiz*, bajo los siguientes términos: i) esta adquisición es conducida por el propio agente; ii) el resultado de esta adquisición de conocimiento es incorporado por el agente aprendiz en sus actividades futuras; iii) esta incorporación de conocimientos y habilidades lleva al agente a lograr un mejor rendimiento [74].

Principalmente distinguimos dos tipos de aprendizaje:

- **Aprendizaje Centralizado**: Este tipo de aprendizaje se caracteriza por que todo el proceso implicado en la adquisición de conocimiento es ejecutado por un solo agente. Aquí no es necesaria la interacción con otros agentes. Es posible que el agente se encuentre situado en un MAS, sin embargo, el proceso de aprendizaje se lleva a cabo como si este estuviera sólo.

- **Aprendizaje Distribuido**: También se le conoce como aprendizaje interactivo. Aquí, varios agentes se encuentran implicados en el proceso de aprendizaje. Puede ser visto como un grupo de agentes tratando tratando de resolver el mismo problema a manera de equipo. Este tipo de aprendizaje se base principalmente en las capacidades particulares (explotar las habilidades) que tiene cada agente.

Existe una clasificación para el aprendizaje, basada en el *nivel de circunspección*[1] que tiene cada agente, es decir, en que forma modela a su entorno social, y el comportamiento de los otros agentes [35, 37]:

- **Nivel 1**: dónde los agentes aprenden a partir de sus propias interacciones con el ambiente, sin intervención directa con ningún otro agente. Aunque los cambios hechos

[1] En inglés *awareness level*.

en el ambiente pueden ser vistos por los demás por cualquier agente, a través de la percepción.

- **Nivel 2**: aquí existe una interacción directa entre los agentes por medio del intercambio de mensajes.
- **Nivel 3**: a este nivel el aprendizaje a partir de la observación de las acciones tomadas por otros agentes también es considerado.

Existe evidencia en el trabajo de Manisha Mundhe and Sandip Sen [63] que prueban que la rapidez y efectividad del aprendizaje mejoran arriba del nivel 2 para simulación y en aprendizaje por reforzamiento.

5.3. ¿Aprendizaje Incremental? El paradigma de Agentes Sapientes

Parece ser que la clave de la inteligencia en el ser humano recae en gran medida sobre su capacidad para inferir relaciones causa-efecto a partir de la experiencia [40]. El aprendizaje incremental da a los sistemas dinámicos robustez y adaptabilidad; provocando comportamientos que exhiben mayor grado de inteligencia. Por ejemplo, un agente que implemente aprendizaje por inducción de manera incremental puede modificar su estado de cognición, y con ello su comportamiento, y permanecer más consistente con su ambiente.

Nos centramos principalmente en conectar la idea de efecto-causa con la de función inversa, y a su vez con la de aprendizaje. En un sistema experto, por ejemplo, podemos llamar al conjunto de los valores de los atributos y al valor de la clase, causa y efecto respectivamente. Si lo vemos de esa manera, la estructura del sistema que permite la clasificación, comúnmente un árbol de decisión, se convierte en la función directa. Es claro entonces que la función inversa sería la construcción de dicho árbol. Pronto notamos que la construcción de dicha función es en tiempo de diseño, es decir, una vez que se incorpora queda estática, es ahí dónde no se cumple con el modelo de sistemas sapientes, siendo que el sistema sólo aplica la función directa en tiempo de ejecución.

Cabe señalar que el modelo que caracteriza a los Sistemas Sapientes [59] no involucra únicamente la aplicación de una función inversa para retroalimentar el sistema, pero también hemos notado el especial énfasis que se hace en esa propiedad.

Intentamos justificar la importancia del aprendizaje incremental (por experiencia) y

aproximarnos además a un sistema sapiente (el paradigma propuesto en el trabajo de René Mayorga [59]) enfocándonos principalmente a los Agentes Sapientes, implementando la idea de función inversa en términos de aprendizaje. Tomando en cuenta:

- Que los agentes sapientes han sido caracterizados como agentes cognitivos que aprenden sus capacidades y estado de cognición a través de la experiencia, además de presentar otras características como interacción con otros agentes, y estar embebidos en ambientes sociales.
- La Sapiencia Computacional (SC) puede ser desarrollada como una extensión natural de los enfoques de la IA. Tal como se comentó anteriormente, lo que se busca aquí es ver a la SC como un paradigma que cohesiona los diferentes enfoques de la IA.

Es necesario también considerar algunas de las limitaciones que se presentan en el modelo BDI al realizar la aproximación al modelo de Sistemas Sapientes:

- La ausencia de los mecanismos necesarios que permiten un aprendizaje incremental.
- La arquitectura BDI no incluye de manera explícita elementos que permitan modelar sistemas multiagentes y sociabilidad.
- No se especifica tampoco un mecanismo para manejar el uso de las emociones [2].

En este trabajo nos enfocamos en la primera carencia, y creemos que la supresión de esta limitación puede aproximar adecuadamente el modelo BDI a la caracterización hecha para los Agentes Sapientes (AS).

En [31] se presenta una extensión a la arquitectura genérica de BDI propuesta en la implementación de dMARS [25]. Se muestra cómo la función directa puede ser caracterizada por todos los componentes comunes a dicha arquitectura. La extensión propone una meta-función que trabaja de forma paralela, como un elemento de auto-observación sobre las acciones resultantes. Además, de ser un regulador entre la función directa y la inversa. También en [40] (véase apéndice A) presentamos una aproximación de agentes sapientes, basada en la incorporación de aprendizaje. Discutimos algunas extensiones al modelo BDI para incorporar tanto aprendizaje intencional como social: el elemento de

[2] Cabe señalar que los agentes dentro del paradigma de sapiencia incluyen mecanismos para tratar con emociones, además de aprendizaje. En [59] se discute la incorporación de emociones a la arquitectura BDI.

aprendizaje retroalimenta el sistema, utilizando aquellas creencias relevantes al plan aplicado que falló, con nuevas creencias y planes que pueden reparar dicha falla. Lo anterior se puede implementar con mecanismos de aprendizaje inductivo (e.g., TILDE, SMILE). Como resultado, nuestro agente BDI capaz de aprender, se puede aproximar a la caracterización hecha para agentes sapientes.

5.4. Aprendizaje Distribuido: SMILE

El protocolo de SMILE (*Sound Multiagent Incremental LEarning)* [11] aborda el problema del aprendizaje colectivo dentro de un SMA. En términos generales SMILE supone que cada agente puede aprender de manera incremental en función de la información que va obteniendo. Cada agente debe contar con un mecanismo de revisión de creencias que mantenga la consistencia en toda actualización realizada a la base de creencias de un agente. El aprendizaje surge mientras se llevan a cabo una serie de interacciones entre los agentes del SMA que tienen en común ciertas creencias, con el propósito de mantener la consistencia global.

Es decir, que un agente r es capaz de actualizar su estado B para mantener la consistencia después de que cierta informació K ha sido percibida (*a-consistencia*). Luego existe un conjunto B_C de creencias que es y debe permanecer común a los agentes del SMA. Si el agente r_i actualización la parte común B_C para mantener su *a-consistencia* entonces cada uno de los agentes debe actualizar su estado para mantener la *a-consistencia* (*sma-consistencia*).

Con SMILE se sugiere que mientras se mantiene la *sma-consistencia* de un agente en tanto nueva información es recibida, el agente mismo juega un rol de aprendiz, e implica una comunicación con otro agente que toma el papel de crítico. Cabe mencionar que los agentes no son especializados y pueden tomar roles diferentes. Nos basamos en que la información está distribuida entre los agentes pero puede ser redundante. En SMILE no existe el uso de una memoria centralizada, ni de agentes que juegan un papel particular.

La idea general es que para que un mecanismo M_s sea *sma-consistente*, se supone que exista una interacción entre el agente *aprendiz* r_i y los otros agentes, con base en que la información se encuentra de manera distribuida dentro del SMA. Con el propósito de lograr la *sma-consistencia* de M_s, se efectúa la aplicación iterada por el agente *aprendiz* r_i de un mecanismo *interno* M encargado de mantener la consistencia del agente, seguido de las interacciones entre r_i y los otros agentes; y así iterativamente hasta que la *sma-*

consistencia de r_i se restaurada (*Figura* 5.4). A continuación se describe el proceso de dicho mecanismo:

El mecanismo M_s es disparado por un agente r_i que recibe una información k la cual necesita de una actualización de estado para restablecer su *a-consistencia*. El estado del agente *aprendiz* después de una actualización de estado se denota como $M(B_i)$; B'_C se denotará como la parte comn modificada por el agente r_i; y $B'j$ el estado de cualquier otro agente r_j inducido por la modificación de la parte comn B_C en B'_C.

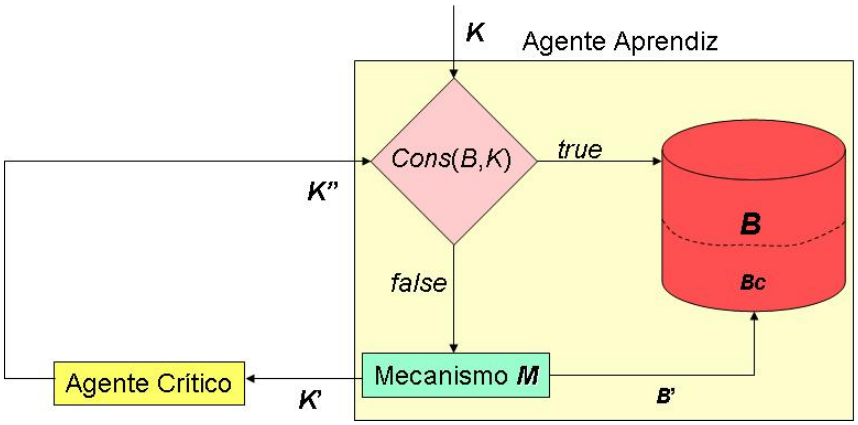

Figura 5.4: Protocolo SMILE

Una interacción $I(r_i, r_j)$ entre el agente *aprendiz* r_i y otro agente r_j jugando el papel de crítico, está constituído por:

- Por el envío del agente r_i de la actualización de estado B'_C de la parte común de creencias de su estado. r_i es *a-consistente* despues de dicha actualización de estado.
- Del examen por el agente r_j de la modificación B'_j de su estado inducido por la actualización de estado B'_C. Si dicha modificación conserva la *a-consistencia* de r_j entonces la modificación es adoptada por r_j.
- Del envio por el agente r_j ya sea sobre la aceptación de B'_j o de una refutación acompañada de la información k' indicando que la propiedad de consistencia del agente entre su estado y la información recibida, denotada por $Cons(B'_j, k')$, es falsa.

Una iteración del mecanismo M_s se constituye de la siguiente manera:

- De la recepción por el agente *aprendiz* r_i de alguna información k y de la actualización de su estado $M(B_i)$ restaurando la *a-consistencia* de r_i.

- De un conjunto de interacciones $I(r_i, r_j)$ (pudiendo intervenir eventualmente más agentes *críticos* r_j) tales que al menos una información k' es transmitida a r_i. El hecho de adjuntar k' y obtener como consecuencia la *a-inconsistencia* de r_i conduce a una nueva iteración.

El mecanismo M_s termina hasta que ningún agente puede proporcionar más información k'. Cuando éste es el caso, la *sma-consistencia* del agente *aprendiz* r_j es restaurada. Después todos los agentes adoptan la actualización de estado B_C hecha durante la ejecución del mecanismo.

Capítulo 6

Aprendizaje en Agentes BDI

En este capítulo exponemos los resultados obtenidos de la implementación hecha en Jason de aprendizaje intencional, tal como se define en [36–39]. En el que el contexto de los planes expresa las razones prácticas para actúar en cierta forma y no en otra [1], y el éxito o falla de la ejecución es la evidencia que los agentes tienen para actualizar su razones prácticas.

En la segunda sección presentamos una versión incremental de aprendizaje SMA, escalando la definición original de SMILE en lógica proposicional, a una versión en primer orden, y adoptando este último como protocolo de aprendizaje para los agentes BDI.

Finalmente, definimos en términos de la semántica operacional de Jason, el conjunto de reglas necesarias para incorporar aprendizaje intencional. Con lo anterior presentamos una primera aproximación a formalizar el proceso de aprendizaje SMA en el modelo BDI.

6.1. Implementación de aprendizaje en Jason

El ejemplo utilizado en esta sección hace referencia al conocido mundo de los bloques. El cuadro 6.1 ejemplifica la situación en este ambiente, el cual está representado por un conjunto de creencias y un plan diseñado para apilar un cubo X sobre uno Y. El plan `stack` se vuelve aplicable si el cubo destino está libre, (`clear(Y)`), y el agente está sosteniendo el cubo que ha de ser apilado, (`holding(X)`). Notemos que si la definición de nuestro plan no considerará el átomo `holding(X)` como parte de su contexto, el plan fallaría al ser ejecutado, con el ciego supuesto de que se está sosteniendo el cubo X y no fuera ese el

[1] El aprendizaje intencional contiende con el problema de mantener dichas razones actualizadas y consistentes, con la finalidad de garantizar la coordinación.

Beliefs	**Plan**
on(b,a).	@stack
clear(b).	+!stack(X,Y) : clear(Y) & holding(X) <-
clear(c).	-clear(Y); -holding(X);
onTable(a).	+armEmpty; +on(X,Y);
onTable(c).	.print("Stacking ",X,.on ", Y).
armEmpty.	

Cuadro 6.1: Algunas creencias y un plan en el mundo de los bloques.

caso. Cuando el plan falla, el agente debería reconsiderar las razones prácticas que éste tiene para adoptar dicho plan, como parte de sus intenciones [2] en el manejo de un evento del tipo `+!stack(X,Y)`.

6.1.1. Aprendizaje centralizado (nivel 1)

Siguiendo con el ejemplo anterior, supongamos que el plan `stack` ha fallado, debido a un contexto mal definido, como se explicó. Desearíamos que el agente tratara de aprender por qué el plan ha fallado, y actualizar sus razones prácticas para adoptarlo como parte de una intención. Con el propósito de ejecutar el proceso de aprendizaje, el agente debería generar un evento del tipo `!pLearn(stack)` al momento en que el agente detecta que el plan `stack` ha fallado. Uno de los planes relevantes para este tipo de evento es como sigue:

```
@learningPlan
+!pLearn(Plan) : flag(newExample)
   <- ia.makeSet(Plan);
      ia.execTilde(Plan);
      ia.updatePContex(Plan,C,V);
      !evalLearnedContext(Plan,C,V).
```

El agente aprendiz mantiene un registro de las intenciones que ha ejecutado. Cuando el *éxito* o *fallo* de alguna intención es detectado, un registro es agregado como una creencia, la lista de esas creencias, integra el conjunto de entrenamiento de forma dinámica.

Si al menos un nuevo ejemplo ha sido colectado por el agente, éste creerá `flag(newExample)`, y el plan `learningPlan` será ejecutable. En Jason es posible volver a intentar los planes fallidos, a diferencia de la implementación de dMARS, en la cual este tipo de estrategia

[2] El contexto para hacer un plan aplicable es visto como las razones prácticas que tiene un agente para adoptar un plan como parte de sus intenciones en el manejo de un evento particular [36–39].

no es considerada. El plan `learningPlan` se constituye por la acción interna `ia.makeSet`, la cual configura la tarea de aprendizaje del agente, generando los archivos requeridos por Tilde para aprender: la base de conocimiento (`.kb`), conocimiento de previo (`.bg`), y el archivo de parámetros y lenguaje *bias* footnoteLenguaje bias es una forma de estructurar el espacio de búsqueda con el propósito de acotarlos y buscar de forma eficiente una hipótesis explicatoria del conjunto de entrenamiento [6]., L (`.s`). El identificador `Plan` nombra estos archivos, que son generados automáticamente por el agente de la siguiente manera.

Cada ejemplo es codificado como un modelo (ejemplo) bajo la especificación del paradigma de aprendizaje por interpretaciones. Un modelo comienza con una etiqueta que indica la etiqueta del ejemplo (*success* o *failure*), indicada por la ejecución del plan. A esta etiqueta le precede un predicado de la forma `plan/2` que es agregado para establecer que el modelo, pertenece a una instancia generada de la ejecución de un plan en particular por cierto agente. El modelo también contiene las creencias que el agente tenía al momento de que el plan fue seleccionado como parte de una intención (estado del sistema). La etiqueta es agregada en la fase de ejecución normal del intérprete BDI. El cuadro 6.2 muestra algunos modelos generados por la ejecución del plan `stack` por el agente $c1$.

Ahora es necesario crear un archivo de configuración indicando tanto el lenguaje *bias* L, como los parámetros para ejecutar Tilde. Siguiendo el ejemplo anterior, el archivo sería `stack.s`. La primera parte del archivo es común para todas las configuraciones, porque especifica la información impresa mientras se ejecuta el proceso de aprendizaje por Tilde (*talking*); el número mínimo de casos en una hoja del árbol; el formato de la salida; y las clases para los ejemplos, e.g., *success* o *failure*.

```
talking(0).
load(models).
minimal_cases(1).
output_options([c45,lp]).
classes([success, failure]).
```

La segunda parte del archivo de configuración especifica los predicados que han de ser considerados para formar parte del árbol inducido. El comando `rmode` se usa para definir el lenguaje bias $\mathcal{L}$ de la siguiente manera: el símbolo '`#`' puede ser visto como una variable aterrizada, es decir, como un comodín que toma todos los posibles valores constantes a partir de los ejemplos de la base de conocimiento. El prefijo '`+`' significa que la variable debería ser instanciada después de los ejemplos en la base de datos, ésta hace referencia a

```
begin(model(1)).
failure.
plan(c1,stack).
clear(b).
clear(c).
onTable(a).
onTable(c).
on(b,a).
armEmpty.
end(model(1)).

begin(model(2)).
success.
plan(c1,stack).
holding(c).
clear(b).
onTable(a).
on(b,a).
end(model(2)).

begin(model(3)).
success.
plan(c1,stack).
holding(b).
clear(a).
onTable(a).
onTable(c).
clear(c).
end(model(3)).

begin(model(4)).
success.
plan(c1,stack).
holding(c).
clear(a).
clear(b).
onTable(a).
onTable(b).
end(model(4)).

begin(model(5)).
success.
plan(c1,stack).
holding(b).
clear(c).
clear(a).
onTable(a).
onTable(c).
end(model(5)).

begin(model(6)).
success.
plan(c1,stack).
holding(b).
clear(c).
on(c,a).
onTable(a).
end(model(6)).

begin(model(7)).
failure.
plan(c1,stack).
clear(b).
on(c,a).
on(b,c).
armEmpty.
onTable(a).
end(model(7)).
```

Cuadro 6.2: Ejemplos de entrenamiento para el plan `stack`, colectados por el agente `c1`..

una variable de entrada [3]. La manera en la que un agente genera este archivo, reside en la definición del agente. El cuadro 6.3 muestra el lenguaje bias, generado automáticamente por el agente `c1` a partir de su propia definición, usado para este ejemplo.

```
rmode(plan(Agent,Plan)).
rmode(clear(X)).
rmode(onTable(X)).
rmode(on(X,Y)).
rmode(armEmpty).
rmode(holding(X)).

rmode(clear(#)).
rmode(clear(#)).
rmode(onTable(#)).
rmode(on(#,+Y)).
rmode(on(+X,#)).
rmode(on(#,#)).
rmode(holding(#)).
```

Cuadro 6.3: Lenguaje *bias* en el archivo de configuración `.s`.

[3]También está disponible el prefijo '`-`' para indicar que la variable es de entrada, es decir, debe haber sido instanciada antes de los ejemplos en la base de datos. Y el prefijo '`+-`', para indicar tanto entrada como salida.

El agente ejecuta una versión modificada (no interactiva) de ACE [7], por medio de la acción primitiva `ia.execTilde(Plan)`; luego extrae automáticamente la hipótesis aprendida desde el archivo `stack.out`, para lograr modificar la definición del plan (`ia.updatePContex(Plan,C,`
También es posible preguntar al usuario por la modificación de los planes después de mostrarle los resultados obtenidos. `!evalLearnedContext(Plan,C,V)`

```
holding(X) ?
+--yes: [true] 5.0 [[true:5.0,false:0.0]]
+--no:  [false] 2.0 [[true:0.0,false:2.0]]
```

El ejemplo anterior empleó siete modelos para su inducción (ver cuadro 6.2), en un tiempo de 0.032 segundos, bajo una plataforma Linux SuSe Intel Centrino 1.73 GHz. El resultado sugiere un nuevo contexto para el plan `stack`: `clear(Y) & holding(X)` (¡el contexto original del plan!).

6.1.2. Aprendizaje Descentralizado (nivel 2)

¿Por qué un agente BDI aprendiendo al nivel 1 debería comunicarse para tratar de aprender? [36]. Tenemos dos situaciones para las cuales un agente podría considerar comunicarse mientras se encuentra en el proceso de aprendizaje: i) cuando el agente no es capaz de iniciar el proceso de aprendizaje, e.g., no tiene suficientes ejemplos; ii) el agente no puede hallar una hipótesis para explicar la falla del plan en cuestión. En ambos casos el agente aprendiz puede preguntar a los demás agentes en el MAS por más ejemplos de entrenamiento.

El compartir ejemplos de esta manera, ha sido una estrategia comúnmente adoptada en otros casos de aprendizaje distribuido. En los cuales el problema de la fragmentación vertical suele ser un asunto delicado, sin embargo no es nuestro caso, ya que los ejemplos están representados como conjuntos etiquetados de cláusulas definidas.

Consideremos los ejemplos de entrenamiento del 6.2. Desde una perspectiva de aprendizaje por interpretaciones, ese conjunto de entrenamiento son modelos para un concepto particular, sin ser necesariamente homogéneos, e.g., el modelo 1 tiene información acerca de `armEmpty`, mientras que el modelo 2 no cuenta con ella. En una representación proposicional lo anterior implica que el atributo pertenece a una fuente de información diferente (fragmentación vertical). El no tener que lidiar con este problema es muy importante, debido a que mientras nuestro modelo de aprendizaje en agentes BDI, presenta sólo el problema de fragmentación horizontal, el intercambio de ejemplos de entrenamiento es

```
@gatherExamples_f
+!gatherExamples([],Plan) : true
   <- !pLearn(Plan).

@gatherExamples_i
+!gatherExamples([Ag|T],Plan) : true
   <- .send(Ag, askAll, value([Plan,X,L,Y],example(Plan,X,L,Y)),List);
      !adoptList(List);
      !gatherExamples(T,Plan).
```

Cuadro 6.4: Planes para colectar ejemplos distribuidos en el MAS.

suficiente para alcanzar aprendizaje al nivel 2.

Se dice que un grupo de agentes tienen la misma *competencia* para un evento dado, si ellos comparten el mismo plan para manejar dicho evento. Definimos competencia en términos de planes, porque el evento con el cual contienden se encuentra ya incluido en la definición del plan. Siguiendo este concepto, existen dos formas de pedir ayuda: i) a través de un mensaje de multidifusión (todos los agentes lo reciben), incluyendo una etiqueta para señalar el plan que ha de ser aprendido, entonces aquellos agentes que comparten el mismo plan pueden aceptar el procesamiento del mensaje; ii) el agente envia dicho mensaje sólamente a los agentes con los cuales comparte esa competencia. En nuestra implementación ambos casos son posibles: la acción interna `ia.setTolearningMode/3` toma como argumentos: los planes para los cuales se activará la habilidad de aprendizaje (es posible especificar `all`); los agentes compartiendo las mismas competencias (`all` para indicar *broadcasting*); y algunas opciones de *debug*. Por ejemplo, la acción `ia.setTolearningMode([stack], [c2,c3], [verbose(true)])` indica que el agente aprenderá acerca del plan `stack`, comunicándose con `c2` y `c3`, mientras reporta algunos detalles del proceso de aprendizaje.

En la implementación en Jason, un agente siempre trata de aprender al nivel 1 cuando un ejemplo nuevo es adquirido ((`flag(newExample)`). El plan `pLearn` sólo falla si el proceso de aprendizaje no produce una hipótesis, esto significa que el agente no puede aprender solo. En el caso anterior, el este agente genera el evento `!gatherExamples`. Se tienen dos planes aplicables para el evento `+!gatherExamples` (ver cuadro 6.4). Primero, el plan `gatherExamples_i` iterará la lista completa de agentes, preguntando a cada uno por ejemplos relevantes al plan `Plan`, mientras adopta, como parte de sus creencias, las respuestas de los demás agentes (`List` es la lista de ejemplos). Posteriormente, el agente

intentará nuevamente aprender(`gatherExamples_f`).

El resultado obtenido en el nivel 2 es el mismo que el obtenido al nivel 1, el agente actualiza el contexto del plan con el contexto hecha en la definición original. Sin embargo, siguiendo la aproximación hecha en dMARS para la falla de planes, el agente `c1` solo puede alcanzar este resultado en el nivel 2. Es imposible para el agente colectar los ejemplos necesarios para aprender exitosamente el contexto. Pero aún bajo el contexto de fallo de planes manejado por Jason, que implica volver a intentar los planes fallidos, los ejemplos colectados por un agente al nivel 1 parecen contener muy poca información. Cabe señalar que gracias a que los agentes colectan los ejemplos de entrenamiento a través de su experiencia, resulta más fácil la obtención de los mismos, en comparación con un enfoque clásico de aprendizaje máquina, en el cual los ejemplos son previamente colectados (*offline*) por un supervisor. En nuestro caso los agentes exploran el espacio de hipótesis de forma natural mientras actúan (*online*). El uso de métodos de programación lógica inductiva ayuda a que el número de ejemplos requeridos por el agente para poder lograr el aprendizaje sea menor; en parte gracias al uso de un conocimiento base. También, el colocar a los agentes en un MAS en el nivel 2, puede contribuir a que la colección de los ejemplos de entrenamiento sea más rápida, tomando ventaja de la experiencia de otros agentes. La table 6.6 muestra la salida del proceso de aprendizaje descentralizado al nivel 2.

El resultado de un proceso de aprendizaje es compartido por los agentes en el MAS. Si el usuario modifica la definicición del plan de acuerdo al árbol lógico de decisión encontrado, el cambio surge efecto automáticamente en la librería de planes de todos los agentes que comparten esa misma competencia. Lo anterior justifica la cooperación en la adopción de metas: compartir ejemplos es fácil, mientras que aprender no lo es, pero aprender algunas veces no es posible sin cooperación, entonces, la cooperación se vuelve ventajosa [59].

6.2. $\mathcal{FOL}$-SMILE: el *saber-cómo* de aprender

En esta sección explicamos los resultados obtenidos de la implementación del protocolo SMILE en Jason. Utilizamos como mecanismo de aprendizaje el algoritmo ILDT [41] que promete ser localmente eficiente, es decir, aprende de forma incremental y siempre restaura la consistencia del agente. ´

Es importante destacar, que a diferencia de la implementación anterior, esta definición de planes es realmente incremental. Aquí no es necesario un mínimo número de ejemplos para lograr aprendizaje, ni utilizar cada vez que se ejecuta el algoritmo de aprendizaje toda

```
[c1] saying: PLAN stack FAILED
[c1] saying: EXECUTE LEARNING FOR stack
[report] CURRENT EXAMPLES
[report] c1: example(stack,[clear(b),clear(c),onTable(a),
                    onTable(c),on(b,a),do,counter(1),
                    armEmpty],failure)
[c1] saying: CAN'T LEARN. CONTEXT IS clear(Y)
[c1] saying: NO NEW EXAMPLES, ASKING FOR HELP
...
[c1] saying: EXECUTE LEARNING FOR stack
[report] CURRENT EXAMPLES
[report] c1: example(stack,[clear(b),clear(c),onTable(a),onTable(c),
                    on(b,a),do,counter(1),armEmpty],failure)
[report] c1: example(stack,[holding(c),clear(b),clear(c),onTable(a),
                    on(b,a),do,counter(1)],success)[source(self)]
...
[report] c1: example(stack,[clear(b),clear(c),onTable(a),onTable(c),
                    on(b,a),do,counter(4),armEmpty],failure)
...
[showLearningRes] THE INDUCED TREE FOR stack IS:
[showLearningRes] holding(-A) ?
[showLearningRes] +--yes: [true] 5.0 [[true:5.0,false:0.0]]
[showLearningRes] +--no:  [false] 4.0 [[true:0.0,false:4.0]]
[c1] saying: THE NEW CONTEXT FOR stack IS clear(Y) & ((holding(X)))
```

Cuadro 6.5: Aprendizaje descentralizado en el mundo de los bloques.

la evidencia acumulada. Los ejemplos formulados para crear una hipótesis son descartados, ya que la información resulta redundante, al encontrarse incluida en la hipótesis actual. También la forma en la que se colectan los ejemplos en el MAS ha sido adaptada de acuerdo al protocolo SMILE. Una sola pieza de información k, representando un ejemplo refutando una hipótesis es enviada en cada iteración entre agentes. Lo anterior reduce considerablemente el tráfico de mensajes usados para la comunicación.

El contexto en los planes es visto como el conocimiento común definido en SMILE, en una instancia intencional; y el éxito o falla de la ejecución es la evidencia que los agentes tienen para actualizar su razones prácticas.

En el cuadro 6.6 se muestran tres agentes en el mundo de los bloques, tratando de aprender por qué el plan `stack` ha fallado. Aquí el mecanismo de aprendizaje empleado para restablecer la consistencia (el algoritmo ILDT [41]), trata de obtener la hipótesis consistente empleando los ejemplos colectados por los tres agentes. Lo que obtenemos en

una primera iteración del protocolo SMILE es una hipótesis incompleta, la cual cubre sólo una parte de la evidencia, la que el agente ag_1 ha obtenido a través de su experiencia. Posteriormente, una nueva iteración se lleva acabo cuando el Ag_2 refuta H y regresa al agente aprendiz una pieza de evidencia nueva que el algoritmo ILDT emplea nuevamente para crear una nueva hipótesis. Finalmente, el contexto del plan es exitosamente refinado despues de la intervención del Ag_3 jugando el rol de crítico, porque con la información que el ejemplo que este último proporciona se tiene la información suficiente para restablecer la $MAS - Consistencia$ del sistema.

6.3. Jason Smiles: extendiendo la semántica para incorporar aprendizaje

A continuación describimos la semántica operacional para aprender intencionalmente adoptando SMILE. Con el propósito de mantener las reglas semánticas coherentes con la definición de Jason, aquí usamos la misma notación empleada en [10, 61, 62].

Dada una estado de configuración de $AgentSpeak(L)$, C, escribimos C_E para hacer referencia al componente E de C. C. De forma similar para todos los demás componentes de C. También usamos $i, i', \ldots$ para denotar las intenciones, y $i[p]$ señala la intención que tiene el plan p en su cima. Asumimos la existencia de las funciones de selección: S_E para los eventos, S_{Ap} para seleccionar un plan aplicable, S_I para intenciones, y S_M para los mensajes en el buzón.

Para hacer más clara la descripción de las reglas, primero definiremos algunas funciones sintácticas. Hemos mencionado anteriormente que un plan (p tiene la forma $label\ te : \phi \leftarrow h$. Usaremos $Head(p)$ para denotar $te : \phi$; y $Body(p)$ para apuntar a h. Así mismo, $Label(p) = label$ dónde $label$ es una literal que identifica al plan. $TE(p) = te$ y $Ctxt(p) = \phi$. Finalmente, $Plan(label) = p$.

Un agente cree, que algunos (o todos) sus planes son supervisados, e.g., pueden ser actualizados por medio de aprendizaje intencional.

$$Supervised(p) = \begin{cases} true, & Beliefs \models supervised(l) \wedge Label(p) = l \\ false, & otherwise \end{cases}$$

Un agente es capaz de recordar, con alguna extensión, las creencias que soportan la adopción de un plan como una intención, así como el resultado de su ejecución, e.g., si un

plan falla o tiene éxito. Así es como un agente puede ir colectando ejemplos de entrenamiento mientras realiza su ejecución normal. Dado un plan p el conjunto de entrenamiento E está dado por:

$$E(p) = \{sample(l, B, C) | Beliefs \models sample(l, B, C) \wedge Label(p) = l\}$$

donde $C \in \{success, failure\}$ es la clase del ejemplo de entrenamiento, y B es el conjunto de creencias del agente cuando éste seleccionó el plan p como parte de una intención.

La función $ENeg(H, E)$ regresa el subconjunto de ejemplos de entrenamiento E refutando la hipótesis H:

$$ENeg(H, E) = \{e | e \in E^+ \wedge H \not\models e \vee e \in E^- \wedge H \models e\}$$

donde B^+ es el conjunto de ejemplos para los cuales la clase es $success$ y B^- aquellos ejemplos para los cuales la clase es $failure$.

Como se mencionó en el capítulo 5, Smile requiere de un mecanismo de aprendizaje M localmente eficiente. $M(H, e)$ denota la ejecución de tal mecanismo, donde H es una hipótesis representada como un árbol lógico de decisión, y e es un ejemplo de entrenamiento, tal que $M(H, e) = H'$ en dónde H' restablece la consistencia con e.

A continuación, describiremos las reglas semánticas para incorporar aprendizaje intencional en Jason. Primero, definimos las reglas para colectar los ejemplos de entrenamiento de aquellos planes supervisados. **RecPlan$_1$** agrega, sin generar ningún evento, el ejemplo de entrenamiento incompleto en la forma $sample(l, B, executing)$, donde $executing$ indica que la clase ($success$ o $failure$) para ese ejemplo aún no se conoce.

$$\textbf{RecPlan}_1 \qquad \frac{Supervised(p)}{C, Beliefs \rightarrow C, Beliefs'} \quad C_\rho = p$$
$$d\acute{o}nde: \quad Beliefs' \models sample(Label(p), Beliefs, executing)$$

una vez que se conoce el resultado de la ejecución del plan, los ejemplos en memoria deben ser completados. El caso de éxito es como sigue:

$$\textbf{RecPlan}_{\textbf{succ}} \quad \frac{Supervised(p)}{C, Beliefs \rightarrow C, Beliefs'} \quad C_\iota = i[Head(p) \leftarrow]$$
$$\textit{dónde}: \quad Beliefs' \not\models sample(Label(p), B, executing)$$
$$Beliefs' \models sample(Label(p), B, success)$$

Las situaciones de fallo incluyen: i) fallo de una sub-meta, esto es un plan $[head \leftarrow !at; h]$ falla si no existe ningún plan para manejar el evento $+!at$; ii) fallo de un *query* ($test$; y iii) desde una perspectiva de metas declarativas, un plan diseñado para alcanzar g falla si el plan termina o el agente no cree g [46]:

$$\textbf{RecPlan}_{\textbf{fail 1}_{\textbf{A}}} \quad \frac{RelPlans(Ps, te) = \{\}}{C, Beliefs \rightarrow C', Beliefs'} \quad C_\epsilon \neq \_, C_\iota = i[p], C_R = C_{Ap} = \emptyset$$
$$\textit{dónde}: \quad Beliefs' \not\models sample(Label(p), Bels, executing)$$
$$Beliefs' \models sample(Label(p), Bels, failure)$$
$$C'_E = C_E \cup \{\langle +fail(Label(p)), T\rangle\}$$

$$\textbf{RecPlan}_{\textbf{fail 1}_{\textbf{B}}} \quad \frac{AppPlans(Beliefs, C_R) = \emptyset}{C, Beliefs \rightarrow C', Beliefs'} C_\epsilon \neq \_, C_\iota = i[p], C_R \neq \emptyset, C_{Ap} = \emptyset$$
$$\textit{dónde}: \quad Beliefs' \not\models sample(Label(p), Bels, executing)$$
$$Beliefs' \models sample(Label(p), Bels, failure)$$
$$C'_E = C_E \cup \{\langle +fail(Label(p)), T\rangle\}$$

$$\textbf{RecPlan}_{\textbf{fail 2}} \quad \frac{Test(Beliefs, at) = \emptyset}{C, Beliefs \rightarrow C, Beliefs'} \quad C_\iota = i[Head(p) \leftarrow ?at, h]]$$
$$\textit{dónde}: \quad Beliefs' \not\models sample(Label(p), B, executing)$$
$$Beliefs' \models sample(Label(p), B, failure)$$
$$C'_E = C_E \cup \{\langle +fail(Label(p)), T\rangle\}$$

$$\textbf{RecPlan}_{\textbf{fail 3}} \quad \frac{Beliefs \not\models g,\ TE(p) = +!g,\ Body(p) = \emptyset}{C, Beliefs \rightarrow C', Beliefs'} \quad C_\iota = i[p]$$

$$\textit{dónde}: \quad Beliefs' \not\models sample(Label(p), B, executing)$$

$$Beliefs' \models sample(Label(p), B, failure)$$

$$C'_E = C_E \cup \{\langle +fail(Label(p)), T\rangle\}$$

Cuando surge un evento de fallo (denotado por $\langle +fail(X), T\rangle$), la *a-consistencia* debe ser restaurada utilizando el mecanismo M. Los ejemplos utilizados por el agente son olvidados, ya que resutaría una información redundante con el nuevo contexto aprendido. Asumimos la existencia de una función de selección $S_{ag}(MAS)$ para seleccionar un agente que juegue el papel de crítico.

$$\textbf{FailEv} \quad \frac{}{C, Beliefs \rightarrow C', Beliefs'} \quad C_\epsilon = \langle +fail(l), T\rangle$$

$$\textit{dónde}: \quad C'_\epsilon = \_$$

$$Ctxt(Plan(l))' = M(Ctxt(Plan(l)), Test(Beliefs, sample(Label(l), B, L)))$$

$$Beliefs' \not\models sample(Label(l), B, L)$$

$$C'_A = C_A \cup \{send(S_{ag}(MAS), AskIf, \langle Ctxt(Plan(l))\rangle)\}$$

Si el agente crítico no encuentra evidencia para refutar el contexto del plan p recibido, este adopta la actualización propuesta por el agente aprendiz (**SmileRec$_1$**). En caso contrario, el agente crítico envia de regreso sus creencias refutando el contexto propuesto(**SmileRec$_2$**).

$$\textbf{SmileRec}_\textbf{1} \quad \frac{S_M(C_M) = \langle AskIf, id, H\rangle,\ ENeg(H, E(Plan(l))) = \emptyset}{C, Beliefs \rightarrow C', Beliefs}$$

$$\textit{dónde}: \quad Ctxt(Plan(l))' = H$$

$$C'_A = C_A \cup \{send(id, Tell, ok)\}$$

$$\textbf{SmileRec}_\textbf{2} \quad \frac{S_M(C_M) = \langle AskIf, id, H\rangle,\ ENeg(H, E(Plan(l))) \neq \emptyset}{C, Beliefs \rightarrow C', Beliefs}$$

$$\textit{dónde}: \quad C'_A = C_A \cup \{send(id, Tell, ENeg(H, E(Plan(l))))\}$$

Cuando el agente aprendiz recibe un *Tell* con evidencia, este adopta la evidencia como parte de sus creencias y genera un evento de fallo que comienza el proceso de aprendizaje otra vez (una nueva iteración).

$$\textbf{SmileRec}_{\mathbf{3}} \qquad \frac{S_M(C_M) = \langle Tell, id, K\rangle,\ K \not\models ok}{C, Beliefs \rightarrow C, Beliefs'}$$

$$\textit{dónde}: \qquad Beliefs' \models K$$

$$C'_E = C_E \cup \{\langle +fail(Label(First(K))), T\rangle\}$$

```
...
[ag1] saying: PLAN stack FAILED
[ag1] saying: EXECUTE ILDT FOR stack
[report] ag1's CURRENT RELEVANT EXAMPLES FOR stack:
[report] sample(stack,[holding(c),clear(b),
   onTable(a),on(b,a),success]).
 sample(stack,[clear(b),clear(c),onTable(a),
   onTable(c),on(b,a),armEmpty],failure).
...
[mechanismM] THE ILDT FOR stack IS:
[mechanismM] <> holding(X) ?
[MechanismM]  (+) leaf(success)
[mechanismM]  (-) leaf(failure)
[ag1] ASKING Ag2 TO REFUTE holding(X) AS CONTEXT FOR stack
[ag2] REFUSE! EVIDENCE: sample(stack,[on(a,b),holding(c),onTable(b),clear(a)],failure).
...
[mechanismM] THE ILDT FOR stack IS:
[mechanismM] <> holding(X)
[mechanismM]  <+> clear(Z)
[mechanismM]        (+) leaf(failure)
[mechanismM]        (-) leaf(success)
[mechanismM]    (-) leaf(failure)
[ag1] ASKING Ag3 TO REFUTE holding(X) & not clear(Z) AS CONTEXT FOR stack
[ag3] REFUSE! EVIDENCE: sample(stack,[holding(c),clear(a),clear(b),onTable(a),onTable(b)],
      failure).
...
[mechanismM] THE ILDT FOR stack IS:
[mechanismM] <> holding(X)
[mechanismM]  <+> clear(Z)
[mechanismM]       <+> clear(Y)
[mechanismM] (+) leaf(success)
[mechanismM] (-) leaf(failure)
[mechanismM]        (-) leaf(success)
[mechanismM]    (-) leaf(failure)
[ag1] ASKING Ag2 TO REFUTE holding(X) & clear(Z) & clear(Y) | holding(X) & not clear(Z) AS
      CONTEXT FOR stack
[ag2] OK
...
```

Cuadro 6.6: Aprendizaje con Smile en el mundo de los bloques

Capítulo 7

Conclusiones

7.1. Conclusiones

Esta tesis se inspiró inicialmente en la caracterización de Agentes Sapientes hecha por el paradigma de Sapiencia Computacional [59]. Se destaca la importancia que tiene el aprendizaje incremental en los sistemas inteligentes, a través de la caracterización hecha para los agentes sapientes, los cuales son vistos como sistemas que aprenden tanto su estado cognitivo como sus habilidades, dentro de un ambiente social, a través de la experiencia [65]. Aquí, conectamos la idea de *efecto-causa*, con la de función inversa [59], y a su vez con la de aprendizaje. Una primera reflexión para incorporar aprendizaje por experiencia en agentes *BDI*, contempla una extensión a la arquitectura *BDI* [31]. Se argumenta cómo la función directa puede ser caracterizada por todos los componentes comunes a una arquitectura *BDI* y, se propone la extensión en forma de una *meta-función* que trabaja de forma paralela, cumpliendo dos tareas: i) emphauto-observación sobre las acciones resultantes; y ii) como un regulador entre la función directa y la inversa, dónde ésta última se incorpora mediante un mecanismo de aprendizaje inductivo que retroalimenta el sistema.

Basándonos en el trabajo hecho sobre aprendizaje intencional [36–39], discutimos algunas extensiones al modelo *BDI* para incorporar tanto aprendizaje intencional como social, en donde:

- El elemento de aprendizaje retroalimenta al sistema (función inversa), utilizando aquellas creencias relevantes al plan ejecutado que falló para corregir su situación. Lo anterior ha sido, actualmente implementado en esta tesis mediante mecanismos de aprendizaje inductivo (*TILDE* y $\mathcal{FOL}$*-SMILE*) [40, 41] .

- La *meta-función* es planteada inicialmente, como un meta-plan que cumple la función de recolectar información sobre los planes ejecutados [40]. El cual es incorporado mediante la detección de fallo de planes [41].

Como resultado, nuestro agente *BDI*, capaz de aprender de forma incremental, y se puede aproximar a la caracterización de agente sapiente.

Aterrizamos esta tesis partiendo del trabajo de [35], en el cual A. Guerra-Hernández aborda exitosamente el problema de aprendizaje *SMA* en términos de una arquitectura *BDI* (programada en *Lisp*). Se ha utilizado como hilo conductor para esta tesis el hecho sobre aprendizaje intencional en [36–39],

En una primera contribución [40], logramos además, reforzar el fundamento pragmático de [35] mediante una implementación en *Jason* (intérprete bien conocido de *AgentSpeak(L)*). También, pasamos de una implementación en un nivel de diseño, a una hecha en un lenguaje formal orientado a agentes *BDI*. Al respecto A. Rao advierte que ha existido una divergencia entre práctica y teoría en el área de investigación de agentes *BDI* [68]. En [40] disminuimos considerablemente dicha brecha para [35].

Al implementar el mecanismo de aprendizaje con el algoritmo de *TILDE* [40], no es posible garantizar un aprendizaje verdaderamente incremental dentro del *SMA*. Con el propósito de sanear dicha carencia y asegurar un aprendizaje incremental en el *SMA*, en una segunda implementación, se escala *SMILE* a lógica de primer orden, para poder ser adoptado como protocolo de aprendizaje por los agentes *BDI*. Para este fin resultó de gran importancia el trabajo llevado por A. Luna, de forma paralela a esta tesis, en la cual aborda el problema de aprendizaje incremental en primer orden, y propone un algoritmo para inducir árboles lógicos de decisión (*ILDT*) [41]. Este algoritmo hizo posible contar con un *Mecanismo-M* localmente eficiente [11] compatible con lógica de primer orden.

Finalmente, al inclinarnos por la tendencia de pasar de arquitecturas de implementación a lenguajes formales de programación, y elegir *AgentSpeak(L)* como formalismo para modelar agentes *BDI*, dimos el primer paso para desarrollar una teoría sobre aprendizaje social: mediante la definición del protocolo *SMILE* en términos del modelo *BDI*. En esta tesis, al mismo tiempo que extendemos la semántica operacional de *AgentSpeak(L)* siguiendo las especificaciones introducidas en *Jason*, presentamos una primera aproximación, de lo que puede ser una teoría formal de aprendizaje.

7.2. Trabajo Futuro

En esta tesis atacamos la problemática de *qué hacer* cuando una intención falla, así mismo definimos cuatro reglas para determinar cuando una intención no tiene éxito. Sin embargo, el poder precisar cuando una acción falla continúa siendo un problema abierto. Estamos convencidos de que es necesaria una formalización para el manejo de las acciones, con la cual se incorpore una manera de precisar cuándo una acción ha fallado.

Con el propósito de hacer de Jason un lenguaje verdaderamente funcional, han sido incorporadas diversas estructuras para hacer más fácil la programación en Jason. Por ejemplo, en relación al fallo de metas, se incorpora un mecanismo que consiste en generar un evento de supresión de meta; propuesto como una forma para que el programador pueda terminar correctamente aquellas intenciones que han fallado. Al respecto, no existe una formalización, ni una noción clara sobre lo que es realmente la supresión de metas. La semántica operacional de Jason no incluye una especificación formal para contender con el problema de fallo de intenciones y acciones. Para retomar la visión inicial de A. Rao, de integrar el aspecto pragmático y el teórico del modelo BDI mediante la definición de *AgentSpeak(L)*, es necesaria la formalización de aspectos como el mencionado, para evitar una bifurcación reincidente entre práctica y teoría.

Con base en un primer intento para formar una teoría de revisión de intenciones, en la cual se toma como punto de vista central, la noción de que un agente debe mantener sus creencias y sus intenciones de una manera racional [44]; el trabajo futuro debe enfocarse principalmente, en extender la noción de revisión de planes (empleada en esta tesis) hacia el dominio de las intenciones. Es decir, reconsiderar las razones prácticas que existen para estar comprometido con una intención en particular, de forma dinámica, en una forma similar a la que revisamos las razones prácticas que tiene un agente para seguir considerando un plan como una opción para manejar un evento en particular.

Finalmente, destacamos que para los experimentos hechos en el presente trabajo, empleamos el conocido mundo de los cubos como ambiente modelo. Con la finalidad de poder ofrecer una verdadera tabla comparativa, sobre la ejecución de *SMILE* en su versión de primer orden, empleando *ILDT* como *Mecanismo-M*, contra otros algoritmos de aprendizaje *SMA*, sería deseable correr dichos algoritmos en diferentes ambientes. Variando la complejidad del ambiente, el número de agentes y la cantidad de conocimiento soportado por cada uno de los agentes.

Bibliografía

[1]

[2] PhD thesis.

[3] John L. Austin. *How to Do Things with Worlds*. Harvard University Press, Harvard, MA., USA, second edition, 1975.

[4] Joseph Bates. The role of emotion in believable agents. Technical Report CMU-CS-94-136, School of Computer Science, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, April 1994.

[5] Hendrik Blockeel. Top-down induction of first order logical decision trees. *AI Commun.*, 12(1-2):119–120, 1999.

[6] Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Min. Knowl. Discov.*, 3(1):59–93, 1999.

[7] Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Ramon, and Henk Vandecasteele. Executing query packs in ilp. In *ILP '00: Proceedings of the 10th International Conference on Inductive Logic Programming*, pages 60–77, London, UK, 2000. Springer-Verlag.

[8] H. Rafael Bordini and Jomi Fred Hübner. *Jason: A Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net*, August 2005.

[9] Rafael Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak.

[10] Rafael H. Bordini, Ana L. C. Bazzan, Rafael de O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. Agentspeak(xl): efficient intention selection in bdi agents

via decision-theoretic task scheduling. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1294–1302, New York, NY, USA, 2002. ACM Press.

[11] Geauvan Bourgne, Ammal El Fallag-Seghrouchni, and Henry Soldano. Smile: Sount multi-agent incremental learning. In *AAMAS'07: Proceedings of the sept international joint conference on Autonomous Agents and Multi-Agent Systems*, pages 14–17, May 2007.

[12] Michael Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1987.

[13] Michael E. Bratman, David Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. In Robert Cummins and John L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.

[14] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[15] Rodney A. Brooks. *Cambrian Intelligence: the Early History of the New AI.* MIT Press, Cambridge, MA., USA, 1999.

[16] Jose C. Brustoloni. Autonomous agents: Characterization and requirements.

[17] David Canfield-Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[18] Cristiano Castelfranchi. Modelling social action for ai agents. *Artif. Intell.*, 103(1-2):157–182, 1998.

[19] Philip R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 221–255. MIT Press, Cambridge, MA, 1990.

[20] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artif. Intell.*, 42(2-3):213–261, 1990.

[21] Philip R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. pages 423–440, 1986.

[22] Arie A. Covrigaru and Robert K. Lindsay. Deterministic autonomous systems. *AI Mag.*, 12(3):110–117, 1991.

[23] Daniel C. Dennett. *The Intentional Stance (Bradford Books)*. The MIT Press, March 1987.

[24] Ian Dickinson and Michael Wooldridge. Towards practical reasoning agents for the semantic web. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 827–834, New York, NY, USA, 2003. ACM Press.

[25] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. In *Agent Theories, Architectures, and Languages*, pages 155–176, 1997.

[26] Oren Etzioni. Intelligence without robots: A reply to brooks. *AI Magazine*, 14(4):7–13, 1993.

[27] Jacques Ferber. *Les systèmes multi-agents : vers une intelligence collective*. Informatique, Intelligence Artificielle. InterÉditions, 1995.

[28] Leonard N. Foner. What's an agent anyway? - a sociological case study. FTP Report - MIT Media Lab, May 1993.

[29] Stan Franklin. Artificial minds. *SIGART Bull.*, 9(1):35–39, 1998. Reviewer-Robert L. Causey.

[30] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III*, pages 21–36. Springer-Verlag, 1996.

[31] Daniel Alejandro García López, Karina Gutiérrez Fragoso, Wulfrano Arturo Luna Ramírez, Rosibelda Mondragón Becerra, Gustavo Ortiz-Hernández, and Carlos Adolfo Piña García. Computational sapience, a new paradigm? In Angélica Mu noz Meléndez and Grogori Sidorov (Eds.), editors, *5th MICAI. Fourth International Workshop on Sapient Systems*, pages 13–17, Instituto Tecnológico de Apizaco, November 2006.

[32] Donald F. Geddis. *Caching and First-Order inference in model elimination theorem provers.* PhD thesis, Stanford University, Stanford, CA., USA, 1995.

[33] Michael P. Georgeff. Situated reasoning and rational behavior. Technical Report 21, Carlton, Victoria, April 1991. printed as invited contribution in PRICAI-90.

[34] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning.

[35] Alejandro Guerra-Hernández. *Learning in intentional BDI Multi-Agent Systems.* PhD thesis, Université de Paris 13. Institut Galilée. LIPN - Laboratoire d'Informatique de Paris Nord, Villetaneuse, France, 2003.

[36] Alejandro Guerra-Hernández, El-Fallah-Seghrouchni Amal, and Henry Soldano. On learning intentionally. *Inteligencia Artificial,* Revista Iberoamericana de Inteligencia Artificial, (25):9–18, 2005.

[37] Alejandro Guerra-Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano. Bdi multi-agent learning based on first-order induction of logical decision trees. In *Intelligent Agent Technology: Research and Development, Proceedings of the 2nd Asia-Pacific Conference on IAT*, pages 160–169, Maebashi, Japan, November 2001. World Scientific.

[38] Alejandro Guerra-Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano. Distributed learning in intentional bdi multi-agent systems. In *ENC '04: Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)*, pages 225–232, Washington, DC, USA, 2004. IEEE Computer Society.

[39] Alejandro Guerra-Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano. Learning on bdi multiagent systems. In *Lecture Notes in Computer Science,* Computational Logic and Multiagent Systems: 4th International Workshop CLIMA IV, pages 218–233, Heidelberg, Germany, 2004. Springer Verlag.

[40] Alejandro Guerra-Hernández and Gustavo Ortiz-Hernández. Towards bdi sapient agents: learning intentionally. In Rene V. Mayorga and Dr Leonid I. Perlovsky, editors, *Towards Computational Sapience: Principles and Systems.* Springer-Verlag, 2007. In press.

[41] Alejandro Guerra-Hernández, Gustavo Ortiz-Hernández, and Wulfrano-Arturo Luna-Ramírez. Jason smiles. In *Proceedings of the 6th Mexican International Conference*

on Artificial Intelligence (MICAI), Poster Session. IEEE Computer Society Press, 2007. In press.

[42] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artif. Intell.*, 72(1-2):329–365, 1995.

[43] José Hierro-Pescador. En torno a la intencionalidad.

[44] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. Towards a theory oj intention revision. In *Knowledge, Rationality and Action*, volume 155, pages 265–290. Springer, Science+business, 2007.

[45] Marcus J. Huber. Jam: a bdi-theoretic mobile agent architecture. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, New York, NY, USA, 1999. ACM Press.

[46] Jomi Fred Hübner, Rafael H. Bordini, and Michael Wooldridge. Plan patterns for declarative goals in agentspeak. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1291–1293, New York, NY, USA, 2006. ACM Press.

[47] Leslie Pack Kaelbling. *Learning in embedded systems*. PhD thesis, 1990. Adviser-Nils J. Nilsson.

[48] N. David Kinny and Michael P. Georgeff. Commitment and effectiveness of situated agents, 1991.

[49] Saul A. Kripke. A completeness theorem in modal logic. *The Journal of Symbolic Logic*, 24:1–14, 1959.

[50] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages, 1999.

[51] Jaeho Lee, M. Jomi. Huber, Patrick G. Kenny, and Edmund H. Durfee. UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications. In *Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS)*, pages 842–849, Houston, Texas, 1994.

[52] Stephen C. Levinson. The essential inadequacies of speech act models of dialogue. In M. Sbisá M. Parret and J. Verschueren, editors, *The essential inadequacies of speech act models of dialogue*, pages 473–492, Benjamins, Amsterdam, 1981.

[53] William E. Lyons. *Approaches to Intentionality.* Oxford University Press Inc., New York, USA, 1995.

[54] Rodrigo Machado and Rafael H. Bordini. Running agentspeak(L) agents on SIM_AGENT. In John-Jules Meyer and Milind Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 376–389, 2001.

[55] Pattie Maes. Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM*, 38(11):108–114, 1995.

[56] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[57] Maja J. Mataric. Interaction and intelligent behavior. Technical report, Cambridge, MA, USA, 1994.

[58] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of KQML as an Agent Communication Language. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Proceedings on the IJCAI Workshop on Intelligent Agents II : Agent Theories, Architectures, and Languages*, volume 1037, pages 347–360. Springer-Verlag: Heidelberg, Germany, 19–20 1996.

[59] René Mayorga. Towards computational sapience (wisdom). a paradigm for sapient (wise) systems. 2005.

[60] John McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationary Office.

[61] Alvaro Moreira and H. Rafael Bordini. An operational semantics for a bdi agent-oriented programming language, 2002.

[62] Alvaro F Moreira, Renata Vieira, and H. Rafael Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing act based

communication. In *In proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT-03), held with AAMAS-03*, Melbourne, Australia, July 2003. Springer, Verlag.

[63] M. Mundhe and S. Sen. Evaluating concurrent reinforcement learners, 2000.

[64] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[65] Martijn van Otterlo, Marco Wiering, Mehdi Dastani, and John-Jules Meyer. A characterization of sapient agents.

[66] Ramesh Patil, Richard Fikes, Peter Patel-Schneider, Donald P. McKay, Tim Finin, Thomas Gruber, and Robert Neches. The DARPA Knowledge Sharing Effort: Progress Report. In Bernhard Nebel, editor, *Proceedings of the Third International Conference on Principles Of Knowledge Representation And Reasoning*. Morgan Kaufman, August 1992.

[67] Martha E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43–68, 1992.

[68] Anand S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, pages 42–55, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[69] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[70] Stuart Russell and D. Subramanian. Provably bounded-optimal agents, 1995.

[71] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 1995.

[72] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, London, 1969.

[73] John R. Searle. *Intentionality: An Essay in the Philosophy of Mind*. Cambridge University Press, New York, 1983.

[74] Sandip Sen and Gerhard Weiss. Learning in multiagent systems. In Gerhard Weiss, editor, *Multiagent Systems*, chapter 6, pages 259–298. MIT Press, 1999.

[75] Munidar P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-how, and Communications.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.

[76] Munindar P. Singh. *Multiagent Systems: A theoretical framework for intentions, know-how, and communication.* Number 799. Berlin-Heidelberg, Germany, 1995.

[77] Leon Sterling and Ehud. Shapiro. *The Art of Prolog.* MIT Press, Cambridge, MA., USA, 2nd edition, 1994.

[78] Michael Wooldridge. *Reasoning about Rational Agents.* MIT Press, 2000.

[79] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: a survey. In *ECAI-94: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pages 1–39, New York, NY, USA, 1995. Springer-Verlag New York, Inc.

Apéndice A

Computational Sapience, a new paradigm in Artificial Intelligence?

Apéndice B

Towards BDI sapient agents: learning intetionally

Apéndice C

Jason smiles: Incremental BDI MAS Learning