Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación

Imaginación en tiempo real

Representación y gestión de creencias temporales para agentes en entornos dinámicos

Tesis doctoral

Dirigida por:
Dra. Eva Onaindía de la Rivaherrera
Dr. Vicente J. Botti Navarro

Presentada por:
Miguel Rebollo Pedruelo

Imaginación en tiempo real

Imaginación en tiempo real

Representación y gestión de creencias temporales para agentes en entornos dinámicos

Miguel Rebollo Pedruelo

Tesis doctoral presentada al
Departamento de Sistemas Informáticos y Computación
para la obtención del grado de

Doctor en Informática

Valencia, 2004

Prefacio

> *"El pensamiento humano involucra la capacidad de imaginar, la capacidad de pensar en algo en ausencia de entradas perceptibles y la capacidad de imaginar sin reaccionar."[Davis, 1998]*

Esta frase de R. Davis, presidente de la *American Association for Artificial Intelligence (AAAI)* entre 1996 y 1998, resume perfectamente la intención del autor en el desarrollo de su trabajo de tesis y justifica la elección, un tanto osada, del título de la presente disertación.

El dotar de inteligencia a entidades artificiales, como son los agentes, no sólo requiere del uso de métodos que podemos calificar de inteligentes. Hoy en día, las computadoras son capaces de llevar a cabo tareas específicas que, antes, únicamente las personas eran capaces de realizar. Pero ahora la inteligencia se cifra en aspectos más cercanos al comportamiento humano, no en máquinas de propósito específico que son capaces de resolver problemas sólo por su extraordinaria capacidad de cálculo.

La imaginación no es una cualidad que uno atribuye normalmente a un robot. En el diccionario de uso del español de María Moliner encontramos la siguiente definición:

imaginar: (del lat. *"imaginari"*) Representarse en la mente la imagen de algo que no existe o que no está presente.

La imaginación, como representación de cosas inexistentes, es una cualidad humana, más acorde con el sentido de creatividad. El trabajo de tesis que aquí se desarrolla se centra en la definición de imaginación como la representación de algo que no está presente en este momento o que se infiere a partir del conocimiento del sujeto.

> *"Una capacidad básica es nuestra capacidad para predecir el futuro, es decir, imaginar cómo van a ocurrir las cosas en lugar de tener que probarlas. La faceta esencial aquí es la* imaginación*, es decir, la desconexión entre pensamiento y acción. Esta desconexión nos proporciona*

la capacidad de imaginar las consecuencias de una acción antes, o en lugar, de experimentarla."[Davis, 1998]

"Imaginación formal en tiempo real" es un término que emplean'Nirkhe y Kraus para describir las actividades de un agente con sentido común en un entorno de tiempo real en general, y en una situación con plazos máximos de ejecución en particular [Nirkhe and Kraus, 1995]. En su artículo se esboza un paralelismo entre imaginación, tal y como la entendemos en un contexto humano, y la capacidad de un agente autónomo para formular imágenes mentales de posibles escenarios y planes de acción en el curso de sus razonamientos.

La capacidad de imaginar es una tarea habitual en Inteligencia Artificial desde sus orígenes. En efecto, cuando hablamos de un programa que es capaz de jugar al ajedrez, tarea para la cual no dudaríamos en afirmar que es necesario un determinado grado de inteligencia, nos hallamos ante un programa que tiene *imaginación*. Aunque lo único que haga el programa en cuestión sea examinar todas las posibilidades que se pueden producir a partir de la jugada actual para decidir cual es su mejor movimiento. La máquina conseguirá un movimiento más provechoso cuantas más jugadas sea capaz de anticipar, de forma que a un espectador humano le parecerá que se trata de un programa más inteligente que el anterior. Y ¿qué es anticipar una jugada, sino pensar en las posibles alternativas del adversario, sin que se produzca realmente ningún movimiento, y elegir aquella que nos proporcionará una situación global más ventajosa? Es decir, nuestro programa es capaz de calcular las posiciones futuras del tablero de juego, evaluarlas y seleccionar el curso de acciones que le reporte mayor beneficio. Nuestro programa es capaz de representar jugadas de ajedrez que no han sucedido: *es capaz de imaginar*.

Incertidumbre, urgencia y desconocimiento caracterizan el complejo mundo de un agente situado en un entorno dinámico. Y para resolver de forma eficaz estos problemas resulta adecuado mantener una representación del tiempo en las creencias del agente. En la presente disertación se expone un modelo temporal que permite a un agente situado en un entorno de tiempo real estricto razonar, además de con información pasada y actual, con datos futuros. Contempla posibilidades de evolución del sistema y trabaja con todas ellas simultáneamente hasta que es capaz de decidir el curso de acción más adecuado o simplemente el estado del mundo exterior le indica cuál de todas las alternativas es la única posible. Es un agente de tiempo real con imaginación.

Valencia, *Miguel Rebollo Pedruelo*
Octubre 2003

Resumen

Uno de los aspectos de interés en el área de los agentes y los sistemas multiagente es el de la construcción de sistemas de tiempo real que satisfagan ciertas restricciones temporales, como plazos máximos de ejecución o periodos de activación. En la literatura sobre agentes inteligentes son escasas las referencias a la construcción de agentes que operen en entornos de tiempo real estricto (*"hard real-time"*) y que integren técnicas acotadas de Inteligencia Artificial y comportamiento de tiempo real.

El objetivo principal de este trabajo es extender el modelo inicial de representación y razonamiento temporal, ideado para la arquitectura de ARTIS, para (i) adaptarlos a los requerimientos de una arquitectura de agente, y (ii) incorporar la resolución de problemas de síntesis. Un segundo objetivo es utilizar algoritmos eficaces para la gestión de la información temporal acotados que se adapten a las características de un entorno de tiempo real.

La disertación comienza con una revisión de las tendencias actuales en el punto de confluencia de las tres áreas de interés relacionadas con este trabajo: agentes inteligentes, sistemas de tiempo real y modelos temporales. Se ha escogido una extensión de la lógica RTCTL para construir el modelo formal de agente de tiempo real. A partir del modelo formal, se propone una arquitectura de agente que se pueda aplicar a problemas tanto de análisis como de síntesis.

Las creencias temporales del agente se representan mediante un grafo temporal. La topología de los grafos generados es diferente en función del tipo de problemas: de análisis o de síntesis. Topologías tan distintas obligan a considerar la construcción del conjunto de creencias y la consulta de las relaciones temporales por separado en cada tipo de problema.

Los experimentos realizados sobre los algoritmos de razonamiento temporal están orientados a la obtención de estimaciones de tiempos de forma empírica. Se ha evaluado el rendimiento de los algoritmos de razonamiento temporal sobre grafos generados aleatoriamente, consiguiendo resolver consultas temporales con coste lineal.

Tabla de contenidos

Índice de figuras

Índice de algoritmos

1

Introducción

1.1. Motivación

Bajo el título

> *"Representación y gestión de creencias temporales para agentes en entornos dinámicos"*

se presenta un modelo computacional que permite a un agente inteligente, situado en un entorno de tiempo real estricto, manipular información temporal en sus procesos cognitivos. El agente está basado en la arquitectura ARTIS, desarrollada en trabajos previos del grupo de investigación [Thomson(Fr.) et al., 1990], que ha sido ampliada paulatinamente en varias tesis doctorales [García-Fornés, 1996] [Onaindía, 1997] [Terrasa, 2000] [Henao, 2001] [Julián, 2002] hasta alcanzar su estado de desarrollo actual.

Durante la última década, la Inteligencia Artificial en Tiempo Real ha demostrado ser una importante línea de investigación para la resolución de problemas complejos en los que se requiere "inteligencia" y respuestas en tiempo real.

De forma paralela, el paradigma de sistemas multiagente (SMA) constituye un área de creciente interés dentro de la Inteligencia Artificial (IA). Este interés se debe, entre otras razones, a que permite abordar adecuadamente la construcción de sistemas complejos, para los que no se habían obtenido resultados satisfactorios mediante el empleo de técnicas clásicas.

El desarrollo de este tipo de agentes está íntimamente ligado con el de los sistemas de tiempo real (STR): sistemas dinámicos que evolucionan con el tiempo y en los que la calidad de sus respuestas dependen del instante de tiempo en el que se éstas proporcionen. Y dado que el tiempo es una característica intrínseca del entorno, su tratamiento debe ser una tarea natural en el modelo empleado. Debe tenerse en cuenta tanto el conocimiento disponible en un instante de tiempo determinado como los cambios que se producen en las creencias del sistema; bien como respuesta a un evento externo o bien como una deducción interna. Los sistemas de tiempo real estricto desarrollados

durante esta última década se caracterizan por su *flexibilidad* y *adaptabilidad*, características difíciles de encontrar en los rígidos sistemas de tiempo real tradicionales [Stankovic and Ramamritham, 1993].

Desde el punto de vista del diseño de agentes individuales, quedan dos cuestiones abiertas que afectan directamente a la aplicación de esta paradigma al desarrollo de sistemas inteligentes en tiempo real: el razonamiento con información incompleta y la operatividad en tiempo real

Razonamiento con información incompleta. Los agentes mantienen una perspectiva parcial de su entorno debido a que habitualmente se trata de sistemas abiertos complejos. Si se desea diseñar agentes robustos, se requieren mecanismos sofisticados para razonar con información incompleta e intentar predecir el estado del entorno y su posible evolución a partir de la información disponible. Por otra parte, una fuente adicional de incertidumbre en sistemas de tiempo real es determinar si las acciones que se han planificado van a realizarse a tiempo con el fin de, por ejemplo, tomar medidas correctivas que palien inminentes fallos o estados no deseados en el sistema.

Operatividad en tiempo real. Los agentes se utilizan cada vez más en entornos en los que considerar el tiempo resulta esencial. El comportamiento de tiempo real es necesario tanto para las acotar las deliberaciones del agente como para que las interacciones con otros agentes sean predecibles. Posibilita, por ejemplo, la delegación de tareas o la consideración tareas urgentes que deben ejecutarse con una prioridad mayor.

El presente trabajo de tesis presenta una aproximación mediante la cual, realizando las modificaciones necesarias a la arquitectura ARTIS para agentes inteligentes de tiempo real, puedan resolverse estos dos problemas abiertos.

1.2. Objetivos

El objetivo principal es extender el modelo de representación y razonamiento temporal inicial para la arquitectura de agente ARTIS [García-Fornés, 1996][Botti et al., 1999], basado en el modelo de pizarra [Nii, 1986b][Nii, 1986a], en tres frentes.

En primer lugar, se debe modificar el modelo de agente ARTIS de manera que se amplíe el tipo de problemas que se pueden resolver utilizando esta arquitectura. Se estudiarán los cambios necesarios para soportar problemas tanto de análisis como de síntesis. Otro aspecto que afecta al modelo de agente es la definición de un modelo formal que integre todas las propuestas realizadas hasta la fecha. Tiene una especial importancia la posibilidad de definir una lógica común, adecuada para la representación de propiedades de tiempo real, la especificación de agentes inteligentes y la representación de información temporal.

En segundo lugar, se estudiará el modelo temporal actual del agente ARTIS, proponiendo un cambio en la lógica temporal del modelo. Se estudian varias extensiones de CTL* [Emerson, 1995] adaptadas para la inclusión de relaciones cualitativas, que permitan expresar propiedades esenciales en sistemas de tiempo real. Otro aspecto fundamental es la posibilidad de representar acciones temporales, indispensables para ampliar el tipo de problemas a los que aplicar la arquitectura. Con todo ello, se define una *red de creencias temporales* como soporte para la representación y gestión de las creencias temporales del agente.

Por último, se buscarán nuevos algoritmos de razonamiento temporal que se ajusten mejor a las necesidades de un sistema de tiempo real. Dada la casi imposibilidad de conseguir que el tiempo de cómputo en el peor caso de las tareas inteligentes sea predecible, se opta por que las operaciones de gestión de la información temporal se implementen como métodos que puedan ser interrumpidos en cualquier paso de ejecución. De esta forma, el sistema es más reactivo y se consigue una mayor coherencia entre la información que maneja el sistema y el estado real del mundo exterior.

1.3. Organización de la disertación

El resto del documento se estructura de la siguiente forma.

En la sección 2 se realiza una revisión del estado del arte sobre las arquitecturas de agente susceptibles de poder aplicarse en entornos de tiempo real estricto: reactivas e híbridas.

La sección 3 describe la arquitectura de agente ARTIS para agentes de tiempo real. Incluye una revisión de los problemas intensivos en conocimiento que pueden resolverse con ARTIS, el modelo conceptual de agente su fomralización. Concluye con el estudio de las propiedades de seguridad y viveza en el modelo de agente.

En la sección 4 revisa las lógicas temporales empleadas en la formalización de agentes, de sistemas de tiempo real y la representación de acciones, buscando los aspectos comunes para definir una lógica con la que especificar un agente ARTIS. A continuación se presenta el modelo temporal del agente, describiendo la teoría de tiempo utilizada y su semántica.

La sección 5 contiene los algoritmos de manipulación de información temporal. Se basan en métodos interrumpibles que siempre tienen disponible una respuesta cuya fiabilidad aumenta con el tiempo. El núcleo de los algoritmos son procesos de búsqueda en tiempo real dotados con cierta capacidad de aprendizaje.

La sección 6 expone un ejemplo de aplicación y muestra los resultados experimentales de los algoritmos para grafos temporales generados aleatoriamente. Se han construido simulando la estructura que se genera por la sucesiva aplicación de reglas de inferencia.

Por último, en la sección 7 se presentan las conclusiones obtenidas con la realización del presente trabajo de tesis y se exponen brevemente las líneas de trabajo futuras para la ampliación y mejora de ciertas cuestiones de interés que quedan fuera del alcance de la presente disertación.

2

Agentes para entornos de tiempo real

2.1. Introducción

El concepto de agente ya se ha tratado ampliamente en la literatura. Sin embargo, existe un pequeño vacío en la consideración de agentes como entidades que operan en entornos de tiempo real. Y se trata de un tema especialmente importante para los agentes, pues una de sus características básicas es que se encuentran situados en un entorno.

El presente capítulo estudia el concepto de agencia desde el punto de vista de su utilización en entornos de tiempo real. Comienza con una revisión de la definición de agente inteligente. A continuación se expone la arquitectura abstracta de agente, junto con la definición de sistema de tiempo real. El objeto es poner de manifiesto la similitud existente entre ambas. Por último, se describen las aproximaciones existentes hasta la fecha para la construcción de agentes adecuados para entornos dinámicos.

2.2. ¿Qué es un agente inteligente?

El principal problema que surge a la hora de decidir la agencialidad del sistema es la variedad de definiciones existentes. Dependiendo de la definición de *agente* seleccionada, un sistema puede ser o no un agente.

2.2.1. Concepto de agencia

En los últimos años ha surgido una nueva área de investigación: el desarrollo de agentes inteligentes. Sin embargo, no hay una definición de agente comúnmente aceptada, dado que se trata de un término con el que se abarca un cuerpo muy heterogéneo de disciplinas de investigación y desarrollo. Además, se han creado varios sinónimos que añaden, si cabe, una mayor confusión al intento de definición y estandarización del concepto de agente.

La definición que proporciona Shoham en su propuesta de AOP *("Agent-Oriented Programming")* es la primera en la que se considera el concepto de agente como una entidad por sí misma, separada de las propuestas del área de la Inteligencia Artificial Distribuida [Shoham, 1993]. En ella se proponen los agentes como la siguiente evolución en la Ingeniería del Software. Considera un agente como

> *"Una entidad cuyo estado está formado por un conjunto de componentes mentales, como creencias, capacidades, elecciones y acuerdos."*

Se trata de una definición poco útil desde el punto de vista de un desarrollador que desea construir un agente. Sin embargo, proporciona una visión del mismo como un componente software a un grado altísimo de abstracción, con características casi humanas.

La definición más sencilla, desde el punto de vista de los requerimientos mínimos para que una entidad *"software"* sea considerada un agente, es la enunciada por Russell y Norving [1995] :

> *"Un* agente *es todo aquello que puede considerarse que* percibe *su ambiente mediante* sensores *y que responde o actúa en tal ambiente por medio de* efectores*."*

La problemática que genera esta definición se hace patente de forma rápida: casi cualquier entidad puede ser un agente. Es tan general que hace que un termostato pueda considerarse un agente. Sin embargo, hay algunas ideas clave que deja claras:

1. el agente se encuentra en un entorno conocido y concreto (su ambiente), y
2. interactua con dicho entorno, percibiendo su estado y actuando sobre él con intención de modificarlo.

Esta definición proviene de la teoría de sistemas clásica: el agente puede considerarse como el controlador de un sistema dinámico (véase Figura 2.1perl). Y, en efecto, un termostato (el controlador de la temperatura de una habitación – entorno –) es un agente.

Tras esta definición simplista de agente, Russell introduce otras características que son comunes en la literatura sobre agentes inteligentes: la racionalidad y la autonomía. En este caso, la *autonomía* no se refiere tanto a la capacidad de actuar por si mismo como a la capacidad de aprendizaje. Un agente tendrá un mayor grado de autonomía cuando sus decisiones se basen en su propia experiencia y no en el "conocimiento integrado", proporcionado al agente por su diseñador.

Una definición más completa, pero que se mantiene en la misma línea, es la sugerida por Weiss, según la cual:

> *"Un* agente *es un sistema informático* situado *en algún* entorno *y que es capaz de* actuar *de forma* autónoma *sobre dicho entorno para cumplir sus objetivos de diseño."*

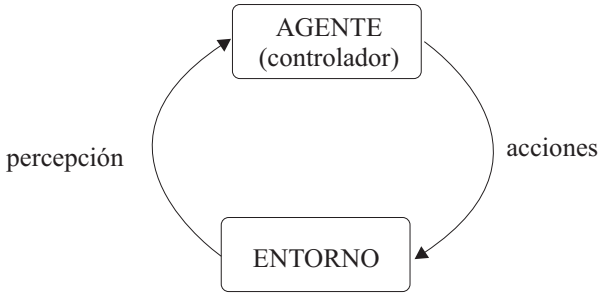


Figura 2.1. Agente como controlador

Esta definición [Weiss, 2000] ya considera a un agente como un sistema informático. En ella se refleja una acepción ligeramente distinta del término *autonomía*: no tanto como capacidad de aprendizaje sino como la capacidad de actuar sin la supervisión de un humano que controle al agente y sea responsable, en última instancia, de su comportamiento. A partir de este momento, consideraremos la autonomía en los agentes inteligentes como un sinónimo de *independencia*.

La definición de agente más aceptada hasta la fecha es la de Wooldridge y Jennings, según la cual un agente es un sistema informático capaz de actuar de forma autónoma y flexible en un entorno [Wooldridge and Jennings, 1995], entendiendo por flexible que sea:

- **reactivo**, responda al entorno en el que se encuentra,
- **proactivo**, sea capaz de intentar cumplir sus propios planes u objetivos,
- **social**, se comunique con otros agentes mediante algún tipo de lenguaje.

Aquí aparece una nueva característica: la *sociabilidad* de los agentes. Tal y como se han ideado, no tiene sentido desarrollarlos como entidades aisladas. A lo largo de su existencia, será habitual que se relacionen con otros agentes para realizar su trabajo.

Y también se considera sociabilidad la comunicación con humanos cuando ésta va más allá de la simple selección de una opción de un menú o de pulsar un botón. Si bien no es necesario que la interacción con personas se realice en lenguaje natural, mediante sistemas de reconocimiento de habla, sí que se les exige a los agentes que sean capaces de desarrollar procesos de comunicación de alto nivel, en los que se tenga en cuenta, además del propio contenido del mensaje, la intención del comunicador al emitir su mensaje.

Aunque el presente trabajo de tesis se centra en el modelo de razonamiento para un único agente, se está trabajando ya en la creación de una arquitectura de SMA, formada por varios agentes ARTIS, orientada al control de *dominios sociales de tiempo real* [Julián et al., 2002]. En este sentido, es necesario incorporar los mecanismos necesarios para incorporar los mensajes recibidos a las creencias del agente y su adecuado tratamiento temporal.

Una definición particularmente interesante es la de Nwana. Centra el concepto de agencia en una cualidad que puede poseer un software determinado, desligándolo de sistemas físicos que se deben controlar [Nwana, 1996]. Define un agente como

> *"Un componente software o hardware que es capaz de actuar exhaustivamente realizando tareas en provecho de su usuario."*

Esta definición se completa con una tipología funcional de los agentes, por aquello que hacen, en lugar de la habitual clasificación tecnológica. Nwana identifica siete tipos de agentes:

- agentes colaborativos
- agentes de interfaz
- agentes móviles
- agentes de información/internet
- agentes reactivos
- agentes híbridos
- agentes inteligentes

Muchas aplicaciones combinan agentes de varias de estas categorías. Así, se puede hablar de agentes estáticos,[1] deliberativos, colaborativos; agentes móviles, reactivos, colaborativos; agentes estáticos, deliberativos de interfaz; etc.

2.2.2. Arquitectura abstracta de agente

En la práctica totalidad de las representaciones de agentes, se considera que éstos mantienen un estado interno más o menos complejo (véase Figura 2.2). Una arquitectura de agente es, en esencia, un mapeo del comportamiento interno del agente ante una entrada (estructuras de datos, operaciones y flujo de control) en el conjunto de acciones que el agente es capaz de realizar sobre su entorno [Weiss, 2000].

Podemos asumir que el entorno del agente está caracterizado por una secuencia de *estados* $S = \{s_1, s_2, \ldots\}$. En un instante de tiempo determinado, el entorno estará en alguno de esos estados. La capacidad efectora del agente se representa mediante un conjunto de *acciones* $A = \{a_1, a_2, \ldots\}$.

La información sobre el estado actual del entorno y su historia pasada se almacena en una estructura interna de datos. Sea I el conjunto de todos los estados internos del agente. El proceso de decisión del agente debe basarse, al menos en parte, en esta información.

Una función *see* representa la capacidad del agente para observar su entorno. Puede estar implementada mediante *"hardware"* en el caso de un agente situado en el mundo físico: por ejemplo, una videocámara o un sensor de infrarrojos para un robot. La salida de la función es una percepción (una entrada perceptual). *see* es una función que se define como

[1] Como contraposición de agente móvil.

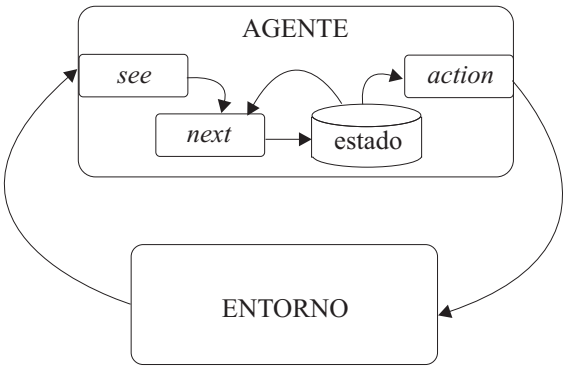


Figura 2.2. Agentes con estado interno

$$see : S \rightarrow P \tag{2.1}$$

La función de selección de acciones, *action*, representa el proceso de toma de decisión del agente. Queda definida de la siguiente forma:

$$action : I \rightarrow A \tag{2.2}$$

entre estados internos y acciones.

Es necesario introducir una función, denominada *next*, que representa la evolución del estado interno del agente. Determina cómo se modifica el estado interno del agente en función de su estado interno actual y de las percepciones a partir del entorno.

$$next : I \times P \rightarrow I \tag{2.3}$$

El comportamiento de un agente con estado interno puede resumirse de la siguiente forma.

1. El agente comienza en un estado interno determinado i_0
2. Observa el estado actual del entorno s y genera una percepción $see(s)$
3. El estado interno del agente se actualiza a través de la función $next(i_0, see(s))$.
4. El agente selecciona una acción para ejecutar con $action(next(i_0, see(s)))$.
5. la acción seleccionada se ejecuta y el agente entra en un nuevo ciclo en el paso 2.

Siguiendo este ciclo de ejecución genérico para un agente, podemos considerar que un agente es la composición de estas tres funciones *see*, *next* y *action*, de la siguiente forma:

$$agente : action \circ next \circ see(S) \tag{2.4}$$

Una característica habitual de los entornos en los que se encuentran los agentes es que se trata de entornos *no deterministas*, en el sentido de que

no se puede garantizar que el estado que se alcanza después de la ejecución de una acción sea el esperado. Este comportamiento se modela mediante una función

$$env : S \times A \rightarrow \wp(S) \tag{2.5}$$

que toma el estado actual del entorno $s \in S$ y una acción $a \in A$ que debe realizar el agente y la mapea en un conjunto de estados, que son todos aquellos a los que se puede llegar desde s aplicando la acción a. Si $env(s, a)$ siempre se mapea sobre conjuntos de un solo elemento, entonces el entorno es determinista: podemos conocer exactamente qué estado se va a alcanzar tras la aplicación de una acción.

Siguiendo la definición de agente vista en la Ecuación 2.4, la consideración del entorno de la Ecuación 2.5 y y s_0 es el estado inicial del agente, entonces la secuencia

$$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots \tag{2.6}$$

representa una posible historia de la ejecución del agente en el entorno sii se verifican las siguientes condiciones:

1. $\forall u \in \mathbb{N}, a_u = action((s_o, s_1, \ldots, s_u))$, y
2. $\forall u \in \mathbb{N}$ con $u > 0, s_u \in env(s_{u-1}, a_{u-1})$

El *comportamiento característico* de un agente en un entorno *env* es el conjunto de todas las historias que satisfacen una determinada propiedad ϕ. Si ϕ se cumple en todas las historias, podemos afirmar que se trata de una *propiedad invariante* del agente en el entorno. Por ejemplo, si el agente es el controlador de un tanque de aguas residuales y en todas las posibles historias del tanque éste nunca se desborda, entonces ésta puede ser una propiedad invariante (y además deseable).

2.3. ¿Qué es un sistema de tiempo real?

En general, se puede considerar un sistema de tiempo real como un conjunto de secuencias de estados temporalizados, cada una de las cuales es un posible comportamiento del sistema [Alur and Henzinger, 1991].

Definición 2.1 (Sistema de tiempo real) *Un* sistema de tiempo real *(STR) es una tupla*

$$STR = \langle \mathcal{S}, \mathcal{P}, \mu, \mathcal{T} \rangle$$

donde:

- $\mathcal{S}$ *es el conjunto de estados del sistema;*
- $\mathcal{P}$ *es un conjunto de observables;*
- $\mu : \mathcal{S} \rightarrow \wp(\mathcal{P})$ *asocia a cada estado s un conjunto de observaciones;*
- $\mathcal{T} : \mathbb{R}^+ \rightarrow \mathcal{S}$ *es un conjunto de secuencias de estados temporalizados.*

El conjunto de observables $\mathcal{P}$ está formado por eventos y proposiciones, de manera que $\mu(s)$ determina tanto el conjunto de eventos que se producen en s como el conjunto de proposiciones que son ciertas en s.

Una secuencia $\tau \in \mathcal{T}$ representa un posible comportamiento del sistema, donde $\tau(t) \in S$ identifica un estado único $\forall t \in \mathbb{R}^+$. La secuencia τ cumple la propiedad de *variabilidad finita*:

Propiedad 2.2 (Variabilidad finita) *Existe una secuencia de intervalos temporales* $\bar{I} = I_0, I_1, I_2, \ldots$ *tal que* $\forall t, t' \in I_i, \mu(\tau(t)) = \mu(\tau(t'))$.

Es decir, que dentro de cada intervalo I_i, la componente observable del estado de sistema no varía.

El conjunto $\mathcal{T}$ es *cerrado bajo fusión* si cada estado del sistema contiene toda la información necesaria para determinar la evolución futura del sistema.

Propiedad 2.3 (Clausura de fusión) $\forall \tau_1, \tau_2 \in \mathcal{T}, \forall t_1, t_2 \in \mathbb{R}$, *si* $\tau_1(t_1) = \tau_2(t_2)$ *entonces* $\exists \tau \in \mathcal{T}$ *tal que* $\tau(t) = \tau_1(t)$ *para* $t \leq t_1$ *y* $\tau(t) = \tau_2(t + t_2 - t_1)$ *para* $t > t_1$.

Sea $\bar{\sigma} = \sigma_0, \sigma_1, \sigma_2, \ldots$ una *secuencia de observaciones*, finita o infinita, donde $\sigma_i \in \wp(\mathcal{P})$. Si añadimos a esta secuencia los intervalos temporales en los que se indica el instante de tiempo en los que cada observación es válida, obtenemos una *secuencia de observaciones temporalizadas* $\rho = (\bar{\sigma}, \bar{I})$, como un par compuesto por una secuencia de observaciones y una secuencia de intervalos, ambas de la misma longitud. Puede reescribirse como una secuencia de pares :

$$(\sigma_0, I_0) \rightarrow (\sigma_1, I_1) \rightarrow (\sigma_2, I_2) \rightarrow \cdots \tag{2.7}$$

Llegados a este punto, la semejanza entre las definiciones de una arquitectura abstracta de agente y de STR es notable. En ambos casos hay una función de percepción del entorno: *see* (2.1) y μ, y la historia del sistema se define como una secuencia de estados (2.6) y (2.7). En el caso de la arquitectura abstracta de agente, se indica además como se efectúa la transición entre estados.

La principal diferencia entre ambas es que

1. los agentes no incorporan información temporal de forma explícita en sus comportamientos (secuencias de estados); y
2. los STR son completamente deterministas y, dado un estado y un conjunto de acciones, el siguiente estado está completamente determinado.

La integración de ambas propuestas nos conduce a un modelo de agente situado en un entorno con características de tiempo real, formando un conjutno agente + entorno en un sistema agencial al que denominaremos *sistema* ARTIS [Rebollo et al., 2003].

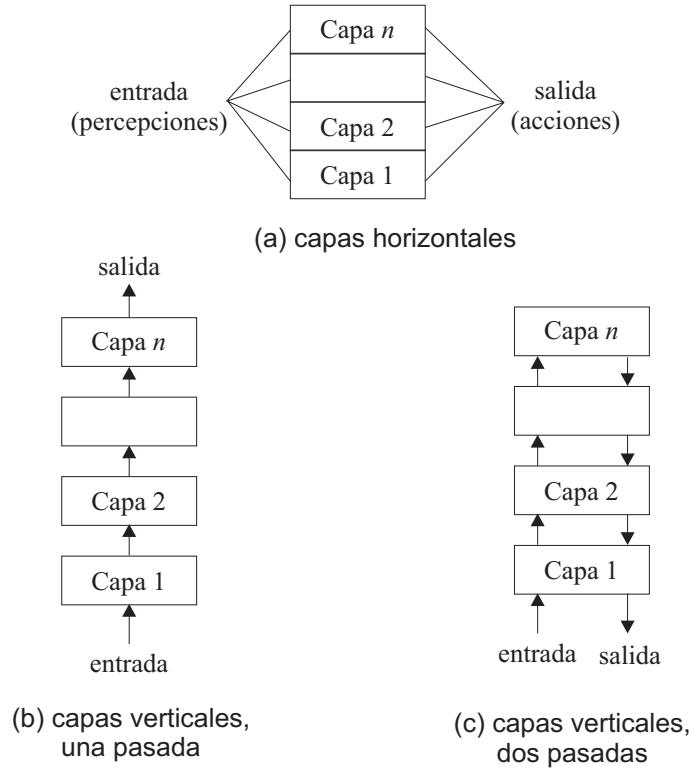


Figura 2.3. Arquitecturas híbridas en capas

2.4. Arquitecturas de agente para tiempo real

Las arquitecturas de agente se clasifican en arquitecturas *deliberativas*, *reactivas* e *híbridas*, dependiendo del modo en el que se organizan los distintos comportamientos del agente.

Las *arquitecturas deliberativas*se basan, normalmente, en la teoría de resolución de problemas de IA: dados un estado inicial, un conjunto de operadores/planes y un objetivo que se desea alcanzar, el proceso de deliberación del agente consiste en determinar la secuencia de pasos que debe realizar para cumplir sus objetivos.

La aparente intratabilidad de ciertos problemas utilizando aproximaciones simbólicas ha llevado a algunos autores a cuestionar su utilidad en la construcción de agentes que operen en entornos dinámicos con restricciones temporales.

La primera alternativa son las *arquitecturas reactivas*. De todas ellas, la *arquitectura de subsumpción* es su principal representante [Brooks, 1991b]. Aunque no faltan las críticas a este tipo de arquitecturas [Etzioni, 1993].

Las *arquitecturas híbridas* surgen para intentar aproximar posturas entre las dos anteriores, utilizando las ventajas que cada una de ellas aporta a la construcción de agentes inteligentes. Combinan módulos reactivos con módulos deliberativos. Los módulos reactivos contienen el comportamiento reactivo del agente, mediante los cuales se responde a los cambios en el entorno mediante estímulos que no necesitan deliberación. Los módulos deliberativos son los responsables del comportamiento proactivo del agente, dirigiendo sus accionesde forma que alcance los objetivos locales o cooperativos del agente.

Las arquitecturas híbridas suelen basarse en arquitecturas por capas (véase Figura 2.3), cada una de las cuales implementa un comportamiento del agente. La estructura más básica es la formada únicamente por dos capas: una capa para el comportamiento reactivo y otra para el comportamiento deliberativo. Dependiendo de cómo se conecten las distintas capas a las entradas y salidas del agente y de la conexión de las capas entre sí, se distingue entre:

Horizontales. Todas las capas se conectan directamente a las entradas y las salidas del agente. Cada capa puede considerarse como un agente que propone las acciones que se deben realizar.

Verticales. La entrada y la salida se conectan a una sola capa. La información y el control pasa de una capa a otra, cada vez con un mayor grado de abstracción o de complejidad.

Dadas las características de los sistemas de tiempo real estricto, en los que es necesario asegurar una respuesta acotada en el tiempo, parece más adecuado emplear una arquitectura reactiva o una arquitectura híbrida. A continuación se presentan las características de las arquitecturas más conocidas.

2.4.1. Arquitecturas puramente reactivas

Arquitecturas de subsumpción (Brooks)

La arquitectura de subsumpción se basa en la idea que el comportamiento de un agente inteligente puede dividirse en un conjunto de *niveles de competencia*[Brooks, 1991b] [Brooks, 1991a]. El nivel más bajo proporciona un comportamiento básico, semejante a los actos reflejos y los instintos en los seres vivos. Los niveles superiores implementan mecanismos de razonamiento cada vez más elaborados que se ejecutan en paralelo. Estos últimos tienen, además, la capacidad de actuar sobre las entradas de los niveles más bajos, modificando así el comportamiento global del agente.

Cada una de las capas se implementa como un conjunto de módulos, cada uno de los cuales es una máquina de estados finitos (véase Figura 2.4). Los estados se clasifican en cuatro categorías:

Salida. Son aquellos por los que la máquina de estados produce una salida en una de sus líneas de salida.

Efectos colaterales. Modifican las variables a partir de la información de entrada y de las variables existentes.

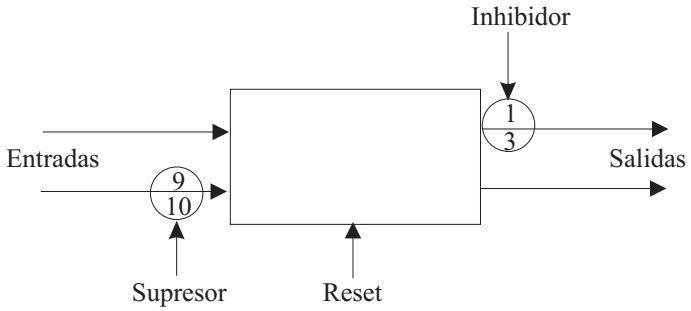


Figura 2.4. Representación esquemática de una máquina de estados finitos

Condición. Se evalúan predicados y valores de entrada y, como resultado, se cambia a uno de los estados siguientes.

Evento. Se monitorizan una conjunto de condiciones y cuando alguna de ellas se cumple, se salta al estado correspondiente,

Esta arquitectura queda definida por dos características. La primera de ellas es que el proceso de toma de decisiones del agente queda dividido en un conjunto de comportamientos, cada uno de los cuales percibe el estado del entorno y selecciona la acción a ejecutar. Cada uno de los módulos de comportamiento queda encargado de resolver una tarea particular, que no incluye ningún tipo de razonamiento simbólico; quedan reducidos a conjuntos de reglas de la forma

$$situación \rightarrow acción$$

La segunda característica es que todos los comportamientos pueden estar activos al mismo tiempo. Debe haber algún mecanismo que permita escoger al agente la acción que va a ejecutar.

Pengi (Chapman)

Es otro ejemplo de sistemas reactivos puros. La idea básica consiste en mostrar que la planificación realmente no es algo requerido, aunque sí lo sea diseñar un sistema reactivo que se comporte como si fuera capaz de planificar. Esto resulta particularmente interesante en aquellos dominios en los que sólo hay una pequeña cantidad de tiempo para reaccionar [Agre and Chapman, 1987].

Para ello descartan la "Planificación" con P-mayúscula, entendiendo como tal la que realizan los sistemas de IA clásicos. En ellas hay una fase inteligente de planificación, seguida por otra fase de ejecución irreversible. Las personas, en cambio, realizan procesos de "planificación" con p-minúscula: desarrollan un plan a seguir; pero existe una componente de improvisación, en la medida que deciden en cada momento qué hacer basándose en el estado actual del entorno.

Pengi está formado por dos subsistemas: el sistema periférico y el sistema central. El *sistema periférico* consta de un sistema visual para reconocer situaciones importantes (que denominan aspectos) y de un sistema motor simple que proporciona el interfaz para las capacidades efectoras del agente. El *sistema central* es el responsable de los procesos de cognición del agente. Se implementa como una red combinatoria, que toma la entrada del sistema visual y produce una salida para el sistema motor.

Programas teleo–reactivos (Nilsson)

Los *programas teleo–reactivos (T-R)* [Nilsson, 1994] son sistemas de producción formados por un conjunto de reglas de producción que se disparan cuando se cumple la condición que aparece en la parte izquierda de las reglas. A diferencia de los sistemas de producción habituales, un programa T-R está diseñado para alcanzar una meta concreta. Sobre las reglas se define una relación de orden total. La producción de más alto nivel es la que consigue el cumplimiento del objetivo. En cada momento se dispara aquella regla que verifica la condición de su parte izquierda y está más arriba en la lista.

Una *secuencia T-R* tiene la siguiente forma:

$$\begin{aligned} c_1 &\rightarrow nil \\ c_2 &\rightarrow a_2 \\ c_3 &\rightarrow a_3 \\ &\vdots \\ c_n &\rightarrow a_n \end{aligned}$$

Cada condición c_i debe satisfacerse para que se pueda realizar la acción a_i correspondiente. Si se cumplen varias condiciones al mismo tiempo, se dispara la producción con menor valor para i y se ejecuta la acción a_i asociada. La primera regla, $c_1 \rightarrow nil$ es aquella que da por conseguido el objetivo: cuando se alcanza c_1 (el objetivo), ya no es necesario ejecutar más acciones.

Un *programa T-R* es una generalización de la noción de secuencia T-R que permite que las reglas contengan variables libres tanto en su parte izquierda como en su parte derecha, que se instancian durante la ejecución del programa.

El paso final que realiza Nilsson en su análisis es la noción de *árboles T-R*. Se trata de una generalización de un programa T-R en el que la relación existente entre las producciones no es de orden total. Cuando se ejecuta el árbol T-R, se toma la acción asociada con el nodo menos profundo del árbol y cuya condición es cierta. La principal cuestión es determinar qué sucede cuando se puede aplicar más de una regla. La solución, no propuesta directamente Nilsson, es considerar el hecho de que un árbol T-R pueda tener un comportamiento no determinista.

La idea de Nilsson es que los árboles T-R se construyan automáticamente usando encadenamiento hacia atrás a partir del objetivo que se desea alcanzar. Es concebible que el agente, una vez alcanza un objetivo específico, pueda construir en línea un árbol T-R siguiendo encadenamiento hacia atrás.

ERE: Motor de reducción de la entropía (Drummon)

Drummond y Bresina proponen una arquitectura que permite la integración de planificación (razonamiento dirigido por objetivos y selección de acciones), *"scheduling"* (secuenciación de las acciones y gestión de recursos) y control (monitorización y adaptación a un entorno dinámico). El escenario que asumen es el manejo de un robot autónomo que se envía a Marte como explorador. Esta arquitectura recibe el nombre de Motor de Reducción de la Entropía (ERE) [Bresina and Drummond, 1990].

La arquitectura ERE comprende los siguientes componentes:

- un *reactor* que produce el comportamiento reactivo en el entorno;
- el *proyector* explora posibles estados futuros y proporciona al reactor los comportamientos adecuados por anticipado; y
- el *reductor* razona sobre las restricciones de los comportamientos y proporciona al proyector la dirección adecuada en la que focalizar las búsquedas.

La arquitectura se organiza siguiendo el denominado *principio de capacidad independiente:* cada componente debe tener la capacidad de ejecutar por sí solo las tareas que se le han asignado. Este principio casa con la idea de algoritmo *"anytime"*. Al separar el sistema en reacción, proyección y reducción, la arquitectura ERE puede explotar las características *"anytime"* de cada uno de estos componentes por separado.

El reactor representa la dinámica del sistema mediante redes de planes [Drummond, 1986], que definen los posibles eventos en el entorno en función de sus precondiciones y los efectos dependientes de la situación (un mismo evento puede tener distintos efectos en situaciones diferentes).

Sea S el dominio de todas las situaciones posibles y O el conjunto de operadores de planes. Se definen:

$$executancy : O \rightarrow \{true, false\}$$
$$enabled : S \rightarrow \wp(\wp(O))$$

La función $enabled(s)$ devuelve el conjunto de operadores de todos los planes que se pueden ejecutar en paralelo en una situación $s \in S$ dada. $executancy(o)$ es una función que se emplea para indicar tanto eventos que se han producido en el entorno como acciones que ejecuta el agente.

La proyección considera los efectos de los eventos sobre el sistema, al igual que los cambios provocados por otros agentes. Se trata de una simple búsqueda de posibles secuencias de eventos, de forma que un camino representa

un posible comportamiento del agente. El proyector define una función que describe los efectos de un conjunto de operadores en el entorno:

$$\forall o \subseteq O, s \in S, apply(o,s) = \begin{cases} s' \in S & \text{if } o \in enabled(s) \\ undefined & \text{otherwise} \end{cases}$$

Cada conjunto de operadores tiene asociada una duración, que se utiliza para estampar con tiempos cada nueva situación. Sobre los comportamientos se pueden definir restricciones, que se construyen a partir de conjunciones y disyunciones de la siguiente forma:

- $(maintain\ \phi\ t_1\ t_2)$ es cierto en el camino de una proyección sii la f.b.f. ϕ es cierta entre los puntos de tiempo t_1 y t_2 en el camino.
- $(prevent\ \phi\ t_1\ t_2)$ es cierto en el camino de una proyección sii la f.b.f. ϕ es falsa entre los puntos de tiempo t_1 y t_2 en el camino.

En el contexto de ERE, un problema es un par formado por una situación y una restricción de comportamiento. El reductor aplica reducciones no terminales para determinar cómo se puede conseguir el comportamiento deseado, descomponiendo las restricciones de comportamiento en otras más simples. La semántica de la reducción no terminal determina que la satisfacción de la conjunción de todas las restricciones de comportamientos especificadas en la descomposición implican la satisfacción de la restricción original. Las reducciones terminales proporcionan una acción $o \in enabled(s)$ que, aplicada a la situación s del subproblema, satisface sus restricciones de comportamiento.

Phoenix (Cohen)

Phoenix [Cohen et al., 1989] es una arquitectura de agente diseñada para trabajar en entornos dinámicos, cambiantes, con restricciones de tiempo real, impredecibles, variados, con múltiples escalas espaciales y temporales y espacialmente distribuidos. Todo esto acompañado de gestión de recursos, incertidumbre, cooperación y planificación.

Los aspectos de tiempo real de la arquitectura de Phoenix se dividen en dos grandes componentes: el componente reflexivo y el componente cognitivo. El *componente cognitivo* realiza un razonamiento basado en casos sobre un conjunto de planes almacenados, entre los cuales escoge el más adecuado para una situación dada y lo instancia para ella.

El *componente reflexivo* se encarga de manejar las situaciones inesperadas que el agente debe resolver. Phoenix responde a este tipo de situaciones de tres formas:

1. Actos reflejos para detener acciones potencialmente perjudiciales para el agente. Los reflejos requieren poco tiempo de procesamiento y reportan poca información.
2. Recuperación de errores y replanificación, provocados por la llegada de un evento inesperado que impide la terminación de la acción o del plan actuales.

3. Una capacidad limitada para monitorizar el progreso del agente. Esta monitorización se acompaña de la generación de expectativas del resultado y de la comparación de dichas expectativas con el rendimiento actual del agente.

Autómatas situados (Rochenstein)

Se basa en la idea de que un agente interactua con un entorno que no se puede predecir completamente [Rochenstein and Kaelbling, 1995]. El comportamiento del agente se define en términos de cómo se produce la interacción con el entorno: qué acción se debe ejecutar en un estado en particular. Un autómata situado debe ser capaz de responder a un entorno cambiante.

La principal diferencia en la aproximación de Rochestein y Kaelbling es que un agente realiza una "transducción" en lugar de aplicar una función que pase de unos datos de entrada a un resultado como salida.

Un *autómata* (o máquina) M es una estructura:

$$M = \langle S, \Sigma, A, \delta, \lambda, s_0 \rangle$$

donde:

- S es un conjunto de estados;
- Σ es el conjunto de estímulos o entradas;
- A es un conjunto de acciones;
- $\delta : S \times \Sigma \rightarrow S$ es la función de próximo estado;
- $\lambda : S \rightarrow A$ es la función de salida;
- $s_0 \in S$ es el estado inicial.

El autómata interactua con su entorno enviando salidas, que cambian el estado del entorno, y el entorno produce estímulos $\sigma \in \Sigma$ que, junto con el estado interno de la máquina $s \in S$, provoca una transición de estados δ que define la próxima salida.

Sea Φ el *conjunto de condiciones* del mundo: todo aquello que puede ser cierto en el mundo. Un estado es un par *(estado-entorno, estado-interno)*. Si w es el estado del entorno, denotamos por $\phi(w)$ el hecho de que ϕ sea cierto en w.

Para cada estado $\sigma \in \Sigma$, se define una función que mapea Φ sobre Φ. La idea es que ϕ/σ es la condición más específica sobre el mundo que debe garantizarse en un instante de tiempo t', dado que ϕ se cumple en t y el estímulo σ ha sido detectado en el intervalo de tiempo $[t, t']$.

CIRCA (Musliner)

CIRCA *("Cooperative Intelligent Real-Time Control Architecture")* es la única arquitectura de agente capaz de manejar entornos de tiempo real estricto. La evolución actual de esta arquitectura recibe el nombre de SA-CIRCA

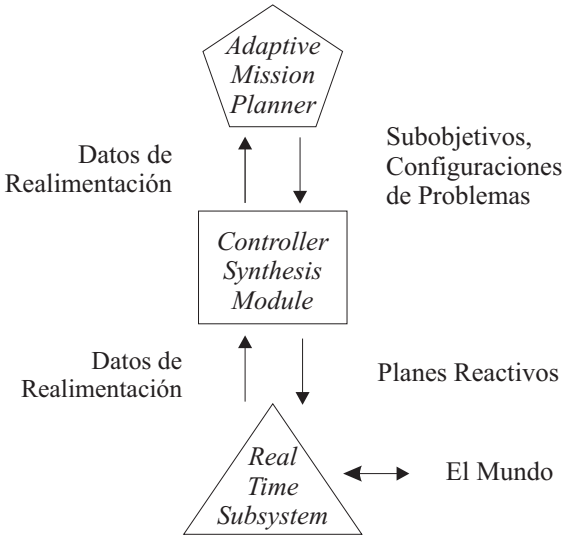


Figura 2.5. División en módulos de la arquitectura CIRCA

(*Self-Adaptive* CIRCA –CIRCA Auto-adaptativo–)[Goldman et al., 2001] [Musliner, 2002].

Aunque dicha arquitectura incorpora aspectos tanto de sistemas en tiempo real como de sistemas de IA, no se puede considerar como una aplicación de la idea que subyace en la IA en tiempo real: adaptar los métodos de IA para trabajar con restricciones de tiempo real. Esto es debido a que CIRCA es una arquitectura compuesta por tres módulos funcionales que operan en paralelo sin competir por los recursos (véase Figura 2.5).

La arquitectura de CIRCA identifica tres módulos o subsistemas:

- **Subsistema de tiempo real** (RTS). Este módulo ejecuta planes de control predecibles en tiempo real, que perciben el estado del mundo y responden con acciones que preservan la seguridad del sistema y tratan de conseguir sus objetivos.
- **Módulo de Síntesis de Controladores** (CSM). Este módulo es el encargado de construir dinámicamente los planes de control reactivo que ejecuta el RTS.
- **Planificador de Misión Adaptativo** (AMP). Este tercer y último módulo es el control de más alto nivel de CIRCA. El AMP es el responsable de dividir la misión global del sistema en fases más pequeñas y que intersectan, denominadas *regiones de competencia*. Cada una de estas regiones pueden quedar cubiertas por un plan de control reactivo sintetizado automáticamente. El AMP le encarga al CSM que cree estos nuevos planes de control; bien por adelantado, antes de que comience la misión, bien en ejecución, cuando las condiciones cambian. Esta síntesis (o planificación) de

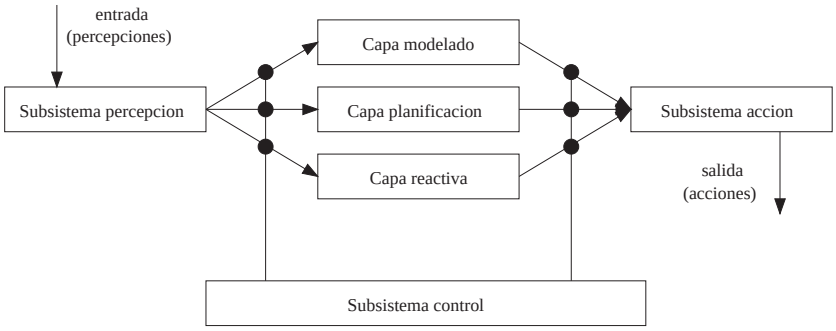


Figura 2.6. Arquitectura de capas de TouringMachines

controladores en línea proporciona a un agente basado en la arquitectura CIRCA un mecanismo de adaptación.

2.4.2. Arquitecturas híbridas

La principal crítica a las arquitecturas reactivas es que, como su propio nombre indica, se centran en la reactividad del agente: atender adecuadamente a los cambios que se produzcan en el entorno. Pero esta no es la única característica de un agente. La proactividad es otra de las características que se deben tener presentes para que un agente pueda considerarse una entidad inteligente.

Las arquitecturas híbridas intentan equilibrar ambas características: reactividad y proactividad, de manera que se obtenga un agente capaz de responder ante cambios en el entorno pero también sean sus propios objetivos los que dirijan su comportamiento. Es este balance entre las dos propiedades lo que hace tan difícil el proceso de construcción de los agentes.

Las arquitecturas híbridas mantienen los niveles de reactividad conseguidos por las arquitecturas reactivas, e intentan autonomía mediante procesos de razonamiento complejos, a menudo simbólicos. A continuación se exponen las arquitecturas híbridas más conocidas que pueden aplicarse a entornos de tiempo real.

TouringMachines (Ferguson)

Ferguson describe en su tesis doctoral [Ferguson, 1992] una arquitectura horizontal para agentes autónomos móviles, que deben moverse en un entorno cambiante. La arquitectura de las TouringMachines está formada por tres capas: una capa reactiva, una capa de planificación y una capa de modelado (véase Figura 2.6). Las tres trabajan de forma concurrente y todas ellas se conectan al subsistema sensorial del agente, del que reciben información acerca de sus percepciones, y a su subsistema de acción, al que envían sus acciones.

La *capa reactiva* está programada para calcular respuestas específicas para el dominio a los estímulos del entorno. Habitualmente se encuentra codificada por hardware. La *capa de planificación* es la responsable de generar y ejecutar planes para el cumplimiento de las tareas de posicionamiento a largo plazo que el agente debe desempeñar. Los planes construidos se almacenan en una biblioteca de planes como planes parciales jerárquicos. A partir de ellos, el agente es capaz de construir planes lineales y compaginar su ejecución. La *capa de modelado* proporciona al agente la capacidad de modificar sus planes basándose en cambios en el entorno que la capa de planificación no puede manejar. Una tarea muy importante de esta capa es la representación de modelos mentales y causales de los otros agentes.

Pero las capacidades de las tres capas anteriores aisladas no pueden garantizar un funcionamiento adecuado. Un módulo de control se encarga de coordinar el funcionamiento de estas tres capas, controlando el acceso a los sensores y a los actuadores. Su conocimiento se describe mediante reglas de control que se activan en función del contexto. Pueden ser de dos tipos: *censores* y *supresores*, según actúen sobre el subsistema sensorial o el subsistema de acción respectivamente.

InterRap (Müller)

La arquitectura de agente de InterRap [Muller, 1996] es una de las primeras en integrar realmente comportamiento reactivo, dirigido por objetivos (proactivo) y social (interacción con otras entidades de forma explícita) de un agente, a través de:

- un conjunto de capas de control jerárquicas,
- una base de conocimiento que permite la representación del conocimiento a diferentes niveles de abstracción, y
- una estructura de control bien definida que asegura la correcta interacción entre las capas de control.

El avance que representa la arquitectura de InterRap frente a las TouringMachines es que los agentes se comunican entre sí y construyen planes compartidos para conseguir un objetivo común.

Se trata de una arquitectura vertical, en la que la capa de conducta (i) toma el control inicial del agente, (ii) accede a su sistema de sensorización y (iii) envía la información hacia capas superiores hasta encontrar una con el nivel de competencia suficiente como para tratar la información. Una vez que la capa determina el conjunto de tareas que se deben realizar con la información, las envía de nuevo hacia la capa de conducta: la única que tiene la capacidad de acceder al medio a través de un interfaz con el mundo exterior (véase Figura 2.7).

La capa de conducta es la que permite a InterRap manejar situaciones de tiempo real, utilizando *patrones de comportamiento* (PoB). Un PoB establece las acciones que se deben realizar cuando se cumple una condición. Incluye

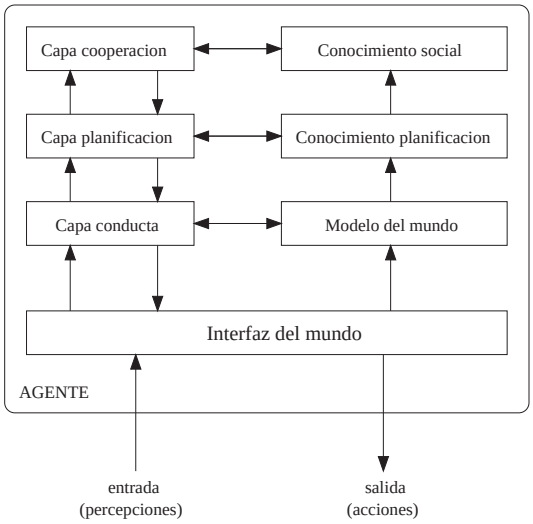


Figura 2.7. Arquitectura de capas de INTERRAP

además una serie de condiciones, que determinan si el comportamiento se ha ejecutado con éxito o si ha fallado, una prioridad estática y otras características. Pueden ser de dos tipos: *reactivas*, que se activan mediante eventos externos, y *procedurales*, que se activan a partir de los requerimientos de la capa de planificación.

Decaf (Graham)

DECAF *(Distributed, Environment-Centered Agent Framework)* es una arquitectura que permite la construcción de agentes para entornos de tiempo real *'soft'*, es decir, entornos en los que las tareas deben intentar cumplir sus plazos de ejecución, pero un fallo no provoca una catástrofe en el sistema [Graham, 2001].

La arquitectura de DECAF proporciona cinco funciones básicas (véase Figura 2.8):

- *Inicialización:* toma un plan como entrada y produce un conjunto de patrones de tareas para ejecutar. Este conjunto de patrones de tareas son las creencias del agente, en el sentido de sus capacidades, en oposición a creencias como conocimiento del dominio.
- *Gestión:* Determina nuevos objetivos para el agente a partir de los mensajes de entrada.
- *Planificación:* Toma un objetivo determinado y los patrones de tareas que lo pueden satisfacer y produce una instanciación concreta del plan que encola en la lista de tareas pendientes.

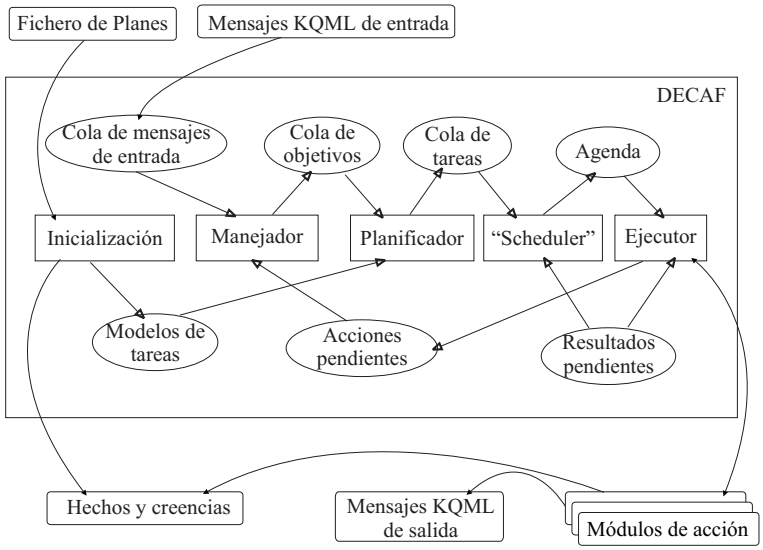


Figura 2.8. Arquitectura de agente de DECAF

- *Scheduling:* Planifica cómo ejecutar las tareas pendientes, construyendo una agenda que establece la secuencia de ejecución del agente. Las acciones mantienen una relación de orden parcial, indicando qué acciones pueden ejecutarse en paralelo y entre cuáles hay una prelación determinada.
- *Ejecución:* Genera las intenciones del agente a partir de la agenda de acciones. Para DECAF, una intención comprende el compromiso a más bajo nivel de los cursos de acción del agente. Su responsabilidad principal es determinar cuándo una acción se ha completado y determinar qué acciones se pueden ejecutar como consecuencia de esta finalización.

Las *acciones* son las unidades fundamentales planificables de DECAF. Tienen asociado un *perfil de ejecución*, que ayuda al *scheduler* a determinar la secuencia óptima de ejecución para el conjunto de tareas que debe ejecutar. Es un vector que consta de tres componentes: el coste, la duración y la utilidad de la acción.

A partir de los perfiles de ejecución de las acciones, DECAF determina la *función de acumulación característica* de cada tarea. Si la tarea está formada por n subtareas, puede ocurrir que sólo deba ejecutarse una de ellas o que sea necesario completarlas todas. La función de acumulación característica permite determinar qué subconjunto de subtareas ejecuta DECAF.

2.5. Conclusiones

La arquitectura abstracta de un agente situado y la definición de un entorno de tiempo real son muy similares. Pero ninguna se adapta exactamente

a los requerimientos de un agente inteligente de tiempo real. Basta con eliminar la exigencia de estados instantáneos de la arquitectura abstracta y el determinismo de los STR para disponer de un agente situado en un entorno dinámico.

De los tres tipos de arquitecturas para la construcción de agentes inteligentes, las arquitecturas reactivas y las híbridas son las más adecuadas para la construcción de agentes que operen en entornos de tiempo real.

Las arquitecturas reactivas huyen de aparatos lógicos y se centran en la construcción de agentes "que funcionen". Es decir, enfocan sus esfuerzos a la construcción de agentes más simples desde el punto de vista formal, pero directamente implementables en una máquina. Su punto fuerte es la gran eficiencia que se consigue en la ejecución. Son arquitecturas centradas sobre todo en el comportamiento reactivo de un agente que trabaja en el mundo real, rodeado de imprevistos a los que debe responder de forma adecuada.

El principal problema de estas arquitecturas es precisamente ése: su falta de formalización, que dificulta la prueba de los posibles comportamientos del agente. En este caso, la única forma de demostrar su funcionamiento es a través de simulaciones o pruebas controladas. Pero la variabilidad tan grande del entorno puede hacer que las pruebas realizadas no sean válidas cuando el agente se pone en funcionamiento en un entorno real.

Las arquitecturas híbridas permiten incorporar procesos de razonamiento más elaborados sin renunciar al alto grado de reactividad conseguido por las capas inferiores de su jerarquía. De esta forma, se puede construir agentes que se comportan bien en un medio físico.

Esta última solución es la más adecuada cuando se trata de construir entidades artificiales que se ejecutan durante largos periodos de tiempo – potencialmente, un tiempo infinito –, con una baja participación humana en su toma de decisiones, que trabajan en entornos dinámicos y deben adaptar su comportamiento a las diferentes situaciones en las que se encuentran, algunas de ellas incluso no predichas, y por lo tanto no preprogramadas, por sus diseñadores.

3

ARTIS: Arquitectura de agente de tiempo real

3.1. Introducción

Hasta el momento, las aplicaciones de ARTIS han sido típicamente dominios de problemas que se denominan de "control de procesos". En estos dominios existe un sistema físico que se desea controlar y cuyo comportamiento queda completamente caracterizado. Dentro de este tipo de sistemas se pueden encontrar desde procesos relativamente sencillos, como un tanque de aguas residuales, hasta problemas muy complejos y cuyo buen funcionamiento es especialmente crítico, como es el caso de una central nuclear.

La arquitectura que da soporte a los agentes ARTIS (AA) permite un enfoque distribuido para la resolución de problemas, donde existen varias entidades con un grado de autonomía elevado que colaboran entre sí para resolver el problema. Cada una de estas entidades (*in-agent*) posee el conocimiento de resolución de un subproblema específico a distintos niveles de abstracción. El método que se aplica depende del tiempo que tenga disponible el sistema para razonar antes de tener que proporcionar una respuesta al entorno.

Al contar con una arquitectura claramente definida y en un grado avanzado de desarrollo, resulta necesario probar la fiabilidad de las aplicaciones que se modelan con distintos ejemplos. Hasta el momento, todas las aplicaciones pertenecían claramente al campo del control de procesos. Sin embargo, hay un interés especial en ampliar el área de aplicación de los AA para resolver cualquier tipo de problema con características de tiempo real estricto cuyos métodos de resolución necesiten un comportamiento inteligente.

En el presente capítulo se plantean los cambios necesarios en el modelo de agente ARTIS para adaptarla a los requerimientos de los nuevos problemas. En primer lugar, se expone el modelo conceptual de AA, que incluye todas las entidades que forman parte de la arquitectura en su estado actual de desarrollo.

En segundo lugar, se propone un modelo de tareas que integra los componentes de los sistemas dinámicos clásicos con componentes no acotadas que permitan mejorar la solución de problemas tanto de análisis co-

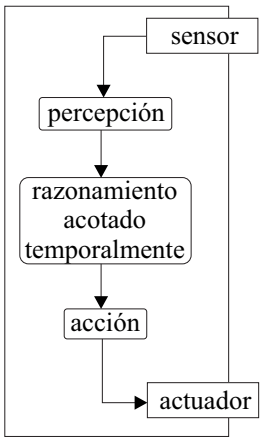


Figura 3.1. Estructura genérica de un agente de tiempo real

mo de síntesis. Se propone adoptar la clasificación de tareas intensivas en conocimiento propuesta en la metodología de desarrollo de CommonKADS [Schreiber et al., 2000]. Esta estructura permite construir plantillas basadas en la jerarquía de entidades de un AA para ayudar al usuario en la implementación de los diferentes métodos de resolución de problemas.

Por último, se construye un modelo formal que integra todas las propuestas actuales en la arquitectura de agente ARTIS. Una parte importante es el estudio de las propiedades de seguridad y viveza, que corresponden en la arquitectura con la definición de restricciones de integridad y objetivos respectivamente.

3.2. Estado actual de ARTIS

ARTIS se plantea inicialmente como una arquitectura para sistemas inteligentes de tiempo real [García-Fornés, 1996]. La arquitectura inicial ha sufrido varias modificaciones desde entonces. Las más relevantes son: (1) la inclusión de un modelo de representación y razonamiento temporal [Onaindía, 1997] [Onaindía and Rebollo, 1998]; (2) la adaptación de ARTIS para el desarrollo de agentes inteligentes de tiempo real [Botti et al., 1999]; y (3) la flexibilización de la estructura de tareas de bajo nivel [Terrasa, 2000][Terrasa et al., 2002].

La definición de agente de tiempo real se apoya especialmente en el concepto de *reactividad*, pues se trata de una característica básica para el buen funcionamiento del sistema. Desde este punto de vista, un agente realiza tres procesos. En primer lugar, *percibe* el estado actual del entorno a través de un conjunto de sensores. A continuación, *razona* con la información de que dispone para calcular la respuesta adecuada a la situación que ha detectado y satisfacer sus objetivos. Por último, *actúa* sobre su entorno con la intención de modificarlo.

Estas tres fases: percepción, cognición y acción, se repiten de forma continua en el ciclo de ejecución del agente y determinan su funcionamiento global. En el caso especial de los agentes de tiempo real, su característica más importante es que la calidad de la respuesta obtenida depende, en gran medida, del tiempo que se tarda en obtenerla. Así, es mejor una respuesta no óptima a tiempo que la mejor respuesta posible cuando ésta ya no es útil para el sistema. Por ello, el proceso de razonamiento debe estar acotado temporalmente, conociendo al menos su tiempo de ejecución en el peor caso.

ARTIS se ha implementado teniendo en cuenta esta característica. Sus procesos de razonamiento siguen un comportamiento que podríamos calificar como *"anytime"*: la calidad de la respuesta obtenida mejora con el tiempo y puede interrumpirse en cualquier momento proporcionando siempre una respuesta.

Los procesos de razonamiento están divididos en dos fases o capas: una *refleja*, que calcula una primera respuesta de baja calidad pero en un tiempo acotado, y una *deliberativa*, que mejora la respuesta durante todo el tiempo que tenga disponible. De esta forma, se sigue garantizando el comportamiento en tiempo real del agente, sin renunciar a utilizar procesos de razonamiento elaborados que incluyan técnicas de IA.

En su estado actual de desarrollo, la arquitectura de agente ARTIS dispone de un soporte para tareas de bajo nivel que garantiza su ejecución en tiempo real. Este módulo trabaja con varias políticas de planificación expulsivas con prioridades estáticas. Se ha implementado como extensiones de RT-Linux, por lo que las tareas de tiempo real se ejecutan a una prioridad mayor que cualquier otro proceso del sistema operativo. Como soporte a esta parte de tiempo real, se dispone de un analizador fuera de línea que es capaz de determinar si el conjunto de tareas que forma el agente ARTIS son planificables o no. Este analizador de la planificabilidad no construye un plan de ejecución; sólo comprueba si las tareas garantizan sus propiedades temporales (*"deadline"* y periodo).

En el tiempo que dejen disponible las tareas críticas de tiempo real (llamado *"slack"*), el agente ejecuta procesos de resolución de problemas más elaborados para mejorar las respuestas básicas que ha proporcionado la primera capa. Para ello, un planificador de segundo nivel determina, para cada hueco de *"slack"*, qué procesos de resolución de problemas van a ejecutarse y en qué orden.

Con el fin de facilitar la construcción de un agente ARTIS, se ha desarrollado un entorno de programación, denominado InSiDE [Julián et al., 2000], que integra la programación de los distintos componentes que forman la arquitectura de agente ARTIS y el test de planificabilidad fuera de línea.

La arquitectura de AA está siendo modificada actualmente en varios frentes:

1. Se ha detectado la necesidad de **inclusión de un planificador** en ARTIS. Por el momento no se plantea como un componente interno del AA, sino

como una tarea más, definida por el usuario, que debe competir con el resto por los recursos del sistema (especialmente, uso del procesador). Debe ser un planificador reactivo, que compagine la creación del plan con su ejecución y, presumiblemente, con capacidades de replanificación. También resulta adecuado que sea capaz de trabajar con distintos niveles de abstracción.

2. **Metaconocimiento de un AA.** Dentro de la investigación del Grupo de investigación de Tecnología Informática e Inteligencia Artificial (GTI–IA), la parte más relevante para el presente trabajo de tesis es la consideración de distintas conductas para un AA que dependen tanto de la situación actual del entorno como del estado interno del AA. Otra parte importante es la necesidad de notificar ciertos eventos, especialmente los que hacen referencia a información temporal (llegada de nuevos valores actuales o cumplimiento de predicciones). Estos eventos se emplean para disparar reglas de control o realizar cambios de conducta. Una última relación es el mantenimiento de información relevante para este metarazonamiento dentro de las creencias del agente. Conceptualmente, se puede considerar que el metaconocimiento de un AA se implementa mediante un conjunto de *in-agent* de control.
3. Adaptación de la arquitectura para la **construcción de SMA de tiempo real** basados en agentes ARTIS. Para ello, es necesario dotar a los agentes de un mecanismo de comunicación que, en determinadas condiciones, pueda acotarse temporalmente. La arquitectura para el SMA se denomina SIMBA [Julián et al., 2002]. Su desarrollo requiere la modificación del AA de forma que permita el envío y recepción de mensajes para su comunicación con otros agentes, que en determinadas ocasiones puede estar acotada temporalmente por necesidades del entorno.
4. **Metodología de construcción de SIMBA.** Asociada a la arquitectura de SIMBA, es necesario proponer una metodología para el desarrollo de sistemas multiagente que sigan este modelo. Ha hecho necesaria la inclusión de nuevos conceptos, tales como roles, responsabilidades o tareas de alto nivel. Es una extensión de MESSAGE [Caire et al., 2001] para incluir las características necesarias para construir agentes de tiempo real. La metodología de análisis y diseño de SIMBA se denomina RT-MESSAGE [Julián, 2002][Julián and Botti, 2004]

3.3. Arquitectura de agente ARTIS

La arquitectura de AA que aquí se presenta aúna características propias de sistemas de tiempo real estricto con un comportamiento inteligente. Es una extensión del modelo de pizarra [Nii, 1986b][Nii, 1986a], adaptado para entornos de tiempo real estricto.

Los aspectos de la arquitectura de AA más relevantes para el presente trabajo de tesis son los siguientes:

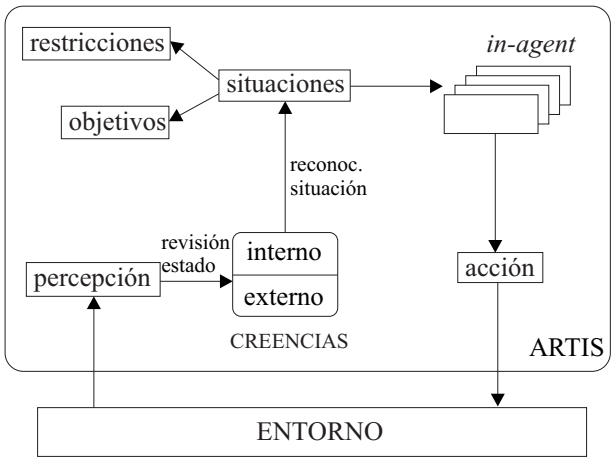


Figura 3.2. Modelo conceptual de AA

1. **Comportamiento reactivo**. El AA está situado en un entorno conocido (aunque no determinista) y debe reaccionar a tiempo ante los eventos que se producen en dicho entorno.
2. **Dirigido por objetivos**. El AA debe seleccionar sus propias acciones basándose en las metas que desea alcanzar.
3. El **tratamiento del tiempo** debe ser algo natural en el modelo que se utilice. Los métodos de gestión de la información temporal deben garantizar dos propiedades: predecibilidad e interrumpibilidad.

3.3.1. Modelo conceptual de agente ARTIS

Una aproximación frecuente en los modelos de agentes es considerar que éstos tienen un *estado mental* determinado. Una de las arquitecturas más habituales en este sentido son las arquitecturas BDI (creencias, deseos e intenciones) [Georgeff and Rao, 1991] [Georgeff and Rao, 1995]. Tiene muchos puntos en común con el modelo que se plantea para ARTIS, pero también tiene una serie de elementos que la diferencian.

El modelo conceptual de AA que aquí se presenta describe los componentes que forman el estado mental del agente y la relación que existe entre ellos. Se basa en el modelo conceptual de INTERRAP [Muller, 1996], el cual a su vez está influenciado por la arquitectura abstracta de agente de Bratman [1988].

Desde el punto de vista del modelo conceptual de un AA (véase Figura 3.2), encontramos los siguientes componentes:

- La *percepción* actual del agente. Está formada por toda la información del entorno que le llega través de sus sensores en un instante de tiempo determinado t. Puede definirse un conjunto Σ formado por los datos de los sensores. Denotamos con Σ^t las percepciones de AA en el instante t.

- Un conjunto de *creencias* que representan tanto el modelo del entorno que mantiene en AA como su estado interno. Puede definirse un conjunto E que denota los estados del entorno y un conjunto L para los estados internos (locales) del agente. El estado del sistema es un par (e, l) donde $e \in E$ y $l \in L$. Si $\Delta = E \times L$, denotamos por E^t, L^t y Δ^t los estados del entorno, del agente y del sistema respectivamente, en el instante de tiempo t.
- Un conjunto de *situaciones* que describen un escenario del entorno. El agente sólo puede encontrarse en una situación. La respuesta que proporciona el agente ante el mismo evento puede ser distinta en situaciones diferentes.
- Un conjunto de objetivos del agente $\mathcal{G}$ para una situación dada. Distinguimos entre *objetivos* propiamente dichos y *restricciones*. Se define un conjunto $\mathcal{G}^+ = \{g, g', \ldots\}$ que son los objetivos actuales del AA, y un conjunto de restricciones $\mathcal{G}^- = \{r, r', \ldots\}$. La diferencia entre ambos es la misma que se da entre las reglas de viveza *("liveness")* y de seguridad *("safety")* de los sistemas concurrentes, definidas formalmente por Alpern y Schneider [1985] y matizadas posteriormente por varios autores [Abadi et al., 1991].
- Un conjunto de *in-agent* que contienen el conocimiento de resolución de problemas y definen cómo actúa el agente en la situación actual.
- La siguiente *acción* que va a realizar el AA, generada por la ejecución de los distintos *in-agent*.

Un *estado* define una *situación* que activa, o permite que siga activo, un *comportamiento* que determina el conjunto de *objetivos* y *restricciones* actuales para el agente, así como el *conocimiento* de resolución de problemas necesario para manejar la situación.

A grandes rasgos, el funcionamiento de un agente ARTIS sigue el ciclo *percepción–cognición–acción* típico de los sistemas de tiempo real. En primer lugar, el agente percibe el estado actual del entorno a través de sus sensores. Cuando los valores difieran significativamente de una lectura anterior, el agente considera que se ha producido un cambio en el entorno y actualiza su conjunto de creencias. El agente determina la situación en la que se encuentra dependiendo del nuevo estado y del conjunto de restricciones y objetivos definidos actualmente. Una vez que ha reconocido la situación, es capaz de aplicar el conocimiento necesario para desenvolverse en la situación actual, activando y desactivando los *in-agent* correspondientes (que tienen el conocimiento de resolución de un subproblema determinado al que se enfrenta el agente). Llegados a este punto se ejecutan todas las partes críticas de los *in-agent*, de forma que se construye una primera respuesta en un plazo de tiempo acotado. Si queda tiempo disponible, se intenta mejorar la calidad de la respuesta mediante procesos de resolución de problemas más elaborados. Tras este proceso, se ejecutan las acciones calculadas sobre el entorno (y se modifica el posible estado interno del agente si es necesario). Estas acciones provocarán un cambio en el entorno que será percibido de nuevo a través de los sensores del agente, comenzando todo el ciclo.

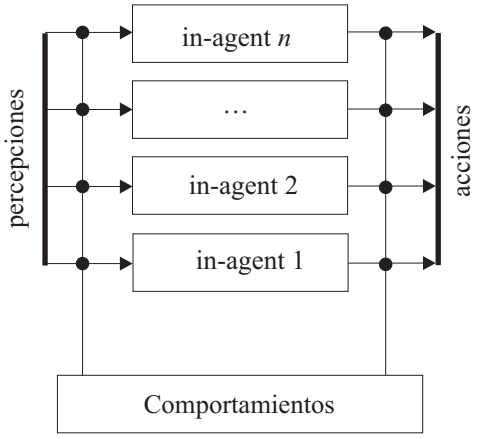


Figura 3.3. Jerarquía de entidades de un agente ARTIS

3.3.2. Jerarquía de entidades de un agente ARTIS

La división en entidades de ARTIS proporciona una jerarquía de niveles de abstracción para organizar el conocimiento de resolución de problemas. Una de las principales razones es ofrecer las ventajas de la descomposición modular de problemas y la reutilización de código.

Un AA está compuesto por una serie de capas horizontales que se ejecutan en paralelo ante una entrada del entorno (véase Figura 3.3). La principal diferencia con las arquitecturas horizontales "tradicionales" es que no se selecciona una única acción para ejecutar, sino que se ejecutan todas las acciones que se han calculado en cada capa.[1] Para ello, previamente se realiza un análisis de la planificabilidad que asegura que habrá tiempo suficiente en el sistema para ejecutar las acciones calculadas. Estas capas, que resuelven un subproblema para el agente, reciben el nombre de agentes internos o *in-agent* .

No es necesario que todas las capas estén activas en un momento dado. También puede ocurrir que los requerimientos temporales para la ejecución de cada *in-agent* cambie a lo largo de la vida del agente. Estos hechos se representan mediante el concepto de situación en general, que en el caso de ARTIS se denominan *comportamientos*.

Siguiendo la representación de AA de la Figura 3.3, puede considerarse que el comportamiento activo se encarga de habilitar únicamente las percepciones y las acciones que pertenecen a dicho comportamiento, deshabilitando todas las demás.

[1] Aunque por simplicidad en los procesos de formalización de la arquitectura de agente ARTIS seguiremos asumiendo que las acciones se ejecutan de forma secuencial.

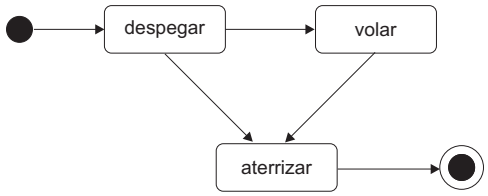


Figura 3.4. Ejemplo de transiciones entre comportamientos en un diagrama de estados

Comportamientos

Un AA va a encontrarse en situaciones muy diferentes a lo largo de su vida. Ni el conocimiento ni las habilidades que se precisan en cada una de ellas son las mismas. Una forma de mejorar el rendimiento del AA es focalizar su atención en un subconjunto de los datos, así como limitar los *in-agent* que se encuentran activos simultáneamente, compitiendo por los recursos del agente.

Un *comportamiento* de un AA es un conjunto de *in-agent*, cada uno caracterizado por unas restricciones de tiempo real, que están activos al mismo tiempo. Dos comportamientos son diferentes cuando:

1. el conjunto de *in-agent* es diferente, o
2. varía alguna de las características temporales (restricciones de tiempo real) de algún *in-agent*.

Cada comportamiento necesita un análisis de la planificabilidad para comprobar que se cumplen los requerimientos de tiempo real de todos los *in-agent* que lo forman. Reciben especial atención los cambios de comportamiento, que se realizan de forma dinámica. El AA no se detiene y debe seguir garantizando la ejecución de las tareas que tiene activas. Es un concepto muy cercano a los *cambios de modo* de los sistemas de tiempo real [Real, 2000].

Por ejemplo, el diagrama de transición de estados de la Figura 3.4 representa un AA que controla el sistema de navegación de un avión. Diferencia tres situaciones: el despegue, el aterrizaje y el vuelo. El AA necesitará unos *in-agent* diferentes para cumplir con las responsabilidades que conlleva cada situación. Antes de poner el sistema en funcionamiento, se realizan varias pruebas de planificabilidad de los componentes fuera de línea; una por cada comportamiento. Además, se debe comprobar que se pueden realizar todas las transiciones posibles entre comportamientos, garantizando la ejecución de los componentes críticos del agente. En este caso, se deben estudiar tres transiciones:

- despegar → volar
- volar → aterrizar
- depegar → aterrizar

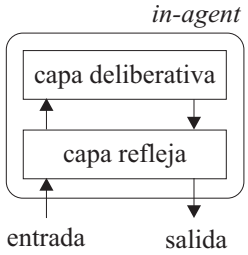


Figura 3.5. Estructura interna de un *in-agent*

Agente interno (in–agent)

Los problemas para los que ARTIS resulta una herramienta útil son demasiado complejos para considerarlos como un todo y abordarlos como un único problema. La arquitectura del AA permite la división del trabajo en entidades que colaboran entre sí: los *in-agent*.

Internamente, un *in-agent* tiene una arquitectura por capas: una *refleja*, que proporciona una respuesta de calidad mínima en un tiempo acotado, y una *deliberativa* de tiempo real. Si tiene tiempo suficiente para finalizar su proceso de razonamiento, la respuesta del *in-agent* es esta última, pues por lo general es una respuesta de mejor calidad.

El *in-agent* puede verse como una arquitectura de capas verticales. En primer lugar se ejecuta la capa refleja, que lee el estado actual del entorno a través de los sensores y modifica las creencias del agente para que reflejen el nuevo estado. A continuación, se realiza un proceso de razonamiento cuyo tiempo de ejecución en el peor caso está acotado. Este proceso determina una primera respuesta, que se traduce en la selección de las acciones a realizar sobre el entorno. Si queda tiempo disponible, se cede el control a la capa deliberativa, que mejora la respuesta hasta agotar el tiempo de ejecución que se le ha asignado. Si le ha dado tiempo a calcular las acciones, serán estas últimas las que toma el agente para su ejecución sobre el entorno. Si la capa deliberativa no proporciona respuesta alguna, se ejecutan las acciones de la capa refleja.

3.4. Resolución de problemas con ARTIS

Los tipos de problemas que se pueden resolver con ARTIS se caracterizan por ser problemas intensivos en conocimiento y con características de tiempo real. Esto hace que debamos revisar algunos de los planteamientos que provienen de la Ingeniería del Conocimiento para aplicarlos a la arquitectura de AA.

Las técnicas de modelado en Ingeniería del Conocimiento están basadas en la Taxonomía de Niveles de Newell [1982]. Siguiendo esta taxonomía, existen

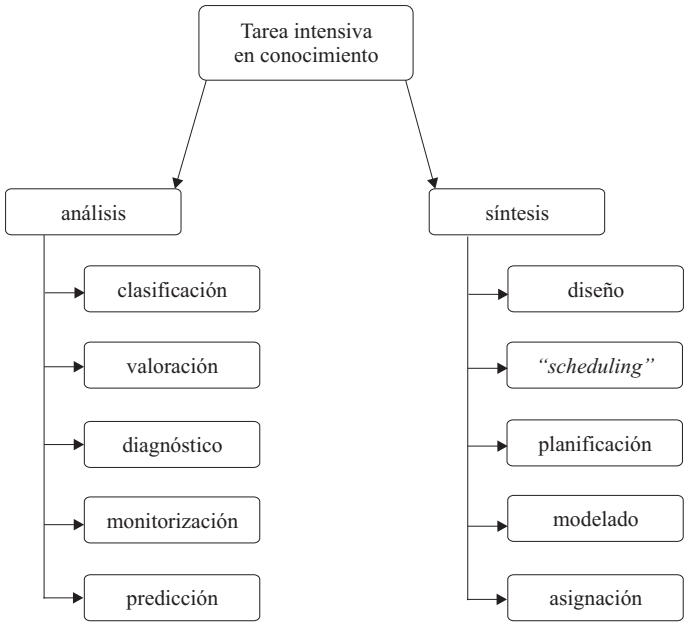


Figura 3.6. Tipos de tareas de CommonKADS

varias clasificaciones, como las propuestas por Hayes-Roth [1983] y Clancey [1992]. De todas ellas, escogeremos la clasificación que se propone para CommonKADS [Schreiber et al., 2000]. El motivo fundamental es que es la que más se ajusta a la línea propuesta en trabajos previos en el grupo de investigación [Henao, 2001].

3.4.1. Modelo de tareas de CommonKADS

En este modelo de tareas se distinguen dos tipos: tareas de *análisis* y tareas de *síntesis*. La diferencia principal entre ambas es que las primeras trabajan con un sistema preexistente, mientras que el objetivo de las segundas es construir el propio sistema: el sistema no existe y es el resultado de la ejecución de la tarea.

CommonKADS propone construir una serie de elementos reutilizables a los que denomina *plantillas*. Cada plantilla proporciona una estructura que permite incorporar el conocimiento de inferencia necesario para resolver un problema–tipo. Las plantillas pueden combinarse entre sí para resolver tareas complejas que requieren simultáneamente varios tipos de conocimiento. La clasificación es la que muestra la Figura 3.6.

Tareas de análisis

Las tareas de análisis pueden ser de cinco clases: *clasificación, valoración, diagnóstico, monitorización* y *predicción*.

Clasificación

La **clasificación** puede considerarse como una tarea avanzada para la lectura de información. A partir de la información que leen los sensores, la tarea es capaz de reconocer el objeto al que corresponden los datos. Este caso es especialmente relevante, por ejemplo, para un robot: debe reconocer las paredes a partir de las señales que le llegan de los infrarrojos, identificar balizas y señales para recalcular la posición actual, o incluso interpretar una imagen que recoja a través de una videocámara para reconocer un objeto.

Valoración

La tarea de **valoración** es, con diferencia, la más simple de todas. Cada uno de los ciclos en una tarea de monitorización puede considerarse una valoración. Un ejemplo típico es un termostato: en función de la temperatura actual de la habitación (valores de entrada de los sensores) se determina la respuesta en términos de una clase de decisión, por ejemplo `subir/bajar/mantener`.

Diagnóstico

Si planteamos un sistema que no se limite a controlar el proceso físico, el siguiente nivel sería conseguir resolver fallos que se produzcan en el sistema y que lo llevan a un malfuncionamiento. El primer paso es el **diagnóstico**, donde se trata de identificar el componente defectuoso. Un ejemplo es el diagnóstico de fallos en una red eléctrica. Los niveles más complejos de tareas de diagnóstico llegan a sugerir la solución para corregir el problema. Pero este tipo de tareas complejas requiere claramente esquemas de conocimiento adicional (habitualmente, tareas de planificación). La diferencia principal entre una tarea de diagnóstico "estándar" y una tarea de monitorización es que esta última asume que el sistema funciona correctamente, ajustándose al modelo que se ha empleado para su diseño. Sin embargo, en las tareas de diagnosis no podemos asumir que el sistema va a comportarse como el modelo, pues un componente defectuoso introduce una gran impredecibilidad en el sistema.

Monitorización

La **monitorización** es el núcleo central de trabajo en un sistema de control de procesos y se relaciona directamente con la definición de agente de Russell [1995], según la cual un agente es una entidad situada en un entorno que percibe y actúa sobre el mismo. El objetivo de una tarea de monitorización es comprobar en todo momento el estado del sistema real e indicar si el sistema se encuentra bajo control o se ha detectado una situación anormal o especial para el usuario.

Predicción

El tipo de tarea de análisis más compleja es la **predicción**. Intenta determinar el estado del sistema en un instante de tiempo futuro, atendiendo al estado actual, a las posibles líneas de razonamiento o al comportamiento observado en situaciones semejantes. Si, además, se combina con una tarea de diagnóstico, nos puede indicar qué componentes tienen mayor probabilidad de fallo en una situación dada, de manera que el sistema pueda incluso evitar llegar a esa situación reduciendo el esfuerzo de los componentes implicados.

Mientras que las demás tareas trabajan con información estática, para realizar predicciones es indispensable disponer de un modelo de representación y razonamiento que sea capaz de manejar el tiempo como una dimensión más de los datos.

En nuestro caso tiene un interés adicional, pues **proporciona la capacidad de imaginación**, que da título al presente trabajo de tesis. Al prever el comportamiento futuro del sistema, un agente es capaz de razonar con posibilidades que todavía no se han producido y adelantar sus reacciones en previsión de que ocurra en el futuro.

Tareas de síntesis

El modelo de tareas de CommonKADS propone cinco tipos de tareas de síntesis: *diseño, "scheduling", planificación, modelado* y *asignación* (véase Figura 3.6). De todas ellas, las tareas de modelado (que precisan una cierta dosis de creatividad) no se suelen automatizar en los sistemas. Su aparición se debe a que en CommonKADS se modela no sólo el conocimiento que se introducirá en el sistema software, sino también cualquier tipo de conocimiento que sea necesario para resolver del problema, independientemente del tipo de entidad que posea el conocimiento (usualmente, humanos).

Por otra parte, existen fuertes dependencias de las tareas de diseño con las tareas de planificación, y de las tareas de asignación con las de *"scheduling"*.

La propiedad más importante de las tareas de síntesis es que son **independientes de las entradas de los sensores**. Los resultados de las tareas se determinan fuera de línea. O si se hace en línea, no están trabajando directamente sobre el sistema real.

Diseño

Las tareas de **diseño** son aquellas que buscan la forma de construir un objeto físico a partir de unidades elementales. No es necesario que dichas unidades existan previamente: por ejemplo, el diseño de un nuevo modelo de coche, que precisa también del diseño de las piezas que lo componen. Un caso especial es el *diseño de configuración*, que parte de un conjunto de componentes conocidos, como las piezas de Lego o los componentes de un ordenador que hay que ensamblar. Este último tipo de tarea de diseño es más susceptible de automatizarse mediante un sistema informático.

El resultado de estas tareas puede verse como la secuencia de acciones que se debe realizar para obtener el producto final, donde una acción representa el ensamblado de uno de los componentes.

Planificación

La principal diferencia con el diseño es que las tareas de **planificación** buscan la ordenación de una serie de actividades en el tiempo. El conocimiento necesario para una tarea de planificación es radicalmente distinto. Ya no se habla de valores de datos en determinados instantes de tiempo, sino de acciones elementales que el planificador es capaz de manejar.

Su principal dificultad estriba en compaginar la construcción de los planes con su ejecución y con la posibilidad de tener que replanificar ante cambios impredecibles en el entorno.

Asignación

La tarea de **asignación** es una tarea bastante sencilla, en especial si la comparamos al resto de tareas de síntesis. Se dispone de dos conjuntos de objetos entre los ques se debe crear una relación (parcial). Pueden definirse restricciones que limiten los posibles emparejamientos. Ejemplos de este tipo de tareas son la asignación de despachos a los empleados o de muelles de atraque a los barcos.

"Scheduling"

Es una tarea que muy a menudo se realiza después de la planificación. El ***"scheduling"*** consiste en asignar una serie de recursos durante unas unidades de tiempo a un conjunto de actividades (que pueden haber sido generadas en un proceso de planificación). El resultado es la asignación de tiempos de inicio y de finalización a las distintas actividades de manera que cada una de ellas disponga de todos los recursos que necesita para su realización.

3.4.2. Adaptación del modelo de tareas en ARTIS

En el mundo real es difícil encontrar problemas puros que correspondan únicamente a uno de los tipos de problemas caracterizados en esta taxonomía. Lo habitual es que se trate de tareas complejas que precisan de la aplicación de varios de los métodos de resolución de problemas expuestos. Sin embargo, considerando que nuestro objetivo es construir un agente de tiempo real, que debe responder ante cambios en el entorno y mostrar cierto grado de iniciativa, se plantea una estructura inicial a la hora de relacionar los tipos de problemas a los que un AA se va a enfrentar.

Los métodos de resolución de cada tipo de tarea se deben integrar dentro de la estructura de capas de la arquitectura de AA. Desde el punto de vista del comportamiento en tiempo real de un AA, recordemos que los procesos

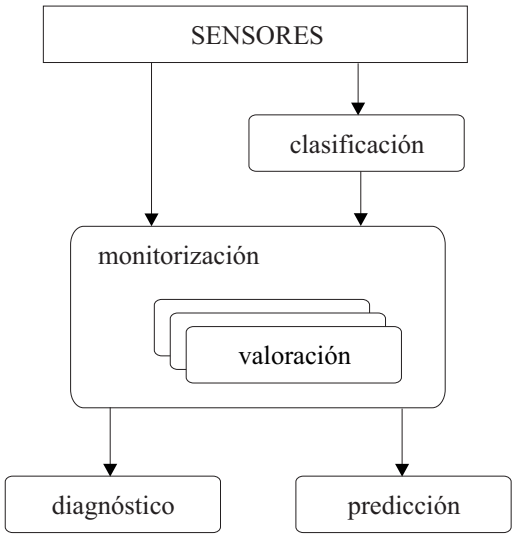


Figura 3.7. Tareas de análisis en ARTIS

cognitivos se descomponen en dos capas: una refleja, acotada temporalmente, y otra deliberativa, sin acotar (véase Figura 3.5).

Si consideramos únicamente el razonamiento que se realiza en la capa refleja, tenemos un agente que se comporta como el controlador de un sistema dinámico según la teoría de control clásica (véase Figura 2.1). Habitualmente, este problema se suele dividir en dos subproblemas:

1. **Estimación del estado actual:** consiste en determinar el estado actual del entorno evaluando las entradas del controlador.
2. **Regulación:** determinar una respuesta adecuada al estado que se ha deducido.

Los procesos de razonamiento de la capa deliberativa involucran a los tipos de problemas intensivos en conocimiento que se han descrito.

En la estructura de tareas pueden distinguirse dos zonas (véase Figura 3.9).

Tareas de análisis

Las tareas de análisis son tareas centradas en los datos de entrada. Por lo tanto, serán tareas próximas a la percepción del AA (véase Figura 3.7). En primer lugar, conectada directamente a los sensores, están las tareas de *clasificación*. Las entradas del entorno (directamente o tras un proceso de clasificación) pasan a las tareas de *monitorización*. Una característica de ARTIS es su aplicación al control de procesos, de las cuales la monitorización es la tarea fundamental y puede considerarse el núcleo central de ARTIS. El resultado de esta fase es un juicio de valor sobre la estabilidad del sistema: indica

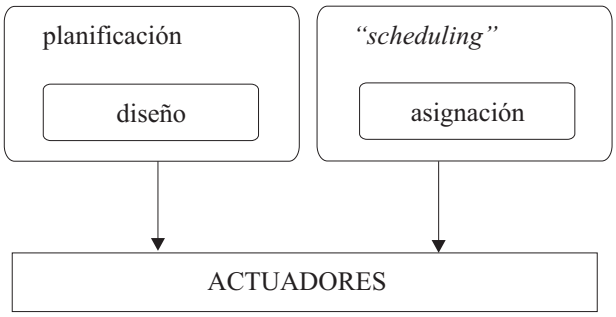


Figura 3.8. Tareas de síntesis en ARTIS

si se encuentra bajo control o si alguna de las variables que se comprueban se ha salido de su rango de aceptación. El estado actual del sistema puede emplearse para localizar posibles fallos en el entorno (tarea de *diagnóstico*) y también para intentar predecir el estado del entorno en un futuro (tarea de *predicción*).

Tareas de síntesis

El segundo grupo está formado por las *tareas de síntesis*, que se encuentran más cerca de los componentes de acción del AA. Básicamente, se trata de tareas de planificación y de *"scheduling"*, pues las tareas de diseño y asignación pueden considerarse incluidas dentro de las primeras (véase Figura 3.8).

En ARTIS, para atender los requerimientos de un sistema de tiempo real, es necesario disponer de dos tipos de planificación. En primer lugar, una planificación inicial clásica para establecer cómo alcanzar alguno de los objetivos del agente. En segundo, capacidades de replanificación que permitan ajustar un plan si se ha producido alguna situación imprevista que impida la ejecución de alguna de las acciones previamente planificadas. En ambos casos, se trata de un proceso complejo que no está acotado temporalmente.

La casuística para las tareas de *"scheduling"* es la misma. En este caso, se tienen en cuenta los recursos disponibles para que el AA realice sus acciones. Este tipo de tareas trata de ajustar el tiempo de ejecución de una secuencia de acciones que forman un plan, asegurando que cada acción se ejecuta dentro de un periodo de tiempo determinado. La ejecución de la acción fuera de este intervalo no es de ninguna utilidad para el sistema.

En otros trabajos del grupo se propone un esquema de planificación y *"scheduling"* que se realizan simultáneamente, comprobando, mientras se construye el plan, si hay recursos disponibles —o va a haberlos— para realizar cada acción en el intervalo temporal correspondiente [Garrido et al., 2001] [Garrido and Barber, 2001] [Tormos et al., 2002]. Utilizando estos resultados, se propone la creación de una tarea que combine planificación y *"scheduling"* en una única tarea.

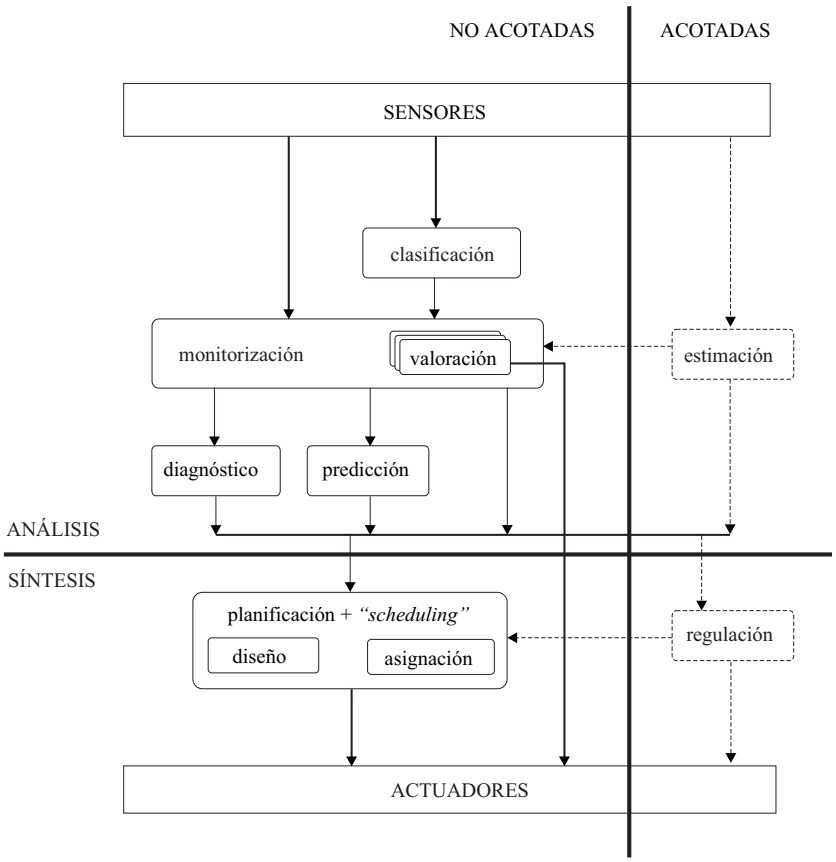


Figura 3.9. Modelo de tareas para ARTIS

Modelo final de tareas

La organización de los distintos tipos de tareas dentro de un agente ARTIS sigue el esquema que muestra la Figura 3.9. Atendiendo a la división en capas, tenemos tareas acotadas (a la derecha de la figura), que corresponden a la capa reactiva del agente, y tareas no acotadas (a la izquierda), asociadas con la capa deliberativa. La capa reactiva, que se ejecuta en primer lugar determina el estado actual del entorno (tarea de estimación) y calcula una primera respuesta simple (tarea de regulación). Si hay tiempo suficiente, la capa deliberativa puede (i) mejorar la precisión en la estimación de la situación actual, o (ii) calcular una respuesta más elaborada o a mayor largo plazo.

Si se atiende al ciclo percepción-cognición-acción, desde los sensores hacia los actuadores, los resultados de las tareas de análisis (en la parte superior de la figura) alimentan la entrada de las tareas de síntesis (en la parte inferior). Los datos que se reciben del exterior a través de los sensores son la entrada de

	estimación	*análisis*
regulación	(1) reflejo	(2) sobreinformado
síntesis	(3) hiperactivo	(4) racional

Figura 3.10. Grados de reactividad de un AA

las tareas de *estimación*, *clasificación* o directamente a la de *monitorización*. Si se invoca a una de las dos primeras, su resultado también es una entrada de la tarea *monitorización*. Ésta puede estar formada por un conjunto de tareas de *valoración*, más simples. El resultado de la monitorización, que consiste en determinar si el sistema se encuentra o no bajo control, se dirige a tareas de (i) *diagnóstico*, para evaluar el fallo de algún componente, o de (ii) *predicción*, para estimar el comportamiento futuro del sistema. El resultado de todas estas tareas conduce a determinar la situación del entorno. A partir de aquí, las tareas de síntesis son las encargadas de calcular las acciones que se deben ejecutar. La tarea más simple es la de *regulación*, que realiza una función de ajuste o *"tunning"*. Si se precisan comportamientos complejos, se emplean tareas de *planificación*, *"scheduling"* o una combinación de ambas, que sólo tiene en cuenta aquellas acciones que se pueden planificar porque hay recursos disponibles. El resultado es la propuesta de un conjunto de acciones a ejecutar sobre el medio.

En función de cómo se combinen los distintos tipos de tareas (de tiempo real e intensivas en conocimiento), aparecen distintos grados de reactividad en el sistema (véase Figura 3.10)que configuran cuatro posibles comportamientos:

1. **reflejo:** ejecutar únicamente tareas de tiempo real: *estimación + regulación*,
2. **sobreinformado:** mejorar la estimación del estado actual mediante tareas de análisis: *análisis + regulación*,
3. **hiperactivo:** mejorar la calidad de la respuesta mediante tareas de síntesis: *estimación + síntesis*, o
4. **racional:** realizar un proceso que se podría denominar *inteligente*, combinando tareas de análisis y de síntesis: *análisis + síntesis*.

3.5. Formalización de un agente ARTIS

Para la definición de agente ARTIS, supondremos que se ha definido previamente un lenguaje lógico de primer orden $\mathcal{L}$, que permite definir símbolos de constante, variables, predicados, funciones, cuantificadores y símbolos de puntuación como se definen en un lenguaje lógico estándar. Todos los términos serán fórmulas bien formadas (f.b.f.) de $\mathcal{L}$. Denotamos con $Form(\mathcal{L})$ el conjunto de todas las f.b.f. que se pueden construir usando elementos de $\mathcal{L}$.

A continuación se definen una serie de términos para simplificar la posterior definición de los conjuntos y funciones que forman las distintas entidades de un AA.

- el conjunto de percepciones del AA, definido como un subconjunto de f.b.f del lenguaje $\mathcal{L}$ que representan datos enlazados directamente al exterior a través de algún sistema de adquisición de datos.

$$\Sigma \subseteq \wp(Form(\mathcal{L}))$$

- el conjunto de creencias del agent e, representado como el subconjunto de f.b.f del lenguaje $\mathcal{L}$

$$Belset \subseteq \wp(Form(\mathcal{L}))$$

- el conjunto f.b.f. que representan de objetivos y restricciones de integridad,

$$Goalset \subseteq \wp(Form(\mathcal{L}))$$

- Un conjunto de reglas (como forma general para la representación del conocimiento de resolución de problemas) se define como un conjunto de pares condición-acción.

$$\begin{array}{rcll} Ruleset & \stackrel{\text{def}}{=} & \wp(ARule) & \text{un cjto de reglas de acción} \\ ARule & \stackrel{\text{def}}{=} & Cond \times Act & \text{define una regla elemental} \\ Cond & \stackrel{\text{def}}{=} & Form(\mathcal{L}) \cup \{true\} & \text{representa una condición} \\ Act & \subseteq & \wp(Form(\mathcal{L})) & \text{es un conjunto de acciones} \end{array}$$

Considerar los requerimientos de tiempo real es de vital importancia en la definición de un AA. Dentro de la arquitectura de agente, la entidad en la que se mantienen estas características son los *in-agent*. Los *in-agent* contienen el conocimiento de resolución de problemas teniendo en cuenta las restricciones temporales que impone el entorno. Se definen de la siguiente forma:

Definición 3.1 (in–agent) *Un* in-agent *es una estructura:*

$$a_i = \langle \Sigma_i, \mathcal{A}_i, \Delta_i, \mathcal{G}_i, r_i, d_i, D_i, T_i \rangle$$

donde:

- $\Sigma_i \in \Sigma$ *es el conjunto de percepciones;*
- $\mathcal{A}_i \in Act$ *es el conjunto de acciones sobre el medio;*
- $\Delta_i \in Belset$ *es el conjunto de creencias;*
- $\mathcal{G}_i \in Goalset$ *es el conjunto de objetivos (metas y restricciones) locales;*
- r_i *es un conjunto de acciones reflejas;*

- d_i *es un conjunto de acciones deliberativas;*
- $D_i \in \mathbb{R}^+$ *es el* deadline*;*
- $T_i \in \mathbb{R}^+$ *es el periodo.*

□

En primer lugar, hacemos explícitas las percepciones del *in-agent*. Si seguimos el convenio seguido en los autómatas situados, denotaremos con $\sigma \in \Sigma_i$ una percepción y con $\bar{\sigma} \in \Sigma_i^*$ una secuencia de percepciones. En principio, es indiferente trabajar con cualquiera de las dos. Por el momento, escogemos la primera para no complicar la formalización con detalles que no aportarían nada a la especificación.

El periodo y el *"deadline"* de un *in-agent* se definen de la misma manera que se hace en los sistemas de tiempo real. El **periodo** determina el tiempo que transcurre entre dos activaciones consecutivas del *in-agent*. El ***"deadline"*** indica el plazo máximo de tiempo de que dispone el *in-agent* para devolver una respuesta una vez que se ha activado.

Se define una *función de revisión de las creencias* del agente de la siguiente forma:

$$brf : \Sigma \times Belset \rightarrow Belset \tag{3.1}$$

Esta función actualiza las creencias del *in-agent* a partir de la información que capta a través de sus sensores, siendo el equivalente a la función *next* de la arquitectura abstracta de agente (véase Ecuación 2.3).

El proceso de resolución de problemas se realiza mediante las acciones que se encuentran almacenadas en r_i y en d_i. Puede tratarse tanto de sistemas basados en reglas como de conocimiento expresado en forma procedural o que utilice cualquier otro mecanismo de resolución de problemas. Distinguiremos dos funciones: *b-solver (problem-bounded solver)*, que representa el proceso de razonamiento reflejo, y *u-solver (unbounded problem-solver)*, que representa el proceso de razonamiento deliberativo7 y mejora, mediante refinamientos o soluciones alternativas, la respuesta proporcionada por la capa refleja si hay tiempo de cómputo disponible.

$$b\text{-}solver : Belset \times Ruleset \times \mathbb{R}^+ \rightarrow Belset \times \mathcal{A} \tag{3.2}$$

calcula una primera respuesta. Esta función modifica las creencias del *in-agent* y propone una primera acción a ejecutar, que será el resultado del *in-agent* si la siguiente capa no concluye su razonamiento y no puede proporcionar una respuesta.

$$u\text{-}solver : Belset \times Ruleset \times \mathcal{A} \times \mathbb{R}^+ \rightarrow Belset \times \mathcal{A} \tag{3.3}$$

implementa el proceso de cognición no acotado del agente. De la misma forma que para las percepciones, se denota con $\alpha \in \mathcal{A}_i$ una acción individual y por $\bar{\alpha} \in \mathcal{A}_i^*$ una secuencia de acciones. Las dos funciones anteriores pueden modificarse de forma que devuelvan una secuencia de acciones. Las funciones

Algoritmo 3.1 Definición de la función *action* para un *in-agent*

```
1: function action_i(Δ_i) : α
2: (Δ_i, α) ← b-solver(Δ_i, r_i, D_i)
3: if hay tiempo disponible suficiente then
4:    (Δ_i, α) ← u-solver(Δ_i, d_i, α, D_i)
5: end if
6: return α
```

b-solver y *u-solver* representan los procesos de inferencia de cada capa del *in-agent*.

La función de decisión $action_i$ de cada *in-agent*, que define su ciclo de ejecución, es de la forma

$$action_i : Belset \rightarrow \mathcal{A}_i \tag{3.4}$$

que se define a partir del pseudocódigo disponible en el Algoritmo 3.1:

Dependiendo del tipo de acciones que realice un *in-agent*, se puede distinguir entre:

- *in-agent reflejos*: son aquellos en los que $d_i = \varnothing$;
- *in-agent deliberativos*: son aquellos en los que $r_i = \varnothing$;
- *in-agent completos*: son aquellos en los que $r_i \neq \varnothing$ y $d_i \neq \varnothing$;

Definición 3.2 *Decimos que dos* in-agent a_i *y* a_j *son* diferentes *sii*

$$r_i \neq r_j \vee d_i \neq d_j \vee D_i \neq D_j \vee T_i \neq T_j$$

□

Denotaremos con $InAg = \{a_i | i \in \mathbb{N}\}$ el conjunto de todos los *in-agent* de un agente ARTIS.

Los *in-agent* se agrupan en comportamientos que responden a escenarios concretos en los que se puede encontrar el agente. Cada comportamiento contiene los *in-agent* que se encuentran activos, permaneciendo el resto inactivos. Un cambio de comportamiento se produce por dos causas: cambios en el estado de un *in-agent* y variación de las restricciones temporales (*deadline* y periodo) que caracterizan al *in-agent*. Un *comportamiento* se define de la siguiente manera:

Definición 3.3 (comportamiento) *Un* comportamiento *es un conjunto no vacío de* in-agent*:*

$$beh \in \wp(InAg) \ - \{\varnothing\}$$

□

Basta con que uno de los *in-agent* que lo forman sea distinto para que dos comportamientos sean diferentes. Por la Definición 3.2, esto puede producirse porque cambien únicamente las restricciones temporales de un agente. Así, dos comportamientos pueden ser diferentes porque el periodo o el *deadline* de alguno de los *in-agent* que lo forman varíe.

Definición 3.4 *Dos comportamientos beh y beh′ se considera que son* diferentes *cuando se cumple:*

$$\exists a_i \in InAg | a_i \in beh \wedge a_i \notin beh'$$

□

Por último, se define un agente ARTIS de la siguiente forma:

Definición 3.5 (Agente ARTIS) *Un agente* ARTIS *es una estructura:*

$$AA = \langle \mathcal{B}, \mathcal{G}, \Delta \rangle$$

donde:

- $\mathcal{B} \subseteq \wp(InAg) - \{\varnothing\}$ *es un conjunto finito de comportamientos;*
- $\mathcal{G} \in Goalset$ *es el conjunto de objetivos del agente en el instante actual*
- $\Delta \in Belset$ *es el conjunto de creencias en el instante actual*

$$\Delta = \bigcup_{\forall a_i \in InAg} \Delta_i$$

□

Se define una función τ, que determina el comportamiento activo del agente, de la siguiente forma:

$$\tau : \mathcal{B} \times Belset \rightarrow \mathcal{B} \tag{3.5}$$

Dicha función calcula cual es el comportamiento activo del AA a partir del comportamiento actual y de las creencias actuales del agente. Un agente ARTIS nunca podrá tener activo más de un comportamiento simultáneamente.

Propiedad 3.6 *Sea $\mathcal{B}$ el conjunto de comportamientos de un agente* ARTIS. *Se verifica que:*

$$\forall beh_i, beh_j \in \mathcal{B}, beh_i \neq beh_j$$

□

El cumplimiento de los objetivos de diseño de un agente ARTIS se puede garantizar a partir del modelo de agente.

Definición 3.7 *Sea Δ el conjunto de creencias del agente, g un objetivo y r_i, d_i un conjunto de reglas de deducción. Entonces $\Delta \overset{r_i,d_i}{\vdash} g$ sii hay una prueba de g desde Δ usando únicamente reglas de r_i o de d_i.* □

Es decir, que se puede cumplir un objetivo si $\exists a_i \in InAg$ tal que, aplicando sus reglas de deducción, se consigue el objetivo marcado a partir de las creencias del agente.

3.6. Seguridad, viveza y equidad

Un tema ampliamente considerado en la verificación formal de programas, habitualmente relacionado con la concurrencia [Lamport, 1977], es la demostración de las propiedades de seguridad (*"safety"*) y viveza (*"liveness"*) [Alpern and Schneider, 1986] [Alpern and Schneider, 1985] [Abadi et al., 1991]. Descrito informalmente, las propiedades de *seguridad* evitan que "algo malo" ocurra durante la ejecución de un programa. Las propiedades de *viveza* expresan que, eventualmente, puede (debe) ocurrir "algo bueno". Inicialmente, se trataba de formalizar propiedades como la ausencia de interbloqueos, garantía de exclusión mutua, funcionamiento FIFO, terminación, garantía de servicio, etc.

Es un tema que ha resultado de interés también en el área de los sistemas de tiempo real. En ellos, es especialmente crítico el correcto funcionamiento del programa bajo cualquier circunstancia. Y sin esta garantía no se puede confiar en que el programa que controla el sistema vaya a mantener a éste siempre en las condiciones adecuadas.

Este tema ha vuelto a cobrar una especial importancia en el campo de los agentes inteligentes. Uno de los principales problemas que los desarrolladores se encuentran a la hora de construir sistemas basados o bien en agentes individuales o bien en sistemas multiagente, es el de la imposibilidad de comprobar experimentalmente la validez de los programas. Y si, además, consideramos la capacidad de aprendizaje de ciertos agentes, queda prácticamente anulada cualquier posibilidad de verificación. Las situaciones en las que se va a encontrar el agente son potencialemente infinitas. Además, el comportamiento puede depender no sólo de la situación, el escenario en el que se encuentre, sino también de la secuencia de situaciones previas por los que ha ido pasando. Posiblemente, la respuesta ante una situación determinada dependerá de sus resultados en situaciones previas parecidas, de la experiencia que ha ido adquiriendo al resolver problemas semejantes.

Construir una batería de pruebas en laboratorio, que luego nos asegure la corrección en la actuación del agente, no es algo que parezca factible. Por este motivo, la verificación formal de ciertas propiedades es necesaria. En el caso de un agente ARTIS, el análisis de la planificabilidad nos asegura que todas las tareas críticas tienen garantizado un tiempo de ejecución. El estudio formal de las propiedades de seguridad y viveza nos permite asegurar que la ejecución de la tarea garantiza los objetivos de diseño del agente.

3.6.1. Definición de las propiedades

En todos estos modelos, la ejecución de un *programa* se define como una secuencia infinita de estados

$$\sigma = s_0, s_1, \ldots$$

Una *propiedad* P es un conjunto de ejecuciones. Y puesto que un programa también define un conjunto de secuencias de estados, se dice que un programa *verifica* una propiedad si la secuencia de estados que define el programa está contenida en la propiedad.

Sean S un conjunto de estados, S^ω el conjunto de secuencias infinitas de estados de S y S^* el conjunto se secuencias finitas de estados de S. Los elementos de S^ω son *ejecuciones* del programa y a los elementos de S^* se los denomina *ejecuciones parciales*. Se denota mediante σ_i una ejecución parcial formada por los i primeros estados de σ. Para denotar una secuencia de estados de i a j usaremos σ_{ij}. Por extensión, $\sigma_{0i} = \sigma_i$ y denotamos con $\sigma_{i\infty}$ la secuencia infinita a partir del estado i.

Seguridad

Definición 3.8 (Seguridad [Alpern and Schneider, 1986]) *Un propiedad P es una propiedad de* seguridad *sii $\forall \sigma \in S^\omega$ se satisface la siguiente condición:*

$$\forall i \geq 0 : (\exists \beta : (\beta \in S^\omega : \sigma_i \beta \models P))$$

□

Si P es una propiedad de seguridad y σ es una secuencia tal que cualquiera de sus prefijos σ_i pueden extenderse a una secuencia de P, entonces $\sigma \in P$.

La definición proporcionada por Lamport [1984] es ligeramente distinta: una propiedad P es de seguridad si y solo si cualquier prefijo de una secuencia $\sigma \in P$ puede extenderse repitiendo hasta el infinito el último estado y la secuencia resultante sigue perteneciendo a P. A este fenómeno de repetir un número arbitrario de veces un estado de la secuencia se de denomina *"stuttering"* .

Alpern [1986] introduce la propiedad de *"stuttering"* en general, según la cual se puede repetir un mismo estado un número finito de veces sin que ello afecte a la propiedad. Muestra que una propiedad cerrada bajo *"stuttering"* cumple la definición de Lamport si y solo si es una propiedad de seguridad. Sistla [1994] se refiere a estas propiedades como propiedades de *seguridad con "stuttering"*, y la utiliza para definir las propiedades de *seguridad fuerte*.

Definición 3.9 (Seguridad fuerte [Sistla, 1994]) *Una propiedad P es una propiedad de* seguridad fuerte *si:*

1. *es una propiedad de seguridad con* "stuttering"*; y*
2. $\forall \sigma = s_0, s_1, \ldots \in P,\ \forall i \geq 0 :$ *la secuencia* $s_0, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots \in P$.

□

Es decir, que podemos eliminar cualquier estado intermedio de la secuencia y ésta sigue cumpliendo la propiedad. Esta condición se debe a que, si se deja de observar el sistema en ciertos instantes de tiempo, el comportamiento

observado debe seguir satisfaciendo la propiedad. Un subconjunto importante de las propiedades de seguridad fuerte son los *invariantes*, que se definen como aquellas fórmulas de la forma $\Box f$, donde f es una fórmula proposicional.

Las propiedades de seguridad fuerte son especialmente importantes en los sistemas de tiempo real, y por extensión para ARTIS, en donde sólo se observa el entorno en ciertos instantes de tiempo. Estas propiedades nos garantizan que se siguen cumpliendo incluso en los periodos entre dos observaciones consecutivas.

Viveza

La idea de las propiedades de viveza es que ninguna ejecución parcial es irremediable. Es decir, que siempre resulta posible que se verifique la propiedad de viveza en un futuro.

Definición 3.10 (Viveza [Alpern and Schneider, 1986]) *Un propiedad P es una propiedad de* viveza *sii $\forall \alpha \in S^*$ se satisface la siguiente condición:*

$$(\exists \beta : (\beta \in S^\omega : \sigma\beta \models P))$$

□

Esta definición es muy general, por lo que Alpern y Schneider [1986] proponen algunas definiciones más restrictivas.

Definición 3.11 (Viveza uniforme [Alpern and Schneider, 1986]) *Una propiedad P es de viveza uniforme sii*

$$(\exists \beta : \beta \in S^\omega : (\forall \sigma : \sigma \in S^* : \sigma\beta \models P))P$$

□

es decir, que siempre hay al menos una ejecución β que se puede añadir a todas las ejecuciones parciales σ.

Definición 3.12 (Viveza absoluta [Alpern and Schneider, 1986]) *Una propiedad P es de* viveza absoluta *sii*

$$(\exists \gamma : \gamma \in S^\omega : \gamma \models P) \wedge (\forall \beta : \beta \in S^\omega : \beta \models P \rightarrow (\forall \sigma : \sigma \in S^* : \sigma\beta \models P))$$

□

es decir, P es una propiedad de viveza absoluta si no es vacía y cualquier ejecución de P puede añadirse a cualquier ejecución parcial σ y la secuencia obtenida sigue estando en P.

Una definición alternativa para la propiedad de viveza absoluta es la dada por Sistla, que utiliza PLTL para la definición de las propiedades:

Teorema 3.13 (Viveza absoluta [Sistla, 1994]) *Una fórmula f en* PLTL *expresa una propiedad de viveza absoluta sii f es satisfacible y es equivalente a $\Diamond f$:*

$$\vdash f \leftrightarrow \Diamond f$$

□

Equidad

Además de las propiedades de seguridad y viveza, existe una propiedad adicional que ha sido también objeto de estudio y que todavía no se ha resuelto de forma satisfactoria: la *equidad ("fairness"*, también referenciada en ocasiones como *"justice")*. Se trata de un nombre genérico que abarca una gran variedad de conceptos [Francez, 1986] en función del contexto en el que se emplea. Aunque su mayor utilización se realiza en la especificación de sistemas concurrentes. Se puede definir de manera informal como una propiedad que garantiza que, si una tarea se activa continuamente, no se postpone de forma indefinida. Implica la ocurrencia eventual de la tarea.

Para Francez constituyen una clase de las propiedades de viveza. Sistla las considera como aquellas propiedades de viveza absoluta que además son estables.

Definición 3.14 (Estabilidad [Sistla, 1994]) *Una propiedad P es* estable *si*

$$(\exists \gamma : \gamma \in S^\omega : \gamma \models P) \wedge (\forall \sigma \in S^\omega : \sigma \models P \rightarrow (\forall i > 0 : \sigma_{i\infty} \models P))$$

□

Una propiedad P se dice que es *estable* si no es vacía y, para cualquier secuencia de P, todos sus sufijos también están en P; es decir, P es cerrada bajo sufijos. Y demuestra que para que una propiedad de equidad debe cumplir ambas propiedades: ser una propiedad de viveza absoluta y estable.

En la literatura sobre la propiedad de equidad suelen definirse varios niveles. Se basan en que, de forma eventual, los elementos de un programa se pueden habilitar (o activar) al mismo tiempo que otros elementos del mismo tipo. Dependiendo de la frecuencia de esa eventualidad tendremos diferentes niveles. Los más habituales (y cuya validación es implementable) son tres:

Equidad incondicional todos los elementos se seleccionan a menudo de forma infinita.

Equidad débil todos los elementos que se activan continuamente se seleccionan a menudo de forma infinita.

Equidad fuerte todos los elementos que se activan a menudo de forma infinita, se seleccionan a menudo de forma infinita.

Estas propiedades se emplean, sobretodo, para garantizar que en programas concurrentes no va a haber elementos discriminados. Pero en el caso de un sistema de tiempo real se debe sustituir el "a menudo" por un "siempre" si se quiere garantizar que el sistema estará bajo control en cualquier situación. Y, de esta forma, las propiedades de equidad se convierten en propiedades de seguridad.

3.6.2. Estudio de las propiedades en ARTIS

Tal y como se detalla en el Capítulo 4, para la formalización del AA se emplea una lógica temporal. Por este motivo, el estudio de las propiedades de seguridad y viveza para el agente se centra únicamente en este tipo de lógicas.

Sistla utiliza una lógica modal, en concreto *Propositional Linear Temporal Logic* (PLTL) [Emerson, 1990] para demostrar estas propiedades. Su adaptación a la lógica temporal de ARTIS, una lógica ramificada, es algo prácticamente inmediato con la incorporación de los cuantificadores sobre caminos A y E.

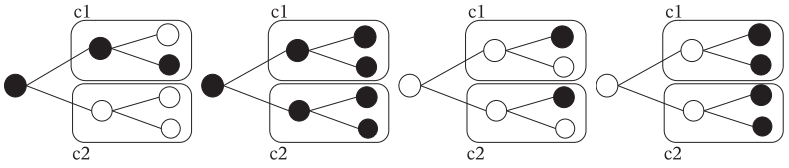
Manolios y Trefler [2001] extienden la caracterización de las propiedades de seguridad y viveza a una lógica temporal ramificada. Distinguen cuatro propiedades: seguridad existencial, seguridad universal, viveza existencial y viveza universal, que surgen de la combinación de los cuantificadores con las correspondientes propiedades.

Clarke, Filkorn y Jha [1993] y Emerson y Sistla [1993] presentan un rango completo de propiedades de corrección en CTL* . Y aunque las propiedades de equidad expresadas en CTL* son muy expresivas, los métodos no son lo suficientemente potentes como para trabajar con la explosión de estados que se produce al evaluar la validez de la fórmula. Una posible solución es la utilización de autómatas, que permiten aprovechar al máximo todas las simetrías existentes en el modelo [Emerson and Sistla, 1997].

Seguridad, viveza y equidad se ha utilizado tradicionalmente para demostrar ciertas propiedades de ejecutabilidad de programas concurrentes. En la arquitectura de AA, la concurrencia se establece a distintos niveles:

- dentro de un comportamiento, los *in-agent* que lo forman se encuentra activos simultáneamente (véase Figura 3.3).
- en la parte deliberativa de los *in-agent*, se planifica el tiempo de ejecución que se le asigna a cada uno para optimizar la calidad de la respuesta [Carrascosa et al., 2003].
- dentro del conocimiento que forma la capa deliberativa, habitualmente en forma de reglas, para la selección de la regla aplicable entre todas las que se han instanciado.

Estas tres características están relacionadas con propiedades de equidad. Pero también se trata de propiedades internas de la arquitectura, que son independientes del conocimiento de resolución del problema concreto. La solución que se ha adoptado es relegar a otros componentes estas funciones.



(a) Seg. exist. (b) Seg. univ. (c) Viv. exist. (d) Viv. univ.

Figura 3.11. Inclusión de un futuro ramificado sobre seguridad y viveza

Así, se realiza un *análisis de la planificabilidad* [García-Fornés et al., 1997] para determinar que todas las tareas cumplen su *"deadline"* y su periodo. En tiempo de ejecución, un *planificador de primer nivel* coordina la ejecución de las capas reflejas de todos los *in-agent* activos. Un *planificador de segundo nivel* [Hernández and Vivancos, 1998] [Botti and Hernández, 1998] se encarga de gestionar la ejecución de las capas deliberativas de los *in-agent* activos con el fin de optimizar la calidad de la respuesta que proporciona el agente. Por último, el motor de inferencia que opera sobre el conocimiento en forma de reglas debe garantizar internamente que no se detiene mientras queden reglas aplicables.

Para los objetivos de la arquitectura de AA, las propiedades de equidad son demasiado relajadas para expresar condiciones sobre el comportamiento de tiempo real del agente, y demasiado estrictas para expresar situaciones deseables para el AA. Nos centraremos en la definición de las propiedades de seguridad y viveza para modelar las restricciones de integridad y los objetivos de un agente ARTIS.

Caracterización de seguridad y viveza en ARTIS

La inclusión de un futuro ramificado requiere reinterpretar el significado de las restricciones. La integración de los cuantificadores A y E sobre caminos requiere la extensión de esta propiedad a árboles de profundidad infinita. Intutivamente, las propiedades de Manolios y Trefler [2001] de seguridad y viveza existenciales y universales generan secuencias de estados como las que muestra la Figura 3.11.

Dentro de ARTIS hablaremos de *restricciones* para hacer referencia a las propiedades de seguridad, y de *objetivos* para las propiedades de viveza. Denotaremos con c una restricción y g con un objetivo.

Propiedades de seguridad: restricciones

Las caracterización de las propiedades de seguridad siguiendo la lógica temporal es la siguiente [Sistla, 1994]:

- toda fórmula positiva es una propiedad de seguridad.
- toda fórmula positiva que utilice únicamente el operador $\mathcal{U}$ es una propiedad de seguridad con *"stuttering"*.

- toda fórmula positiva que utilice únicamente el operador $\Box$ (o su equivalente $p\,\mathcal{U}\,\mathit{false}$) es una propiedad de seguridad fuerte.

De todas ellas, son las propiedades de seguridad fuerte las que nos aseguran que una determinada propiedad se cumple en todos y cada uno de los estados que conforman la secuencia de una computación. El resto no se consideran una propiedad de seguridad para un sistema de tiempo real estricto. En general, una propiedad de seguridad fuerte tiene dos expresiones, en función de si es necesario que se cumpla en todos los estados de todas las posibles computaciones (véase Figura 3.11(b)) o si basta con que se cumpla al menos en una de las computaciones posibles (véase Figura 3.11(a)).

$$A\Box f \qquad E\Box f \tag{3.6}$$

Otra limitación de la definición de las propiedades de seguridad necesaria por las características de ARTIS es la consideración de la propiedad de *"stuttering"* . Una ejecución de ARTIS no genera una secuencia de estados, sino que éstos se encuentran etiquetados con un intervalo temporal que marca el periodo de tiempo durante el cual se mantiene el estado.

$$(s_0, I_0) \rightarrow (s_1, I_1) \rightarrow (s_2, I_2) \rightarrow \cdots \tag{3.7}$$

En general, podemos considerar que todos los intervalos son de la forma $[t_i^-, t_i^+)$ donde t_i^- es el instante de inicio del estado s_i y $t_i^+ = t_{i+1}^-$ es el instante de inicio del estado s_{i+1}. En estas condiciones, no son válidas las secuencias resultantes de la repetición de cualquier estado cualquier número de veces. Y aunque se repita el estado, la marca de tiempo asociada debe variar.

Sea una secuencia de estado como la que muestra la Expresión 3.7. Para mantener la clausura bajo *"stuttering"* ,

1. un estado s_i sólo puede repetirse dentro del intervalo que tiene asociado; y
2. la repetición de un estado obliga a la división del intervalo I_i en dos intervalos I_i' y I_i'' tales que $I_i = I_i' \cup I_i''$

$$(s_0, I_0) \rightarrow (s_1, I_1') \rightarrow (s_1, I_1'') \rightarrow (s_2, I_2) \rightarrow \cdots \tag{3.8}$$

A esta propiedad de *"stuttering"* que conserva la clausura en la secuencia de estados temporalizada la llamaremos *"stuttering" limitado*. En adelante, se considera que *una restricción es siempre una propiedad de seguridad fuerte con "stuttering" limitado*.

Dadas las fuertes limitaciones temporales de los problemas que se van a resolver con ARTIS, se exige al diseñador que el mantenimiento de una restricción sea responsabilidad de un único *in-agent*. La colaboración entre varios *in-agent* para resolver un problema es algo que queda reservado para la capa deliberativa.

Propiedad 3.15 *Una* restricción de integridad *es una restricción c tal que verifica:*

$$c \in \mathcal{G}, \exists a_i, \Delta \overset{\rho_i}{\vdash} c$$

□

Propiedades de viveza: objetivos

A partir de la Definición 3.6.1 de viveza absoluta, podemos ver que que las únicas propiedades de viveza que tienen sentido para garantizar el cumplimiento de objetivos son las propiedades de viveza absoluta, pues con ellas podemos asegurar que una determinada fórmula f será cierta en algún momento.

De la misma forma que se ha procedido con las restricciones, esta definición se extiende para una lógica ramificada incorporando los cuantificadores sobre caminos. de esta forma, si exigimos que el objetivo se alcance alguna vez cualquiera que sea la computación (véase Figura 3.11(d)), es decir, que el cumplimiento del objetivo sea algo inexorable, o si basta con que se cumpla en alguna de las computaciones (véase Figura 3.11(c)).

$$A\Diamond f \qquad E\Diamond f \tag{3.9}$$

Para poder garantizar que un AA puede cumplir sus objetivos de diseño, el conocimiento de resolución de problemas que se integra en los *in-agent* deben verificar que:

1. exista algún *in-agent* que consigue el objetivo, y
2. el periodo del *in-agent* sea tal que permita la detección a tiempo de la situación y el cumplimiento del objetivo.

Propiedad 3.16 *Los objetivos definidos para un* in-agent *deben cumplir:*

$$\forall g^{[a,b]} \in \mathcal{G}_i, \exists a_i, (\Delta \overset{r_i}{\vdash} g \vee \Delta \overset{d_i}{\vdash} g) \wedge T_i \leq a$$

donde $g^{[a,b]}$ denota que el objetivo g debe cumplirse entre los instantes de tiempo a y b. □

Si g es un objetivo de supervivencia (una restricción), un solo in-agent $a_i \in InAg$ debe ser capaz de conseguirlo aplicando únicamente reglas de r_i. El resto de objetivos puede conseguirse empleando indistintamente reglas de r_i y de d_i de cualquier *in-agent* $a_i \in InAg$. Esto queda recogido en la Propiedad 3.15 y en la Propiedad 3.16.

Un agente ARTIS que cumple la Propiedad 3.16 se dice que es un agente *realizable*, en tanto en cuanto su especificación es coherente. Es decir, que su diseño no impide que en algún momento se incumpla alguno de sus objetivos.

Clasificación de restricciones y objetivos

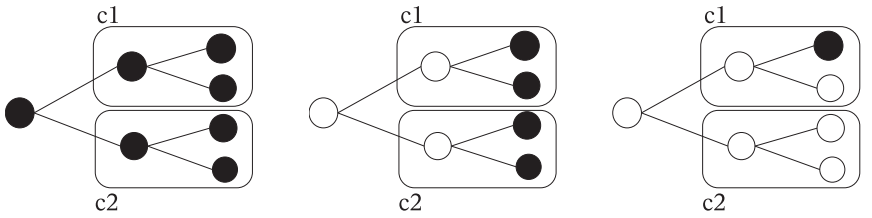
Hasta ahora se ha hablado de la clasificación de los objetivos generales de un AA en restricciones y objetivos propiamente dichos. Por otra parte, en la definición de la arquitectura de AA se ha visto que los objetivos pueden definirse en distintos niveles de la jerarquía de entidades. En cada caso, las restricciones y los objetivos tienen unas propiedades adicionales que afectan a su caracterización, consiguiendo formas más específicas que ayudan al desarrollador del sistema a especificar el comportamiento del agente.

a. *por su tipo:*
   Restricciones. Equivalen a propiedades de seguridad fuertes. Deben verificarse en todos los estados de la ejecución del AA.
   Objetivos. Corresponden a propiedades de viveza absoluta. Se cumplen de forma eventual, o incluso pueden no llegar a cumplirse nunca. Se asume que si se plantea inicialmente un objetivo que no es alcanzable se trata de un error de diseño.
   Esta clasificación es la que se ha expuesto anteriormente como integración en ARTIS de las propiedades de seguridad y viveza.

b. *por su logro:*
   Inevitables. Son aquellas propiedades que se cumplen en todos los caminos posibles.
   Deseables. Son las propiedades que se cumplen al menos en uno de los caminos.
   Las restricciones siempre son inevitables, pues deben garantizarse en todos los estados de una secuencia. Los objetivos pueden ser inevitables o deseables.

c. *por su alcance:*
   Globales. Están definidas sobre el agente completo.
   Locales. Se definen sobre alguna de las entidades de la jerarquía.
   No es obligatorio que sean responsabilidad de uno de los elementos de la jerarquía de entidades en particular. Cualquiera de ellas pueden conseguirse mediante la colaboración de varias entidades de los niveles inferiores. Las restricciones deben ser locales, pues se trata de condiciones esenciales para mantener el sistema bajo control.

d. *por su responsabilidad:*
   Individuales. Un única entidad es la responsable de alcanzar o mantener la propiedad.
   Colectivas. La responsabilidad de su consecución es compartida entre varias entidades.
   Con el nivel de detalle con el que se ha expuesto la arquitectura del AA, *se considera que las entidades más elementales son los* in-agent . Sin embargo, en la arquitectura existen otras entidades que modelan conceptos como



(a) Restricción (b) Obj. inevitable (c) Obj. deseable

Figura 3.12. Restricción y objetivos globales del AA

los refinamientos sucesivos o las soluciones alternativas para la resolución de un problema[García-Fornés, 1996].

Atendiendo a los tres primeros criterios: tipo, logro y alcance, se consiguen diferentes secuencias de estados que garantizan cada una de las propiedades. En primer lugar, los objetivos globales del AA generan secuencias de estados como las que muestra la Figura 3.12.

La caracterización en RTAL de estas propiedades es la siguiente:

- *Restricción global* (Figura 3.12(a)). Se cumple en todos los estados de todos los caminos posibles.

$$A\Box f$$

- *Objetivo global inevitable* (Figura 3.12(b)). Se cumple en algún estado (no tiene porqué ser a la misma distancia) en todos los caminos posibles.

$$A\Diamond f$$

- *Objetivo global deseable* (Figura 3.12(c)). Se cumple en algún estado de alguno de los caminos posibles.

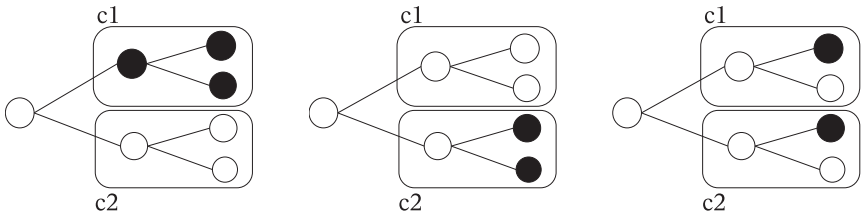
$$E\Diamond f$$

Por otra parte, existen objetivos que deben o pueden cumplirse sólo en alguno de los comportamientos del agente. En ese caso, los objetivos locales dado un comportamiento del AA generan secuencias de estados como las que muestra la Figura 3.13.

La caracterización temporal de las propiedades de un comportamiento, desde una aproximación local, es la misma que para el agente completo. Pero considerando estos objetivos desde el punto de vista del AA, la caracterización en RTAL de estas propiedades es la siguiente:

- *Restricción local* (Figura 3.13(a)). Se cumple en todos los estados de todos los caminos posibles dentro de un comportamiento. Para el AA, se cumple en todos los estados de todos los caminos posibles a partir de un estado determinado, es decir, de forma eventual.

$$E\Diamond A\Box f$$



(a) Restricción (b) Obj. inevitable (c) Obj. deseable

Figura 3.13. Restricción y objetivos de los comportamientos

- *Objetivo local inevitable* (Figura 3.13(b)). Se cumple en algún estado de todos los caminos posibles dentro del comportamiento. Para el AA, se cumple en algún estado de todos los caminos a partir de un estado determinado.

$$E\Diamond A\Diamond f$$

- *Objetivo local deseable*. Se cumple en algún estado de alguno de los caminos posibles del comportamiento. En esta ocasión aparecen dos posibilidades. Si es suficiente con que uno de los comportamientos garanticen la propiedad, entonces es análogo a un objetivo global deseable. Si, por el contrario, se exige que haya al menos una secuencia de estados en *todos los comportamientos* que garantice la propiedad, estamos ante la situación que muestra la Figura 3.13(c). La fórmula que expresa este último caso es:

$$A\Diamond E\Diamond f$$

A todas las propiedades anteriores se les puede exigir que haya una única entidad responsable de mantener el objetivo o que se trate de una tarea compartida entre varias entidades. Como ya se ha comentado, basándose en el grado de abstracción con el que se ha descrito la arquitectura de agente, las entidades elementales en este proceso son los *in-agent*. De esta forma, una *propiedad individual* será responsabilidad de un único *in-agent*, mientras que una *propiedad colectiva* se distribuye entre dos o más *in-agent* para su mantenimiento (si es una restricción) o su consecución (si es un objetivo).

En el caso particular de la arquitectura de AA, destinada a sistemas de tiempo real estricto, se asume que las restricciones, globales o locales, son siempre individuales y además el *in-agent* debe ser capaz de atenderlas mediante la ejecución de su capa refleja (Propiedad 3.15).

Todos los demás objetivos pueden obtenerse indistintamente de forma individual o colectiva, siempre y cuando se garantice la equidad de la arquitectura y no existan unos tiempos estrictos para su cumplimiento. Habitualmente, los objetivos inevitables suelen estar asociados a restricciones de integridad del sistema que no son críticas. Los objetivos deseables son los que marcan el curso de acción que sigue el agente para resolver los problemas. Este tipo de objetivos suele ser dinámicos: se crean objetivos nuevos y se destruyen los

conseguidos a lo largo de la vida del agente. Sin embargo, los anteriores suelen ser objetivos fijos cuya única alteración consiste en la realización de algún ajuste sobre sus valores.

3.7. Conclusiones

En este capítulo se han presentado las aportaciones realizadas a la arquitectura de agente ARTIS para dar soporte a la construcción de agentes inteligentes completos que puedan situarse en entornos de tiempo real estricto.

En primer lugar, se propone un modelo conceptual de agente que integra todos los elementos de la arquitectura. En él aparecen los elementos principales para el desarrollo de la tesis son la consideración de objetivos y restricciones de integridad, la representación de las creencias temporales del agente.

Para facilitar la programación de un agente, se facilita a los desarrolladores un modelo de tipos de tareas que ayudan a especificar el comportamiento del agente para la resolución de problemas intensivos en conocimiento. La propuesta permite distribuir fácilmente los procesos de resolución de problemas por capas: reactiva y deliberativa. También proporciona un esquema para resolver problemas complejos, que involucren varios tipos de tareas.

Dada la dificultad de comprobar empíricamente que el comportamiento de un agente es el deseado, es necesario disponer de un modelo formal que permita especificar por completo el agente. Se ha realizado un esfuerzo importante para integrar en un único modelo todas las modificaciones que se han realizado a la arquitectura desde sus orígenes. La formalización afecta a los distintos elementos que forman la jerarquía de entidades de un agente ARTIS: *in-agent*, que contienen el conocimiento para la resolución de problemas; comportamientos, que definen grupos de *in-agent* activos para responder a las distintas situaciones en las que se encuentra el agente; y el propio agente ARTIS.

Con el objeto de poder validar que los agentes construidos cumplen con sus objetivos de diseño, y para integrar el modelo formal general con el modelo temporal empleado para la representación de las creencias temporales, se ha realizado un análisis de las propiedades de seguridad y viveza y cómo expresar estas propiedades en una lógica modal ramificada como CTL*. Como se expone en el próximo capítulo, esta lógica se utiliza ampliamente en la especificación de agentes inteligentes. Para que una fórmula pueda considerarse una restricción de integridad se le exige que sea responsabilidad de un único *in-agent*. A los objetivos se les pide que al menos exista un *in-agent* que pueda alcanzar dicho objetivo y que los parámetros temporales del *in-agent* (periodo) sean tales que permitan detectar la situación y cumplir el objetivo. Por último, se propone una clasificación de los objetivos según cuatro factores: su tipo (restricciones u objetivos), su logro (inevitable o deseable), su alcance (global o local) y su responsabilidad (individual o colectiva). Esta clasificación ayuda al diseñador a reconocer y formalizar el comportamiento del agente

en cada situación. El objetivo es poder realizar verificaciones "manuales" de ciertas características de especial relevancia para el correcto funcionamiento del sistema, en la línea de las ideas de Emerson sobre sistemas reactivos [Emerson, 1995].

4

Modelo de representación y razonamiento temporal

4.1. Introducción

Los problemas susceptibles de ser resueltos mediante la utilización de la arquitectura de agente ARTIS son aquellos que conciernen a los denominados *procesos de eventos discretos* [Ostroff, 1989]: control de procesos, fabricación flexible, robótica, redes de comunicaciones, gestores de tráfico, aviación, sistemas computacionales de tiempo real, etc. Este tipo de procesos tiene dos componentes principales: *estados*, que se caracterizan por tener una duración en el tiempo, y *eventos*, que ocurren de forma instantánea.

Según Ostroff, las ventajas que aportan los estudios formales para este tipo de sistemas de tiempo real son importantes:

- se descubren ambigüedades, omisiones y contradicciones;
- los modelos formales conducen a métodos de desarrollo de sistemas semi-automáticos (o incluso automáticos);
- su corrección puede verificarse mediante métodos matemáticos;
- su composición para la construcción de sistemas mayores, con una alta confianza de que se comporte siguiendo su especificación;
- permite la comparación de varios diseños.

En el caso de la construcción de agentes inteligentes, la importancia de estas características se acentúa todavía más. Y resulta de especial relevancia ante la imposibilidad de realizar pruebas exhaustivas para comprobar de forma empírica la corrección del modelo en todas las situaciones. Ya se ha visto en el capítulo anterior, al comentar las propiedades de seguridad y viveza de la arquitectura, la imposibilidad de demostrar de forma experimental que un agente va a cumplir sus objetivos en todas las circunstancias.

Una de las razones que produce esta incertidumbre en la corrección del diseño es que, en cualquier sistema situado, es prácticamente imposible que el agente pueda disponer de una visión completa del entorno en el que se encuentra. El agente dispone de una visión reducida, y, en ocasiones, no actualizada, del mismo.

Las *creencias* del agente ARTIS son esa visión parcial de su entorno. Incluyen tanto la información que el agente es capaz de percibir a través de sus sensores como los datos que estima a partir de un proceso de inferencia. Y para poder representar de la forma lo más exacta y natural posible el mundo exterior, debe mantenerse la referencia al instante de tiempo en el que éstos se produjeron.

La consideración de la información temporal en sistemas generales de resolución en IA introduce una problemática adicional, que debe ser planteada y resuelta de forma lo suficientemente expresiva y eficiente [Shoham, 1988]. Podemos definir un sistema de razonamiento temporal como aquel que está formado por una base de conocimiento temporal (las creencias del agente), un procedimiento que permita comprobar su consistencia y un mecanismo de inferencia que permita derivar nueva información y obtener una o todas las soluciones para las consultas que se planteen.

Para representar la evolución que ha seguido el entorno a lo largo del tiempo y los posibles estados futuros que se pueden alcanzar a partir de la situación actual, necesitamos distinguir el estado temporal de los datos. Así, el agente debe ser capaz de mantener información pasada, actual y futura y razonar sobre ella.

Para una adecuada aplicación del modelo temporal, con todas las consecuencias que conlleva, la representación escogida para la información temporal debe permitirnos [Allen, 1983]:

Imprecisión. En un entorno de tiempo real no se puede predecir con exactitud el instante en el que van a ocurrir los eventos. Únicamente se puede trabajar con estimaciones. La representación tiene que permitirnos esta imprecisión y debe ser capaz de razonar con ella.

Incertidumbre. Las creencias del agente no siempre son correctas. La visión parcial del entorno y la inclusión de información futura, y por lo tanto no verificada, en los procesos de razonamiento son los principales responsables de que la información que se maneja no tenga una garantía absoluta de ser cierta.

Modificar la granularidad. La granularidad del sistema es el intervalo mínimo de tiempo con el que el agente es capaz de trabajar. Establece la precisión con la que se mide el tiempo, se razona y se percibe y se actúa sobre el entorno. No se puede medir ningún evento que se produzca con una frecuencia inferior la precisión empleada. La representación escogida debe permitir modificar la granularidad y mantener datos que trabajen con distinta precisión.

Soportar persistencia. El término persistencia hace referencia al hecho de que la información permanece inmutable hasta que se dice explícitamente lo contrario. Es uno de los mecanismos necesarios para garantizar una rápida actualización de la base de datos.

La forma de garantizar estas características en un agentes ARTIS es la siguiente. La *imprecisión* se representa mediante la inclusión de ventanas de

validez temporal para las entidades temporales, en lugar de usar fechas precisas. La *incertidumbre* se consigue mediante la incorporación de información futura: predicciones sobre el estado del sistema, que pueden cumplirse o no. El modelo temporal que se utilice será un modelo de futuro ramificado. La consideración del tiempo como un conjunto denso (valores reales) facilita, entre otras cosas, utilizar distintas *granularidades* para representar hechos temporales. Por último, la no monotonía inherente a la lógica modal proporciona soporte a la *persistencia* de la información temporal.

En este capítulo se analizan las deficiencias del modelo temporal inicial de ARTIS para la construcción de agentes inteligentes de tiempo real capaces de resolver problemas de análisis y de síntesis. Se exponen los formalismos más relevantes para el tratamiento explícito del tiempo en las áreas de influencia del presente proyecto: los agentes y los sistemas de tiempo real. Se define el modelo temporal a través de (i) la lógica temporal propuesta para ARTIS, (ii) una ontología y una teoría del tiempo, que define las primitivas y las entidades temporales), y (iii) la representación de restricciones temporales a través de un mapa de tiempos.

4.2. Limitaciones del modelo temporal actual de ARTIS

El modelo de representación y razonamiento temporal que la arquitectura de agente ARTIS ha incorporado hasta la fecha está basado en trabajos previos del grupo de investigación [Thomson(Fr.) et al., 1990] [Thomson(Fr.) et al., 1993] [Crespo et al., 1994] [Barber et al., 1994] [Botti et al., 1995] [Onaindía, 1997] [Onaindía and Rebollo, 1998]. La adaptación a un modelo de agentes y la ampliación del tipo de problemas a los que se aplica la arquitectura hacen que resulte necesario una revisión del modelo para ajustarlo a los nuevos requerimientos.

Tipos de problemas

La arquitectura inicial de ARTIS [García-Fornés, 1996] es una arquitectura para sistemas basados en el conocimiento de tiempo real. Como tal, estaba pensada para problemas de análisis, en los que el conocimiento se mantiene en una pizarra temporal y en conjuntos de reglas que contenían en conocimiento de resolución de problemas. La ampliación de la arquitectura para tratar también problemas de síntesis requiere la consideración de *acciones* dentro del proceso de razonamiento, que permite realizar procesos de planificación y *"scheduling"* en la parte deliberativa del agente. No todo el conocimiento tiene porqué expresarse en forma de reglas, aunque sigue siendo lo común para la parte deliberativa. En la parte refleja, es habitual el uso de código procedural para mantener el conocimiento de resolución de problemas, que debe garantizar las restricciones temporales críticas del agente.

Lógica "reified" *como lógica temporal*

Desde el punto de vista de la lógica temporal utilizada, se escogió una lógica *"reified"* por la representación explícita del tiempo y la expresividad que proporciona [Shoham, 1987][Ma and Knight, 2001]. Inicialmente era la única que permitía cuantificaciones en sus proposiciones, por lo que era la única capaz de expresar *conocimiento temporal general*, como "las causas siempre preceden a los efectos". Sin embargo, la gran aceptación de las lógicas modales para la especificación tanto de agentes inteligentes como de sistemas de tiempo real, junto con la posibilidad de utilizar esta misma lógica para construir un modelo temporal, justifica el cambio en la lógica para dar una mayor continuidad al modelo. De esta manera, se permite emplear el mismo tipo de lógica para definir el agente ARTIS completo; comenzando por la especificación formal del agente al más alto nivel hasta llegar al modelo temporal interno.

Modelo de puntos de tiempo con restricciones métricas

El modelo temporal de ARTIS era un modelo basado en puntos de tiempo con restricciones métricas. La definición de intervalos se realiza a través de dos puntos de tiempo que señalan su instante de inicio y su instante de finalización. Este modelo es el más adecuado para el modelado de relaciones temporales en sistemas de tiempo real. Pero siguiendo las tendencias actuales [Gerevini et al., 1996] [Barber, 2000] [Jonsson et al., 1999] [Krokhin et al., 2001] [Krokhin and Jonsson, 2002], y para permitir la representación de restricciones de forma más natural, se incluyen relaciones cualitativas en el grafo temporal.

Comprobación de la consistencia

Aunque el grafo temporal de ARTIS no es una red de restricciones temporales, sino que sigue el modelo de un mapa de tiempos [Dean and McDermott, 1987], es necesario asegurar la consistencia de la información que se almacena en el grafo. Para ello, se realizan dos pruebas de consistencia, denominadas *prueba de consistencia sintáctica* y *prueba de consistencia semántica*. La primera comprueba que la nueva información introducida en el grafo es posible. La segunda se asegura de que la nueva información no produce inconsistencias con el resto de información del grafo. Bajo estos conceptos, se encuentran los equivalentes a las pruebas de consistencia de *arco* y de *camino* utilizadas, para comprobar la 2-consistencia y 3-consistencia, de las redes de restricciones temporales.

Sin embargo, dadas las características especiales de los grafos temporales que se generan a partir del proceso de resolución de problemas de un agente ARTIS, podemos considerarlo como un subconjunto de las álgebras de puntos convexas [Meiri, 1996], lo que afecta a los procesos para la comprobación de la consistencia.

Algoritmos de razonamiento en tiempo real

Por último, la forma de adecuar los algoritmos de razonamiento temporal a las necesidades de un sistema de tiempo real es implementando los procesos de búsqueda como métodos interrumpibles. Estos son los procesos más costosos y en ellos se basan la mayor parte de las operaciones sobre el grafo temporal.

Los algoritmos implementados con anterioridad al presente trabajo de tesis proponen un proceso que se descompone en tres fases: intervalos, búsqueda directa y nodos comunes, cada una de las cuales proporciona una respuesta más precisa pero con un mayor consumo de tiempo de cómputo. Si no se puede deducir una relación temporal que determine la verdad o falsedad de la consulta realizada, se pasa a las siguientes fases. Pero no se obtiene una nueva respuesta hasta que cada fase finaliza.

Pero los métodos que se proponen en el presente trabajo tienen dos características adicionales:

- los procesos de búsqueda en el grafo pueden interrumpirse en cualquier paso de ejecución y siempre tienen una respuesta intermedia disponible, sin que sea necesario que finalice la búsqueda completa en el grafo; y
- se emplean métodos con aprendizaje, que aprovechan los resultados de búsquedas anteriores para dirigir búsquedas sucesivas en el grafo.

Este última parte se describe con detalle en el Capítulo 5 del presente trabajo de tesis. Los resultados empíricos obtenidos para comprobar el rendimiento de estos algoritmos aparece en el Capítulo 6

4.3. Lógica temporal

La incorporación de información temporal a los procesos de razonamiento lógicos busca formalizar la noción de tiempo: las proposiciones no tienen un valor de verdad universal e inmutable, sino que éste varía a lo largo del tiempo. Las lógicas temporales proporcionan un mecanismo para representar estos hechos, considerando que el dominio de las proposiciones es una mezcla de espacio y tiempo en el que evaluar la veracidad o falsedad de éstas.

Tradicionalmente, las alternativas que se presentan en el campo de la IA para incorporar la representación explícita de información temporal están basados en uno de los siguientes formalismos lógicos:

Argumentos temporales añaden a los argumentos de un predicado información temporal que indica cuándo el predicado se cumple

$$\textit{is-hot}(Valencia, t, t')$$

Lógicas temporales *"reified"* incorporan un predicado adicional que toma como parámetros un predicado de la lógica estándar y el tiempo en el que el predicado es cierto

$$TRUE(\textit{is-hot}(Valencia), t, t')$$

Lógicas temporales modales que incorporan un operador temporal $At()$ para especificar cuándo el predicado es cierto.

$$At(t, t')is\text{-}hot(Valencia)$$

Las lógicas más empleadas para la formalización de agentes inteligentes son las lógicas intencionales y las lógicas modales. En el caso de los sistemas de tiempo real, existen una gran cantidad de formalizaciones. Las lógicas modales son una de ellas, y es la que se escoge por su cercanía a las lógicas empleadas en la formalización de agentes.

Si a esto añadimos que las lógicas modales son uno de los formalismos válidos para incorporar el razonamiento temporal en las lógicas, no cabe duda que resulta la mejor opción para continuar construyendo el modelo de agente ARTIS (AA), esta vez bajo el punto de vista de la construcción de un modelo temporal robusto que se integre fácilmente con la representación de objetivos y restricciones (objetivos de seguridad, viveza y equidad) realizada en el Capítulo 3.

4.3.1. Lógicas temporales para agentes

Las aproximaciones basadas en lógicas provienen de las arquitecturas deliberativas, descritas en la Sección 2.4. En ellas se emplean métodos formales, especialmente basados en lógicas, para especificar y verificar el correcto funcionamiento de este tipo de software.

Los dos formalismos más habituales en este campo son las *lógicas intencionales*, que describen los sistemas según sus creencias, deseos e intenciones, y las *lógicas temporales*, heredadas del razonamiento sobre sistemas reactivos [Wooldridge, 1995].

Si bien inicialmente se encontraban arquitecturas basadas en una de estas lógicas, en la actualidad la gran mayoría de formalizaciones se basan en una combinación de ambas, utilizando una extensión de la lógica modal CTL* [Emerson, 1995] que incluye predicados para manipular las creencias, deseos e intenciones del agente.

Lógicas intencionales

Una de las definiciones más extendidas de agente lo identifica con un sistema intencional, entendiendo por tal aquellas entidades cuyo comportamiento puede predecirse a través de la atribución de creencias, deseos y perspicacia [Dennet, 1987].

Su formalización se basa en la definición de su semántica a través de una interpretación de mundos posibles [Haplern and Moses, 1992]. Si bien presenta ciertas ventajas, como unos fundamentos teóricamente atractivos y bien definidos, tiene como principales inconvenientes el conocido problema de la

"omnisciencia lógica", así como el de ser excesivamente teórica para su aplicación directa a la construcción de agentes [Wooldridge, 1995].

Es una lógica ampliamente utilizada en la especificación de SMA. Su máximo representante es la arquitectura BDI [Georgeff and Rao, 1991], que tuvo que ser replanteada en aras de una mayor cercanía a la implementación final de los agentes [Georgeff and Rao, 1995]. Otras implementaciones conocidas son IRMA [Bratman et al., 1988], Phoenix [Cohen et al., 1989], PRS [Georgeff and Lansky, 1987] o dMARS [d'Inverno et al., 1998].

Tiene sus raíces en el *razonamiento práctico*: decidir en cada instante de tiempo la siguiente acción que se debe realizar para alcanzar un objetivo determinado. Este proceso involucra dos procesos: en primer lugar, decidir qué objetivos son los que el agente debe alcanzar y en segundo lugar determinar cómo podemos alcanzar el objetivo seleccionado.

Su semántica habitualmente se realiza a través de una interpretación de mundos posibles [Kripke, 1963]. Esta semántica tiene ciertas ventajas, con unos fundamentos teóricos bien establecidos [Chellas, 1980]. Sin embargo, su principal desventaja, que tiene una especial importancia en el campo de la IA, es el de la *omnisciencia lógica*, según la cual los agentes deben ser razonadores perfectos.

Lógicas modales

El segundo procedimiento para describir SMA parte de la utilización de Pnueli de las lógicas temporales para razonar sobre sistemas reactivos [Pnueli, 1977], quien fue el primero en reconocer la necesidad de disponer de un formalismo capaz de describir un comportamiento continuo. El concepto de *sistema reactivo* de Pnueli es el de un sistema que mantiene una relación con su entorno y debe describirse siguiendo este comportamiento [Pnueli, 1986].

Por otro lado, desde el punto de vista de la planificación en IA un sistema reactivo es aquel que es capaz de responder dinámicamente a cambios en el entorno. Su ejecución viene dada por una secuencia, potencialmente infinita, de estados, los cuales se suelen organizar en forma de árboles de secuencias. Esta ramificación de las secuencias modela el comportamiento no determinista de los sistemas reactivos.

La lógica modal inicial proporciona operadores temporales básicos, como $\Diamond p$ (alguna vez p) y $\Box p$ (siempre p), que pueden combinarse para expresar propiedades de corrección de interés para los sistemas reactivos.

Las lógicas temporales pueden clasificarse atendiendo a distintas dimensiones [Emerson, 1990]:

- proposicional o de primer orden;
- ramificadas o lineales;
- basadas en puntos de tiempo o en intervalos;
- discreta o continua;
- razonamiento sobre el futuro o razonamiento sobre el pasado y futuro.

La mayoría de sistemas reactivos suelen emplear una lógica proposicional, basada en puntos de tiempo y con razonamiento sobre el futuro [Emerson, 1995]. Para sistemas de tiempo real suelen utilizarse estructuras temporales continuas, basadas en la recta real; por ejemplo, RTCTL . Y para modelar agentes inteligentes es más frecuente el uso de lógicas ramificadas, como CTL* .

4.3.2. Lógicas temporales para sistemas de tiempo real

Se han propuesto una gran cantidad de formalismos que permiten modelar, especificar y probar las propiedades temporales de los sistemas reactivos. La aproximación más cercanas a los formalismos empleados para especificación de agentes inteligentes son las aproximaciones basadas en *lógicas modales*.

Previamente a la definición de las aproximaciones formales, Alur y Henzinger [1991] definen la semántica de un sistema de tiempo real, tal y como se ha visto en la Sección 2.3. También define una clasificación de las semánticas en función de diferentes dimensiones:

- de secuencias de estados o de observaciones;
- puntos de tiempo o intervalos;
- estrictamente monótonas o débilmente monótonas
- tiempo continuo (real) o discreto (entero)

Dada la gran variabilidad existente en la especificación de sistemas de tiempo real, podemos encontrar algún representante de cualquiera de las posibles combinaciones entre las distintas semánticas.

Las aproximaciones basadas en la lógica emplean la lógica temporal, tomando como tal una lógica modal en la que los operadores modales se interpretan asignándoles un significado temporal. Así, se emplea el operador $\Box$ para denotar *siempre* y $\Diamond$ para denotar *alguna vez*.

El inconveniente de esta aproximación es que no permite expresar relaciones cuantitativas, no resultando válida para expresar las restricciones de tiempo real estricto que establecen los *deadline* en el comportamiento del sistema reactivo. Emerson [1990] realiza una revisión de diversos tipos de lógicas temporales que se pueden emplear para especificar sistemas reactivos. Cualquiera de ellas puede extenderse para expresar restricciones temporales.

Se presentan tres posibilidades a la hora de incorporar el tiempo en las f.b.f de la lógica correspondiente [Alur and Henzinger, 1991]:

Operadores acotados temporalmente. Reemplaza los operadores temporales por sus versiones correspondientes acotadas. Por ejemplo $\Diamond_{[2,4]}$ se interpreta como *alguna vez dentro de 2 a 4 unidades de tiempo*. En general, la fórmula $\Box_t\phi$ (ó $\Diamond_t\phi$) se cumple en $t \in \mathbb{R}$ de una secuencia $(\bar{\sigma}, \bar{I})$ de observaciones temporalizadas (véase Ecuación 2.7) sii ϕ se cumple en todos los instantes (o en algún instante) dentro del intervalo $t + I$. La

Ecuación 4.1 se interpreta: *siempre que se observe p, alguna vez antes de 3 unidades de tiempo se cumple q.*

$$\Box(p \rightarrow \Diamond_{\leq 3}\ q) \tag{4.1}$$

Cuantificación inmóvil *(freeze).* Asocia a cada estado el instante de tiempo en el que se cumple mediante un cuantificador inmóvil "$x.$". Una fórmula $x.\phi(t)$ se cumple en el instante t sii $\phi(t)$ también lo hace. Por ejemplo, en la fórmula $\Diamond y.\phi$, la variable temporal y está acotada a aquellos estados en los que ϕ es cierta *alguna vez.* Esta alternativa permite escribir cosas como:

$$\Box x.(p \rightarrow \Diamond y.(q \wedge y \leq x + 3)) \tag{4.2}$$

que se lee: *en todos los estados en el instante x, si se cumple p entonces hay un estado posterior en el instante y en el que se cumple q y además y se encuentra antes de 3 unidades de tiempo de x.* Este tipo de expresiones no se pueden expresar de forma directa con la opción de los operadores acotados temporalmente.

Variable de reloj explícita. Se define una variable de estado dinámica T, que representa el reloj, y la cuantificación de variables temporales. La Ecuación 4.2 se reescribe como:

$$\forall x, \Box((p \wedge T = x) \rightarrow (q \wedge y\ T \leq x + 3)) \tag{4.3}$$

La variable x toma valores en todos aquellos instantes de tiempo en los que se observa p.

Existen algunas extensiones a la lógica RTCTL que permiten representar de manera adecuada relaciones entre intervalos de tiempo, como la *"Parametrized Real-Time Computation Tree Logic"* (PRTCTL) [Emerson and Trefler, 1999]. De esta forma, se puede expresar la relación que aparece en las ecuaciones 4.2 y 4.3 empleando esta lógica:

$$\forall x, \Box(p \rightarrow \Diamond_{\leq x+3}\ q) \tag{4.4}$$

4.3.3. Lógicas temporales de acción

Dada la extensión de tipo de problemas a problemas también de síntesis, es necesario incluir acciones en la lógica, pues son el fundamento de cualquier problema de planificación y de *"scheduling"*.

Nos centraremos en la posibilidad de modelar acciones con lógicas modales. En esta línea, existen algunas aproximaciones que permiten incorporar acciones a una lógica modal, en lo que se denomina en general *lógicas temporales de acción* [Lamport, 1990] [Lamport, 1993].

La mayoría de las aplicaciones utilizan como base una lógica temporal lineal (LTL), no ramificada en el futuro. La lógica LTL se ha utilizado ampliamente en la especificación y verificación de propiedades de sistemas dinámicos, como las de seguridad, viveza y equidad [Emerson, 1995] [Vardi, 1995].

Pero las lógicas lineales no son adecuadas para poder realizar aserciones sobre la validez de las acciones futuras en una situación dada [Calvanese et al., 2002]. Para ello, es necesario disponer de una lógica que utilice un tiempo ramificado.

En el caso de las lógicas temporales ramificadas, existen algunos ejemplos sobre procesos de planificación sobre técnicas de *"model checking"* [Cimatti et al., 1997] [Bertoli et al., 2001]. Existen trabajos para expresar objetivos de varios tipos utilizando CTL [Pistore and Traverso, 2001].

Además, este tipo de lógicas y técnicas de planificación, basadas en *"model checking"*, también se han utilizado en el área de los sistemas multiagente para la planificación distribuida entre los agentes que forman el sistema. Con ellas se puede hacer referencia a objetivos *epistémicos*, es decir, objetivos comunes para un conjunto de agentes [van der Hoek and Wooldridge, 2002]

4.3.4. Lógica temporal del agente ARTIS

La propuesta que se realiza en el presente trabajo es la de emplear una adaptación de la *Computation Tree Logic* (CTL) [Emerson, 1990], definida inicialmente para el modelado de sistemas concurrentes. Es también la base para la especificación de agentes inteligentes. Pero no es suficiente para las necesidades de un sistema de tiempo real, por lo que se adoptarán extensiones a la lógica adecuadas a las necesidades de los agentes ARTIS.

- para la formalización de agentes que siguen el modelo de creencias, deseos e intenciones (BDI)[Georgeff and Rao, 1991] [Georgeff and Rao, 1995] y otras arquitecturas deliberativas de agente, la variante más empleada es la CTL* .
- dada la imposibilidad de especificar instantes precisos de tiempo sobre la lógica inicial, Emerson propone una extensión a CTL que permite incorporar de forma explícita aserciones cuantitativas sobre los operadores temporales, denominada *Real-Time* CTL (RTCTL) [Emerson et al., 1992]. Esta lógica se construye para poder demostrar la corrección de las propiedades de tiempo real de los programas.
- la lógica RTCTL solo permite restricciones temporales de la forma $\Box_{<x}$ o $\Diamond_{<x}$, donde x es un valor entero. Aunque se trata de un requerimiento bastante estricto, garantiza que el proceso de *"model checking"* para estudiar si se cumple una propiedad puede resolverse con un coste temporal de $O(n^2)$. La incorporación de las restricciones de igualdad e inferioridad provoca el aumento del coste de los algoritmos a $O(n^3)$. A esta extensión de RTCTL, que incluye como posibles restricciones $\{>, =, <\}$, se la denomina RTCTL *completa* (CRTCTL). Dadas las características de nuestro grafo temporal, y la similitud con las relaciones cualitativas entre puntos de tiempo, es más adecuado emplear la variante completa de esta lógica para especificar el modelo temporal del AA.

- una carencia importante de RTCTL, como queda reflejado en el Apartado 4.3.2, es la imposibilidad de establecer relaciones entre las variables que determinan las restricciones métricas del modelo. Para mejorar la expresividad y la aplicabilidad de la lógica, Emerson propone una nueva adaptación denominada *Parametrized Real-Time Computation Tree Logic* (PRTCTL) [Emerson and Trefler, 1999].
- por último, se debe incorporar la representación de acciones en la lógica para unificar la especificación y el tratamiento de problemas de análisis y de síntesis en un único modelo. Se emplea el mecanismo usado por Lamport en la definición de TLA$^+$ para sistemas híbridos [Lamport, 1990].

La lógica propuesta para la arquitectura de AA podría definirse como *"Parametric, Qualitative, Complete, Real-Time Computation Tree Action Logic"* (PCRTCTAL*). Por motivos obvios de simplificación, en adelante nos referiremos a ella como RTAL *(Real-Time Agent Logic)*. Su semántica se encuentra en el Apéndice A.

4.4. Ontología y teoría del tiempo

Una *ontología* está formada por las *primitivas* de tiempo y las *relaciones temporales* que existen entre ellas. La teoría del tiempo está formada por un conjunto de axiomas que definen la estructura de estas primitivas (la estructura del tiempo) [Vila, 1995].

Para la selección de la primitiva de tiempo se plantean dos posibilidades: puntos de tiempo o intervalos. Existe una tercera posibilidad desarrollada ampliamente durante esta última década: integrar las dos primitivas en una única ontología de puntos e intervalos. De esta forma, no es necesario construir ninguna representación auxiliar para modelar puntos con intervalos o viceversa. Se pueden utilizar unos u otros indistintamente, en función de lo que sea más adecuado para aquello que se desea representar. Naturalmente, las relaciones temporales varían, existiendo relaciones entre puntos, entre intervalos y entre puntos y relaciones entre intervalos en un mismo modelo. El principal problema de los modelos de intervalos es que se trata de álgebras en las que resolver el problema de la consistencia es NP-duro. Existen trabajos recientes enfocados en identificar las subálgebras tratables y, más en concreto, identificar los conjuntos máximos de subálgebras tratables de cada tipo: de puntos, de intervalos y ambas [Krokhin and Jonsson, 2002], así como de disponer de modelos de gran expresividad que permitan integrar restricciones cualitativas y métricas tanto sobre puntos como sobre intervalos e incluir duraciones [Barber, 2000].

4.4.1. Primitivas temporales

La primitiva temporal es el punto de tiempo, ya que resulta lo más adecuado para los sistemas de tiempo real, sacrificando la expresividad por la eficiencia y tratabilidad de los algoritmos.[1]

En la mayoría de las aproximaciones basadas en puntos de tiempo, el tiempo se ve como una estructura $(\mathcal{P}, \prec)$, donde $\mathcal{P}$ es un conjunto de puntos de tiempo y $\prec$ una relación de orden parcial sobre $\mathcal{P}$. Las propiedades *mínimas* son las de un *conjunto parcialmente ordenado*:

Irreflexiva $\forall p, p' \in \mathcal{P},\ \neg(p \prec p')$
Asimétrica $\forall p, p' \in \mathcal{P},\ p \prec p' \rightarrow \neg(p' \prec p)$
Transitiva $\forall p, p', p'' \in \mathcal{P},\ p \prec p' \wedge p' \prec p'' \rightarrow p \prec p''$

Además, en el caso del modelo temporal del AA, se verifican las siguientes propiedades adicionales:

No-acotado $\forall p \in \mathcal{P}\ \exists p', p'' \in \mathcal{P},\ (p' \prec p \prec p'')$
Ramificado $\forall p, p', p'' \in \mathcal{P},\ p' \prec p \wedge p'' \prec p \rightarrow p' \prec p'' \vee p' = p'' \vee p'' \prec p'$
Denso $\forall p, p' \in \mathcal{P},\ (p \prec p' \rightarrow \exists p''(p \prec p'' \prec p')$

que nos proporcionan el concepto de infinito en el tiempo, de ramificado en el futuro y de continuo en el tiempo respectivamente.

4.4.2. Entidades temporales

Hechos temporales

Se denomina *hecho temporal* al conjunto de valores que toma una determinada variable a lo largo del tiempo. Los hechos temporales se pueden clasificar de diferentes formas, en función de su estado temporal, su accesibilidad y su origen [Onaindía, 1997].

Definición 4.1 (Hecho temporal) *Un* hecho temporal *es una fórmula*

$$At(a, b, c, d)\phi, con\ a \leq b \wedge a < c \leq d$$

donde:

- $a, b, c, d \in \mathbb{R}^+$ *son puntos de tiempo que determinan el instante de inicio y de finalización del hecho temporal*
- $\phi \in \Delta$ *es una fórmula que expresa una creencia del agente sobre un atributo temporal.*

□

[1] En el Apartado 4.5 se trata con más profundidad este tema.

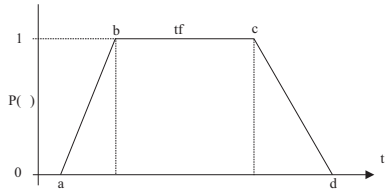


Figura 4.1. Representación gráfica de un hecho temporal

El operador $At()$ proporciona una forma de extender la lógica RTAL para expresar que una fórmula ϕ se verifica en un instante de tiempo determinado [Reichgelt, 1989]. Puesto que en nuestro caso las fórmulas no son instantáneas, sino que tienen un periodo de validez, el operador $At()$ se modifica para que considere cuatro instantes: dos que marcan el instante de inicio y dos que determinan el instante de finalización de la fórmula.

Gráficamente, podemos ver la representación de un hecho temporal *tf* como muestra la Figura 4.1. El intervalo que definen los puntos de tiempo a y b delimitan el instante de inicio del hecho temporal. El intervalo que definen los puntos c y d determinan su instante de finalización. Así, podemos afirmar que ϕ comienza a ser cierto en algún momento entre a y b y deja de serlo en algún momento entre c y d. Y podemos asegurar que entre b y c el hecho temporal es cierto. Nótese que esta representación es muy cercana a la que utilizan aproximaciones difusa de redes de restricciones temporales, definidas por Barro [1994] y empleada en otras propuestas de inclusión de incertidumbre en dichas redes [Godo and Vila, 1995] [Dubois et al., 1996] [Badaloni et al., 2002]. El trabajo de Dechter y Larkin [2001] define un tipo de grafo al que denominan *red de creencias*. Tiene asociada una tabla de probabilidad condicionada que contiene la probabilidad de los predecesores de cada nodo. La información puede ser determinista (con probabilidad 0 ó 1) o probabilista. Esta red se emplea para determinar la probabilidad de que una consulta sea cierta.

Sea *tf* un hecho temporal. Se definen dos funciones, *begin* y *end*, que permiten recuperar el instante de inicio y fin de *tf*, de la siguiente forma:

$$\begin{aligned} begin &: \Delta \rightarrow \mathbb{R} \times \mathbb{R} \\ begin(tf) &\stackrel{\text{def}}{=} \langle a, b \rangle,\ tf = At(a,b,c,d)\phi \end{aligned} \tag{4.5}$$

$$\begin{aligned} end &: \Delta \rightarrow \mathbb{R} \times \mathbb{R} \\ end(tf) &\stackrel{\text{def}}{=} \langle c, d \rangle,\ tf = At(a,b,c,d)\phi \end{aligned} \tag{4.6}$$

Para poder manejar individualmente los límites del intervalo de validez de un punto de tiempo *begin* o *end*, se definen dos funciones *inf* y *sup* que recuperan el límite inferior y el superior respectivamente del intervalo de validez. Sea *tp* un punto de tiempo que corresponde al instante de inicio o de

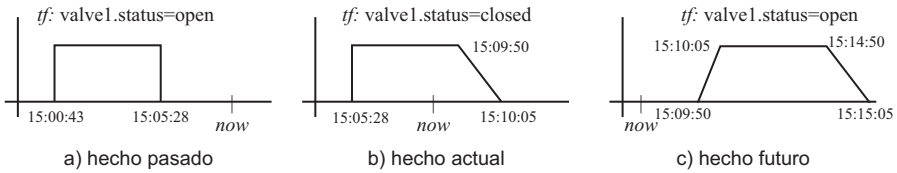


Figura 4.2. Estados temporales posibles para los hechos

finalización de un hecho temporal y cuyo intervalo de validez es $\langle x, y\rangle \in \mathbb{R}\times\mathbb{R}$. Se definen

$$\begin{aligned} inf: &\quad \mathbb{R}\times\mathbb{R}\rightarrow\mathbb{R} \\ inf(tp) &\stackrel{\text{def}}{=} x, tp = \langle x, y\rangle \end{aligned} \tag{4.7}$$

$$\begin{aligned} sup: &\quad \mathbb{R}\times\mathbb{R}\rightarrow\mathbb{R} \\ sup(tp) &\stackrel{\text{def}}{=} y, tp = \langle x, y\rangle \end{aligned} \tag{4.8}$$

Sea Δ el conjunto de creencias del agente en el instante actual.[2] La representación de los hechos temporales atendiendo a su estado, expresados en RTAL , es la siguiente:

- *tf* es un *hecho pasado* si se cumple

$$\Delta \models \blacksquare^{[b,e]}\phi, b = begin(tf), e = end(tf) \tag{4.9}$$

- *tf* es un *hecho actual* si se cumple

$$\Delta \models \blacksquare^{\geq b}\phi \wedge E\Diamond^{<e}\phi, b = begin(tf), e = sup(end(tf)) \tag{4.10}$$

- *tf* es un *hecho futuro* si se cumple

$$\Delta \models E\Diamond^{[b,e]}\phi, b = inf(begin(tf)), e = sup(end(tf)) \tag{4.11}$$

Un hecho temporal sólo puede estar en uno de estos tres estados. La Figura 4.2 representa gráficamente cada uno de estos estados. Como puede observarse, para un hecho pasado sólo son necesarios dos puntos de tiempo, para un hecho actual hacen falta tres puntos y sólo los hechos futuros necesitan los cuatro puntos de tiempo.

Para facilitar la nomenclatura, se pueden usar las siguientes expresiones equivalentes:

$$\begin{aligned} At(a, d)\phi &\equiv At(a, a, d, d)\phi \\ At(a, c, d)\phi &\equiv At(a, a, c, d)\phi \end{aligned} \tag{4.12}$$

para formalizar hechos pasados y actuales respectivamente.

[2] $\blacksquare$ y $\blacklozenge$ son las versiones sobre el pasado de los operadores temporales $\Box$ y $\Diamond$.

Eventos

Una de las características de los agentes es que éstos se hayan situados dentro de un entorno. Y en el caso de los agentes de tiempo real, su comportamiento es especialmente sensible y debe prestar una mayor atención a los cambios que se produzcan en su entorno para poder responder ante ellos.

La representación de la evolución del sistema se realiza a través de las percepciones que provienen del entorno. Estas observaciones provocan cambios en los atributos directamente vinculados con la sensorización del agente, y se modela en ARTIS mediante *eventos*.

Cuando se recibe un evento, el agente actualiza sus creencias con la información que llega del exterior. Este comportamiento es el que refleja la función *bnf* que se define sobre el AA (véase Definición 3.5).

Por otra parte, el propio proceso de razonamiento del AA modifica las creencias del agente, modificando los valores de atributos temporales no observables. Se denomina *evento* a cualquiera de estos cambios instantáneos en las creencias del AA.

Definición 4.2 (Evento) *Un* evento *e para el modelo temporal es un hecho* $tf = At(t)\phi$*, donde* t *es el instante en el que el agente detecta el evento.* □

En este caso, dado que la llegada de un evento es algo conocido e instantáneo, sólo se le asocia un punto de tiempo. Siguiendo el mismo criterio que se ha utilizado en la Expresión 4.12, podemos resumir un evento como

$$At(a)\phi \equiv At(a, a, a, a)\phi \tag{4.13}$$

Atendiendo a la modificación que producen en el estado temporal de los hechos, un evento puede provocar cambios de estado de tres tipos. de actual a pasado, de futuro a actual o la inserción de un nuevo valora actual [Onaindía, 1997]

Acciones

Un problema de la planificación en los STR es que el tiempo que transcurre entre la construcción del plan y su ejecución es relevante para el entorno. En ese periodo, el estado del mundo ha podido variar de tal forma que, cuando se intenta ejecutar una acción del plan, haya ocurrido algún evento que imposibilite la ejecución de la acción.

La consideración del tiempo de razonamiento dentro del propio proceso de razonamiento no es nuevo [Perlis et al., 1991]. El razonamiento consume tiempo. Y es un hecho que no se puede ignorar. Un agente que no es consciente de que el tiempo a medida que razona no será efectivo en muchas situaciones habituales.

Para poder representar este tipo de razonamiento, debemos ser capaces de razonar sobre el intento de realización de acciones [McDermott, 1982][Allen, 1991].

Se define un predicado $Try()$ con el que se especifica la acción que se desea realizar y sus restricciones temporales. De esta manera, un agente planifica una serie de acciones en el futuro, pero antes de ejecutarlas se comprueba que el estado actual del entorno sea el adecuado. Nótese que el comportamiento de las acciones no sigue estrictamente las relaciones causales, pues es necesaria una comprobación adicional antes de aceptar definitivamente las consecuencias: que las condiciones siguen siendo ciertas en el momento de ejecutar la acción.

Por otra parte, desde el área de los agentes inteligentes también se han realizado propuestas semejantes para el tratamiento de las acciones. En el marco propuesto en KARO [van Linder et al., 1998], se habla de la *posibilidad práctica* de ejecución de una acción, como la suma de la capacidad del agente para realizarla y la oportunidad que se le ofrece para hacerlo. Para ello, se define un predicado $do_i(\alpha)$ para indicar que el agente i ejecuta la acción α. De modo que la fórmula $[do_i(\alpha)]\varphi$ denota que φ es el resultado de la ejecución de α por parte del agente i.

En esta linea, se define un predicado para la generación de acciones del agente ARTIS.

Definición 4.3 (Acción) *Una* acción *es una fórmula $Try(a)\phi$, mediante la cual se expresa que el agente intenta realizar la acción a para conseguir que la fórmula ϕ sea cierta.*

$$\begin{aligned} Try: &\quad \Delta \times \mathcal{A} \rightarrow \Delta \\ Try(\alpha)\phi &\stackrel{\text{def}}{=} \Delta \cup A\Box(\alpha \rightarrow E\Diamond\phi) \end{aligned} \tag{4.14}$$

□

La ejecución de esta función implica la creación de un nuevo hecho temporal futuro asociado a la acción. El cumplimiento del hecho temporal provocará la ejecución en el agente de la acción correspondiente, pues garantiza que en el entorno se dan las condiciones adecuadas.

Para la representación de las acciones se plantean varias alternativas, que no son excluyentes. Por una parte, se encuentran las propuestas de las lógicas de acción. Y dentro de estas, por su proximidad a las propuestas del presente trabajo de tesis, las Lógicas Temporales de Acción (TLA—*"Temporal Logic of Actions"*—) [Lamport, 1990] y la la variación TLA$^+$ para sistemas híbridos [Lamport, 1993], acepción con la que el autor denota los sistemas de tiempo real. A esta propuesta se le puede añadir la solución adoptada en KARO para la realización de acciones.

En general, se denota con

$$[a]\phi \tag{4.15}$$

que ϕ es cierto tras la ejecución de la acción a. El uso de corchetes simples indica que la acción *puede* ejecutarse en paralelo con otras acciones, sin que ello afecte a la obtención de ϕ. La expresión

$$[\![a]\!]\phi \tag{4.16}$$

indica que **sólo** se ejecuta la acción a. Puede considerarse una forma de representar el concepto de *exclusión mutua*.

En general, usaremos la siguiente expresión

$$\pi \rightarrow [a]\phi \tag{4.17}$$

para denotar que cuando se cumpla la condición π, al ejecutar la acción a se consigue que ϕ sea cierto.

Añadiendo a la Ecuación 4.17 el predicado para la propuesta de acciones y la inserción de operadores modales para establecer las restricciones temporales también sobre la ejecución de las acciones, un ejemplo de regla que representa una acción sobre el entorno se expresaría de la siguiente forma:

$$A\Box(\pi \rightarrow E\Box^{[a,b]}[Try(\alpha)]\phi) \tag{4.18}$$

que se lee: *siempre que se cumpla π, se verifica que habrá al menos una camino en el que siempre, tras la ejecución de la acción α entre los instantes de tiempo a y b su cumple que ϕ es cierta.*

Desde el punto de vista del modelo temporal, la gestión de acciones es análoga a la que se realiza con los hechos futuros, pero con algunas diferencias en la propagación de las modificaciones que se explican con más detalle en el Capítulo 5. Siguiendo la definición de $Try()$ que aparece en la Definición 4.3, la ejecución de las acciones se postpone para un tiempo futuro.

Al usar la misma estructura de hechos temporales, las acciones quedan temporalmente caracterizadas por cuatro puntos de tiempo que establecen su instante de inicio y de finalización. De esta forma, pueden expresarse duraciones, retardos, condiciones de disparo (que sólo necesitan cumplirse en un instante de tiempo) o condiciones que deben cumplirse durante toda la ejecución de la acción: por ejemplo, que una puerta esté abierta todo el tiempo mientras la estamos cruzando.

4.5. Restricciones temporales

Referir los datos a un tiempo requiere no sólo manejar restricciones cualitativas [van Beek, 1990], sino también gestionar información métrica [Dean and McDermott, 1987]. Las redes de restricciones temporales (TCN—*Temporal Constraint Network*—) [Dechter et al., 1991] son una de las aproximaciones más conocidas para gestionar la información temporal. Sin embargo, comprobar la consistencia en una TCN en un problema NP-duro [Vilain and Kautz, 1986] [Dechter et al., 1991] [Golumbic and Shamir, 1993] e incluso los algoritmos de consistencia local incurren en costes exponenciales. Otras aproximaciones, como los grafos temporales o los gestores de mapas de tiempos [Dean and McDermott, 1987], parecen más adecuados para aplicaciones con gran cantidad de información, donde se producen frecuentes actualizaciones en los datos [Yampratoon and Allen, 1993].

Debido a las características de los dominios de tiempo real, el número de actualizaciones de información es elevado frente al número de consultas. Por ese motivo, la utilización de un grafo temporal resulta más eficiente para la representación interna del tiempo que una representación basada en TCN, pues prima el rendimiento de las operaciones de modificación frente al de las consultas temporales.

Además de la aproximación elegida para la representación interna del tiempo, debe decidirse si se van a utilizar restricciones cualitativas, métricas o ambas entre los puntos de tiempo, que es la primitiva temporal elegida. La tendencia actual es la de incluir ambos tipos de restricciones para definir un único álgebra [Kautz and Ladkin, 1991] [Meiri, 1996]. La propuesta de Kautz y Ladkin y la de Meiri se diferencian en la forma de convertir las restricciones cualitativas en métricas y viceversa. Pero dada la estructura y las necesidades de representación temporal de un AA, parece más adecuada la utilización de la denominada *Álgebra Cualitativa Extendida* de Meiri [1996]. Y dada la equivalencia que puede definirse entre restricciones métricas y cualitativas, podemos seguir utilizando un modelo basado en puntos de tiempo con restricciones métricas, convirtiendo a éstas las restricciones cualitativas cuando sea necesario.

Existen modelos más expresivos, como el propuesto por Barber [2000], que permite incluir duraciones en un modelo métrico de puntos e intervalos. Otra aportación importante de este trabajo es la consideración de varios contextos, en función de los cuales las restricciones temporales pueden variar de un contexto a otro. En ese caso, tanto la consistencia del grafo como los modelos que le corresponden dependerán del contexto en el que nos encontremos. Nótese la similitud de esta aplicación con la consideración de los diferentes escenarios en los que pueden encontrarse el AA, cada uno de los cuales determina el comportamiento activo del agente. La aproximación de Barber permitiría modificar también el conjunto de creencias del AA en cada situación, y no solo los métodos de resolución de problemas.

Por otro lado, una restricción importante en los sistemas de tiempo real es disponer de unos métodos tratables de forma que se pueda garantizar unos tiempos de respuesta aceptables para el peor caso. En este sentido, existen diversos trabajos que estudian la tratabilidad de las distintas álgebras y subálgebras generadas para modelos que definen relaciones

- *punto-a-punto* (PP), entre dos puntos de tiempo,
- *punto-a-intervalo* (PI) o *intervalo-a-punto* (IP), entre un punto de tiempo y un intervalos,
- *intervalo-a-intervalo* (II), entre pares de intervalos.

Si denotamos por $\mathcal{PP}$, $\mathcal{PI}$ y $\mathcal{II}$ los conjuntos formados por todas las relaciones PP, PI e II respectivamente, puede demostrarse que $\mathcal{PP}$ es tratable [Vilain et al., 1989], $\mathcal{PI}$ contiene 5 subclases tratables maximales [Jonsson et al., 1999] y $\mathcal{II}$ contiene 18 subclases tratables maximales [Krokhin et al., 2001]. Con todos estos resultados, Krokhin [2002] determina

todas las subclases tratables maximales para el conjunto $\mathcal{QA} = \mathcal{PP} \cup \mathcal{PI} \cup \mathcal{II}$, formado por puntos e intervalos con relaciones cualitativas entre ellos.

Estos resultados sugieren que el modelo temporal empleado en ARTIS puede extenderse a estas nuevas álgebras, pues es posible su consideración para sistemas de tiempo real.

4.5.1. Representación de restricciones en ARTIS

Como se ha comentado en el Apartado 4.4, la arquitectura de AA emplea un modelo basado en puntos de tiempo como primitiva temporal. El modelo temporal que se emplea es una red métrica de restricciones temporales [Dechter et al., 1991] [Dean and McDermott, 1987] [McDermott, 1982]. Estas relaciones restringen la *distancia* temporal que puede haber entre dos puntos de tiempo.

Las restricciones temporales pueden ser de dos tipos: *unarias*, que restringen el dominio de valores en los que pueden ocurrir los puntos de tiempo, y *binarias* que determinan la relación temporal existente entre dos puntos de tiempo.

Las *restricciones unarias* representan el concepto de ventana temporal del punto de tiempo. Viene dada por un intervalo que delimita el instante de ocurrencia del punto de tiempo.

Definición 4.4 (Restricción unaria) *Sea tp_i un punto de tiempo. Se define una* restricción unaria u_i *como un intervalo*

$$u_i = \langle a, b \rangle$$

de forma que $p_i \in \langle a, b \rangle$.[3] □

Las *restricciones binarias* entre dos puntos de tiempo tp_i y tp_j determinan la distancia mínima y máxima que separa a ambos puntos de tiempo. O dicho de otra forma, determina el conjunto de valores permisibles para la distancia $tp_j - tp_i$.

Definición 4.5 (Restricción binaria) *Sean tp_i, tp_j dos puntos de tiempo. Se define una* restricción temporal c_{ij} *como un intervalo:*

$$c_{ij} = \langle d_{ij}^-, d_{ij}^+ \rangle$$

donde: $d_{ij}^-, d_{ij}^+ \in \mathbb{R}$ representan la distancia mínima y máxima que separa ambos puntos de tiempo, de forma que se verifica:

$$tp_j - tp_i \in \langle d_{ij}^-, d_{ij}^+ \rangle$$

□

[3] Utilizamos los símbolos $\langle$ y $\rangle$ para indicar que los límites del intervalo pueden ser abiertos o cerrados indistintamente.

En ambos casos, las restricciones temporales quedan representadas mediante un intervalo, que en el caso más general puede ser abierto o cerrado en cualquier extremo.

Las operaciones de intersección temporal $\oplus$ y composición temporal $\otimes$ siguen la misma definición de Mairi [1996] y Barber [2000].

Definición 4.6 (Intersección temporal) *Sean c y c' dos restricciones métricas representadas por los intervalos I e I' respectivamente. Su intersección temporal se define como:*

$$c \oplus c' = \{x | x \in I \wedge x \in I'\}$$

□

Definición 4.7 (Composición temporal) *Sean $c_i = \langle d_i^-, d_i^+ \rangle$ y $c_j = \langle d_j^-, d_j^+ \rangle$ dos restricciones métricas. Su intersección temporal se define como:*

$$c_i \otimes c_j = \{z | \exists x \in c_i \wedge \exists y \in c_j : z = x + y\}$$

□

La inclusión de restricciones cualitativas, que se denotan mediante el conjunto $\{<, =, >\}$, se hace de la siguiente forma [Meiri, 1996]. Sea c una restricción cualitativa entre dos puntos de tiempo. La restricción métrica supuesta $QUAN(c)$ se define como:

- si $'<' \in c$ entonces $(0, \infty) \in QUAN(c)$
- si $'=' \in c$ entonces $0 \in QUAN(c)$
- si $'>' \in c$ entonces $(-\infty, 0) \in QUAN(c)$

Las operaciones de intersección y composición con restricciones entre una restricción métrica y una cualitativa tiene como resultado una restricción métrica. Sic es una restricción métrica y c' una restricción cualitativa. La intersección se define:

$$c \oplus c' = c \oplus QUAN(c')$$

y la composición:

$$c \otimes c' = c \otimes QUAN(c')$$

Por ejemplo, sea $c_1 = [2, 3]$ y $c_2 = \{<, =\}$. La intersección $c_1 \oplus c_2 = [2, 3] \oplus [0, \infty) = [2, 3]$, y la composición $c_1 \otimes c_2 = [2, 3] \otimes [0, \infty) = [2, \infty)$.

4.5.2. Red de creencias temporales

El conjunto de relaciones temporales en las creencias del agente se modelan a través de un grafo acíclico, dirigido y ponderado, donde los nodos representan los puntos de tiempo asociados a un hecho temporal y los arcos las restricciones temporales que existen entre ellos. Este grafo temporal constituye lo que denominaremos una *red de creencias temporales* (TBN—*Temporal Belief Network*—).

Debido a las características del proceso que sigue un AA en la generación de sus creencias temporales, que sigue el *principio de causalidad* (Onaindía realiza un examen detallado de la implicación de este principio en un grafo temporal [Onaindía, 1997, p. 44–51]), existen algunas restricciones implícitas en el modelo temporal de AA que afectan a la topología resultante para la TBN.

La representación en la TBN de un hecho temporal se realiza de la siguiente forma. Sea $tf = At(b^-, b^+, e^-, e^+)\phi$ un hecho temporal. Se verifica que $begin(tf) \in \langle b^-, b^+ \rangle$ y que $end(tf) \in \langle e^-, e^+ \rangle$, denotando estos intervalos las restricciones temporales unarias definidas sobre el instante de inicio y de finalización de un hecho temporal. Además, se cumple la siguiente propiedad:

Propiedad 4.8 *El instante de finalización de un hecho temporal siempre debe ser posterior a su instante de inicio.*

$$\forall tf \in \Delta, \exists c = \{<\}\ end(tf) - begin(tf) \in QUAN(c)$$

□

Por otra parte, las relaciones causales que existen en el conocimiento del agente definen relaciones de causa-efecto entre los hechos temporales. Y por extensión, también entre los puntos de tiempo. La existencia de una dependencia causal entre dos puntos de tiempo se representa mediante un nodo especial de intersección. Estos nodos no se corresponden con ningún hecho temporal de las creencias del agente; representan la existencia de una regla de inferencia o la ejecución de una acción.

La TBN se construye de forma incremental, mediante la sucesiva aplicación de reglas de inferencia [Onaindía, 1997] [Onaindía and Rebollo, 1998] [Rebollo and Onaindía, 1999]. La intersección temporal se produce cuando todos los hechos temporales de la LHS son ciertos simultáneamente (véase Figura 4.3). Las restricciones que esto implica son las siguientes:

Sobre el *begin*. Entre cada punto de tiempo *begin* de la LHS y el inicio de la intersección existirá al menos una restricción $r_b = \{<\}$. El último hecho temporal que sea cierto marcará el comienzo de la intersección, luego al menos para uno se cumplirá una restricción $r'_b = \{=\}$. Como a priori no se conoce cuál será el hecho que produzca el disparo de la regla, se incluye la igualdad en todas las restricciones, de forma que todos los puntos de tiempo *begin* de la LHS mantienen una restricción temporal con los puntos de tiempo *begin* de la intersección $r_b = \{<, =\}$.

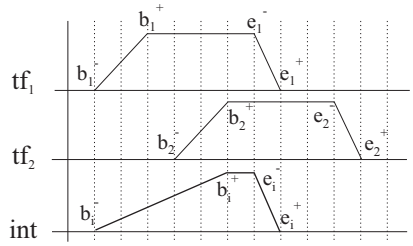


Figura 4.3. Ejemplo de intersección entre dos hechos temporales

Sobre el *end*. Debe garantizarse que ningún hecho temporal de la LHS deje de ser cierto antes de que se hayan cumplido todos al menos en un instante de tiempo. Esto implica que los puntos de tiempo *end* de la LHS deben ser posteriores al *begin* de la intersección, es decir, existe una relación $r_e = \{>\}$ con todos los *end*.

La intersección temporal resultante es un hecho temporal

$$Int = At(b^-, b^+, e^-, e^+)TRUE.$$

Para completar la aplicación de las reglas y la formación de la TBN, se insertan los hechos temporales de la RHS y se añaden las relaciones que establece el usuario (retardos en el comienzo de la conclusión y persistencia de los hechos temporales) entre los puntos de tiempo de la intersección y los nuevos hechos temporales. La ventana temporal de estos hechos se calcula como la composición de la ventana temporal de la intersección (una restricción unaria) con las restricciones temporales adicionales (o se asume $r = \{=\}$ si no hay ninguna).

4.5.3. Comprobación de la consistencia

Meiri [1996] indica una jerarquía en las redes cualitativas en función de las relaciones temporales que contienen y muestra que sólo las álgebras de puntos (PA) y las álgebras de puntos convexas[4] (CPA) son tratables en todos los casos. Posteriormente, propone las *redes cualitativas aumentadas*, en las que incluye restricciones unarias sobre los dominios de las variables (ventanas temporales). La TBN es equivalente al caso de intervalos simples en una red acíclica. Por último, referencia las redes generales incluyendo también restricciones métricas.

El proceso de generación de la TBN, siguiendo el principio de causalidad, impide la presencia de arcos de retroceso, por lo que podemos asegurar que se obtiene un grafo acíclico. Y en esta situación, podemos usar el siguiente teorema para decidir la consistencia de la TBN:

[4] Eliminando la desigualdad de las posibles relaciones.

Teorema 4.9 ([Meiri, 1996]) *Una red CPA acíclica, no vacía y arco-consistente sobre un dominio de intervalos múltiples es consistente.* □

En el caso del AA, un dominio de intervalos simples puede considerarse un caso especial de los intervalos múltiples, por lo que el teorema sigue siendo aplicable.

Una consecuencia inmediata del Teorema 4.9 es que basta con comprobar la consistencia de arco en la TBN para asegurar que la red es consistente. La prueba de consistencia sintáctica [Onaindía, 1997] [Onaindía and Rebollo, 1998] [Rebollo and Onaindía, 1999] es la encargada de comprobar la consistencia de arco.

La *consistencia sintáctica* garantiza que las ventanas temporales de todos los puntos del tiempo del grafo son consistentes. Es decir, que siempre se cumple que su límite inferior es menor o igual al límite superior. Cada vez que se inserta un nuevo punto de tiempo en el grafo, se construye su ventana temporal a partir de las ventanas temporales de sus antecesores y las restricciones temporales establecidas entre ellos. Si la ventana temporal resultante para el nuevo punto de tiempo no es válida, el punto de tiempo no se inserta en el grafo. Y como consecuencia, el hecho temporal asociado no se añade a las creencias del agente por ser inconsistente con alguna de las creencias actuales. En el Capítulo 5 se detalla este proceso.

Definición 4.10 (Consistencia sintáctica) *El grafo temporal es* sintácticamente consistente *sii*

$$\forall tp_i \in \Delta,\ sup(tp_i) - inf(tp_i) \geq 0$$

□

4.5.4. Equivalencia entre RTAL y TBN

En este apartado simplemente se muestra la representación en RTAL de la TBN que modela las creencias del AA. De esta manera, se puede usar el álgebra para razonar sobre el modelo temporal y comprobar sus propiedades mediante técnicas bien conocidas de grafos. Pero también se dispone de un modelo de alto nivel que permite incluir el razonamiento sobre el tiempo en técnicas de *"model checking"*, que es la tendencia actual para la demostración de propiedades en agentes inteligentes deliberativos.

Dado $u^b = begin(tf)$ el punto de tiempo que marca el inicio de un hecho temporal $tf = At(b^-, b^+, e^-, e^+)\phi$. se puede expresar mediante RTAL la ocurrencia de un hecho temporal dentro del intervalo que establece su ventana temporal mediante la expresión:

$$A\Box^{[b^-,b^+]}u^b \equiv A\Box^{\geq b^-}u^b \wedge A\Box^{\leq b^+}u^b \equiv A\Box^{\geq b^-}A\Box^{\leq b^+-b^-}u^b \qquad (4.19)$$

Se puede definir de la misma forma la ocurrencia temporal del instante de finalización.

La definición en RTAL de una restricción temporal entre dos puntos de tiempo viene dada por la siguiente expresión:

$$A\Box(tp_i \rightarrow A\Diamond^{[d_{ij}^-, d_{ij}^+]} tp_j) \tag{4.20}$$

En cuanto a la *temporalidad* de los datos, se asume que el instante de finalización de un hecho temporal siempre será posterior a su instante de inicio (véase Definición 4.1). Esto, que se representa en el grafo mediante una relación $r_{be} = \{<\}$ entre el punto de tiempo *begin* y en *end* del hecho temporal, se expresa mediante:

$$A\Box(begin(tf) \rightarrow A\Diamond^{\geq 1} end(tf)) \tag{4.21}$$

Sean tp_i y tp_j dos puntos de tiempo con ventanas temporales $[u_i^-, u_i^+]$ y $[u_j^-, u_j^+]$ respectivamente. La intersección temporal entre los puntos de tiempo verifica que

$$\begin{array}{l} A\Box^{[u_i^-, u_i^+]} tp_i \wedge A\Box^{[u_j^-, u_j^+]} tp_j \wedge \\ \wedge A\Box(tp_i \rightarrow A\Diamond^{[d_{ik}^-, d_{ik}^+]} tp_k) \wedge \\ \wedge A\Box(tp_j \rightarrow A\Diamond^{[d_{jk}^-, d_{jk}^+]} tp_k) \\ \Rightarrow \\ A\Box^{[u_k^-, u_k^+]} tp_k \end{array}$$

donde:

- $u_k^- = \text{máx}(u_i^- + d_{ik}^-, u_j^- + d_{jk}^-)$
- $u_k^+ = \text{mín}(u_i^+ + d_{ik}^+, u_j^+ + d_{jk}^+)$

No todas las expresiones válidas de la lógica son aceptables para ARTIS. Para incluir acciones en una lógica modal y mantener la causalidad temporal es necesario exigir ciertas limitaciones a las expresiones que pueden construirse. Sea la expresión

$$A((\bigcirc a) \rightarrow f) \tag{4.22}$$

que se lee *"si en el siguiente estado se ejecuta la acción a, entonces el fluente f es cierto ahora"*. Es una regla que rompe el principio de causalidad, pues la consecuencia de la regla (el cumplimiento de f) es temporalmente anterior a sus causas (la ejecución en el próximo estado de la acción a). Se debe restringir el conjunto de fórmulas válidas en la lógica de forma que no permita este tipo de situaciones. Para evitarlo, en el lenguaje de RTAL se impide que la LHS de una regla contenga expresiones con los operadores $\bigcirc$ *(siguiente)* y $\mathcal{U}$ *(hasta)*.

4.5.5. Representación de acciones

Para especificar acciones concretas para el planificador y diferenciarlas de las simples reglas de producción para modelar el conocimiento de resolución de

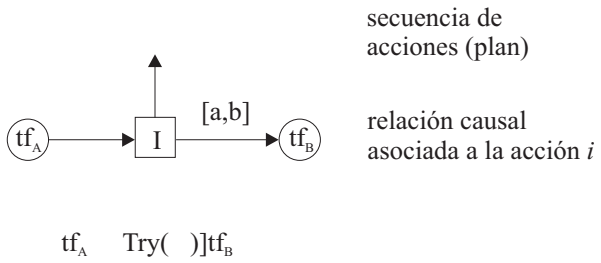


Figura 4.4. Enlace de las acciones con el grafo temporal

problemas, se incorpora un nuevo predicado: $Try()$, en la línea de los trabajos de Allen [1991] y McDermott [1982] . En ambos se considera la planificación desde el punto de vista del razonamiento temporal. La construcción de un plan no presupone la ejecución de las acciones, sino que éstas se proponen. Después, en tiempo de ejecución, habrá que comprobar que realmente se dan las condiciones necesarias para ejecutar la acción. Desde este punto de vista, las acciones son equivalentes a las predicciones sobre el estado futuro del entorno en el modelo temporal de ARTIS.

En la TBN , la representación de las acciones es la misma que se emplea para las predicciones sobre hechos temporales futuros. La diferencia fundamental en la representación de hechos temporales correspondientes a acciones y aquellos que hacen referencia a relaciones causales es que, para los primeros, existe una estructura adicional que mantiene un plan formado por secuencias de acciones. Los hechos temporales se enlazan con las acciones correspondientes a través de los nodos de intersección.

Para incorporar las acciones a la TBN a través de los nodos de la intersección, la fórmula asociada al hecho temporal incluye la ejecución de la acción de la siguiente forma:

$$At(b^-, b^+, e^-, e^+)[Try(\alpha_i)] \tag{4.23}$$

indicando que, si se produce intersección temporal (se dan las condiciones necesarias para la ejecución de la acción), se ejecutará la acción α_i de un plan previamente establecido para lograr sus efectos.

Una acción está representada por dos puntos de tiempo, que corresponden al instante de inicio y de finalización previsto para la acción. Nótese que, implícitamente, la representación de las acciones les asigna una duración: tienen asociado un punto de tiempo para su instante de finalización, que permite la especificación de persistencias. Así mismo, mediante el uso de restricciones temporales explícitas, el usuario puede establecer retrasos desde que se cumplen las condiciones hasta la ejecución real de la acción. De esta forma, ARTIS proporciona al usuario un mecanismo transparente para la planificación temporal.

La forma de distinguir un hecho temporal futuro de una acción planificada es etiquetando los nodos de intersección que establecen la relación causal entre

los puntos de tiempo del grafo. (véase Figura 4.4). El nodo intersección que representa la intersección temporal de las condiciones de la acción mantiene un enlace a la acción correspondiente dentro de un plan, que se habrá generado en una tarea de planificación (ver Sección 3.4)

4.6. Semántica del modelo temporal

La evolución de las creencias de un agente ARTIS es el resultado de tres procesos diferentes:

1. El proceso de razonamiento, que sigue las *relaciones causales* existentes entre las condiciones y los efectos de las reglas en las que se expresa el conocimiento del agente.
2. La *generación de planes*, que puede verse como secuencias de acciones que modifican tanto el estado interno del agente (acciones cognitivas) como su entorno (acciones efectoras).
3. El propio *paso del tiempo*, que se refleja en la propagación que se produce en las ventanas temporales que representan los instantes de inicio y finalización de los hechos temporales.

Se puede asumir, sin pérdida de generalidad, que las relaciones causales en el conocimiento de resolución de problemas de un agente ARTIS vienen dadas por un conjunto de reglas de producción.[5] Estas reglas representan el contenido de la capa refleja y de la capa deliberativa de cada uno de los *in-agent* que componen un AA.

En la parte izquierda de la regla (LHS) puede haber dos tipos de premisas: premisas *de valor* y premisas *de tiempo*. La RHS está formada por una serie de acciones que se ejecutan cuando se cumplen las condiciones de la LHS. La particularidad de disponer de razonamiento temporal es que no basta con que se cumplan todas y cada una de las condiciones, sino que deben cumplirse a la vez. Es decir, que exista al menos un instante de tiempo en el que todas las condiciones de la LHS de la regla se evalúen a cierto.

Las interacciones con el conjunto de creencias del agente pueden plantearse a través de tres tipos de acciones [Burmeister and Sundermeyer, 1991]:

- *acciones cognitivas:* modifican las creencias mediante procesos de inferencia (razonamiento).
- *acciones efectoras:* modifican el entorno a través de los efectores.
- *acciones comunicativas:* envío y recepción de mensajes.

Dentro de los procesos de razonamiento de los *in-agent* distinguiremos solamente dos tipos de acciones: acciones *cognitivas* y acciones *efectoras*.

[5] El código procedural en el que se suelen codificar los comportamientos críticos en el tiempo del AA pueden equipararse a una función.

a) premisas de valor

El predicado $Hold(tf)$ evalúa la certeza de un hecho temporal tf. Este predicado comprueba en el conjunto de creencias del AA si el slot temporal al que hace referencia tiene el valor indicado. Si no se especifica ninguna restricción temporal, se compara con el valor actual del hecho temporal. En otro caso, se evalúan las restricciones y se comprueba si existe algún estado en las creencias del agente en el que se cumpla el hecho temporal en el intervalo temporal resultante de la aplicación de todas las restricciones.

$$\begin{aligned} Hold &: \Delta \rightarrow \mathbb{B} \\ Hold(tf) &\stackrel{\text{def}}{=} E\Diamond(At(b^-, b^+, e^-, e^+)\phi) \end{aligned} \tag{4.24}$$

Una particularidad de trabajar con datos actuales y con predicciones es que el predicado $Holds$ se instancia tanto con datos actuales como con datos futuros. De esta forma, puede avanzarse el proceso de razonamiento y evaluar las condiciones en las que se puede encontrar el entorno en un estado futuro.

b) premisas de tiempo

El predicado $Test(tp_i, tp_j, c_{ij})$ permite realizar consultas acerca de la relación temporal existente entre dos puntos de tiempo o la ocurrencia de un hecho temporal tomando como referencia el instante de tiempo actual.

Sean $tf_i = At(b_i^-, b_i^+, e_i^-, e_i^+)\phi$ y $tf_j = At(b_j^-, b_j^+, e_j^-, e_j^+)\varphi$. c_{ij} es una expresión que relaciona los instantes de inicio o de finalización de tf_i y de tf_j. El test temporal devuelve *TRUE* si encuentra un modelo del agente en el cual existe una relación temporal más restrictiva entre los puntos de tiempo involucrados en la consulta.

$$\begin{aligned} Test: & \quad \Delta \rightarrow \mathbb{B} \\ Test(tp_i, tp_j, c_{ij}) &\stackrel{\text{def}}{=} E\Diamond(At(b_i^-, b_i^+, e_i^-, e_i^+)\phi \\ & \quad \wedge At(b_j^-, b_j^+, e_j^-, e_j^+)\varphi) \\ & \quad \wedge (tp_i = t_i \vee tp_i = t_i') \\ & \quad \wedge (tp_j = t_j \vee tp_j = t_j') \\ & \quad \wedge \Delta \vdash c_{ij} \end{aligned} \tag{4.25}$$

c) acciones cognitivas

Son las acciones que modifican el estado interno del agente, mediante cambios en sus creencias actuales. Encontramos dos tipos de acciones cognitivas, en función de su grado de certeza. Por una parte, tendremos *acciones seguras*, que son aquellas que sabemos con total seguridad que se van a producir y

únicamente esperamos la confirmación del exterior de que el el hecho temporal ha tenido lugar. Para insertar este tipo de acciones se define una función *Know* de la siguiente forma:

$$\begin{aligned} Know: &\quad \Delta \rightarrow \Delta \\ Know(tf) &\stackrel{\text{def}}{=} \Delta \cup A\Diamond At(t,t')\phi \wedge t \geq NOW \end{aligned} \tag{4.26}$$

Se considera una *predicción* aquella información cuya certeza no se puede garantizar. Es una de las características que surgen al trabajar con un futuro ramificado: se mantienen varias alternativas posibles al mismo tiempo, sin determinar a priori cuál es el camino elegido. Para insertar predicciones en el conjunto de creencias de un agente ARTIS se define una función *Believe*:

$$\begin{aligned} Believe: &\quad \Delta \rightarrow \Delta \\ Believe(tf) &\stackrel{\text{def}}{=} \Delta \cup E\Diamond At(t,t')\phi \wedge t > NOW \end{aligned} \tag{4.27}$$

d) acciones efectoras

Se trata de acciones sobre el entorno, que el agente lleva a cabo a través de sus actuadores.

Las acciones sobre el entorno suelen ser el resultado de procesos de planificación complejos que abarcan grandes periodos de tiempo. Por esa razón, es posible que, cuando se intente ejecutar la acción, ya no se den las condiciones adecuadas en el entorno. Por ese motivo, la realización de acciones no va a ser algo seguro. En los apartados 4.4.2 y 4.5.5 se revisa esta cuestión con más detalle.

En el caso de que se trate de acciones inmediatas, resultado de procesos simples (como un ajuste tras una monitorización del entorno), las consideraciones son las mismas que con las acciones cognitivas.

$$\begin{aligned} Try: &\quad \Delta \times \mathcal{A} \rightarrow \Delta \\ Try(\alpha)\phi &\stackrel{\text{def}}{=} \Delta \cup A\Box(\alpha \rightarrow E\Diamond\phi) \end{aligned} \tag{4.28}$$

4.7. Conclusiones

En este capítulo se ha descrito el modelo temporal que define las creencias de un AA en una *red de creencias temporales*. Dicho modelo cumple las características exigidas inicialmente [Allen, 1983]: imprecisión, incertidumbre, granularidad y persistencia.

Tras un breve análisis de las lógicas empleadas para la especificación y verificación de programas en las dos principales áreas de investigación relacionadas: los agentes inteligentes y los sistemas de tiempo real, se ha expuesto

la lógica temporal escogida para formalizar el modelo temporal del agente, denominada RTAL. Esta lógica, basada en CTL*, permite la expresión de restricciones cuantitativas sobre los operadores temporales, relaciones entre las variables que determinan estas restricciones métricas y la consideración de acciones.

El principal motivo de la elección de RTAL como lógica temporal es que permite especificar las propiedades del modelo desde los niveles más altos de la jerarquía de entidades de ARTIS hasta los procesos internos de representación y razonamiento sobre el tiempo. El modelo temporal se ha reescrito empleando expresiones en RTAL para definir las entidades temporales (hechos, eventos y acciones) y las restricciones temporales en la TBN.

El modelo interno del tiempo de un agente ARTIS se mantiene en una red de creencias temporales, con una estructura de grafo acíclico dirigido y ponderado, donde los nodos representan puntos de tiempo asociados a hechos temporales actuales y futuros y los arcos indican las distancias mínima y máxima que puede haber entre cada pareja de puntos de tiempo. La estructura del grafo es equivalente a un red CPA acíclica, no vacía sobre un dominio de intervalos simples, por lo que es suficiente comprobar la consistencia de arco (prueba de consistencia sintáctica) para verificar que la TBN es consistente.

El modelo de representación y razonamiento empleado permite representar de forma sencilla y transparente para el usuario las relaciones causales existentes entre los hechos temporales. El presente trabajo de tesis extiende el modelo anterior de creencias del agente ARTIS de forma que incluye en el mismo grafo la representación de acciones, y no únicamente conocimiento sobre el entorno. Estas acciones tienen la consideración de predicciones futuras, pues no se garantiza su ejecución. Sólo se ejecuta una acción si en el momento de su lanzamiento se siguen cumpliendo todas las condiciones necesarias para su ejecución. La construcción del plan no implica su ejecución: el plan es la propuesta de una secuencia de acciones.

5

Algoritmos de razonamiento temporal

5.1. Introducción

Para adaptar el modelo de pizarra a un entorno de tiempo real, es necesario conocer el tiempo de ejecución (al menos en el peor caso) de todas las tareas que integran el sistema. Existen varios trabajos enfocados a reducir la complejidad del razonamiento temporal, como la aplicación de técnicas heurísticas que permite alcanzar tiempos de ejecución razonables para problemas de gran tamaño [van Beek, 1990] en algunos contextos determinados. Sin embargo, la búsqueda en entornos dinámicos o aplicaciones de tiempo real debe garantizar algunas propiedades como:

- **predecibilidad:** el proceso de búsqueda debe obtener una respuesta dentro de un intervalo de tiempo acotado.
- **interrumpibilidad:** el proceso debe ser capaz de detenerse en cualquier instante de tiempo y devolver una respuesta válida.

La mayor responsabilidad para conseguir un modelo de agente válido para entornos de tiempo real recae sobre los algoritmos de razonamiento. Podemos conseguir una representación adecuada, una lógica muy expresiva, un lenguaje de manipulación intuitivo, ... pero si los procedimientos que se emplean para razonar con los datos no están acotados no se podrá resolver ningún problema que presente requerimientos de tiempo real estricto.

Siguiendo la arquitectura de AA (véase Definición 3.5), podemos distinguir dos tipos de razonamiento muy diferentes, según se realicen en la capa refleja o en la capa deliberativa de los *in-agent* . La primera precisa una respuesta con un tiempo de ejecución conocido y acotado para el peor caso (*wcet —worst case execution time—*). La segunda, si bien no necesita que el tiempo esté acotado, sí que dispone de tiempo limitado para finalizar su razonamiento. Los métodos que se utilicen, o bien están acotados y podemos determinar si pueden finalizar en el tiempo que tienen asignado, o bien se pueden interrumpir en cualquier momento y proporcionan siempre una respuesta.

En la sección 5.2 se examinan las propuestas actuales para realizar búsquedas en grafos en tiempo real. Las alternativas examinadas son las que plantean Korf [1990] e Ishida [1998]. Se caracterizan por cierta capacidad de aprendizaje que se pone de manifiesto en sucesivas soluciones del mismo problema. Se observa que, a medida que aumentan las pruebas realizadas, estos algoritmos convergen a la solución óptima, es decir, al camino mínimo entre dos nodos del grafo. Esta convergencia es interesante para los agentes cuando el entorno se encuentra en equilibrio. Sin embargo, la tendencia a recorrer el espacio de soluciones completo hace que sean desaconsejables para ambientes muy dinámicos.

Las tareas de gestión de la información temporal son fundamentalmente tres:

1. creación de nuevos hechos temporales,
2. actualización de la información existente en la TBN, y
3. recuperación de la información a través de *tests temporales*.

El contenido de la TBN evoluciona mediante la repetida aplicación de estas operaciones. En la sección 5.3 se detallan estas operaciones y los algoritmos empleados.

Todas las operaciones que se realicen sobre la TBN se han implementado como métodos de búsqueda de caminos en grafos. Se basan en consultas temporales sobre la relación existente entre dos puntos de tiempo. El contar con un algoritmo de búsqueda interrumpible, acotado en el tiempo y eficiente (espacial y temporalmente) es un prerrequisito para que el resto de operaciones pueda aplicarse en un entorno de tiempo real. En la sección 5.5 se examina con detalle el algoritmo de búsqueda que se propone.

5.2. Algoritmos para tiempo real

Los algoritmos de búsqueda pueden dividirse en dos tipos: búsquedas fuera de línea, como el algoritmo A* [Hart et al., 1968], y búsquedas en tiempo real,[1] como el algoritmo RTA* [Korf, 1990]. La diferencia entre ambas es que las primeras necesitan determinar por completo el camino hasta el nodo final antes de empezar a recorrerlo, mientras que los algoritmos llamados de tiempo real seleccionan cada paso en tiempo constante y uno a uno, y notifican esa decisión al entorno. De esta forma, permiten intercalar, la planificación con la ejecución de las acciones.

Las búsquedas en tiempo real no garantizan que se encuentre la solución óptima: los algoritmos utilizan óptimos locales en sus procesos de razonamiento que pueden podar la mejor solución. Sin embargo, no es algo crítico para

[1] Aquí, el término *tiempo real* no tiene las mismas connotaciones que las asumidas en ARTIS. Se mantiene el término por respeto a la nomenclatura original usada por los autores.

ARTIS. Al tratarse de un sistema de tiempo real estricto, el tiempo que se tarda en obtener la respuesta es tan importante o más que la respuesta en sí misma. Es mejor una buena respuesta a tiempo que la respuesta óptima demasiado tarde. Y este criterio es algo frecuente en el mundo real.

> *"nature is a satisficer, not an optimizer"* [Davis, 1998].

Si pensamos en un robot, su horizonte de búsqueda queda limitado por el alcance de sus sensores. Con esa información tiene que tomar ciertas decisiones (por ejemplo, qué dirección debe seguir) sin que se pueda garantizar que ésta es la óptima. Su razonamiento se basa en óptimos locales. Incluso si dispone un mapa detallado, no se puede garantizar que el entorno no haya cambiado y siga teniendo libre un camino por el que pasó con anterioridad.

5.2.1. RTA*

Los algoritmos de búsqueda tradicionales utilizan funciones heurísticas que habitualmente provocan exploraciones en anchura, expandiendo todas las alternativas hasta un determinado nivel. Las funciones que se utilizan para calcular el coste de una solución son del tipo $f(n) = g(n) + h(n)$, donde $g(n)$ es el coste acumulado hasta el nodo actual n y $h(n)$ es una estimación del coste hasta el final del camino. Habitualmente, las funciones $h(n)$ que se emplean nunca sobreestiman el coste real del camino restante [Pearl, 1984]. De esta forma, a medida que se profundiza en el árbol de búsqueda, los costes asociados a cada nodo suelen ser mayores que los costes de los nodos que están más cerca del inicio, que son los elegidos para su expansión en detrimento de las soluciones en curso.

El algoritmo RTA* [Korf, 1990] intenta resolver este problema forzando a que sólo se realice una vuelta atrás cuando el camino que se sigue no sea bueno. Se basa en el siguiente principio: sólo se debe volver a un estado ya visitado cuando el coste estimado para resolver el problema desde él más el coste de volver a ese estado sea menor que el coste estimado de continuar desde el estado actual. Para ello, la función $g(n)$ representa la distancia del nodo n partiendo del estado actual en lugar del coste acumulado desde el estado inicial. Además, se actualizan el valor de h para el estado actual con el valor de la segunda mejor $f(n)$ (para todo n sucesor del estado actual).

El algoritmo RTA* es el siguiente:

[RTA]*

1. Exploración:
   Calcular $f(n') = k(n, n') + h(n')$ para cada sucesor n' del estado actual n, donde $h(n')$ es la estimación del coste desde n' hasta el nodo meta y $k(n, n')$ es el coste del arco ente n y n'.

2. Mantenimiento de la consistencia:
   Actualizar la función heurística del estado n de la siguiente forma:

$$h(n) \leftarrow \text{secondmin}_{(n')} f(n') \tag{5.1}$$

3. Selección:
   Escoger aquel sucesor n' con el menor valor para $f(n')$. Los empates se resuelven aleatoriamente.

5.2.2. LRTA*

El algoritmo RTA*, tal y como lo describió Korf, es válido para ejecuciones simples. Sin embargo, en muchas ocasiones necesitamos resolver varias instancias de un mismo problema (frecuentemente para intentar alcanzar el mismo resultado partiendo de estados iniciales diferentes). En cada ejecución se pierden los cálculos realizados para los distintos estados y deben repetirse en distintas ejecuciones del algoritmo. Una mejora importante consiste en poder utilizar los valores almacenados en ejecuciones anteriores y emplearlos en el nuevo problema. Nótese que es un caso frecuente en el mundo de los agentes, en los que se deben enfrentar una y otra vez a lo largo de su vida a los mismos problemas (otra vez debido a la continuidad temporal de los agentes).

Es necesario realizar algunas modificaciones al algoritmo para que se comporte de esta manera. No es suficiente con guardar los valores de $h(n)$ actualizados, pues al almacenar el segundo mejor valor puede hacer que, en algunos casos, se sobreestimen las distancias al objetivo . Se soluciona actualizando el valor de $h(n)$ con el de la mejor solución, y no con el segundo. El algoritmo así modificado se conoce como *Learning*-RTA* o LRTA*.

[LRTA]*

1. Exploración:
   Calcular $f(n') = k(n, n') + h(n')$ para cada sucesor n' del estado actual n, donde $h(n')$ es la estimación del coste desde n' hasta el nodo meta y $k(n, n')$ es el coste del arco ente n y n'.
2. Mantenimiento de la consistencia:
   Actualizar la función heurística del estado n de la siguiente forma:

$$h(n) \leftarrow \min_{(n')} f(n') \tag{5.2}$$

3. Selección:
   Escoger aquel sucesor n' con el menor valor para $f(n')$. Los empates se resuelven aleatoriamente.

LRTA* mantiene una tabla que contiene los valores de la función heurística $h(n)$ que estima el coste de cada estado en el espacio de soluciones. Inicialmente, la tabla tiene la evaluación de la función o cero si no se dispone del valor. Se asume que estos valores son cotas inferiores de los costes reales. A medida que se realizan ejecuciones, se van aprendiendo valores más ajustados para la función heurística en cada estado. Este algoritmo converge a una solución óptima a medida que se realizan más repeticiones.

Sin embargo, el algoritmo LRTA* presenta dos deficiencias:

- Búsqueda de todas las soluciones.
  Aunque el algoritmo haya encontrado una solución aceptablemente buena, siempre continúa buscando la solución óptima.[2] LRTA* no puede determinar si la solución que ha hallado es la óptima o no (el valor de $f(n)$ siempre es una cota inferior).
- Inestabilidad de la solución.
  Al usar una función que nunca sobrestima el valor real de $h(n)$ y actualizar los valores de $h(n)$ en los nodos visitados de la solución con los valores reales, habitualmente las soluciones halladas tendrán un coste superior a aquellos caminos que no se han recorrido. En sucesivas iteraciones, LRTA* tiende a recorrer caminos que no se han explorado previamente. Aunque el algoritmo converja a la solución óptima, no se garantiza que se mejore la calidad de la solución en cada ejecución.

Ishida propone dos variaciones al algoritmo LRTA* para paliar estos problemas [Ishida, 1998]: ε-search y δ-search. ε-search limita la exploración de nuevas soluciones y permite soluciones subóptimas con un error ε. δ-search restringe la búsqueda en sucesivas ejecuciones para que no se aleje de la solución encontrada en ejecuciones previas.

5.2.3. ε-SEARCH

Esta variación consiste en incrementar en un factor ε el valor de la función heurística. Sea $h^*(n)$ el coste real del estado n hasta el objetivo y $h(n)$ una cota inferior de dicho coste. Denotaremos por $h_\varepsilon(n)$ a la función que devuelve una ε-cota inferior. Inicialmente, $h_\varepsilon(n) = (1+\varepsilon) \cdot h(n)$.

[ε-search]

1. Exploración:
   Calcular $f_\varepsilon(n') = k(n,n') + h_\varepsilon(n')$ para cada sucesor n' del estado actual n, donde $h_\varepsilon(n')$ es una ε-cota inferior del coste desde n' hasta el nodo meta y $k(n,n')$ es el coste del arco ente n y n'.

[2] Si hay más de una, no se detiene hasta que las encuentra todas

2. Mantenimiento de la consistencia:
   Actualizar la ε-cota inferior del estado n de la siguiente forma:

$$h_\varepsilon(n) \leftarrow \text{máx} \left\{ \begin{array}{l} \text{mín}_{(n')}\, f_\varepsilon(n') \\ h_\varepsilon(n) \end{array} \right\} \tag{5.3}$$

3. Selección:
   Escoger aquel sucesor n' con el menor valor para $f_\varepsilon(n')$. Los empates se resuelven aleatoriamente.

Cuando $\varepsilon = 0$, ε-search se comporta exactamente igual que LRTA*. Tras repetidas ejecuciones sobre el mismo problema, el algoritmo converge a una solución subóptima con un error de $\varepsilon \cdot h^*(n)$. El camino $P(n_0, n_1, \ldots, n_{l-1}, n_l)$ se denomina ε-óptimo si se satisface

$$\sum_{i=0}^{l-1} k(n_i, n_{i+1}) \leq (1+\varepsilon) h^*(n) \tag{5.4}$$

A medida que el valor de ε crece, resulta más difícil la convergencia a soluciones óptimas.

El rendimiento de la búsqueda en tiempo real aumenta considerablemente al permitir soluciones subóptimas. Sin embargo, un problema pendiente de resolver es la inestabilidad de la calidad de la solución durante el proceso de convergencia.

5.2.4. δ-SEARCH

A diferencia de LRTA*, δ-search mantiene cotas superiores del coste al estado objetivo. Las cotas superiores son útiles para balancear exploración con explotación del resultado.

Sea $c(j)$ el coste total de realizar j movimientos por δ-search.

$$c(j) = \sum_{i=0}^{j-1} k(n_i, n_{i+1}) \tag{5.5}$$

El algoritmo de búsqueda con cotas superiores es siguiente.

[δ-search]

1. Exploración:
   Calcular $f_u(n') = k(n, n') + h_u(n')$ para cada sucesor n' del estado actual n, donde $h_u(n')$ es una cota superior del coste desde n' hasta el nodo meta y $k(n, n')$ es el coste del arco ente n y n'.
2. Mantenimiento de la consistencia:
   Actualizar la cota superior del estado n de la siguiente forma:

$$h_u(n) \leftarrow \text{mín} \left\{ \begin{array}{l} \text{mín}'_n\, f_u(n') \\ h_u(n) \end{array} \right\} \tag{5.6}$$

3. Selección:
   Para cada sucesor n' del estado actual n, actualizar las cotas superiores de la siguiente forma:

$$h_u(n') \leftarrow \text{mín} \left\{ \begin{array}{l} k(n', n) + h_u(n) \\ h_u(n') \end{array} \right\} \tag{5.7}$$

   Escoger aquel sucesor n' con el menor valor para $f(n')$ entre aquellos estados que satisfagan la siguiente condición

$$c(j) + f_u(n') \leq (1 + \delta)h_0 \tag{5.8}$$

   donde h_0 es la cota superior del estado inicial cuando comenzó la ejecución actual. Los empates se resuelven aleatoriamente.

Cuando $\delta = \infty$, δ-search se comporta exactamente como LRTA*. En otro caso, se asegura que el coste de la solución hallada es menor que $(1 + \delta) \cdot h_0$. A medida que el valor de δ decrece, la solución se estabiliza más rápidamente. Sin embargo, también es más difícil obtener soluciones óptimas.

Se pueden combinar las ideas de estos dos últimos algoritmos, obteniendo un nuevo algoritmo de búsqueda en tiempo real denominado *$\varepsilon\delta$-search* , en el cual se modifica δ-search para que se seleccione el nodo con menor valor para $f_\varepsilon(n')$. *$\varepsilon\delta$-search* reduce el aprendizaje que necesita δ-search y disminuye la inestabilidad de las soluciones de ε-search.

5.3. Algoritmos para la gestión de información temporal

La gestión de la información temporal se lleva a cabo mediante tres tareas: la creación de nuevos hechos temporales, la actualización de la información existente y la recuperación de la información utilizando consultas temporales. El contenido de la TBN evoluciona por la repetida aplicación de estas tres operaciones.

Todas las operaciones tienen como base procesos de búsqueda de caminos o recorridos en un grafo, pues es necesario propagar todos los cambios en la TBN para mantener un grafo mínimo. El contar con un algoritmo interrumpible, acotado en el tiempo y eficiente (temporal y espacialmente) son prerrequisitos para que las operaciones que se describen a continuación puedan aplicarse en entornos dinámicos [Rebollo and Onaindía, 1999].

5.3.1. Creación de hechos temporales

La creación de un nuevo hecho temporal tiene lugar en dos casos: (i) tras la llegada de un evento externo que indica la modificación de algún aspecto

observable del entorno y (ii) por la sucesiva aplicación de las reglas que contienen el conocimiento del agente, incorporando nuevas relaciones causales en el conjunto de creencias de un AA.

En el primer caso, se inserta un hecho completamente conocido en la TBN a través de la función $Know(\phi)$. En el segundo caso, puede tratarse tanto de un dato seguro como de una predicción. Las primeras usan la misma función $Know(\phi)$, mientras que las predicciones se crean mediante la función $Believe(\phi)$. En todos los casos, ϕ es una fórmula que denota un nuevo hecho temporal junto con sus restricciones temporales unarias y binarias.

La inserción de un nuevo hecho temporal sobre las creencias del agente se divide en los pasos siguientes (véase Figura 5.1):

1. construir el punto de tiempo *begin* para la intersección y comprobar que se satisface el test de intersección temporal;
2. construir el punto de tiempo *end* de la intersección y relacionarlo con su correspondiente *begin* para incluir las restricciones temporales que indique el usuario;
3. añadir los puntos de tiempo correspondientes a los instantes de inicio de los hechos que aparecen en la conclusión de la regla, enlazándolos con los puntos de tiempo adecuados de la intersección;
4. insertar los puntos de tiempo de la finalización de las conclusiones y enlazarlos de manera adecuada con el resto del grafo.

Si al realizar alguno de estos pasos aparecen inconsistencias, se interrumpe el proceso. La comprobación de la consistencia del grafo en cada una de estas etapas se realiza mediante una *prueba de consistencia sintáctica*, que comprueba la validez de la ventana temporal de un punto de tiempo. Tal y como se pone de manifiesto en el Apartado 4.5.3, dadas las características de la TBN se elimina la necesidad de realizar un prueba de consistencia sobre los caminos, denominada en trabajos previos *prueba de consistencia semántica* [Onaindía and Rebollo, 1998].

La inserción en la TBN del resultado de la ejecución de una acción se modela como una predicción. Se debe tener en cuenta que las acciones se planifican para un futuro (se intentan), pues no se tiene la absoluta seguridad de que, en el momento en el que se vayan a ejecutar, se sigan dando en el entorno las condiciones requeridas.

Este proceso con las acciones presenta una pequeña variación respecto a la inserción de una predicción. Por ese motivo se define una función especial para la inserción de acciones en la TBN: $Try(\alpha)$. El cambio se debe a que toma como argumento una acción a ejecutar. La Figura 4.4 muestra cómo se enlaza una plan generado de forma externa con la información temporal almacenada en la TBN.

5.3.2. Actualización de la información

El proceso de actualización de la información temporal se lanza cuando se produce un cambio en los límites de la ventana temporal de cualquier punto de tiempo de la TBN. Los cambios que provocan esta modificación en la ventana temporal pueden ser dos:

1. la llegada de un nuevo valora actual que convierte el valor anterior en un hecho pasado; o
2. la confirmación de una predicción, que cambia el estado temporal del hecho de futuro a actual.

Es decir, los eventos que provocan cambios en el grafo temporal son aquellos que indican un cambio de estado de alguno de los hechos temporales que se encuentran almacenados en las creencias del agente.

Cualquiera de estos cambios provoca a su vez una modificación en las ventanas temporales de todos los puntos de tiempo que dependen de él. A este proceso lo denominamos *propagación*. El proceso de propagación consiste, entonces, en la aplicación sucesiva de modificaciones de ventanas temporales y la prueba de consistencia sintáctica correspondiente. Dado que este proceso no altera las restricciones binarias entre los puntos de tiempo, no varían las longitudes de los caminos del grafo.

El proceso de actualización que genera el cumplimiento de una predicción es el siguiente:

1. La ventana temporal del punto de tiempo asociado al instante de inicio de la predicción se actualiza a $[t, t]$, donde t es el instante de detección del evento.
2. Se actualiza la ventana del punto de tiempo correspondiente al instante de finalización, teniendo en cuenta la restricción que lo une con el inicio.
3. Se recalcula la intersección temporal con el resto de puntos de tiempo "hermanos" de su mismo punto de intersección.
4. Si hay intersección temporal, se recalcula la nueva ventana para la intersección (prueba de consistencia sintáctica).
5. Se aplican las restricciones que unen el punto de intersección con sus sucesores directos.

La modificación de la ventana temporal de un punto de tiempo siempre se realiza en el sentido de reducción del intervalo de validez del punto de tiempo. Sea $[a, b]$ la ventana temporal de un punto de tiempo en el instante t, y sea $[a', b']$ la ventana de tiempo del mismo punto en un instante posterior $t' > t$. Siempre se cumple que $a' \geq a \wedge b' \leq b \equiv [a', b'] \subseteq [a, b]$.

Sea el grafo de la Figura 5.1. Supongamos que el hecho temporal tf_A comienza a ser cierto en el instante de tiempo $t = 4$. El proceso de actualización modifica el grafo de la siguiente forma:

1. Se establece el instante de inicio de tf_A en $[4, 4]$.

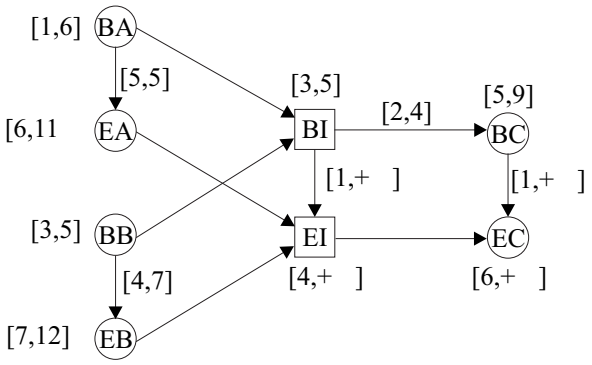


Figura 5.1. Estructura del grafo temporal

2. Se actualiza la ventana temporal del instante de finalización de tf_A, resultando un intervalo $[4+5, 4+7] = [9, 11]$.
3. Se realiza la prueba de consistencia sintáctica para comprobar que se mantiene la intersección temporal. Dicha fase devuelve una ventana temporal $[4, 5]$ como instante de inicio de la intersección.
4. Si la prueba es válida, se recalculan las ventanas temporales de los puntos de tiempo de la RHS. La ventana resultante para el instante de inicio de tf_C es $[4+2, 5+4] = [6, 9]$.

Propagación de predicciones

El proceso de propagación de una actualización recorre todos los caminos del grafo que parten del nodo modificado hasta que llega hasta un nodo hoja o hasta que la restricción unaria que define la ventana temporal sobre un punto de tiempo no varía. Este proceso está ampliamente descrito en trabajos previos [Onaindía, 1997] [Onaindía and Rebollo, 1998].

La única consideración es que, para evitar los retrasos que pueden producir los cambios en la TBN en la ejecución de otras tareas críticas para el agente, se opta por dejar pendiente la propagación de la información hasta que haya tiempo disponible entre las capas deliberativas de los *in-agent*.

Dado que el proceso de propagación mantiene el grafo mínimo, se puede afirmar que el grafo sin propagar es un grafo equivalente, pero nunca erróneo o inconsistente.

Propagación de acciones

En los sistemas clásicos de planificación, el mundo se detiene mientras el planificador construye su plan. Una vez que el plan está completo, se comienza con su ejecución, asumiendo que se mantienen todas las condiciones para la ejecución de las acciones elementales. Pero esto no ocurre así en los entornos

dinámicos. En primer lugar, el mundo sigue evolucionando mientras el planificador construye su secuencia de acciones. Y esto puede provocar que el estado de partida haya variado lo suficiente como para que la solución no sea factible. Por otro lado, el no determinismo que caracteriza los sistemas de tiempo real no garantiza que el resultado de la ejecución de las acciones sea la deseada. Hay eventos externos no controlados que varían las condiciones en las que se ejecutan las acciones. Por estos motivos, el modelo de planificación del AA se basa en la *propuesta de acciones*, cuyas condiciones vuelven a comprobarse inmediatamente antes de ejecutarlas.

Al modelar las acciones como predicciones, se vuelven a comprobar las condiciones cuando se extrae una acción del conjunto de creencias. La construcción del plan propone la ejecución de una acción, que se almacena en el conjunto de creencias como una predicción. Cuando en el entorno se cumplen las condiciones para la ejecución de la acción, la predicción se convierte en un valor actual, lanzando un evento de confirmación de una predicción y ejecutando las acciones elementales asociadas.

Las acciones, al estar asociadas a un punto de tiempo que modela la intersección, tienen también asociadas restricciones unarias, que se modifican como consecuencia de la actualización de la información temporal. La diferencia en el proceso de propagación se encuentra en el comportamiento de la propagación cuando no se puede realizar la acción.

En el caso de las predicciones sobre hechos futuros, el incumplimiento de la predicción invalida todo el proceso de inferencia que se había realizado asumiendo que se daba la situación representada por el hecho temporal. Esto desencadena un proceso de propagación del borrado que elimina todos los hechos temporales que dependen de la predicción. Es decir, se eliminan del grafo todos los sucesores del punto de tiempo asociado a la predicción.

Sin embargo, sólo por que no se den las condiciones para ejecutar una acción no quiere decir que no se pueda alcanzar un estado posterior en el que se pueda retomar el curso de acciones establecido. Supongamos que el usuario dispone de un *in-agent* que es capaz de planificar acciones. La ejecución del *in-agent* ha construido una posible secuencia de acciones para conseguir un determinado objetivo. El agente comienza a ejecutar las acciones pero, llegado un punto del plan, comprueba que no se dan las condiciones que el agente había previsto para poder ejecutar la acción. En ese caso no se debe rechazar el resto del plan. La opción mas razonable es intentar tomar acciones correctoras, reutilizando la mayor parte posible del plan original, para que engarce en algún punto del plan inicial. Las únicas acciones predichas que se deben eliminar serán aquellas que se encontraban entre la acción que ha fallado y el punto de inserción del nuevo plan.

5.3.3. Borrado de hechos temporales

La información temporal no se elimina como cualquier otro dato, como si nunca hubiera existido. Eliminar un dato temporal actual simplemente quiere

decir que *ahora* no es cierto. Pero sí lo ha sido durante todo su periodo de validez.

El modelo temporal definido para el agente ARTIS, aunque almacena los valores pasados de un atributo temporal, no razona con ellos. Simplemente se guardan para usarlos como un histórico, pero no se incluyen en los procesos de razonamiento del agente. Una posible extensión de la arquitectura puede consistir en la inclusión de una lógica temporal modal lineal en el pasado, que se puede integrar con RTAL para razonar con información temporal de cualquier tipo.

Con las información futura el proceso es diferente. Eliminar una predicción implica que nunca va a producirse, y por lo tanto el la secuencia de estados que genera la ejecución del AA nunca va a cumplirla, es decir, se prevé que nunca se va a dar ese dato en el intervalo de tiempo esperado. Como consecuencia, todo el proceso de razonamiento que se basaba en esa suposición deja de ser válido.

La desaparición de un hecho temporal implica que puede haber también otros hechos temporales que deban convertirse en pasado (por ejemplo, si había alguna restricción sobre el cumplimiento de las premisas). Si se borra un hecho futuro, deben eliminarse todos los hechos temporales que dependían de su cumplimiento.

La única salvedad, como ha quedado de manifiesto en la propagación de las acciones, es el borrado de un hecho temporal de intersección que tiene asociada una acción. La propagación del borrado provocaría la eliminación del plan por completo.

Y en ciertas ocasiones no es necesario. Un planificador puede ser capaz de recuperarse de posibles fallos en la ejecución de acciones y proponer planes alternativos para alcanzar el objetivo. Estos planes alternativos pueden coincidir en parte del plan antiguo, por lo que se puede reaprovechar el plan fallido para simplemente añadir sobre él las acciones correctoras necesarias.

Para favorecer este comportamiento, se ha tomado la determinación de no propagar el borrado de una acción. Será el propio planificador el responsable de eliminar de forma individual todas aquellas acciones que no sean válidas y reenganchar las restantes con el nuevo plan reconstruido. El planificador debe disponer de la información necesaria para seguir este funcionamiento. Básicamente, tener la referencia del hecho temporal asociado a cada acción.

5.3.4. Prueba de consistencia sintáctica

Su función es determinar si la ventana temporal resultante para un punto de tiempo es consistente, es decir, si el límite inferior de su ventana temporal es menor o igual que su límite superior.

El cálculo se realiza a partir de las ventanas temporales de sus predecesores y las restricciones temporales que se establecen entre cada uno de ellos y el nuevo punto de tiempo. Esta prueba se realiza cuando se añade a las creencias del agente una nueva relación causal.

Algoritmo 5.1 Prueba de consistencia sintáctica

1: var: tp_i, tp_j, int_B, int_E : puntos de tiempo
2: $Pred = \{tp_i | tp_i$pertenece a la LHS$\}$
3: new int_B
4: $l(int_B) = \text{máx}(l(tp_i)), \forall tp_i \in Pred, tp_i = begin(tf)$
5: $r(int_B) = \text{mín}(\text{máx}(r(tp_i)), \text{máx}(r(tp_j)))\forall tp_i, tp_j \in Pred, tp_i = begin(tf), tp_j = end(tf)$
6: **if** $l(int_B) \leq r(int_B)$ **then**
7: return int_B
8: **else**
9: return null
10: **end if**

Si la prueba de consistencia genera un nodo de intersección con una temporal válida si las restricciones temporales definidas son consistentes. En caso contrario, indica que no se hay intersección temporal entre los puntos de tiempo asociados a los hechos temporales de la LHS de la regla.

Veamos el funcionamiento de la prueba de consistencia sintáctica con un ejemplo. Supongamos que tenemos una regla de la forma:

$$tf_A \wedge tf_B \rightarrow E\Diamond^{[2,4]}tf_c \tag{5.9}$$

y sean $At(1,6,6,11)\phi = tf_A$ y $At(3,5,7,12)\varphi = tf_B$ los hechos temporales de la LHS de la regla. La Figura 5.1 muestra el subgrafo correspondiente a la regla.

Siguiendo los pasos del Algoritmo 5.1 obtenemos:

1. $Pred = \{begin(tf_A), end(tf_A), begin(tf_B), end(tf_B)\}$
2. se crea el punto de tiempo de intersección int_B
3. $inf(int_B) = \text{máx}(inf(begin(tf_A)), inf(begin(tf_B))) = \text{máx}(1,3) = 3$
4. Para que resulte más claro, dividiremos el cálculo de $sup(int_B)$ en tres etapas:
   a) $\text{máx}(sup(begin(tf_A)), sup(begin(tf_B))) = \text{máx}(4,6) = 6$
   b) $\text{máx}(sup(end(tf_A)), sup(end(tf_B))) = \text{máx}(11,12) = 12$
   c) $sup(int_B) = \text{mín}(6,12) = 6$
5. se cumple que $3 \leq 6$, luego la ventana temporal es válida y hay intersección temporal entre los puntos de tiempo de la LHS.

5.4. Búsqueda interrumpible

Todas las operaciones que se realizan en el grafo temporal se basan en procesos de búsquedas en grafos. En concreto, se trata de determinar si existe un camino (directo o indirecto) entre dos puntos de tiempo y, en caso afirmativo, cuál es su coste. De esta forma se puede determinar la relación temporal existente entre dos puntos de tiempo.

Los procesos de búsqueda en grafos son temporalmente costosos: se trata de métodos con un coste exponencial. El método que se propone es emplear algoritmos interrumpibles, cuya ejecución pueda detenerse en cualquier instante y siempre tienen una respuesta disponible. La interrumpibilidad de los algoritmos se consigue de dos formas. La primera es resolver la consulta en varias fases, cada una de las cuales proporciona una respuesta más restrictiva, pero consumiendo más tiempo. La segunda es implementar los procesos de búsqueda como procesos *"anytime"*, que pueden interrumpirse desde el exterior en cualquier momento. Este tipo de algoritmos tiene sentido en las fases más avanzadas, en las que se realizan recorridos en el grafo temporal para buscar el camino mínimo entre dos puntos de tiempo.

Para poder utilizar el mismo algoritmo tanto en las capa refleja como en la capa deliberativa, el algoritmo calcula una primera respuesta en tiempo constante $O(1)$. Si hay tiempo disponible, mejora esta respuesta buscando un camino más restrictivo en el grafo. Esta búsqueda no está acotada: depende de la posición relativa de los puntos de tiempo. Para garantizar una solución se realizan varias búsquedas, cada una de ellas con distintos grados de confianza. Cada vez que termina una de estas fases, se obtiene una respuesta con una mayor probabilidad de ser exacta.

Si, además de ser interrumpibles, utilizamos algoritmos de búsqueda en tiempo real para las distintas fases, en cada paso de ejecución podemos estimar una respuesta. De esta forma, no es necesario esperar a que termine la fase para proporcionar la respuesta y se incrementa el rendimiento de los algoritmos de razonamiento temporal. Incluso se puede obtener una respuesta sin necesidad de recorrer el camino completo entre dos puntos de tiempo. No debemos olvidar que buscamos una determinada relación temporal entre dos puntos de tiempo. Si las cotas (superiores o inferiores) ya son más restrictivas que nuestra condición, no es necesario seguir buscando.

La interrumpibilidad en el proceso de búsqueda se consigue trabajando con un espacio compartido en el que el agente deposita sus consultas y el gestor del grafo temporal escribe la respuesta obtenida hasta el momento. De esta manera, se garantiza que siempre se tiene acceso a la última respuesta calculada.

Para que los métodos sean realmente interrumpibles, se ha optado por implementar cada una de las fases como un hilo independiente. Cada hilo trabaja con una lista de consultas pendientes de resolver. Cuando se ejecuta el hilo correspondiente a una determinada fase, ésta lee el siguiente elemento de su lista e intenta resolverlo. Si lo consigue, escribe las respuesta a la consulta. En caso contrario, la consulta se pasa a la lista de la fase siguiente (véase Figura 5.2).

Si ninguna fase es capaz de resolver la consulta, ésta queda almacenada en una lista de consultas pendientes. Cuando se produce una modificación en el grafo temporal que afecta a las ventanas temporales de los puntos de tiempo, vuelven a lanzarse estas consultas, por si, en la nueva situación, puede conseguirse una respuesta firme para ellas.

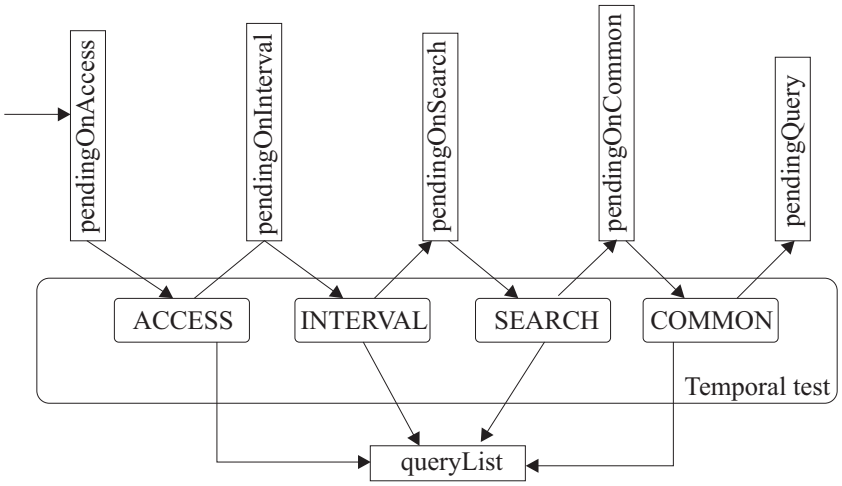


Figura 5.2. Resolución de consultas. Estructura de hilos de ejecución

5.5. Proceso de búsqueda

El proceso de búsqueda en la TBN se divide en cuatro etapas. Cada una de ellas calcula una respuesta teniendo en cuanta información adicional, de manera que la solución que aporta es más precisa, si bien también necesita un mayor tiempo de ejecución para llegar a ella. El proceso de búsqueda global es interrumpible en la medida de que cada etapa almacena su respuesta para tenerla disponible cuando se le interrumpa.

Las cuatro fases de búsqueda en el grafo temporal son las siguientes:

1. **Fase de accesibilidad.** Responde consultado la matriz de accesibilidad asociada al grafo temporal que informa de la existencia de una camino entre dos puntos de tiempo.
2. **Fase de intervalos.** Resuelve la consulta calculando la distancia existente entre las ventanas temporales de los puntos de tiempo.
3. **Fase de búsqueda directa.** Recorre el camino que une los dos puntos de tiempo almacenando la distancia real que separa ambos puntos de tiempo.
4. **Fase de nodos comunes.** Busca un nodo antecesor a ambos puntos de tiempo y calcula la distancia que los separa a partir de la composición de los caminos desde el nodo común hasta cada uno de los puntos de tiempo.

A continuación se describen cada una de las fases que constituyen el proceso de búsqueda del modelo temporal. Asumiremos que el grafo se ha creado completamente en algún instante anterior y se mantiene invariable mientras se realiza la búsqueda.

Las respuestas que proporciona el algoritmo de búsqueda son de dos tipos. Hablaremos de una respuesta *firme* cuando el algoritmo de búsqueda haya encontrado una relación más restrictiva que asegure o impida el cumplimiento de

Algoritmo 5.2 Proceso de búsqueda

```
 1: function temporal-test( tp1, type, dist, tp2 ): {true, false, possible}
 2: resp : {true, false, possible};
 3: resp ← idontknow;
 4: resp ← access-phase( tp1, type, dist, tp2 );
 5: if resp <> possible then
 6:    return resp;
 7: end if
 8: resp ← interval-phase( tp1, type, dist, tp2 );
 9: if resp <> idontknow then
10:    return resp;
11: end if
12: resp ← search-phase( tp1, type, dist, tp2 );
13: if resp <> idontknow then
14:    return resp;
15: end if
16: resp ← common-phase( tp1, type, dist, tp2 );
17: if resp <> idontknow then
18:    return resp;
19: end if
```

la consulta. Por el contrario, hablaremos de respuestas *revisable* para indicar que la información disponible en el grafo no es suficiente para asegurar una respuesta. Las respuestas firmes son *TRUE* y *FALSE*, y la respuesta revisable es *POSSIBLE*.

Si ninguna fase proporciona una respuesta firme, debemos esperar a que el grafo evolucione en el tiempo y, al restringir más las ventanas temporales de los puntos de tiempo, podremos determinar con más exactitud la relación temporal entre ellos.

5.5.1. Fase de accesibilidad

El problema fundamental de la búsqueda es determinar si existe un camino entre los puntos de tiempo. Para facilitar esta información, se mantiene una *matriz de accesibilidad*, que informa de la existencia de un camino entre cualquier par de nodos. Una matriz de accesibilidad estándar sólo contiene valores booleanos. Se puede emplear esa misma estructura para almacenar información adicional, útil para mejorar el rendimiento de ciertas búsquedas. En ella se almacena la distancia mínima y máxima entre cada pareja de puntos de tiempo.

El coste de construir la matriz de accesibilidad para un grafo ya construido es del orden de $O(n^3)$. Este coste es excesivo para las operaciones que deseamos realizar en el grafo temporal, por lo que se construye incrementalmente, a medida que añadimos nuevos nodos en el grafo. El coste de construcción para cada inserción es del orden de $O(n)$, siendo n el número de nodos en el grafo temporal.

Algoritmo 5.3 Fase de accesibilidad

```
1: if acc[tp1][tp2] cumple la restricción then
2:     return true
3: end if
4: if acc[tp1][tp2] no cumple la restricción then
5:     return false
6: end if
7: return possible
```

La *fase de accesibilidad* busca en la matriz de accesibilidad un camino entre los dos puntos de tiempo. Si existe un camino, el elemento correspondiente tendrá los valores correspondientes a las distancias mínima y máxima entre ambos. En caso afirmativo devuelve la distancia mínima o máxima (en función del tipo de consulta realizada) almacenada en la intersección de los dos puntos de tiempo. Si la distancia es más restrictiva que la restricción exigida en la consulta la respuesta es *TRUE*. En cualquier otro caso, devuelve *POSSIBLE*. El coste de la consulta a la matriz de accesibilidad es constante.

5.5.2. Fase de intervalos

Se invoca cuando la fase de accesibilidad no ha podido encontrar una respuesta firme (ha devuelto *POSSIBLE*). En ese caso, se realiza una comprobación sobre las ventanas temporales de los puntos de tiempo involucrados en la consulta temporal para intentar deducir una relación más restrictiva que la hallada hasta el momento.

Sean dos puntos de tiempo tp_i y tp_j, con ventanas temporales $[u_i^-, u_i^+]$ y $[u_j^-, u_j^+]$ respectivamente

Definición 5.1 *Se define la* ***distancia paralela*** *como la diferencia* $u_j^- - u_i^-$ *(dist. paralela mínima) ó* $u_j^+ - u_i^+$ *(dist. paralela máxima).* □

En primer lugar, se comprueba si los nodos cumplen la relación de la consulta. Para ello, se obtiene la distancia paralela (*after* o *before*, dependiendo del tipo de consulta) entre los dos puntos de tiempo. Si no se encuentra una relación temporal más restrictiva, se busca la relación contraria para poder asegurarse de que no se puede dar esa relación temporal. Si tampoco se ha encontrado la relación contraria, debe resolverse en la siguiente fase.

Dada una consulta temporal

$$\texttt{(time-test BEGIN } tp_j \texttt{ AFTER } dist_{ij} \texttt{ BEGIN } tp_i\texttt{)}$$

las repuestas posibles de esta fase son:

$$\begin{cases} u_j^- - u_i^- \geq dist_{ij} & \text{TRUE,} \\ u_j^+ - u_i^+ \leq dist_{ij} & \text{FALSE,} \\ \text{en otro caso} & \text{POSSIBLE} \end{cases}$$

Algoritmo 5.4 Fase de intervalos

```
1: l_i, l_j, r_i, r_j : ℝ+
2: l_i = after(tp_1), r_i = before(tp_1)
3: l_j = after(tp_2), r_j = before(tp_2)
4: if (l_j − l_i) ≥ da_ij then
5:     return true
6: end if
7: if (r_j − r_i) ≤ da_ij then
8:     return false
9: end if
10: return possible
```

Sigue siendo un método con un coste del orden de $O(1)$, pero restringe más la distancia posible entre dos puntos de tiempo, pues tiene en cuenta, además de los posibles caminos entre los dos puntos de tiempo involucrados en la consulta, las relaciones temporales derivadas de otros puntos de tiempo.

5.5.3. Fase de búsqueda directa

Si la fase de intervalos tampoco es capaz de encontrar una respuesta firme, es necesario buscar en la TBN el camino que une a los dos puntos de tiempo evaluando la distancia real entre ellos. Para ello, se utilizan los algoritmos de búsqueda descritos en la sección 5.2. Se han contrastado los resultados obtenidos usando RTA*, LRTA*, δ-search y ε-search . El rendiemiento de estos algoritmos puede consultarse en la sección 6.3. Se utiliza como base las búsquedas basadas en el algoritmo LRTA*, adaptado a las características especiales de la TBN. Los motivos de la elección de este algoritmo se justifica por los resultados obtenidos en las pruebas realizadas, cuyos resultados se exponen en el Capítulo 6.

El principal inconveniente de los algoritmos de búsqueda en general es que, si no existe un camino en el grafo, no lo detectan hasta que se ha recorrido completo o, al menos, una parte importante. De hecho, los algoritmos que se han descrito en la sección 5.2 asumen que todos los nodos están conectados con el estado meta. Para superar esta limitación es útil la información de la matriz de accesibilidad. Dada una consulta temporal, si la celda de intersección entre los puntos de tiempo involucrados está vacía indica que no hay un camino directo en el grafo entre ambos nodos. En ese caso, la *fase de búsqueda* devuelve una respuesta *POSSIBLE* y se pasa a la siguiente fase.

Si, por el contrario, se detecta la existencia de un camino entre los dos nodos, esta fase recorre el grafo para obtener la distancia real entre ellos. El coste total de la fase de búsqueda es el coste del algoritmo de búsqueda.

Al tratarse de un grafo generado por un conjunto de reglas, tiene una serie de características especiales que se pueden aprovechar para mejorar el rendimiento de las búsquedas.

Algoritmo 5.5 Fase de búsqueda directa

```
 1: nodesLst, succLst : list of nodes
 2: theNode : node
 3: push( nodesLst, fatherNode )
 4: while NOT nodesLst is empty do
 5:    theNode ← pop( nodesLst )
 6:    succLst ← expand-node( theNode )
 7:    for all succ in succLst do
 8:       ordered-insert( nodesLst, succ )
 9:    end for
10: end while
```

1. Cada nodo tp_j tiene un nivel asociado que se calcula como sigue:

$$Level(tp_j) = \begin{cases} 0 & \text{si } Pred(tp_j) = \varnothing \\ max_{\forall tp_i \in Pred(tp_j)}(Level(tp_i)) & \text{en otro caso} \end{cases} \quad (5.10)$$

   Si, en el proceso de búsqueda, en algún momento excedemos el nivel del nodo meta, podemos descartar ese camino porque no nos va a llevar a la solución. De forma general, podemos plantearlo como un proceso de búsqueda limitado en profundidad (a la profundidad del nodo meta), garantizando la existencia de la solución óptima dentro del subgrafo escogido.
2. Puesto que una consulta temporal no requiere el encontrar el camino óptimo, sino uno más restrictivo que el exigido en la consulta, en determinadas ocasiones podemos interrumpir la búsqueda por haber encontrado un camino suficientemente restrictivo como para responder la consulta.
3. Tomamos como $h(tp_k)$ la distancia paralela mínima o máxima, tratándose de una heurística admisible por ser en todo momento una cota inferior de la longitud real del camino hasta el nodo meta.

La *fase de búsqueda directa* tiene un coste teórico del orden de $O(pred^p)$, donde *pred* es el número máximo de predecesores en las LHS de las reglas y p es la profundidad del grafo temporal. Los resultados empíricos nos indican que esta fase puede resolverse con un coste mucho menor (véase Capítulo 6), debido a las características especiales del grafo y al uso de la información almacenada en la matriz de accesibilidad.

En el algoritmo de búsqueda (véase Algoritmo 5.5), la implementación de las funciones *expand-node* y *ordered-insert* son las que proporcionan los distintos métodos de búsqueda en el grafo. Los métodos de búsqueda son implementaciones de las propuestas de Korf e Ishida, con las variaciones anteriormente expuestas para mejorar su rendimiento.

Algoritmo 5.6 Fase de nodos comunes

```
1: theCommon : node
2: theCommon ← search-common-node( tp1, tp2 )
3: solve query using access phase
4: if resp <> possible then
5:    solve query using interval phase
6:    if resp <> possible then
7:       solve query using direct search phase
8:    end if
9: end if
10: return resp
```

5.5.4. Fase de nodos comunes

Esta fase se dispara cuando no hay un camino directo entre los dos nodos. En ese caso, se busca un punto de tiempo que sea un predecesor común a los dos nodos. En caso de encontrarlo, esta fase se resuelve lanzando dos búsquedas, cada una para buscar el camino mínimo hasta cada nodo.

El proceso general consiste en, dados dos puntos de tiempo que no se encuentran conectados por un camino directo, buscar todos los nodos predecesores a ambos y, para cada nodo común, buscar los caminos que lo unen con los dos puntos de tiempo que aparecen en la consulta. El principal inconveniente de este proceso es encontrar los nodos comunes; para ello emplearemos la información que se almacena en la matriz de accesibilidad.

Otra mejora consiste en no obtener una lista de nodos comunes, sino sólo el más restrictivo. De esta manera, sólo es necesario realizar dos búsquedas para resolver esta fase.

Un primer paso es desestimar todos los nodos comunes que sean predecesor de algún otro nodo común. De esta forma, eliminamos un gran número de posibles nodos comunes, quedándonos con el que se encuentra más próximo a los dos puntos de tiempo involucrados en la consulta temporal (se trata del nodo común de mayor nivel), consiguiendo realizar búsquedas sobre caminos lo más cortos posible.

Una vez seleccionados los nodos comunes más "prometedores", sólo queda lanzar las dos búsquedas necesarias para determinar la relación entre los puntos de tiempo.

Para realizar las búsquedas requeridas para esta fase, se utiliza la misma metodología que en una búsqueda general: un algoritmo interrumpible, formado por varias fases, que proporciona una respuesta más precisa cuanto más tiempo de ejecución tiene asignado. Para determinar la distancia temporal entre el nodo común y cada punto de tiempo iremos aplicando las fases anteriores: accesibilidad, intervalos y búsqueda. En este caso, no es necesario llegar a una nueva etapa de nodos comunes puesto que sabemos con certeza que hay un camino entre los dos nodos.

En resumen. La fase de nodos comunes se descompone en dos partes. En primer lugar, se busca un nodo antecesor de los dos puntos de tiempo. Consultando la matriz de accesibilidad, es una operación con un coste de $O(n)$, siendo n el número de nodos del grafo. A continuación, esta fase lanza dos búsquedas: desde el nodo común hasta cada uno de los puntos de tiempo, y luego construye la distancia entre los puntos de tiempo a partir de las distancias obtenidas. Para calcular las distancias emplea las tres fases anteriores, por lo que el coste puede variar entre $O(2 \times 1)$ y $O(2 \times pred^p)$.

5.6. Conclusiones

Los algoritmos aquí expuestos permiten realizar procesos de búsqueda en el grafo temporal con características que los hace adecuados para sistemas de tiempo real. La principal característica a resaltar es que se trata de métodos interrumpibles, que siempre tienen una respuesta disponible.

La mejora de las respuestas no depende tan sólo del tiempo asignado a una sola ejecución. La utilización de algoritmos que mantienen el estado de una ejecución a otra permite aprender el estado de las creencias del agente, de modo que cada nueva búsqueda no parta de cero. Esto sólo resulta adecuado en entornos con una dinámica lenta, o en los que el volumen de consultas sea muy elevado frente al de actualizaciones.

Por otra parte, los algoritmos de búsqueda pueden adaptarse a distintos grados de reactividad del agente, quien puede establecer en un momento determinado si le interesa que el proceso de búsqueda explore nuevas alternativas cuando hay tiempo disponible. O, por el contrario, si necesita búsquedas muy dirigidas por esta en una situación de estrés.

En este capítulo han descrito los algoritmos internos que se incorporan a la arquitectura de agente ARTIS para gestionar en tiempo real las creencias temporales que forman el conocimiento del dominio del agente.

La implementación de la TBN sigue la estructura de un grafo acíclico, donde los nodos representan puntos de tiempo. Los procesos de inserción, actualización y borrado de información temporal, las pruebas de consistencia de los datos y las consultas sobre la relación temporal existente entre dos puntos de tiempo se traducen en procesos de búsqueda dentro del grafo.

Se han estudiado las propuestas de Korf e Ishida para realizar búsquedas en grafos en tiempo real. En concreto, se han examinado los algoritmos RTA*, LRTA*, δ-search, ε-search y $\varepsilon\delta$-*search*. De todos ellos, tal y como demostrarán los resultados empíricos obtenidos y expuestos en el Capítulo 6, se ha escogido LRTA* para la implementación de los procesos de búsqueda en el grafo.

Por último, se han modificado los algoritmos de gestión temporal, de manera que los procesos de creación de nuevos hechos temporales, actualización de la información y borrado son capaces de tratar también con acciones.

El proceso de búsqueda incorpora una nueva fase, basada en la utilización de la información que se almacena en una matriz de accesibilidad. Estos

datos permiten además mejorar la respuesta del resto de fases, mejorando la respuesta global de los algoritmos.

6

Pruebas de evaluación

6.1. Introducción

La evaluación de las propuestas realizadas en el presente trabajo de tesis se dividen en dos partes claramente diferenciadas. Por una parte, se comprueba la expresividad de la arquitectura de agente, especialmente en lo que se refiere al conocimiento del AA y a la definición de las propiedades de seguridad y viveza, es decir, restricciones y objetivos. Por otro lado, para determinar la utilidad de los algoritmos de gestión de la información temporal, su adecuación a los requerimientos de un agente de tiempo real y el rendimiento observado tanto de tareas de análisis como de síntesis, se ha realizado un conjunto de pruebas intensivas con TBN generadas al azar con diversas estructuras.

Para poder comparar los resultados obtenidos con las propuestas actuales, frente a la última implementación evaluada [Rebollo and Onaindía, 1999], se mantienen los mismo parámetros para los grafos.

6.2. Expresividad de ARTIS

Existen ciertos ejemplos que se han tomado como casos de prueba para el diseño de agentes y sistemas multiagente [Hanks et al., 1993]. Se trata de casos en los que no se mide únicamente la eficiencia computacional y la capacidad de tratar ciertos problemas que aparecen en casos concretos (como la anomalía de Sussman). Entre ellos que se encuentra lo que podríamos considerar una adaptación del mundo de bloques y que denominaremos *el problema del recolector*.

6.2.1. Descripción del problema

El mundo consiste en una rejilla rectangular sobre la que se sitúa el agente (varios si se trata de un sistema multiagente), fichas, huecos y obstáculos.

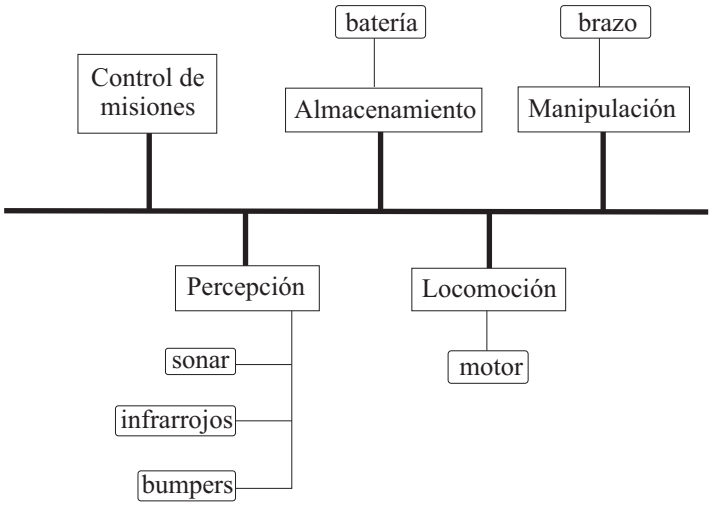


Figura 6.1. Diagrama de bloques para el robot

Cada uno de estos objetos ocupa una casilla de la rejilla. El robot-agente puede moverse en cuatro direcciones: arriba, abajo, izquierda y derecha, excepto si hay un obstáculo en alguna de las direcciones o si se encuentra en los límites de la rejilla. Cuando una ficha está en una celda adyacente al agente, éste puede empujarla para moverla en su dirección. El objetivo del agente es rellenar los huecos con fichas. Cada hueco tiene asociada una capacidad C y una puntuación S, de forma que cuando el agente consigue introducir C fichas en el hueco obtiene una puntuación S. Cada prueba tiene un límite de tiempo y el rendimiento del agente está marcado por la puntuación total obtenida al final de la prueba.

Este problema puede tratarse en simulación o en el mundo real. Naturalmente, el mundo real puede ser mucho más complejo y es necesario tener en cuenta situaciones que podemos asumir como ideales en un proceso de simulación. Por ejemplo, al empujar una ficha, ésta no tiene porqué (no suele) moverse en línea recta, de forma que el resultado de empujar una ficha durante 2 casillas puede no ser el esperado. Para nuestro caso, consideraremos que las fichas, en lugar de empujarse, se cogen con un brazo y se depositan en el hueco. El robot debe situarse exactamente en la casilla en la que están tanto la ficha como el hueco para realizar las operaciones correspondientes.

A la hora de diseñar un agente encargado de la operación sobre el robot, tendremos en cuenta una serie de subsistemas que trabajan de forma colaborativa para modelar el comportamiento del robot. Cada subsistema es el responsable de controlar una parte, y el conjunto de todas ellas, ejecutándose simultáneamente, proporcionan la funcionalidad completa requerida.

Los subsistemas de los que consta el robot son los que muestra la Figura 6.1.

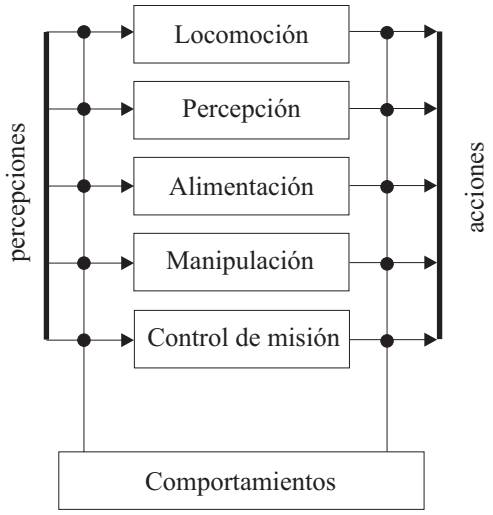


Figura 6.2. Estructura de *in-agent* para ARTIS

Locomotor: encargado de controlar los movimientos del robot a través de los motores de las distintas ruedas y la odometría.

Percepción: su misión es determinar su posición en el entorno y el estado actual del mismo a través de la sensorización que tenga disponible. Está compuesto por un sensores de infrarrojos, ultrasonidos y *bumpers*. A este subsistema puede añadirse sensorizaciones más avanzadas, como sistemas de visión artificial.

Alimentación: controla la energía disponible en el sistema para determinar si el robot tiene la autonomía suficiente como para concluir la misión.

Manipulación: controla una brazo que le permite coger objetos para que el robot los transporte de un sitio a otro.

Control de misiones: es el encargado de supervisar la actuación del robot y coordinar el resto de componentes para que el robot alcance los objetivosestablecidos.

6.2.2. Modelado del agente ARTIS

La descomposición del comportamiento del robot en varios subsistemas independientes facilita el diseño de un agente utilizando las entidades que forman la arquitectura de AA.

El agente ARTIS que controla un robot como el anteriormente descrito está formado por cinco *in-agent* , uno para cada subsistema. Los *in-agent* se dividen en dos comportamientos que podemos denominar *libre* y *ocupado*. El robot está *libre* cuando no lleva ninguna ficha y *ocupado* cuando lleva ficha. Podemos asumir que el comportamiento *libre* describe al robot cuando va a

buscar una ficha y el comportamiento *ocupado* describe al robot cuando va a depositar la ficha en el hueco. Cada comportamiento está formado por los siguientes conjuntos de agentes:

- **Libre:** locomoción, percepción, alimentación y control de misiones.
- **Ocupado:** locomoción, percepción, alimentación, **manipulación** y control de misiones.

Consideraremos que el robot pasa al estado *ocupado* cuando llega a la casilla que contiene la ficha y activa el brazo para cogerla.

El conocimiento del problema se modela a través de dos objetos: un mapa de la rejilla y el robot. El mapa es información estática, almacena el estado actual de la rejilla. Está formado por una matriz en la que se ubican las fichas, los obstáculos y los huecos. Proporciona 3 predicados *IsATile(x,y)*, *IsAHole(x,y)* e *IsAWall(x,y)*, que determinan si en la posición (x, y) de la rejilla hay una ficha, un hueco o un obstáculo respectivamente. También se puede determinar la capacidad de un hueco o la puntuación de una ficha determinada mediante *Capacity(x,y,c)* y *Score(x,y,s)*.

El robot se caracteriza por su posición en x y en y, un atributo *hasTile* que indica si tienen cogida una ficha y el atributo *score* la puntuación obtenida hasta el momento. Todos estos atributos son temporales (tienen asociado un instante de tiempo que indica cuándo el atributo toma el valor) y van a permitir adelantar la secuencia de acciones. Las acciones que puede realizar el robot son *Move(dir)* para moverse una casilla en la direción indicada (norte, sur, este, oeste), *TakeTile(x,y,s)* para coger la ficha de la celda actual, *DropTile(x,y,s)* para dejar la ficha en la casilla en la que se encuentra. Además se definen funciones de consulta para determinar el estado del robot: *Position(x,y)* determina la posición actual, *TotalScore(s)* la puntuación alcanzada y *HasTile()* si tiene o no una ficha.

La resolución del problema consiste en conseguir la mayor cantidad de puntos posible al final del tiempo asignado t_{max}. Además, nosotros exigiremos al sistema una puntuación mínima S_{min} que se debe garantizar. El *objetivo global* del agente es

$$E\Diamond^{\leq t_{max}} TotalScore(robot) = S_{min}$$

Las *restricciones globales* le impiden chocar con los obstáculos o salirse de la rejilla (por ejemplo, porque está sobre una mesa):

$$A\Box(\neg IsAWall(Position(robot)))$$

Una *restricción local* al comportamiento *ocupado* indica que un robot sólo puede llevar una ficha cada vez:

$$A\Box(HasTile(robot) \rightarrow (\neg HasTile(robot)\ \mathcal{U}\ Try(TakeTile(robot))))$$

Para garantizar la evolución del agente en un sentido determinado pueden insertarse objetivos parciales, como recoger la ficha de mayor valor antes de la mitad del recorrido

$$A\Diamond^{\leq \frac{t_{max}}{2}}(IsATile(Position(robot)) \\ \wedge \forall x, y : (IsATile(x, y) \rightarrow Score(x, y) \leq Score(Position(robot))))$$

La parte central del agente queda dentro del *in-agent* encargado del control de misiones. Consideraremos únicamente dos capas: la capa refleja y sólo una capa deliberativa. Podría dividirse en más capas y proponer más soluciones alternativas, pero con dos es suficiente para mostrar el funcionamiento del agente.

La capa refleja construye una solución siguiendo una estrategia voraz, determinando que sigue estos pasos:

1. determinar cuál es la ficha más próxima
2. moverse a esa posición y coger la ficha
3. determinar cuál es el hueco más cercano
4. moverse a esa posición y dejar la ficha

Cada vez que se ejecuta, determina sólo una de estas acciones, dependiendo de su estado actual.

La capa deliberativa construye un plan que obtiene una respuesta de una calidad aceptable. Al menos debe asegurar una puntuación S_{min}. Una primera aproximación puede ser usar también una estrategia voraz, construyendo el plan a partir de los pasos 1 a 4 anteriores. El coste de esta solución es del $O(n)$, donde n es el número de pasos de la solución. Una estrategia más avanzada trata de rentabilizar los movimientos, tratando de conseguir los máximos puntos posibles moviéndose lo menos posible.

Un plan posible está formado por una secuencia:

$$GoTo(x, y) \rightarrow Try(TakeTile(x, y)) \rightarrow GoTo(x', y') \rightarrow Try(DropTile(x', y')) \\ \rightarrow GoTo(x'', y'') \rightarrow Try(TakeTile(x'', y'')) \rightarrow \ldots$$

El *in-agent* de locomoción es el encargado de traducir la acción *GoTo(x,y)* en una secuencia de acciones elementales *Move(dir)* que lleve al robot de la posición actual al destino. El *in-agent* de manipulación mantiene el conocimiento que le permite manejar el brazo del robot para coger y dejar piezas.

La regla que permite la acción de coger una ficha puede especificarse de la siguiente forma:

$$Hold(Position(?x, ?y)) \wedge \\ Hold(IsATile(?x, ?y)) \wedge \\ \neg Hold(HasTile()) \wedge \\ \Rightarrow \\ [\![Try(TakeTile(?x?y?s))]\!] \\ E\Diamond^{[60,120]}(Know(\neg IsATile(?x, y?)) \\ \wedge Know(HasTile()))$$

X	0	0	0	0	0	0	0	0	0
0	0	0	0	-1	0	-1	1	0	1
0	0	0	0	0	0	0	2	0	-1
0	-1	0	3	3	0	3	0	1	1
0	0	0	0	0	0	0	2	1	-1
0	0	2	3	-1	0	2	2	0	3
0	0	1	0	1	1	-1	0	-1	0
0	0	-1	2	1	1	0	3	0	3
0	0	0	2	0	1	-1	2	3	0
0	0	3	0	0	2	2	0	0	0

Figura 6.3. Ejemplo de configuración para el problema del recolector

que se lee *"si el robot está en una celda donde hay una ficha y no lleva ninguna, puede tratar de coger la ficha y, entre 60 y 120 minutos después, el robot habrá conseguido coger la ficha"*.

6.2.3. Ejemplos de funcionamiento

A continuación mostraremos algunos ejemplos del funcionamiento del AA diseñado para resolver el problema arriba expuesto.

Con el fin de poder realizar comparaciones sobre las distintas ejecuciones, se plantea el escenario que muestra la Figura 6.3: un mundo de 10x10, con el robot inicialmente en la casilla marcada con una "X". Las celdas con un valor negativo (-1) indican la presencia de un hueco y su capacidad (en número de fichas). Las celdas con valor positivo (en este ejemplo, entre 1 y 3) indican que contienen una ficha y su valor. Las celdas con un 0 son celdas vacías.

Un robot en un entorno estático

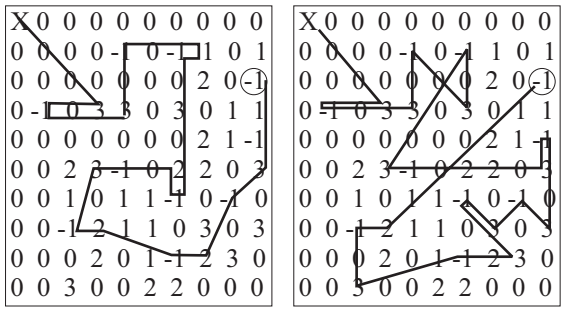
La primera aproximación consiste en emplear una estrategia completamente voraz para la capa refleja del *in-agent* de control de misiones. Se han considerado dos estrategias posibles:

- buscar la posición no vacía más cercana, y
- buscar la ficha con el valor más alto.

Los empates de resuelven aleatoriamente. Para comparar los resultados, se ha optado por examinar el estado del sistema cuando se han recogido 10 fichas y no se ha tenido en cuenta el tiempo de razonamiento, sólo el empleado por el robot.

La Figura 6.4(a) muestra el recorrido que realiza el robot cuando sólo se ejecuta la capa refleja con la estrategia de proximidad. La puntuación final alcanzada ha sido de 23 puntos y ha tardado 63 unidades de tiempo (u.d.t.) en terminar el recorrido.

El resultado de la estrategia de valor más alto se muestra en la Figura 6.4(b). Igual que en el caso anterior, el robot sólo ha utilizado su capa refleja



(a) proximidad (b) valor

Figura 6.4. Decisiones de la capa refleja

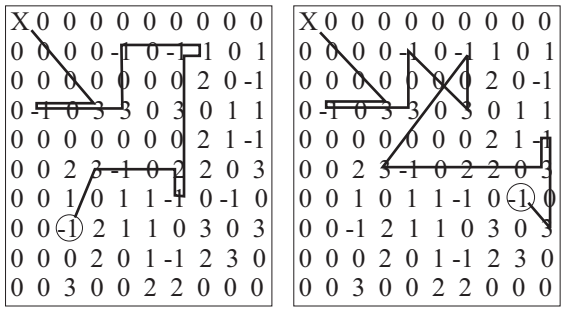
para calcular la siguiente posición a la que debe desplazarse. En este caso, recoger 10 fichas le ha costado 87 u.d.t., pero la puntuación alcanzada ha sido de 29 puntos.

Como os obvio, la estrategia que atiende al valor de las fichas consigue una mejor puntuación. Pero esto no quiere decir que sea la mejor estrategia en todos los casos. De hecho, cuando han pasado 63 u.d.t., la estrategia de valor más alto tan sólo ha conseguido reunir 15 puntos, frente a los 23 de la estrategia de proximidad.

No puede afirmarse que una estrategia sea, en general, mejor que otra. Dependerá de la dinámica del sistema (el tiempo que le cuesta al robot ejecutar las acciones) y también de la dinámica del entorno (el tiempo disponible para completar la tarea). Por ejemplo, si la disposición de las fichas y los huecos cambia cada 50 u.d.t., posiblemente la estrategia por proximidad sea más efectiva. Pero si se realiza cada 100 u.d.t. es preferible en este caso la estrategia de valor más alto.

Otras disposiciones del entorno, o tamaños diferentes de la rejilla afectan al rendimiento de las estrategias reflejas. En este caso, la solución que se propone es comenzar con una estrategia conservadora: por proximidad, y, si se observa que siempre tiene tiempo suficiente y acaba, cambiarla por la estrategia de valor.

En el siguiente ejemplo se ha tenido en cuenta el tiempo de razonamiento que ha utilizado el robot para construir tomar cada una de las decisiones. De esta forma, el tiempo ya no depende sólo de la longitud del camino. Pero ahora la respuesta del sistema dependerá del tiempo de respuesta del robot para realizar los movimientos frente al tiempo que requiere el ordenador para efectuar los cálculos. En un sistema real, el tiempo de respuesta es superior en varios órdenes de magnitud (puede llegar a minutos) al tiempo de cálculo. En los siguientes ejemplos, se ha optado por mantener las medidas en u.d.t. y considerar que las operaciones atómicas del robot siguen necesitando tan sólo 1 u.d.t.



(a) proximidad (b) valor

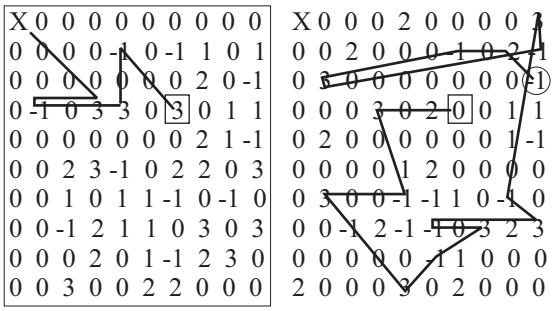
Figura 6.5. Decisiones de la capa refleja considerando el tiempo de razonamiento

Se ha optado por dar un tiempo máximo para resolver el problema de 300 u.d.t. El recorrido que ha realizado el robot en ese tiempo es el que muestra la Figura 6.5. En ella puede observarse que en ningún caso ha habido tiempo suficiente para completar el problema. La estrategia de proximidad ha tardado 281 u.d.t. y ha obtenido 15 puntos, frente a los 18 puntos de la estrategia de mayor valor en 290 u.d.t. El mayor éxito de ésta última se debe a que cerca del origen hay varias piezas con 3 puntos (el máximo), por lo que el agente que escoge por valor parte con cierta ventaja. No se puede afirmar esto en general, y habrá que realizar pruebas con varios mapas para determinar el mejor criterio. La estrategia de proximidad ha realizado 41 movimientos y ha dedicado 259 u.d.t. en determinar el siguiente movimiento a realizar cada vez. La estrategia de valor más alto ha realizado 50 movimientos, dejando las 250 u.d.t. restantes para razonar sobre la solución.

El siguiente paso es utilizar la capa deliberativa para realizar un proceso de planificación que ofrezca el camino de coste mínimo. Para compararlo con los resultados anteriores, la capa refleja realiza un proceso de razonamiento voraz siguiendo la estrategia de mayor valor. Un siguiente paso (o un segundo nivel de razonamiento) puede consistir en conseguir el camino de coste mínimo (de menor longitud posible) o incluso utilizar una función de valuación que pondere el valor de cada ficha por el coste de ir a recogerla o depositarla. Obviamente, estos procesos tienen un coste temporal todavía mayor.

El proceso de construcción de un plan que obtenga el recorrido para recoger las 10 piezas de mayor valor tiene un coste temporal de 437 u.d.t. en la configuración actual (véase Figura 6.4(b)). Así, si asignamos 300 u.d.t. para resolver el problema, el robo habrá invertido todo el tiempo en calcular los movimientos sin poder realizar ninguno.

Sin embargo, en el mundo real, el tiempo que dedica el robot a realizar las acciones es mucho mayor que es disponible para realizar cálculos. Además, es posible utilizar el tiempo durante el cual el robot se está desplazando para continuar trabajando sobre la ruta a seguir. De esta forma, puede usarse la capa refleja para proponer la primera acción *GoTo(3,3)*, y mientras el robot



(a) antes del cambio (b) después del cambio

Figura 6.6. Replanificación ante un cambio en el entorno

se desplaza con los movimientos correspondientes, puede construirse el plan que construye la ruta que recoge las 10 piezas de mayor valor:

$$Try(TakeTile(3,3)) \rightarrow GoTo(3,1) \rightarrow Try(DropTile(3,1)) \rightarrow GoTo(3,4) \rightarrow \cdots \\ \rightarrow GoTo(2,9) \rightarrow Try(DropTile(2,9))$$

De esta forma, cuando el robot llega a la posición destino (3,3), ya tiene calculada la secuencia completa de acciones. Los predicados $Try()$ indican que antes de intentar ejecutar las acciones de coger o dejar una ficha debe comprobarse que esto es posible. De esta forma se contempla que los planes puedan fallar.

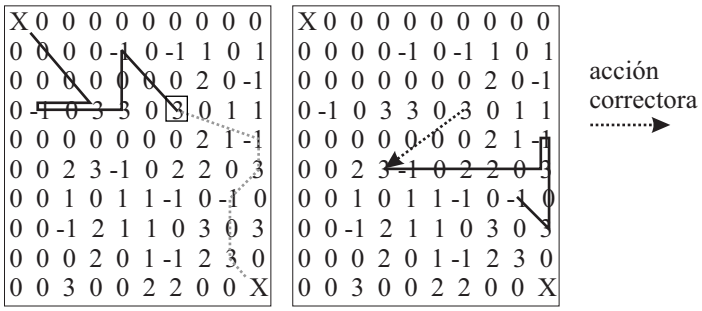
Un plan puede fallar por dos motivos en este entorno:

1. ha habido un cambio en la configuración de huecos y fichas
2. estamos en un entorno competitivo (dos robots que tratan de conseguir el mayor número de puntos) y hay otros agentes que modifican nuestro entorno.

En ambos casos, el resultado para nuestro agente-robot es que la acción de coger o dejar una pieza no puede realizarse porque la ficha ya no está, o porque el hueco ha cambiado de lugar o está lleno con la ficha de otro robot. Aunque en ambos casos el resultado es el mismo: la acción falla, las medidas que el robot puede tomar para adaptarse al nuevo entorno son diferentes.

Un robot en un entorno dinámico

Consideremos el primer caso: que el entorno cambia al azar mientras el robot sigue el plan diseñado. En este caso el cambio puede afectar a todas las celdas de la rejilla. En este entorno, cuando se detecta que una ficha o un hueco ya no están en su sitio, posiblemente hayan cambiado todas. El plan construido no es válido y el que alguna celda se mantenga inalterada no es sino casualidad. La solución que se debe plantear en este caso es eliminar por completo el plan anterior y volver a comenzar desde el principio: (1) leer el



(a) primer plan (b) plan alternativo

Figura 6.7. Replanificación ante un un fallo parcial

entorno, (2) buscar la pieza más cercana (o la de mayor valor), (3) indicar al robot que se mueva a la celda correspondiente, y (4) mientras el robot se mueve, construir un nuevo plan para la nueva situación.

Sea el estado que muestra la Figura 6.6(a). El robot ha construido el plan completo (véase Figura 6.4(b)) pero cuando llega a la celda (3,6) no encuentra la ficha —falla la acción $Try(TakeTile(3,6))$—. En ese caso debe construir un nuevo plan completo, resultando la ruta alternativa de la Figura 6.6(b).

Dos robots en un entorno competitivo

El segundo caso corresponde a un entorno competitivo, en el que dos o más robots comparten el mismo entorno y tratan de conseguir una puntuación superior a la de sus rivales. Para simplificarlo, supondremos que no hay cambios aleatorios en la distribución de las fichas y huecos. Cuando se detecta un fallo en el plan, el motivo es que uno de los contrincantes ha usado la ficha o el hueco de la celda. De nuevo, el plan no es válido, pero no hay porqué descartar el plan completo. Si el robot es lo suficientemente rápido, puede llegar a la siguiente pieza el primero y continuar el plan desde ese punto. El control de misiones tendrá que eliminar manualmente las predicciones correspondientes a las acciones anuladas. El resto de acciones posiblemente requiera una adaptación en las ventanas temporales, pero este cambio se realizará de forma automática como consecuencia de los procesos de propagación.

En la Figura 6.7 se observa qué ocurre cuando un robot llega a una celda y su contenido no es el esperado. El segundo robot (que empieza en la esquina inferior derecha), llega antes a la posición (3,6) y toma la ficha. Cuando el primer robot alcanza la misma posición, la acción $Try(TakeTile(3,6))$ falla y debe reconstruir su plan. La Figura 6.7(b) muestra el curso alternativo que calcula el primer robot tras una nueva lectura. A medida que evoluciona, si el robot detecta más fallos deberá seguir el mismo proceso hasta que se acabe el tiempo, las fichas o los huecos.

6.3. Rendimiento de los algoritmos

Para la evaluación del modelo temporal, se ha implementado un prototipo de agente ARTIS en C++. El objetivo de las pruebas no era la construcción y evaluación de un agente ARTIS completo, sino sólamente los componentes encargados del mantenimiento y gestión de la información temporal.

No se ha empleado ninguna de las características de C++ que impliquen la resolución de referencias en tiempo de ejecución. Todas las referencias a instancias de clases y acceso a funciones miembro se resuelve en tiempo de compilación, por lo que no hay ninguna diferencia en el rendimiento de los algoritmos con las versiones implementadas en C.

Las pruebas de rendimiento de los algoritmos de razonamiento temporal se han aplicado a las fases de creación del grafo temporal y a la resolución de consultas temporales para la recuperación de relaciones entre puntos de tiempo.

Para que sea posible realizar comparaciones del rendimiento de los algoritmos con las versiones anteriores de ARTIS, se ha mantenido los parámetros empleados en las implementaciones anteriores [Onaindía, 1997] [Onaindía and Rebollo, 1998] [Rebollo and Onaindía, 1999].

1. **Número de premisas** de la parte izquierda de la regla. Se han hecho simulaciones con 2, 3 y 4 premisas en la parte izquierda de la regla.
2. **Proporción** de datos **actuales y futuros**. Los hechos temporales iniciales corresponden a hechos actuales, que no dependen de ningún otro hecho temporal, o a predicciones que tan sólo esperan la confirmación del exterior del dato temporal. Los hechos intermedios son predicciones que dependen de algún hecho temporal, presente o futuro.
3. Número total de **puntos de tiempo** a generar. Se realizaron pruebas para conjuntos desde 200 a 3000 puntos de tiempo. Nótese que se trata del número de puntos de tiempo que se intentan crear. En algunos de ellos, falla la prueba de consistencia y se rechaza, por lo que el número de puntos de tiempo finales es menor.

Para cada topología, se construyeron 100 grafos al azar. Los resultados que se presentan en este trabajo corresponden a las medias de cada uno de estos conjuntos de 100 grafos.

Así mismo, con el fin de comprobar el rendimiento de los algoritmos de búsqueda, sobre cada uno de los grafos temporales generados se lanzaron 100 consultas, también generadas aleatoriamente.

La topología de los grafos generados depende del tipo de problemas que se vaya a resolver. Para los problemas de análisis, los grafos son grafos poco profundos. No tiene sentido trabajar con más de seis u ocho niveles futuros en un sistema basado en el conocimiento, especialmente en entornos de tiempo real.

Por el contrario, los problemas de síntesis generan grafos de mayor profundidad, ya que plantean secuencias de acciones, eventualmente infinitas, para

alcanzar un objetivo. Sin embargo, dado el elevado coste de construcción de la solución, no pueden elaborar demasiadas alternativas, por lo que su anchura es mucho menor que la anchura de los grafos generados en problemas de análisis.

Esta diferencia afecta especialmente a los procesos de búsqueda en el grafo, por lo que el rendimiento de los algoritmos en cada tipo de grafo no es comparable entre sí. Por esa razón, se ha optado por elaborar dos baterías de pruebas separadas para cada topología.

6.4. Grafos de problemas de análisis

Son grafos de poca profundidad con respecto al número de nodos que lo componen: con el número máximo de puntos de tiempo, los grafos no superan una profundidad de 10 nodos.

Para generar este tipo de grafos, se permite que cualquier nodo actúe como antecesor de los nodos nuevos que se introducen en el grafo. El aspecto de este tipo de grafos es el que muestra la Figura 6.8. En ellos es frecuente que se generen subgrafos no conectados entre sí, por lo que las consultas que posteriormente se realicen sobre puntos de tiempo de componentes no conexas sólo pueden resolverse en la fase de intervalos, consultando las ventanas temporales.

6.4.1. Creación del grafo temporal

Para comparar tiempos de ejecución, cada gráfica recoge el resultado de la creación de grafos con distinto número de predecesores. En cada una de ellas se comparan distintas proporciones de nodos iniciales e intermedios. Los nodos iniciales representan o hechos actuales o hechos futuros que tan sólo están pendientes de recibir confirmación del exterior.

Se han elegido proporciones 30-70, 50-50 y 70-30 para evaluar diferentes estructuras en el grafo. Una proporción 30-70 indica que el 30 % de los nodos generados son nodos iniciales (nodos de nivel 0), de los cuales depende el 70 % restante. Representa procesos en los que el sistema dispone de tiempo suficiente para razonar sobre el futuro y adelantar posibles situaciones previstas del entorno. Por el contrario, los grafos con una estructura 70-30 representan sistemas eminentemente reactivos, donde la mayor parte del tiempo de cómputo se consume en supervisar la ejecución actual del sistema y apenas queda tiempo disponible para razonar sobre el futuro.

Los resultados de las pruebas de generación de los grafos pueden observarse en las Figuras 6.9 a 6.11.

El coste de creación del grafo temporal es de orden cuadrático. Es un resultado esperado, pues cada inserción tiene un coste del orden de $O(n)$, debido a la creación de la matriz de accesibilidad. La creación de un grafo

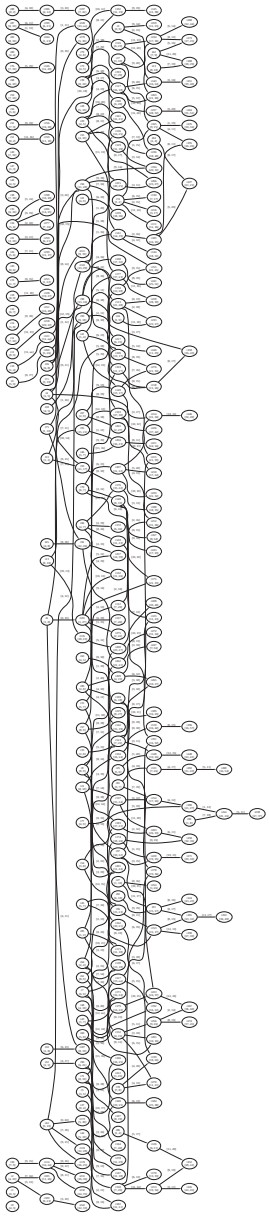


Figura 6.8. Estructura de un grafo para problemas de análisis

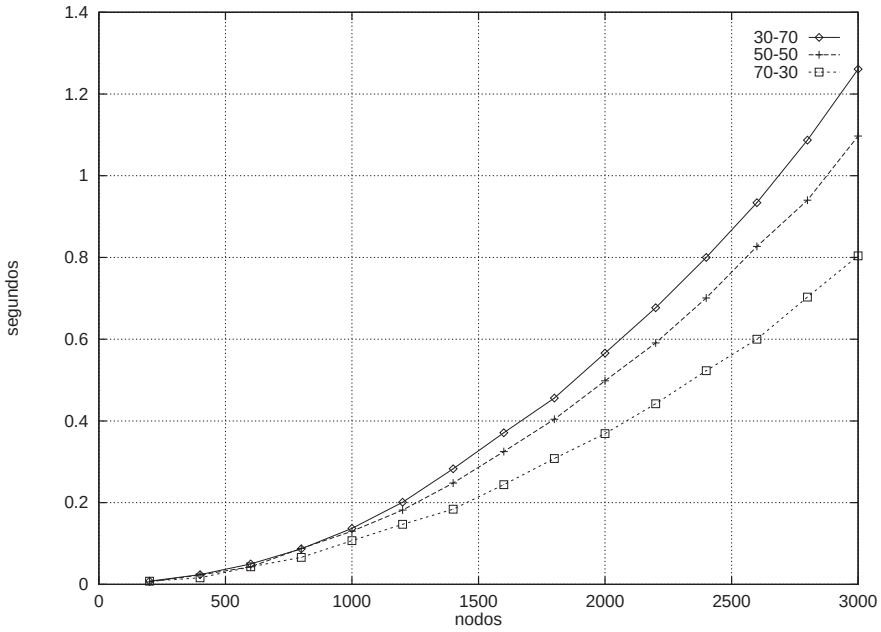


Figura 6.9. Creación del grafo temporal. 2 predecesores

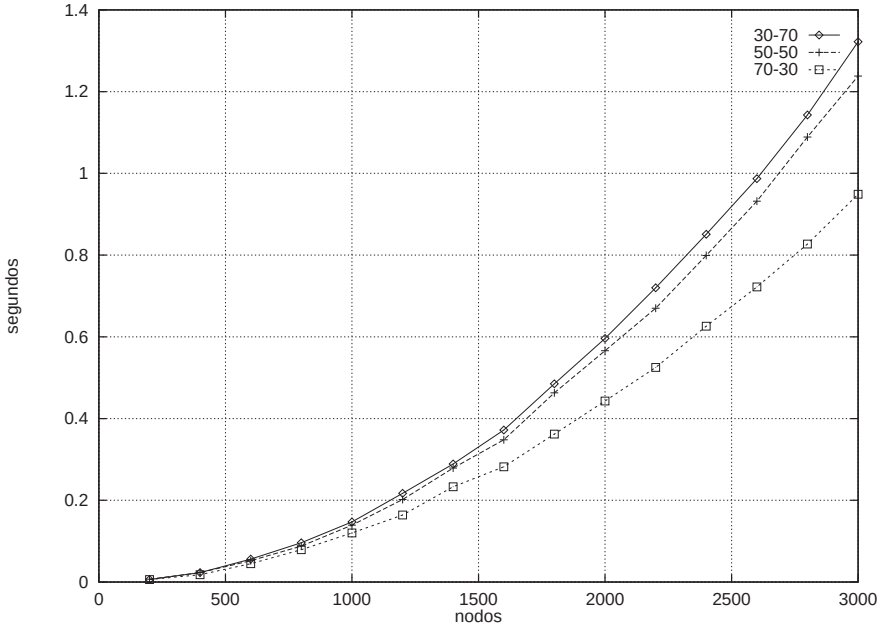


Figura 6.10. Creación del grafo temporal. 3 predecesores

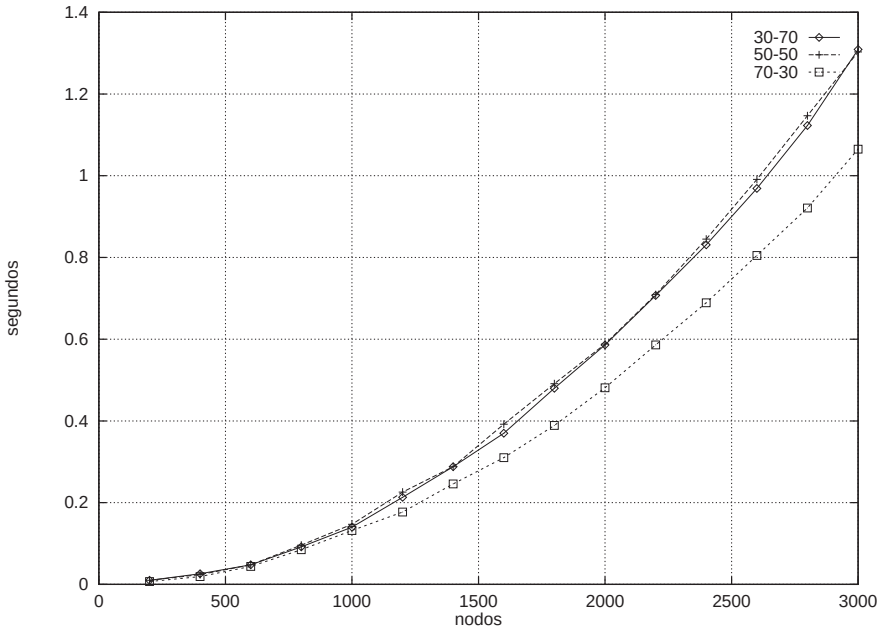


Figura 6.11. Creación del grafo temporal. 4 predecesores

temporal con n nodos realiza n operaciones de inserción, con lo que el coste resultante del proceso de creación del grafo temporal es del orden de $O(n^2)$.

El efecto que se observa es que, a medida que aumenta el número de predecesores, los tiempos de ejecución se van aproximando entre las distintas proporciones de nodos iniciales y generados. La tendencia es aproximarse a un grafo con una estructura 30-70, es decir, un 30 % de nodos iniciales —que corresponden a hechos temporales presentes— y un 70 % de nodos generados, que corresponden a hechos futuros, deducidos. Podemos emplear esta tendencia como una cota superior, de forma que se pueden medir los tiempos sobre grafos con esta estructura y usar los resultados para acotar los tiempos de respuesta de las operaciones de inserción de información en el grafo temporal.

6.4.2. Resolución de consultas

Las pruebas de resolución de consultas temporales se han realizado sobre los grafos generados en la fase anterior. Se han efectuado 100 consultas sobre cada grafo, utilizando distintos algoritmos para la fase de búsqueda directa.

Se han descartado los algoritmos que realizan búsquedas intensivas por su elevado coste. Las medidas se han tomado con métodos de búsqueda basados en el algoritmo A^* para los métodos fuera de línea y los algoritmos expuestos en el Apartado 5.2 para los métodos de tiempo real. Dado que no se han observado diferencias significativas entre LRTA* y RTA* y sus varian-

tes parametrizadas (δ-search y ε-search), y para mejorar la legibilidad de los resultados, en los gráficos sólo se incluyen los resultados obtenidos con LRTA* y RTA*.

Se han realizado pruebas con grafos de 2 y 3 predecesores. Para cada uno de ellos, se han generado estructuras con proporciones 40-60 y 50-50 de nodos iniciales e intermedios (ver Figuras 6.12 a 6.15)

Los algoritmos de resolución de consultas temporales sobre grafos para problemas de análisis consiguen costes lineales. Sin embargo, no se puede asegurar que los métodos que emplean búsqueda en tiempo real (RTA* y LRTA*) sean mejores que los métodos tradicionales.

Con el fin de comprobar el rendimiento del número de ejecuciones sobre las heurísticas de los métodos de tiempo real, se volvieron a lanzar consultas generadas aleatoriamente sobre los mismos grafos temporales. En esta ocasión, en lugar de 100 consultas, se lanzaron 1000 consultas temporales sobre cada grafo (ver Figuras 6.16 a 6.19).

El efecto que se observa al aumentar el número de consultas es el de la reducción de la variabilidad de los tiempos de respuesta. Debemos recordar que los algoritmos de tiempo real mantienen una estructura de datos propia en la que almacenan las mejores distancias conseguidas con el fin de mejorar la heurística a medida que aumenta el número de ejecuciones. Podemos considerar que el algoritmo aprende a qué distancia está un determinado puntos de tiempo. Cuantas más consultas se hagan, más se ajustan las heurísticas. Por otra parte, también hay una mayor probabilidad de recorrer caminos más de una vez. De esta forma, los resultados que se obtienen son cada vez más precisos y es probable incluso que se alcance el camino mínimo.

Sin embargo, no se puede determinar a priori el número de consultas que se van a a realizar. Depende del tipo de problema que se resuelva e incluso del estado actual del entorno y del grado de reactividad que se desee conseguir. Por todo esto, no se puede asumir que un método siempre será mejor que el resto.

6.5. Grafos de problemas de síntesis

A diferencia de las pruebas realizadas en versiones previas de los algoritmos [Onaindía, 1997], en el caso de los problemas de síntesis los grafos que se generan tienen una profundidad mayor. Por el contrario, su anchura es mucho menor (véase Figura 6.20). Es decir, los caminos son más largos, pero hay menos caminos que lleguen a un mismo puntos de tiempo.

6.5.1. Creación del grafo temporal

Las Figuras 6.21 a 6.23 muestran los resultados obtenidos en la creación de grafos generados aleatoriamente.

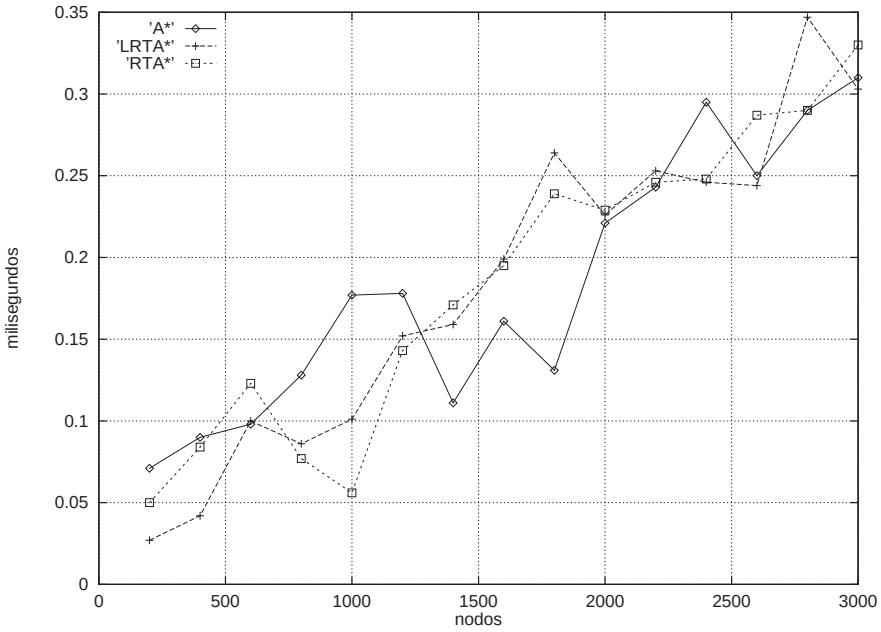


Figura 6.12. Consultas en grafos 40-60. 2 predecesores

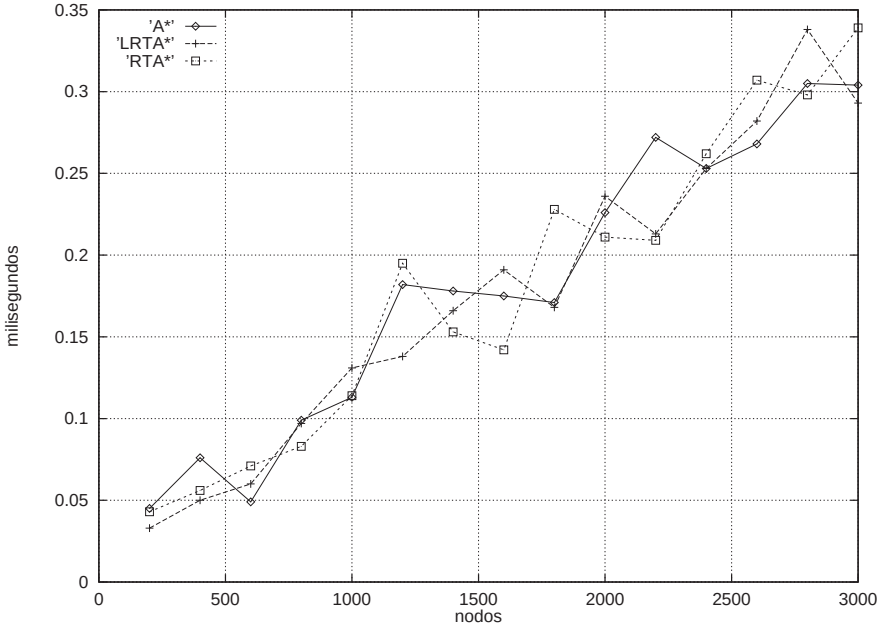


Figura 6.13. Consultas en grafos 50-50. 2 predecesores

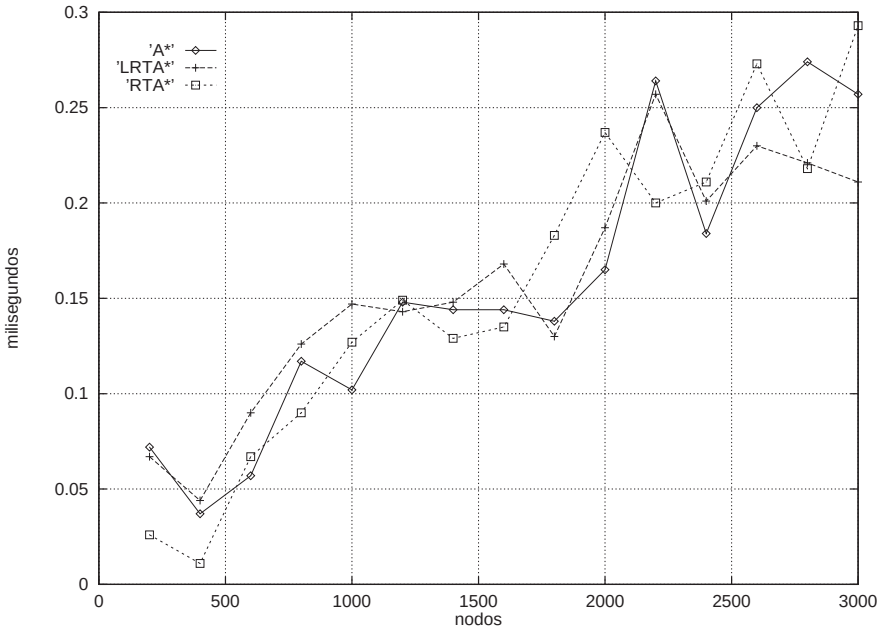


Figura 6.14. Consultas en grafos 40-60. 3 predecesores

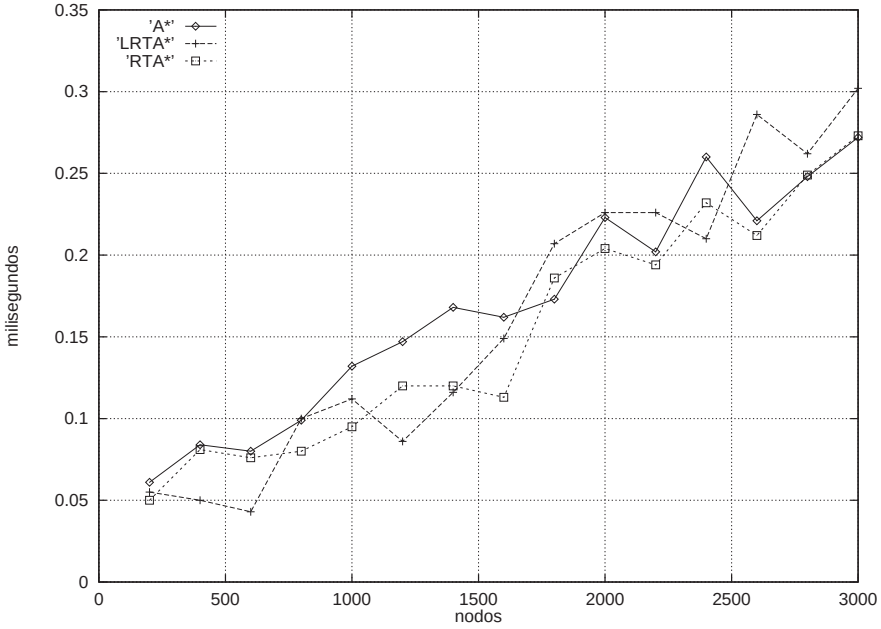


Figura 6.15. Consultas en grafos 50-50. 3 predecesores

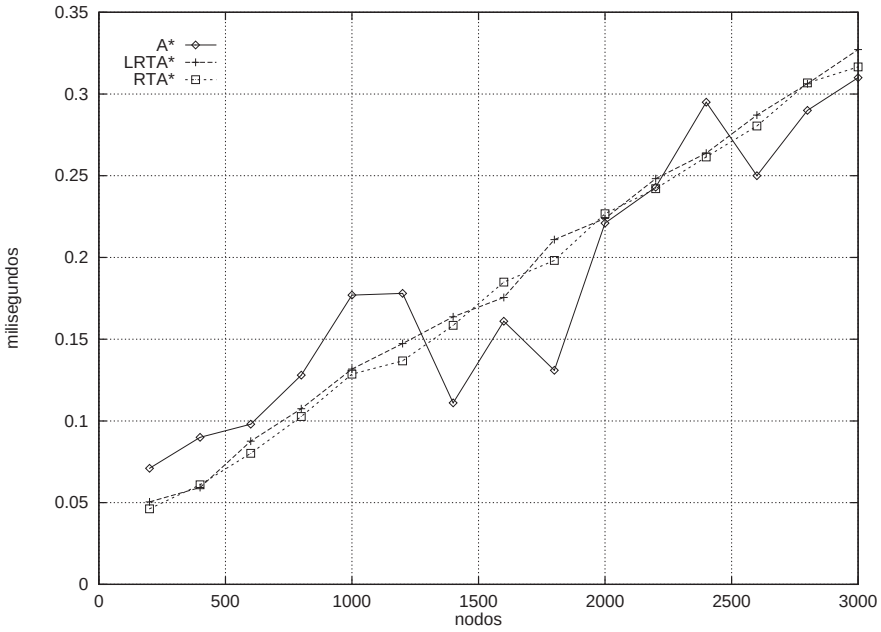


Figura 6.16. 1000 Consultas en grafos 40-60. 2 predecesores

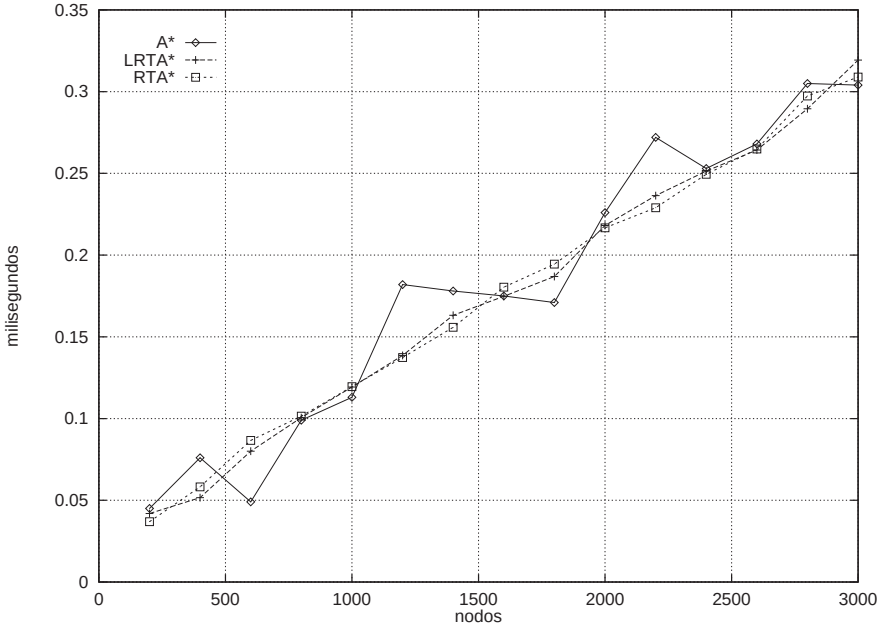


Figura 6.17. 1000 Consultas en grafos 50-50. 2 predecesores

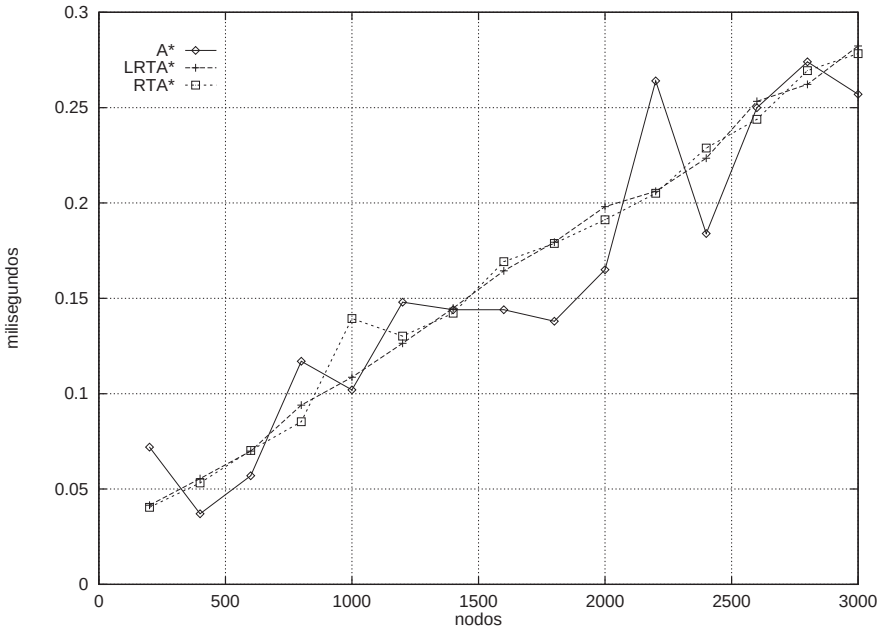


Figura 6.18. 1000 Consultas en grafos 40-60. 3 predecesores

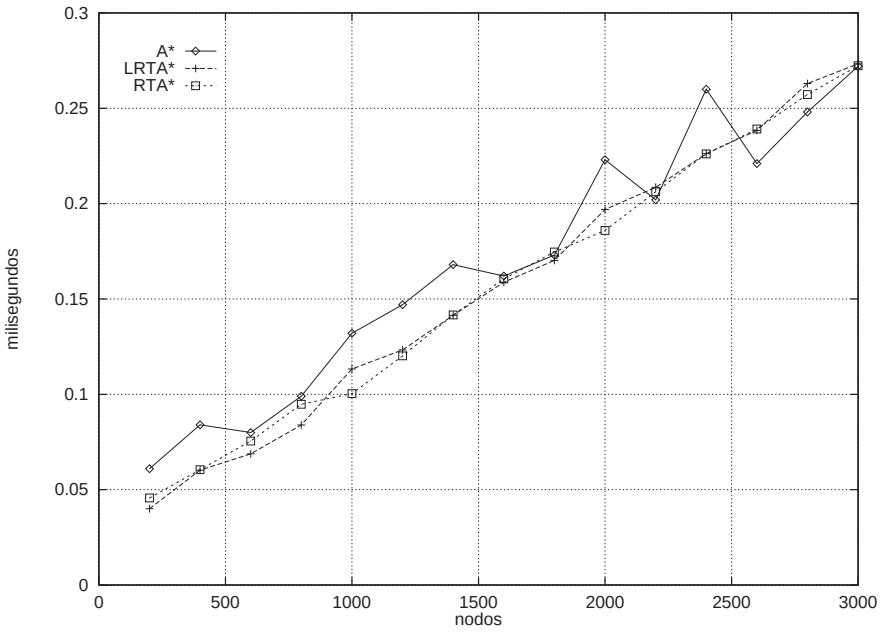


Figura 6.19. 1000 Consultas en grafos 50-50. 3 predecesores

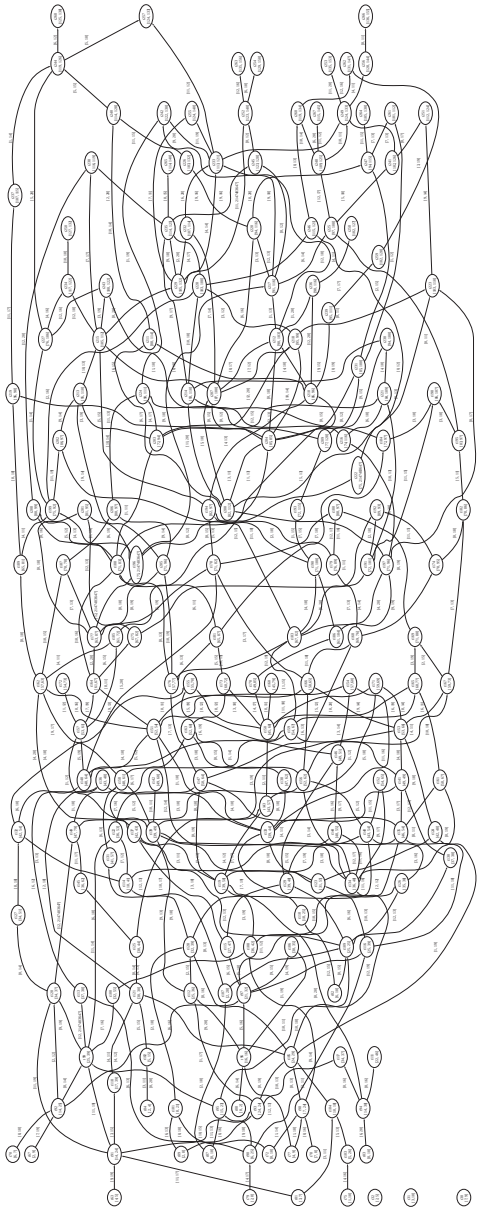


Figura 6.20. Estructura de un grafo para problemas de síntesis

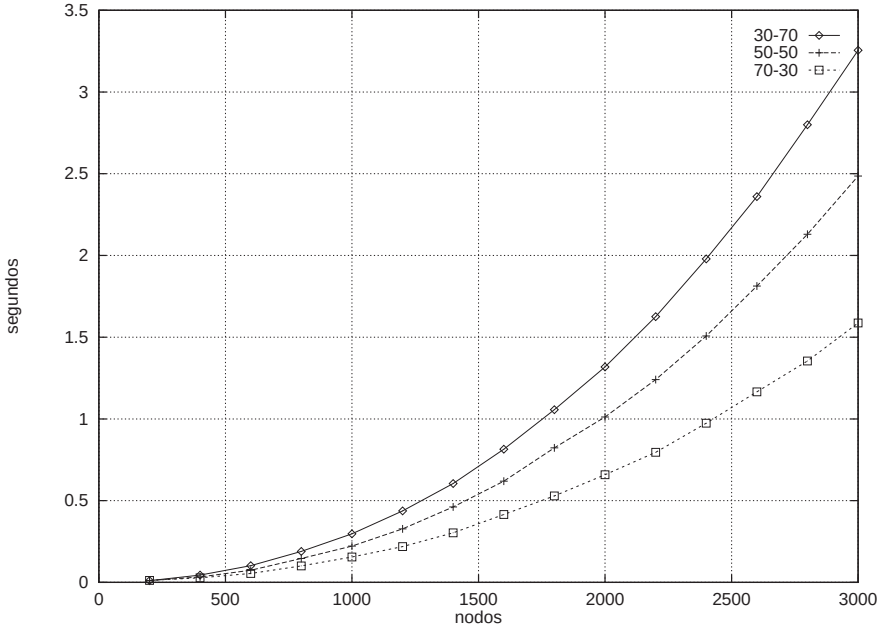


Figura 6.21. Creación del grafo temporal. 2 predecesores

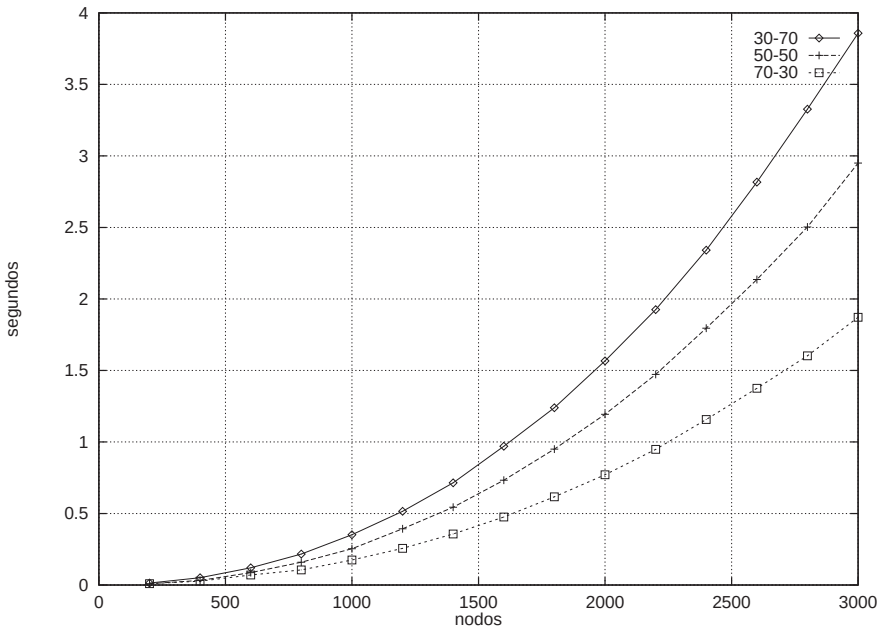


Figura 6.22. Creación del grafo temporal. 3 predecesores

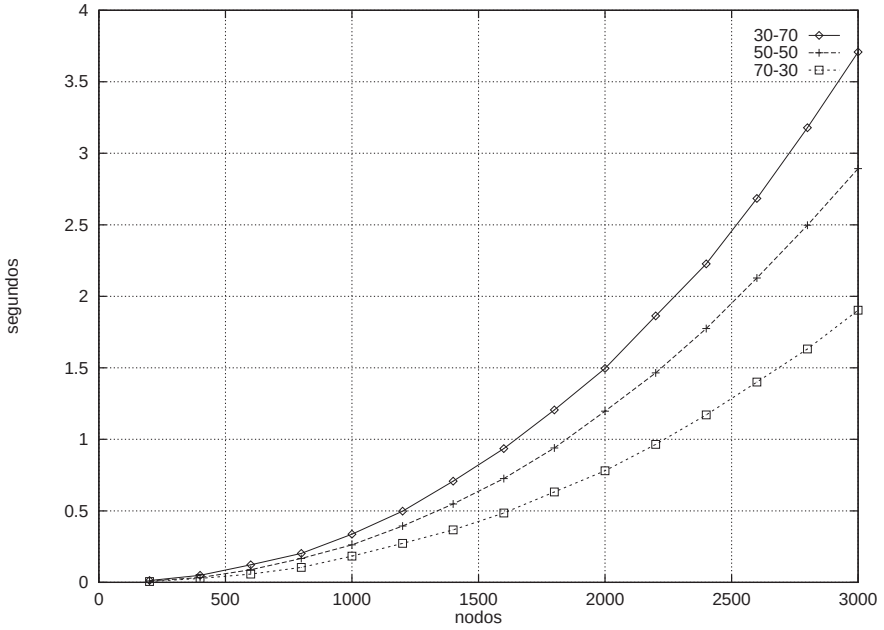


Figura 6.23. Creación del grafo temporal. 4 predecesores

Debe notarse que la tendencia es la misma que en el caso anterior: se trata de una operación con un coste del orden de $O(n^2)$. También se puede observar que el tiempo de ejecución requerido es mayor: aproximadamente el doble que en el caso de grafos para problemas de análisis.

El motivo es la longitud de los caminos. La inserción de un nodo provoca la inserción de un mayor número de elementos en la matriz de accesibilidad. Por ejemplo, si insertamos un nodo de profundidad 100 en el grafo, es necesario recalcular la distancia del camino desde cada uno de los 99 nodos predecesores hasta el nuevo nodo. En el caso de los problemas de análisis, nunca se ha superado una profundidad mayor de 10 nodos. Para los grafos que se obtienen de problemas de síntesis, se alcanzan profundidades cercanas a los 200 nodos.

6.5.2. Resolución de consultas

Las consultas temporales se han realizado con el mismo criterio que en los grafos de problemas de análisis.

Del resultado obtenido no puede extraerse un comportamiento que permita acotar el tiempo de respuesta de una consulta. Si bien el tiempo de ejecución es menor (apenas excede 0,1 milisegundo para cada consulta en el peor caso), no se observa una tendencia clara. Tampoco se produce ninguna mejora al utilizar algoritmos basados en métodos de tiempo real (ver Figuras 6.24 a 6.27).

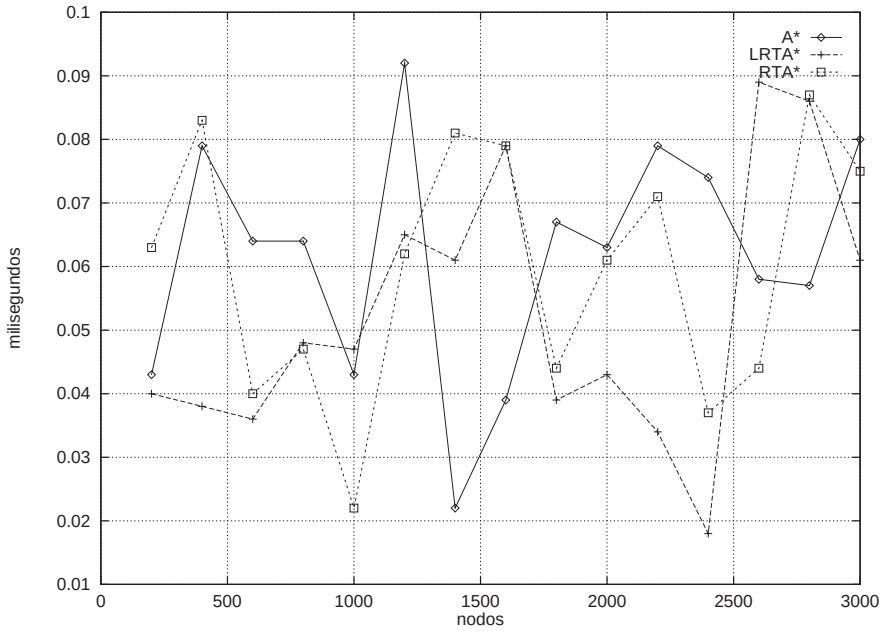


Figura 6.24. Consultas en grafos 40-60. 2 predecesores

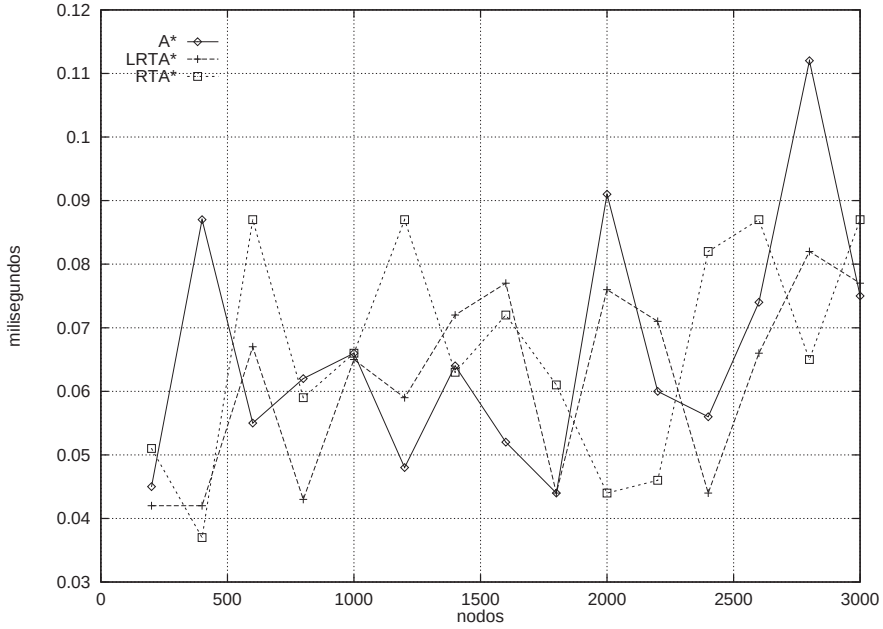


Figura 6.25. Consultas en grafos 50-50. 2 predecesores

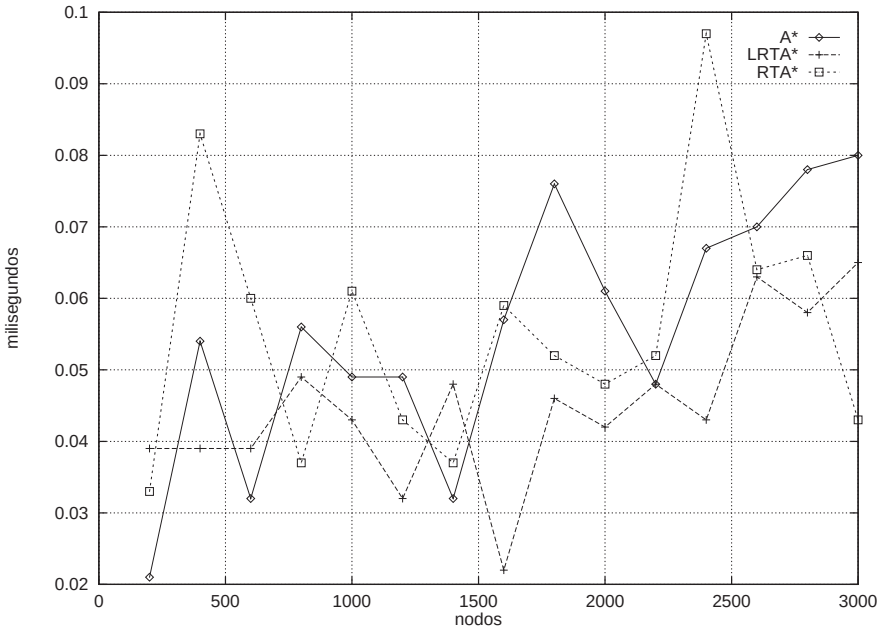


Figura 6.26. Consultas en grafos 40-60. 3 predecesores

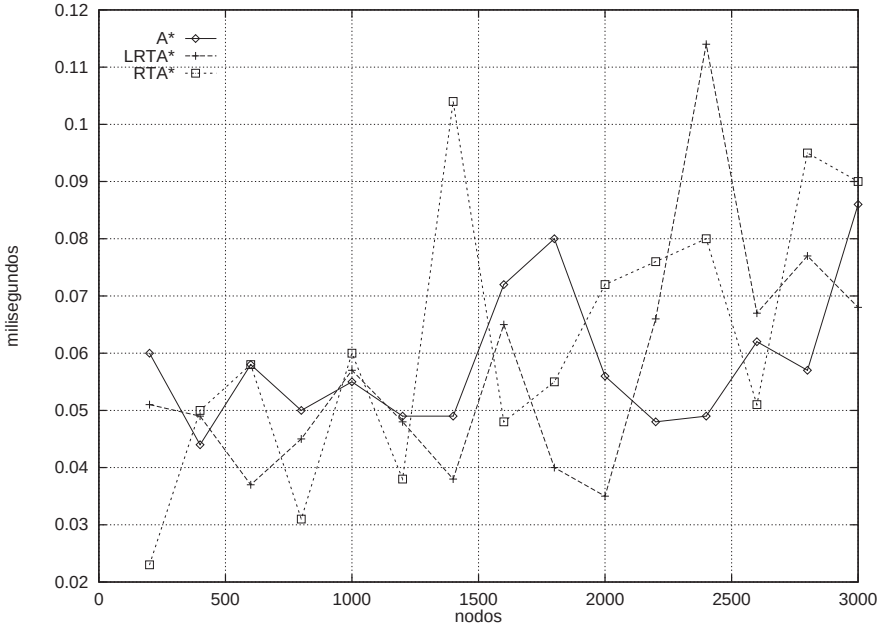


Figura 6.27. Consultas en grafos 50-50. 3 predecesores

En esta situación, la solución más adecuada que se puede aplicar teniendo en cuenta estos resultados es la de emplear métodos de búsqueda tradicionales en IA, pues el exceso de información que manejan los algoritmos de tiempo real no produce ninguna mejoría.

Al igual que en el caso de los grafos de problemas de análisis, se han vuelto a realizar otra batería de pruebas, esta vez con 1000 consultas sobre los mismos grafos. Los resultados aparecen en las Figuras 6.28 a 6.31.

De nuevo, al igual que en el caso anterior, se puede observar claramente cómo, a medida que aumenta el número de consultas realizadas sobre el mismo grafo, disminuye la variabilidad del tiempo de respuesta.

La Figura 6.32 compara el rendimiento de las consultas temporales en grafos generados por problemas de análisis y de síntesis. Se han tomado como muestra los datos de grafos con 2 predecesores y una proporción 40-60 entre nodos iniciales y nodos intermedios.

En ella observamos cómo, en ambos casos, se consigue un rendimiento lineal para los algoritmos de resolución de consultas temporales. Los algoritmos de búsqueda con capacidad de aprendizaje disminuyen considerablemente la variabilidad en los tiempos de respuesta cuando el número de consultas realizadas es lo suficientemente grande.

6.6. Conclusiones

Los resultados obtenidos tras la realización del presente trabajo de tesis se dividen en dos partes. Por un lado, se estudia la potencia de la arquitectura de agente para modelar cualquier tipo de problema intensivo en conocimiento. En segundo lugar, se ha analizado empíricamente el rendimiento de los algoritmos de gestión temporal.

Para demostrar la expresividad del modelo, se ha utilizado el problema del recolector. Sobre él, se han definido los distintos elementos de la jerarquía de entidades: conjunto de *in-agent* necesarios para resolver el problema y comportamientos que determinan cómo se comporta el agente en cada situación. También se muestran ejemplos de distintas propiedades de seguridad y viveza, como objetivos globales y restricciones globales (para todos los comportamientos) y locales (sólo en uno de los comportamientos).

Se ha probado la resolución de los subproblemas con distintos criterios de calidad, y por consiguiente con un coste computacional diferente. De esta forma puede verse la disposición de los métodos de resolución de problemas por capas (reactiva y deliberativa). Por último, se han expuesto varios ejemplos de funcionamiento, que exponen las decisiones que el robot ha tomado para resolver cada problema. Se han considerado la situación en entornos ideales y estáticos para un único robot, entornos dinámicos cuya disposición cambia en el tiempo y entornos competitivos.

Sobre el rendimiento de los algoritmos para la gestión del grafo temporal, y tras la vista de los resultados obtenidos, podemos afirmar que los nue-

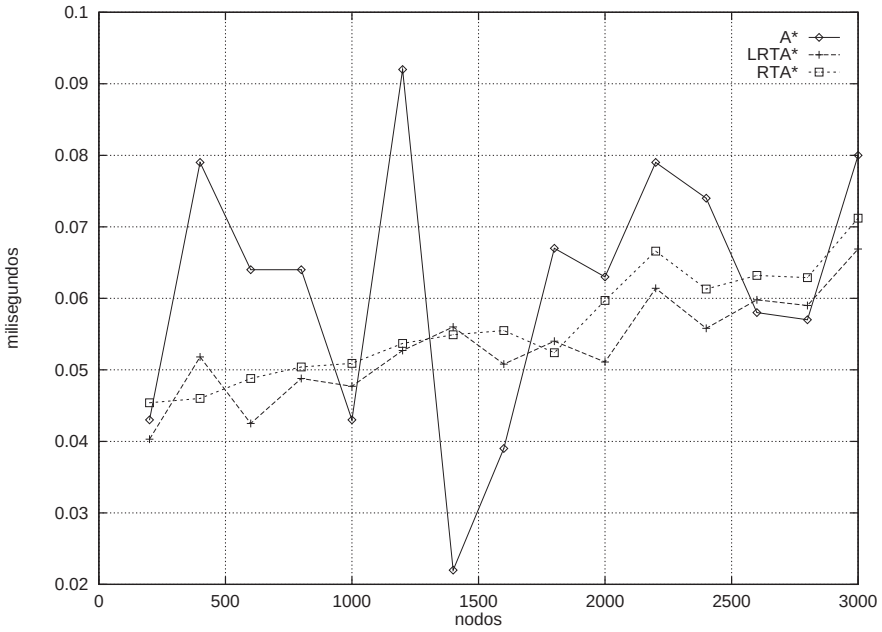


Figura 6.28. 1000 Consultas en grafos 40-60. 2 predecesores

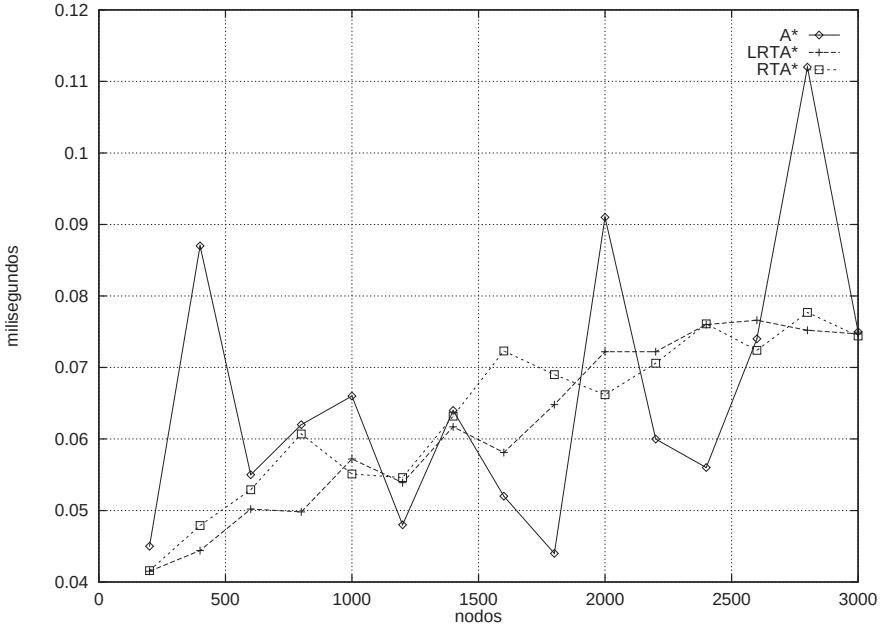


Figura 6.29. 1000 Consultas en grafos 50-50. 2 predecesores

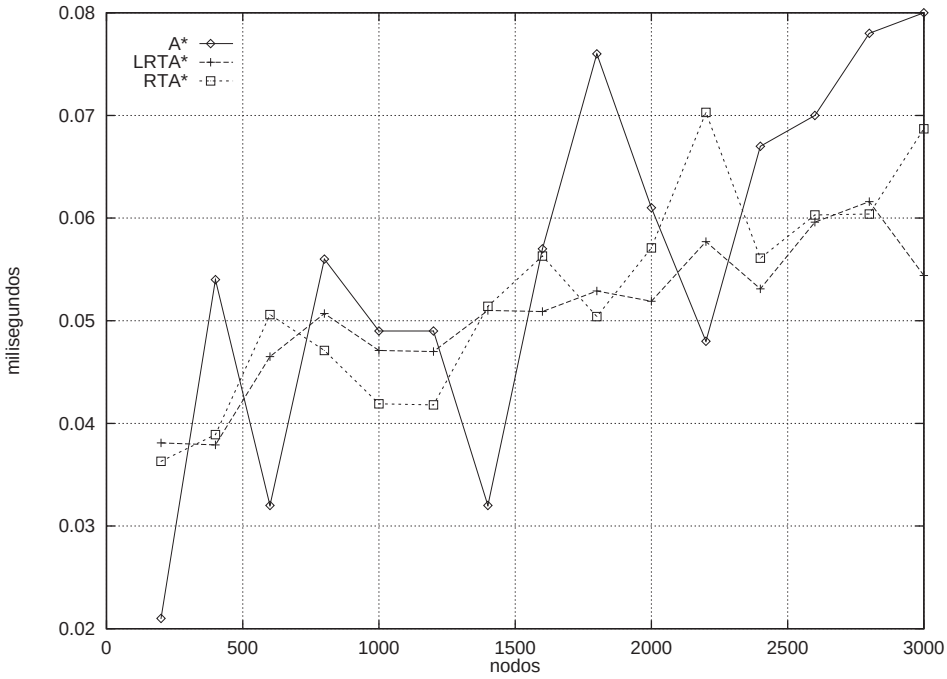


Figura 6.30. 1000 Consultas en grafos 40-60. 3 predecesores

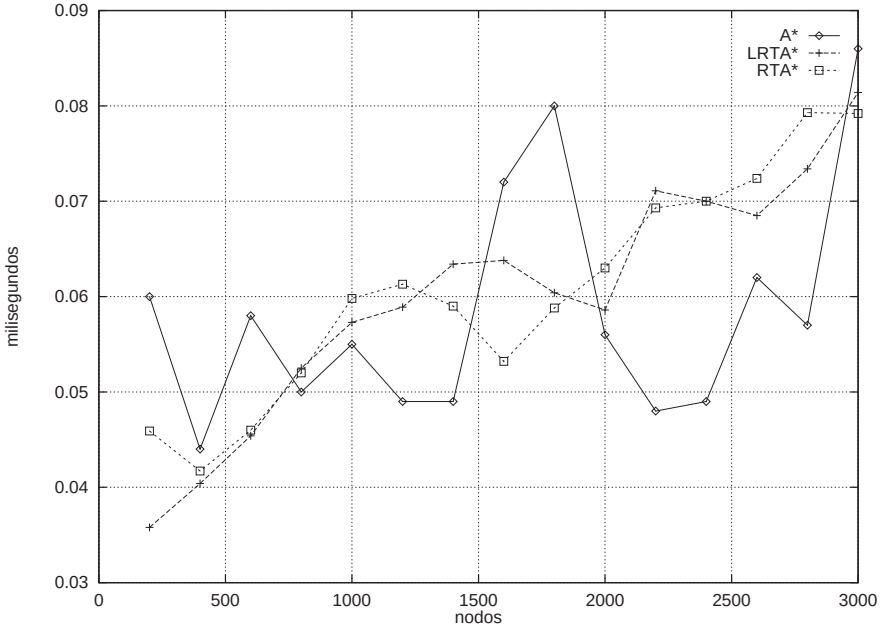


Figura 6.31. 1000 Consultas en grafos 50-50. 3 predecesores

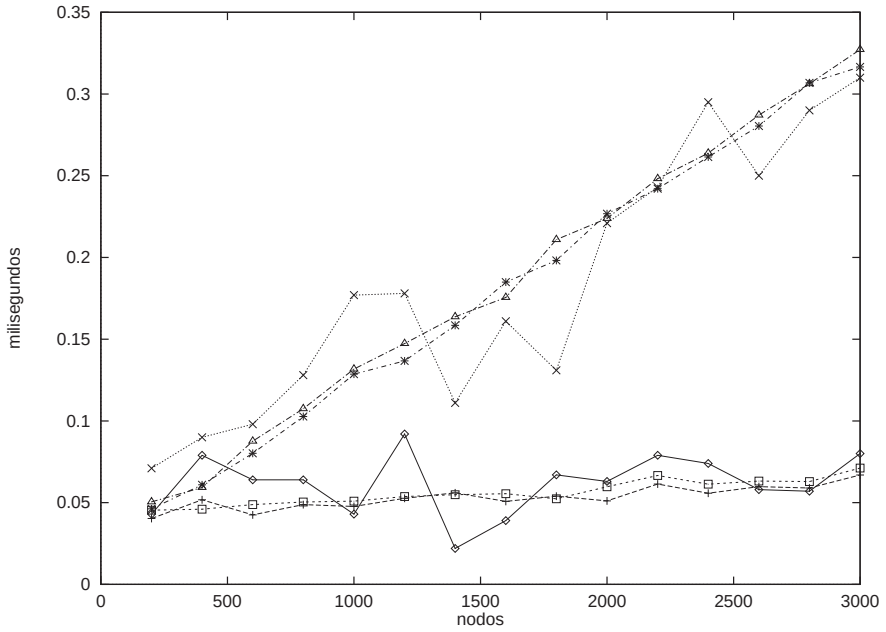


Figura 6.32. Comparación de grafos de análisis y síntesis (2 pred., 40-60)

vos métodos propuestos en el presente trabajo mejoran considerablemente el rendimiento de los algoritmos de razonamiento temporal para problemas de análisis, y resultan adecuados para tratar problemas de síntesis.

Se ha conseguido mantener el coste de creación de los grafos temporales aunque se haya añadido la creación en tiempo de ejecución de la matriz de accesibilidad, que implica un coste del orden de $O(n)$.

Por otro lado, se ha reducido considerablemente el tiempo de ejecución de las consultas. Y, lo que resulta aún más interesante, se han conseguido tiempos de ejecución lineales para resolver consultas sobre las relaciones existentes entre puntos de tiempo. Este hecho permite acotar el tiempo de ejecución de las respuestas basándose en resultados empíricos.

7

Conclusiones

7.1. Contribuciones

Las contribuciones principales que se derivan del presente trabajo afectan a la arquitectura de agente ARTIS en tres niveles:

1. a la **arquitectura general** del agente, mediante
   - la definición del modelo conceptual de agente,
   - la ampliación del tipo de problemas que se pueden resolver con ARTIS,
   - la formalización integral de todos los componentes de la arquitectura que permita especificar cualquier componente del agente, independientemente de su nivel de abstracción, y
   - el estudio de las propiedades de seguridad y viveza del agente;
2. al **modelo temporal** empleado para el mantenimiento de las creencias temporales del agente. En este aspecto:
   - se ha definido una lógica modal basada en diversas extensiones de CTL* que permite definir propiedades a distintos niveles de abstracción,
   - se han redefinido las entidades temporales, incluyendo hechos, eventos y acciones
   - se define una *red de creencias temporales*, con propiedades equivalentes a las álgebras de puntos convexas, que facilita la comprobación de la consistencia sobre el grafo temporal.
3. a los **algoritmos de razonamiento temporal** internos de la arquitectura, que incorporan mecanismos de aprendizaje para mejorar el rendimiento en la resolución de distintas instancias del mismo problema e implementan métodos interrumpibles en cualquier paso de ejecución y siempre tienen una respuesta disponible más exacta. De esta forma, se consiguen algoritmos más cercanos a las necesidades de un sistema de tiempo real. Las aportaciones en este aspecto son:
   - el uso de algoritmos de búsqueda en tiempo real con capacidad de aprendizaje para implementar los procesos de búsqueda en la TBN,

- se han modificado los algoritmos de gestión de información temporal para que incluya la creación, actualización y borrado de acciones junto con los hechos temporales,
- se calcula una matriz de accesibilidad para la TBN. el coste espacial necesario para mantenerla queda ampliamente compensado por la reducción en los tiempos de respuesta de los algoritmos, al emplear esta información para dirigir las búsquedas en el grafo.
- se construye un método basado en hilos de ejecución con listas de consultas pendientes de resolver que facilita la división en fases de la búsqueda y las propagaciones de las modificaciones.

7.1.1. Arquitectura de agente ARTIS

Modelo conceptual de ARTIS

Se ha elaborado un modelo conceptual de agente ARTIS, que muestra todos los componentes que forman el agente y cómo se relacionan entre ellos. Ofrece una visión del agente desde el punto de vista de su modelo mental, identificando las creencias, objetivos y situaciones en las que el agente se pueden encontrar.

Un *estado*, representado a través de las creencias temporales del agente, activa, o permite que siga activo, un *comportamiento* que determina el conjunto de *objetivos* y *restricciones* actuales para el agente ARTIS, así como el *conocimiento* de resolución de problemas necesario para manejar la situación, que se mantiene en un conjunto de *in-agent*. El resultado de la ejecución de estos componentes es un conjunto de acciones a realizar sobre el entorno, siguiendo un ciclo de percepción-cognición-acción.

Tipos de problemas

Hasta el momento, la arquitectura de agente ARTIS era adecuada para la resolución de los denominados *procesos de eventos discretos*, especialmente a problemas de control de procesos. Estos problemas corresponden a las denominadas tareas de análisis por la Ingeniería del Conocimiento.

La apertura de la arquitectura de agente a nuevos problemas, correspondientes a tareas de síntesis, ha mostrado algunos inconvenientes del modelo para especificar ciertos comportamientos.

En el Capítulo 3 se muestra cómo afectan a la arquitectura los distintos tipos de tareas, viendo que puede mantenerse la arquitectura general sin cambios profundos para integrar las nuevas tareas. Combinando las tareas de análisis y de síntesis con los estados de la teoría de control clásica, se obtienen 4 posibles grados de reactividad para los agentes, dependiendo de cómo se reparta el peso de las tareas inteligentes en las tareas de análisis de la información o de síntesis de resultados:

- agente reflejo
- agente sobreinformado
- agente hiperactivo
- agente racional

Formalización de ARTIS con CTL*

La especificación de la arquitectura de agente ARTIS se ha realizado desde diferentes frentes, y cada uno de ellos a aportado su propia visión del agente. El resultado la ausencia de una formalización común del agente.

En la presente tesis se ha hecho un esfuerzo especial por integrar todas las definiciones de los distintos elementos de la arquitectura, además de ofrecer un marco común que permite la especificación formal completa de un agente. Por el momento, el propósito de la construcción de un modelo formal no es la validación del diseño del agente mediante técnicas formales, sino disponer de una herramienta que permita la validación manual de ciertas propiedades.

Por otra parte, se ha intentado integrar las aproximaciones formales empleadas en las tres áreas principales que afectan a este trabajo: los agentes inteligentes, los sistemas de tiempo real y las aproximaciones temporales. Las lógicas modales, y en concreto ciertas extensiones de CTL*, es una de las lógicas empleadas en los tres campos para expresar propiedades temporales de los objetos y para construir y validar los sistemas construidos. Por este motivo, ha sido la lógica que se ha tomado como base para modelar un AA.

Propiedades de seguridad y viveza

Las propiedades de seguridad y viveza se han empleado tradicionalmente en la verificación de sistemas concurrentes. Su interés en el campo de los sistemas multiagente se centra en la consideración de éstos como sistemas distribuidos, que operan simultáneamente sobre un mismo entorno.

Pero estas mismas propiedades pueden reinterpretarse también para un único agente, equiparándose con restricciones de seguridad y objetivos respectivamente. Bajo este punto de vista, las restricciones de integridad de ARTIS quedan caracterizadas como propiedades de seguridad fuerte con *"stuttering"* limitado, y los objetivos como propiedades de viveza absoluta. Esta caracterización se debe a las propiedades adicionales que se deben verificar para sistemas de tiempo real estricto.

Combinando estas propiedades con los cuantificadores temporales, su alcance dentro de la jerarquía de entidades de la arquitectura e incluso la responsabilidad de su cumplimiento, se obtienen:

- restricciones globales (para todo el agente) y locales (para algún comportamiento),
- objetivos inevitables (se asegura su cumplimiento en todos los caminos posibles) globales y locales, y

- objetivos deseables (no se asegura su cumplimiento) globales y locales.

Los objetivos inevitables pueden considerarse *restricciones débiles* desde el punto de vista de un sistema de tiempo real, pues su no cumplimiento, si bien perjudica el rendimiento global del sistema, no representa una catástrofe para el mismo.

7.1.2. Aportaciones al modelo temporal

Lógica modal RTAL

La lógica definida, denominada RTAL, extiende CTL* en de la siguiente forma:

- permite la inclusión de relaciones cualitativas sobre los operadores temporales y $\{<, =, >\}$ como restricciones binarias.
- incorpora relaciones entre las variables que determinan las relaciones métricas del modelo
- incluye la representación de acciones y el razonamiento sobre ellas como si se tratara de predicciones futuras, de modo que se vuelven a comprobar sus condicione de disparo cuando llega el momento de ejecutarlas, para comprobar que el estado del entorno sigue permitiendo su ejecución.

Se utiliza para la especificación de las propiedades de la arquitectura y del modelo temporal, así como para la representación de los objetivos del agente, de sus restricciones de integridad y también para definir el conocimiento de resolución de problemas.

De esta forma, disponemos de una lógica a través de la cual se puede validar de manera formal que un agente cumple sus objetivos de diseño. El objetivo es dotar a los desarrolladores de una herramienta que les facilite la construcción de agentes correctos.

Ontología y teoría del tiempo

El modelo temporal propuesto emplea el punto de tiempo como primitiva temporal. El modelo que se construye a partir de ellos es un modelo no acotado (infinito en el tiempo), ramificado en el futuro y continuo.

El modelo define tres entidades temporales: los hechos temporales, caracterizados por cuatro instantes de tiempo que establecen su instante de inicio y de finalización; los eventos, de ocurrencia instantánea, y acciones, que indican la intención del agente de ejecutarlas.

Red de creencias temporales

Los puntos de tiempo y las relaciones existentes entre ellos forman un mapa de tiempos donde los nodos representan a estos puntos y los arcos

son las restricciones que se definen sobre ellos. Estas restricciones pueden ser métricas o cualitativas. Las restricciones son de dos tipos: unarias, que definen la ventana de validez de un hecho temporal, y binarias, que delimitan la relación existente entre dos puntos de tiempo.

El grafo que representa la ocurrencia de hechos temporales y acciones actuales y futuras recibe el nombre de *red de creencias temporales*. Para calcular la consistencia de estas TBN, es suficiente comprobar la consistencia de arco, pues se trata de un red CPA acíclica y no vacía sobre un dominio de intervalos simples.

Las acciones se incluyen a partir de los puntos de tiempo que definen la intersección temporal de las condiciones de una regla. Atendiendo a su tratamiento temporal, son equivalentes a predicciones futuras, pues estamos insertando acciones que pertenecen a un plan que todavía no se ha ejecutado.

La evolución de las creencias temporales de un agente ARTIS, y en consecuencia la modificación de la TBN, se produce por tres causas: la aplicación de relaciones de causalidad expresadas en las reglas que modelan el conocimiento de resolución de problemas, la generación de planes que insertan nuevas acciones para modificar el estado interno del agente o su entorno. Para construir las reglas, se ponen a disposición del usuario predicados que permiten expresar premisas de valor y de tiempo: $Hold(tf)$ y $Test(tp_i, tp_j, c_{ij})$, y también acciones cognitivas (modifican el estado interno) y efectoras (actúan sobre el entorno): $Know(\phi)$, $Believe(\phi)$ y $Try(\alpha)\phi$.

7.1.3. Algoritmos de razonamiento temporal en tiempo real

Algoritmos de búsqueda en tiempo real

Por último, se han modificado los algoritmos de razonamiento temporal para ajustarlos a los requerimientos de un sistema de tiempo real. Los algoritmos anteriores no eran realmente interrumpibles en cualquier paso de ejecución, sino que debían terminar por completo el recorrido en el grafo antes de cambiar su respuesta.

La utilización de algoritmos de búsqueda en tiempo real permite tener disponible una respuesta en cada paso de ejecución, consiguiendo que la calidad de la respuesta mejore gradualmente con el tiempo, y no lo haga bruscamente cada vez que termina una de las fases.

Además, se han incluido algoritmos con aprendizaje, que se guardan los resultados de ejecuciones anteriores para resolver otras instancias de las búsquedas aprovechando los resultados de las heurísticas previas. Cuando el número de ejecuciones sobre el mismo caso aumenta, se ha observado una reducción de la variabilidad en el tiempo que necesita para la resolución de una consulta temporal. De esta manera, en entornos "lentos" o con un gran volumen de consultas frente al de actualizaciones, se mejora la predicción del tiempo de respuesta de los algoritmos.

Se han estudiado los algoritmos de búsqueda en tiempo real disponibles: RTA*, LRTA*, δ-search, ε-search y $\varepsilon\delta$-*search*. De todos ellos, después de los resultados empíricos obtenidos, se ha optado por utilizar LRTA* en las fases de los algoritmos de búsqueda que realizan recorridos en el grafo temporal.

Gestión temporal de acciones

Las operaciones para la gestión de información temporal son tres: la creación, la actualización y el borrado de hechos temporales. A estas tareas se les añade la obligación de considerar las acciones en el proceso de razonamiento, lo que afecta a los procesos de actualización y borrado de la información.

Así como en el caso de la propagación de la modificación de una predicción simplemente debemos recorrer todos los caminos del grafo a partir del nodo modificado, hasta que se llegue a un nodo hoja o no varía la ventana temporal de alguno de los nodos, en el caso de las acciones el comportamiento de los procesos debe ser distinto.

Los cambios son más importantes cuando se debe eliminar un punto de tiempo de intersección que contiene una acción, porque dejan de darse las condiciones para que la acción se ejecute. En el caso de las predicciones, el borrado de un punto de tiempo implica el borrado en cascada de todos los puntos de tiempo que dependen de él. Sin embargo, en el caso de una acción podemos aprovechar partes del plan ya construido, de manera que sólo sea necesario realizar una serie de tareas correctoras y se pueda aprovechar parte del plan inicial construido.

Rediseño de las fases de búsqueda

El proceso de búsqueda añade una fase inicial adicional: la fase de accesibilidad, en la que se consulta la información almacenada en una matriz de accesibilidad, que se construye de forma incremental a medida que se inserta nueva información en el grafo.

Además de esta nueva fase, la utilización de la información de la matriz de accesibilidad en el resto de las fases ha mejorado el rendimiento global de los algoritmos. A la hora de seleccionar los caminos a recorrer, permite considerar exclusivamente aquellos caminos que conducen directamente al nodo final (eliminando vueltas atrás innecesarias) y buscar rápidamente antecesores comunes a dos nodos determinados para calcular de forma indirecta la distancia entre dos puntos de tiempo.

7.2. Líneas y ampliaciones futuras

7.2.1. Extensión de RTAL para sistemas multiagente

La especificación formal usando lógicas modales aquí expuesta es válida para un solo agente. Sin embargo, el interés actual del grupo se centra en

la extensión de la arquitectura a sistemas multiagente de tiempo real, en los denominados *dominios sociales de tiempo real* [Julián et al., 2002].

En este sentido, puede resultar de utilidad la generalización de CTL realizada recientemente por Alur, Henzinger y Hupferman [1997] , denominada *Alternating-time Temporal Logic* (ATL). Su característica principal es el cambio de los cuantificadores de CTL por fórmulas que expresan formas de colaboración: $\langle\langle\Gamma\rangle\rangle\varphi$, indicando que los agentes Γ pueden cooperar para conseguir que se cumpla φ. Todo esto sin modificar la tratabilidad de la validación de fórmulas mediante técnicas de *"model checking"*.

Posteriormente, van der Hoek y Wooldridge [2003] han extendido la lógica ATL mediante modalidades de conocimiento [Fagin et al., 1995] para hacer posible la representación formal de expresiones como "el grupo Γ puede cooperar para conseguir φ sii en Γ es conocimiento común ψ". La lógica resultante, denominada *Alternating-time Temporal Epistemic Logic* (ATEL), combina la tratabilidad del *"model checking"* de ATL con la expresividad del lenguaje para el razonamiento de sistemas multiagente.

De la misma forma que se ha extendido la lógica CTL* para adaptarla a un agente situado en un entorno de tiempo real, posiblemente puedan realizarse extensiones a la ATEL en el mismo sentido para conseguir una lógica válida para modelar y validar la especificación de un sistema multiagente de tiempo real.

7.2.2. Planificación en ARTIS

Para dotar al AA de comportamientos más inteligentes, es importante extender la arquitectura con componentes adicionales que proporcionen habilidades de planificación. Las alternativas que se barajan actualmente son dos.

La primera consiste en incluir en la arquitectura un *in-agent* especializado en planificación. Este *in-agent* se encargaría de construir todos los planes del agente de forma centralizada.

La segunda posibilidad en la que se trabaja es dotar a cada *in-agent* de capacidades de planificación. Así, cada *in-agent* podría trabajar de forma autónoma para resolver aquellos problemas para los que tiene conocimiento suficiente. cuando el problema que se desea resolver necesita de la colaboración de varios *in-agent*, éstos deben construir la secuencia final de acciones trabajando como un planificador distribuido para el agente.

Por otro lado, un *in-agent* de planificación necesita conocimiento específico de planificación. Además, es necesario almacenar los planes que han tenido éxito (y quizá los que no lo han tenido). De esa forma, pueden reutilizarse o bien los planes completos o bien partes de otros planes para conseguir otros objetivos. Esta forma de razonar es más parecida a la de las personas: cuando llevan a cabo un plan que tiene éxito, lo recuerdan para volver a repetirlo si es necesario.

En el caso de los agentes, también puede resultar adecuado *olvidar* cosas. Debemos recordar que una de las características de los agentes es la de su

continuidad temporal: un agente es una entidad que permanece *"viva"* durante un largo periodo de tiempo. Podríamos considerar incluso que una agente *"nace"* una sola vez y también *"muere"* una sola vez tras un largo periodo de actividad (potencialmente infinito). La capacidad de almacenamiento de un agente es limitada. Y si no lo es, al menos se debe considerar así en el caso de ARTIS porque las búsquedas de planes no estarían acotadas en el tiempo y el agente no sería útil. Por ello, *un agente debe ser capaz de olvidar*.

Desde el punto de vista estrictamente temporal, se limita el número de valores pasados que se pueden almacenar para cada hecho temporal. También se limita el número de valores futuros. Es una forma de acotar la memoria del agente. Desde el punto de vista de los planes (la biblioteca de planes), un agente no puede recordar todas las acciones que ha realizado. Los planes que más se repiten se recuerdan mejor que aquellos que hace más tiempo que no se ejecutan. Además, cuando se olvida un plan no se olvida por completo repentinamente. Primero se olvidan los detalles, quedando acciones abstractas (planes abstractos). Si en algún momento es necesario recuperar un plan parcialmente olvidado (debemos caminar por una calle por la que hace tiempo que no pasamos) tenemos el apoyo del plan abstracto y no es necesario reconstruir el plan comenzando desde cero. Por último, si un plan realmente no es necesario se olvida por completo para dejar memoria para otros planes.

Una posible solución antes de olvidar un plan, o para olvidarlo definitivamente pero poder recuperarlo, es *"anotarlo en una agenda"*. Es decir, proporcionar un espacio en memoria secundaria para que el AA pueda anotar un plan del que no quiera olvidarse. Esto sería útil para planes de uso infrecuente pero cuya construcción es costosa. Si en un momento futuro es necesario, podemos recuperarlo, pero sin perder espacio en la memoria del agente manteniendo algo que quizá no se utilice.

7.2.3. Patrones de comportamiento

Una de las principales dificultades con la que nos encontramos al construir un AA para la resolución de un problema es que no tenemos ninguna ayuda o guía para saber cómo dividir el problema, qué entidades necesitamos y cuál es el papel de cada una de ellas.

Una primera sugerencia es utilizar la metodología propuesta en la tesis de Henao para identificar las tareas intensivas en conocimiento [Henao, 2001]. Con esta guía y la propuesta de estructuración de tareas de la sección 3.4 se puede hacer un primer esbozo de la solución.

InSiDE, la herramienta de desarrollo de AA, debe proporcionar al programador una serie de utilidades adicionales que le faciliten la construcción de agentes para la solución de problemas. ARTIS, a través de su jerarquía de entidades, pretende favorecer la reutilización del código.

La propuesta es incluir *patrones* que proporcionen un esqueleto para la resolución de un problema. Estos patrones (o plantillas) pueden definirse en distintos niveles de abstracción de la jerarquía de entidades de ARTIS. Incluso

pueden llegar a definirse patrones de agentes ARTIS para problemas–tipo. El usuario (programador) no tendría más que incluir el conocimiento específico para la resolución del problema en concreto que desee resolver.

Un patrón para un AA debe incorporar un esqueleto básico de agente, proporcionando soluciones en todos los niveles de la jerarquía, meta–conocimiento y, quizá, conocimiento del dominio. Se puede considerar un AA genérico dedicado a trabajar en un entorno concreto: manejo de un robot, monitorización de un paciente, gestión de una red de distribución eléctrica, etc. Cada uno de estos problemas tiene un conjunto de responsabilidades características, unas creencias en común, una caracterización sobre la asignación de tiempos para cada tarea, etc.

El patrón generaría la estructura completa del AA, incluyendo patrones para todos las entidades de los niveles inferiores. Incluso los agentes que haya creado en usuario pueden utilizarse para crear nuevos patrones que le sean de utilidad en un futuro.

Este tipo de patrones, tan general, sería de utilidad para un desarrollador que necesita programar un equipo completo de agentes ARTIS; por ejemplo, un grupo formado por robots autónomos iguales entre sí. El usuario sólo tendría que ocuparse de generar la estructura de agente una vez y después replicarla como un patrón en los demás.

Los patrones de *in-agent* se corresponden con las plantillas de los tipos de tareas de CommonKADS [Schreiber et al., 2000] en su variante más compleja. Es un tema pendiente de discusión y cuya aplicación conllevaría cambios profundos en la arquitectura, pues afecta al propio control interno del AA.

7.2.4. Conocimiento social

Al plantear la extensión de ARTIS para emplearlo en sistemas multi–agente (SIMBA), es necesario incorporar una serie de características que permitan resolver problemas mediante un conjunto de agentes autónomos. Se trata de lo que, en sistemas distribuidos, se conoce como *sistemas complejos adaptativos*.

Las extensiones principales de ARTIS en este sentido deben ser:

- un lenguaje de comunicación de alto nivel,
- mecanismos para la resolución colaborativa de problemas,
- habilidades de negociación y argumentación, y
- formación dinámica de coaliciones.

Estos cambios afectan al modelo mental del AA, pues es necesario incluir conocimiento sobre la sociedad de agentes que constituye SIMBA. Forman parte de este conocimiento las creencias del agente sobre lo que conoce el resto, objetivos y restricciones comunes de la organización, coordinación de planes, etc.

A

Definición de la lógica RTAL

En este documento se especifica el modelo formal temporal utilizando para representar el tiempo en un agente ARTIS (AA). El modelo está basado en la lógica modal RTCTL [Emerson et al., 1992][Emerson, 1990], extendida de forma que permita trabajar cómodamente con las creencias de los agentes. Esta lógica, a la que denominaremos RTAL, permite una adecuada representación de la información atendiendo a su estado temporal (pasada, actual y futura) y de las restricciones temporales que pueden existir. La lógica temporal que usaremos para representar el conocimiento de un AA debe ser: **de primer orden, composicional, ramificada, de puntos, discreta y futura**. La lógica CTL* (*Computation Tree Logic*)[Clarke et al., 1986] cumple todas estas propiedades, a excepción de ser de primer orden (es una lógica proposicional). Sin embargo, la extensión de CTL* para que sea una lógica de primer orden es sencilla.

A.1. Sintaxis de RTAL

El alfabeto del lenguaje de primer orden $\mathcal{L}$ está formado por

- *conectivas lógicas* $\neg$, $\wedge$, $\vee$, $\rightarrow$
- *operadores temporales* $\Diamond$, $\Box$, $\bigcirc$, $\mathcal{U}$.
- *cuantificadores* A, e
- *variables*: un conjunto contable $x_0, x_1, x_2, \ldots$
- *símbolos auxiliares*), (,], [
- *símbolos de funciones*: un conjunto contable $f_0, f_1, f_2 \ldots$. Cada símbolo de función tiene asociada un aridad ≥ 0.
- *símbolos de constantes*: un conjunto contable $c_0, c_1, c_2, \ldots$
- *símbolos de predicados*: un conjunto contable $P_0, P_1, P_2, \ldots$. Cada símbolo de predicado tiene asociada un aridad ≥ 0.

Los términos del lenguaje $\mathcal{L}$ se definen de la siguiente forma:

1. las variables y las constantes son términos
2. si f es un símbolo de función de aridad n y $t_1, t_2, ..., t_n$ son términos de $\mathcal{L}$, entonces $f(t_1, t_2, ..., t_n)$ es un término.
3. el conjunto de todos los términos se generan por 1 y 2.

Una fórmula bien formada (*fbf*) del lenguaje de primer orden $\mathcal{L}$ se define como sigue:

S1 si P es un símbolo de predicado de aridad n y $t_1, t_2, ..., t_n$ son términos de $\mathcal{L}$, entonces $P(t_1, t_2, ..., t_n)$ es una fórmula de estado.
S2 si p, q son fórmulas de estado, entonces también lo son (p), $\neg p$ y $p \wedge q$.
S3 si p es una fórmula de camino, entonces Ap y Ep son fórmulas de estado.
P1 cada fórmula de estado es también una fórmula de camino
P2 si p, q son fórmulas de camino, entonces también lo son (p), $\neg p$ y $p \wedge q$.
P3 si p, q son fórmulas de camino, entonces también lo son $p\,\mathcal{U}\,q$ y $\bigcirc p$.
P4 si p y q son fórmulas de camino y t, s son términos de $\mathcal{L}$ que se evalúan a un valor entero, entonces también lo son $p\,\mathcal{U}^{[t,s]}\,q$ y $\bigcirc^{[t,s]}\,p$
P5 sea $f(x)$ una fórmula de camino en la que x es una variable libre y sean $a, b \in \mathbb{N}$. Entonces $\forall x f(x)$, $\forall x \in [a,b], f(x)$, $\exists x f(x)$ y $\exists x \in [a,b], f(x)$ son fórmulas de camino.

El conjunto de fórmulas de estado y de camino que generan las reglas S1–3 y P1–5 respectivamente, forman el lenguaje RTAL. El resto de expresiones con las conectivas y operadores temporales restantes pueden obtenerse como abreviaciones:

La expresión...	abreviaría...
$A\Diamond q$	$A(true\,\mathcal{U}\,q)$
$E\Diamond q$	$E(true\,\mathcal{U}\,q)$
$A\Box q$	$\neg E\Diamond\neg q$
$E\Box q$	$\neg A\Diamond\neg q$
$A\bigcirc q$	$\neg E\bigcirc\neg q$

Las expresiones con limitadores temporales para los operadores son análogas a las que muestra la tabla.

A.2. Semántica

Una fórmula RTAL se interpreta sobre una estructura temporal $M = (S, R, \mathcal{L})$ donde S es un conjunto de estados, R es una relación binaria total sobre S.... Un *camino completo* es una secuencia infinita $s_0, s_1, s_2, \ldots$ de estados tales que $\forall i (s_i, s_{i+1}) \in R$. Usaremos como convención que $x = (s_0, s_1, s_2, \ldots)$ denota un camino completo y que x^i denota el sufijo $(s_i, s_{i+1}, s_{i+2}, \ldots)$. Denotaremos con $M, s_0 \models p$ $(M, x \models p)$ para indicar que la fórmula de estado p

(fórmula de camino p) es cierta en la estructura M en el estado s_0 (en el camino x). Cuando se sobrentiende M, escribiremos únicamente $s \models p$. Definimos $\models$ por inducción de la siguiente forma:

S1 $s_0 \models p \iff p \in \mathcal{L}(s_0)$
S2 $s_0 \models p \wedge q \iff s_0 \models p \wedge s_0 \models q$
$s_0 \models \neg p \iff not(s_0 \models p)$
S3 $s_0 \models Ep \iff \exists$ camino completo $x = (s_0, s_1, s_2, \ldots)$ en $M, x \models p$
$s_0 \models Ap \iff \forall$ camino completo $x = (s_0, s_1, s_2, \ldots)$ en $M, x \models q$
P1 $x \models p \iff s_0 \models p$
P2 $x \models p \wedge q \iff x \models p \wedge x \models q$
$x \models \neg p \iff not(x \models p)$
P3 $x \models p\,\mathcal{U}\,q \iff \exists i[x^i \models q \wedge \forall j(j < i \rightarrow x^j \models p)]$
$x \models \bigcirc p \iff x^1 \models p$
P4 $x \models p\,\mathcal{U}^{[t,s]}\,q \iff \exists i, t \leq i \leq s$ tal que $x^i \models q$ y $\forall j, 0 \leq j < i, x^j \models p$
$x \models \bigcirc^{[t,s]} p \iff \exists i, a \leq i \leq b$ tal que $x^i \models p$
P5 $x \models \forall x f(x) \iff \forall n \in \mathbb{N}, x \models f(n)$
$x \models \forall x \in [a,b], f(x) \iff \forall a \leq n \leq b, x \models f(n)$
$x \models \exists x f(x) \iff \exists n \in \mathbb{N}$ tal que $x \models f(n)$
$x \models \exists x \in [a,b], f(x) \iff \exists a \leq n \leq b$ tal que $x \models f(n)$

donde x^i denota el sufijo de un camino x a partir del estado s_i.

B

Funciones de gestión de información temporal

El uso de RTAL para la especificación de un agente es útil para demostrar ciertas propiedades del agente y garantizar que se cumplen los objetivos de diseño.

Pero, a pesar de su potencia expresiva, no resultan intuitivos para programar sistemas complejos, especialmente cuando éstos tienen características de tiempo real. Otro problema es que los lenguajes lógicos no presentan utilidades básicas para la programación de sistemas de tiempo real, como gestión de concurrencia o definición de zonas en exclusión mutua.

Por esa razón, se ha optado por definir un lenguaje de usuario intuitivo y sencillo, que pueda utilizarse fácilmente desde lenguajes procedurales, como C, así como desde lenguajes de representación del conocimiento mediante reglas de inferencia, con sintaxis tipo CLIPS.

Las operaciones que se ejecutan sobre la información temporal asociada a las creencias del AA pueden clasificarse en tres categorías:

- inserción de nuevos hechos temporales
- borrado de hechos temporales que no son ciertos o corresponden a estados pasados
- consultas sobre la relación temporal existente entre dos puntos de tiempo.

Se hace una mención especial a la gestión de acciones pues, pese a que desde el punto de vista temporal son equivalentes a las predicciones sobre hechos temporales futuros, resultan importantes para la aplicación de la arquitectura de AA a nuevos tipos de problemas.

B.1. Inserción de hechos temporales

La propuesta que se realiza es la de unificar la forma de insertar la información temporal en las creencias del AA.

Se proporciona una única función para incluir valores *observables* y *no observables* [Onaindía, 1997]. En la definición del conjunto de creencias, el

usuario especifica qué atributos están conectados al exterior a través de un puerto de lectura. Así, cuando el sistema detecta que se emplea en una regla de inferencia este tipo de atributos, sabe que se encuentra asociado a un hecho temporal observable.

Otra mejora respecto a versiones previas es la posibilidad de establecer cualquier tipo de restricción temporal entre los hechos involucrados en una regla. Esta parte se encuentra detallada en el Apartado B.5

La única diferencia que se mantiene es la posibilidad de separar el tratamiento de valores actuales y predicciones. Esta distinción permite que el usuario trate de diferente forma la información que sabe con seguridad que es cierta —o que va a serlo en función de la temporalidad de los hechos de los que depende— de la información que es simplemente una posibilidad en el futuro y que dependerá de la evolución del sistema.

B.1.1. Valores seguros

La función `know()` se utiliza para insertar en un agente ARTIS información "segura", es decir, creencias del mundo real cuyo grado de certeza es del 100 %. Se trata de nuevos valores actuales o de predicciones que únicamente están a la espera de recibir la confirmación de su valor del exterior.

Si toda la información de la que depende son datos actuales, el nuevo hecho temporal será también un valor actual. Si, por el contrario, depende de algún dato pendiente de confirmar, se esperará a la llegada de ese dato antes de validar el nuevo hecho temporal. En ese caso, el nuevo valor es un valor futuro.

```
(know ?instancia (<slot> <valor>)+ [<rest>])
```

La función `know()` debe indicar el objeto al que hace referencia (puede ser una variable que se instancia en otra parte de la regla) y una o más parejas *(slot,valor)* que indican los valores que tendrán los slots correspondientes de la instancia. Opcionalmente, se puede incluir las restricciones temporales adicionales que se deben cumplir para que el nuevo hecho temporal se considere como válido.

No se va a diferenciar entre información observable y no observable desde el punto de vista del usuario, aunque esta distinción se siga realizando dentro de los procesos internos de gestión de información temporal.

B.1.2. Predicciones

Consideraremos una *predicción* aquella información de cuya certeza no estamos seguros. Se emplea para añadir como conocimiento una situación posible en el mundo actual. Al igual que con `know()`, se va a mantener la

sintaxis de `believe()` y la diferencia fundamental está en cómo se establecen las restricciones temporales (véase Sección B.5).

```
(believe ?instancia (<slot> <valor>)+ [<rest>])
```

Tal y como puede observarse, la sintaxis es exactamente la misma que hemos utilizado para `know()`: indicar la instancia sobre la que se realiza la predicción y después una o más parejas *slot-valor* para establecer la predicción del valor. De manera opcional, el usuario puede establecer una serie de restricciones temporales adicionales que limitan el instante de cumplimiento de la predicción.

Al igual que en el caso anterior, no se distingue entre datos *observables* y *no observables*, dejando al propio AA que determine el tipo de datos correspondiente.

B.2. Borrado de hechos temporales

El borrado de información temporal se produce por dos motivos. El más habitual es un proceso automático que se realiza por la propia evolución del sistema, al eliminar información obsoleta que es reemplazada por nuevos valores. El usuario no tiene control sobre este proceso.

Por otro lado, el usuario puede decidir eliminar de forma explícita un dato que ya no tiene validez. En ese caso, tiene disponible una función `retract()` que se detalla a continuación.

```
(retract ?tsv)
```

El borrado de hechos temporales nos permite eliminar información actual y futura que se encuentra almacenada en el conjunto de creencias del AA. En función del estado temporal del hecho que se desee eliminar, las acciones que se realizan son diferentes.

B.3. Consultas temporales

La consulta de la información temporal almacenada en las creencias del agente es de dos tipos:

1. determinar si un hecho temporal tiene cierto valor en un intervalo de tiempo dado
2. determinar la relación temporal existente entre dos puntos de tiempo

Para ello, se proporcionan dos predicados, que se emplean en la LHS, y sirven para chequear condiciones sobre el conjunto de creencias del AA: `hold()`, y `test()`, cada una de las cuales se encarga de uno de los tipos de consulta.

El predicado `hold()` evalúa la certeza de un hecho temporal. Utiliza la siguiente sintaxis:

```
(hold ?instancia (<slot> <valor>) [<rest>])
```

Este predicado comprueba en el conjunto de creencias del AA si el slot temporal al que hace referencia tiene el valor indicado. Si no se especifica ninguna restricción temporal, se compara con el valor actual del hecho temporal. En otro caso, se evalúan las restricciones y se comprueba si existe algún estado en las creencias del agente en el que se cumpla el hecho temporal en el intervalo temporal resultante de la aplicación de todas las restricciones.

El predicado `test()` permite realizar consultas acerca de la relación temporal existente entre dos puntos de tiempo o la ocurrencia de un hecho temporal tomando como referencia el instante de tiempo actual. Su sintaxis es:

```
(test BEGIN|END ?tf1 AFTER|BEFORE <distancia>
  BEGIN|END ?tf2|NOW)
```

donde las variables `?tf1` y `?tf2` corresponden a hechos temporales de las creencias del agente. Para resolver este tipo de consulta, debe examinarse el conjunto de creencias y buscar la relación temporal existente entre los puntos de tiempo. El predicado se evalúa a cierto si la relación entre los puntos de tiempo es más restrictiva que la solicitada.

Sean tp_1 y tp_2 dos puntos de tiempo cualesquiera. Las relaciones temporales que se pueden dar entre ellos son:

- tp_1 ocurra **antes** que tp_2
- tp_1 ocurra **después** de tp_2
- tp_1 ocurra **mientras** se dé tp_2
- tp_1 ocurra **hasta** que se dé tp_2

Muchas de estas relaciones pueden expresarse como combinaciones de varios `time-test` que hacen referencia a los instantes de inicio (`BEGIN`) y de finalización (`END`) de los hechos temporales involucrados. Sin embargo, es más cómodo para el usuario incrementar el número de relaciones temporales posibles (y no sólo `AFTER` y `BEFORE`).

La elección de estas relaciones temporales se ha realizado siguiendo los operadores básicos que proporciona la lógica CRTCTL. Mediante la composición de estos operadores pueden expresarse cualquier relación temporal entre puntos de tiempo. En el Apartado B.5 se expone con más detalle la especificación de restricciones temporales.

B.4. Propuesta de acciones

A pesar de que el manejo de acciones es análogo al de predicciones sobre valores futuros, resulta más intuitivo disponer de otra función específica para

la propuesta de acciones. Se incluye una función `try()` con la que se especifica la acción que se desea realizar y sus restricciones temporales. De esta manera, un agente planifica una serie de acciones en el futuro, pero antes de ejecutarlas comprueba que el estado actual del entorno sea el adecuado.

```
(try <accion> [<rest>]+)
```

La ejecución de esta función implica la creación de un nuevo hecho temporal futuro asociado a la acción. El cumplimiento del hecho temporal provocará la ejecución en el agente de la acción correspondiente, pues garantiza que en el entorno se dan las condiciones adecuadas.

B.5. Restricciones temporales explícitas

Además de las restricciones implícitas del propio modelo temporal, el usuario puede añadir restricciones temporales adicionales entre hechos temporales.

La forma de establecer restricciones temporales es única para todas las funciones de gestión de información temporal. Así, se unifica la gestión de información temporal para los agentes ARTIS.

Si se indica más de una restricción sobre un mismo hecho temporal, la restricción resultante se construye como la intersección de todos los intervalos temporales que resultan de la aplicación de cada restricción sobre el hecho temporal involucrado. El cálculo de los intervalos temporales y la determinación de la validez de las restricciones se detalla en el Apartado 4.5.3. Si no hay ningún intervalo temporal en el que el hecho sea válido, se rechazará. En ocasiones, ésto puede detectarse al especificar las reglas. En ese caso, el sistema debería avisar al usuario de que la regla nunca va a poder aplicarse.

B.5.1. Restricciones sobre el instante de inicio de un hecho temporal

Restringen el instante de tiempo a partir del cual puede comenzar a ser cierto el hecho temporal. Inicialmente, sólo se podían definir sobre las predicciones, pero se ha ampliado para que se puedan definir sobre cualquier tipo de dato temporal, incluyendo las acciones.

Se expresan mediante la siguiente sintaxis:

```
(BEGIN (?tf|LHS|NOW <min> [<max>])+ )
```

donde `?tf` representa un hecho temporal, `LHS` denota el instante de tiempo en el que se cumplen todos los hechos temporales de la parte izquierda de la regla en la que se encuentra, y `NOW` es una etiqueta con la que se representa el instante de tiempo actual.

B.5.2. Restricciones sobre el instante de finalización de un hecho temporal

Restringen el instante de tiempo en el que el hecho temporal deja de ser cierto. Inicialmente, sólo se definían sobre los hechos actuales o las predicciones pendientes sólo de confirmar del exterior. Al igual que en el caso anterior, su uso se ha ampliado para permitir definir este tipo de restricciones también sobre predicciones y acciones.

Se expresan empleando la siguiente sintaxis:

```
(END (?tf|LHS|NOW <min> [<max>])+ )
```

con la misma interpretación que para las restricciones sobre el instante de inicio.

B.5.3. Duración de los hechos temporales

La duración de un hecho temporal podría deducirse a partir de construcciones sobre el instante de finalización del hecho temporal. Sin embargo, resulta más intuitivo permitir directamente la definición de la duración del hecho temporal o de la acción, a través del concepto de *persistencia*.

La restricción

```
(PERSISTENCE <min> [<max>])
```

define la duración mínima y, de forma opcional, una duración máxima para el hecho temporal.

B.5.4. Restricciones que vinculan la validez del hecho temporal a la validez de otros hechos

La mayoría de estas nuevas restricciones pueden construirse mediante combinaciones de las restricciones anteriores. Sin embargo, aunque el lenguaje de usuario se complique algo más, mejora la claridad a la hora de especificar el comportamiento del agente.

- El nuevo hecho temporal es válido mientras se cumpla una condición (que puede ser todo lo complicada que sea necesario) o mientras un hecho temporal sea válido
  `(WHILE ?tf|<condición>)`
  Equivale a una restricción `(END (?tf 0 0))`
- El nuevo hecho temporal será válido hasta que se cumpla una condición o hasta que un hecho temporal deje de ser válido
  `(UNTIL <condición>)`
  En el momento en el que se cumpla la condición, el hecho temporal se convierte en un valor pasado automáticamente.

Referencias

Abadi et al., 1991. Abadi, M., Alpern, B., Apt, K., Francez, N., Katz, S., Lamport, L., and Schneider, F. (1991). Preserving liveness: Comments on "safety and liveness from a methodological point of view". *Information Processing Letters*.

Agre and Chapman, 1987. Agre, P. and Chapman, D. (1987). PENGI: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA.

Allen, 1983. Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Allen, 1991. Allen, J. (1991). Planning as temporal reasoning. In Allen, J. F., Fikes, R., and Sandewall, E., editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 3–14. Morgan Kaufmann, San Mateo, California.

Alpern et al., 1986. Alpern, B., Deemer, A., and Schneider, F. (1986). Safety without stuttering. *Information Processing Letters*, 23(4):177–180.

Alpern and Schneider, 1985. Alpern, B. and Schneider, F. (1985). Defining liveness. *Information Processing Letters*, 21(4):181–185.

Alpern and Schneider, 1986. Alpern, B. and Schneider, F. (1986). Recognizing safety and liveness. Tech. Rep. TR 86-727, Computer Science Department, Cornell University.

Alur and Henzinger, 1991. Alur, R. and Henzinger, T. (1991). Logics and Models of Real-Time: A Survey. In *Real Time: Theory in Practice*, volume 600, pages 74–106. Springer-Verlag.

Alur et al., 1997. Alur, R., Henzinger, T., and Kupferman, O. (1997). Alternating-time temporal logic. In *Proc. of 38th IEEE Symposium on Fundations of Computer Science*, pages 100–109, Florida.

Badaloni et al., 2002. Badaloni, S., Falda, M., and Giacomin, M. (2002). Qualitative and quantitative fuzzy temporal constraints. In *Proc. of AIIA '02*.

Barber, 2000. Barber, F. (2000). Reasoning on interval and point-based disjunctive metric constraints in temporal contexts. *Journal of Artificial Intelligence Research*, 12:35–86.

Barber et al., 1994. Barber, F., Botti, V., Onaindía, E., and Crespo, A. (1994). REAKT: An environment for real-time knowledge-based systems. *AI Communications*, 7(3):175–202.

Barro et al., 1994. Barro, S., Marín, R., Mira, J., and Patón, A. (1994). A model and a language for the fuzzy representation and handling of time. *Fuzzy Sets and Systems*, 61:153–175.

Bertoli et al., 2001. Bertoli, P., Cimatti, A., and Roveri, M. (2001). Heuristic search + symbolic model checking = efficient conformant planning. In Press, A., editor, *In Proc. 7 th International Joint Conference on Artificial Intelligence (IJCAI-01)*.

Botti et al., 1995. Botti, V., Barber, F., Crespo, A., Onaindía, E., García-Fornés, A., Ripoll, I., Gallardo, D., and Hernández, L. (1995). A temporal blackboard for a multi-agent environment. *Journal of Data & Knowledge Engineering*, 15(3).

Botti et al., 1999. Botti, V., Carrascosa, C., Julián, V., and Soler, J. (1999). Modelling agents in hard real-time environments. In *Proc. of MAAMAW'99*, volume 1647 of *LNAI*, pages 63–76. Springer-Verlag.

Botti and Hernández, 1998. Botti, V. and Hernández, L. (1998). Control in real-time multiagent systems. In *Proc. of II Workshop on Distributed Artificial Intelligece and Multiagent Systems*, pages 137–148.

Bratman et al., 1988. Bratman, M., Israel, D., and Pollack, M. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

Bresina and Drummond, 1990. Bresina, J. and Drummond, M. (1990). Integrating planning and reaction. In *Proc. of the 1990 Stanford Spring Symposium*.

Brooks, 1991a. Brooks, R. (1991a). Intelligence without reason. Technical Report A.I.Memo n. 1293, Massachusetts Institute of Technology.

Brooks, 1991b. Brooks, R. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.

Burmeister and Sundermeyer, 1991. Burmeister, B. and Sundermeyer, L. (1991). Cooperative problem solving guided by intentions and perceptions. In *Proc. of 3th European Worshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW'91)*, pages 217–230, Amsterdam. Elsevier Science.

Caire et al., 2001. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., and Massonet, P. (2001). Agent Oriented Analysis Using Message/UML. In *AOSE*, pages 119–135.

Calvanese et al., 2002. Calvanese, D., Giacomo, G. D., and Vardi, M. (2002). Reasoning about actions and planning in LTL action theories. In *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 593–602.

Carrascosa et al., 2003. Carrascosa, C., Julián, V., Rebollo, M., and Botti, V. (2003). Deliberative server for real-time agents. In *Conference Eastern Europe Multiagent Systems (CEEMAS '03)*, Lecture Notes in Artificial Intelligence, pages 485–496. Springer-Verlag.

Chellas, 1980. Chellas, B. (1980). *Modal Logic: An Introduction*. Cambridge University Press.

Cimatti et al., 1997. Cimatti, A., Giunchiglia, E., Giunchiglia, F., and Traverso, P. (1997). Planning via model checking. In *Proc. of the 4th Eur. Conf. on Planning (ECP'97)*.

Clancey, 1992. Clancey, W. (1992). Model constructions operators. *Artificial Intelligence*, 53(1):1–115.

Clarke et al., 1986. Clarke, E., Emerson, E., and Sistla, A. (1986). Research on automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. On Progr. Lang. And Syst.*, 2(8).

Clarke et al., 1996. Clarke, E., Enders, R., Filkorn, T., and Jha, S. (1996). Exploiting symmetry in temporal logic model checking. *Formal Methods In System Design*, 9:77–104.

Cohen et al., 1989. Cohen, P., Greenberg, M., Hart, D., and Howe, A. (1989). Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*.

Crespo et al., 1994. Crespo, A., Botti, V., Barber, F., Gallardo, D., and Onaindía, E. (1994). A temporal blackboard for real-time process control. *Engineering Applications of Artificial Intelligence*, 7(3):225–256.

Davis, 1998. Davis, R. (1998). What Are Artificial Intelligence? And Why? *AI Magazine*, 19(4):91–110.

Dean and McDermott, 1987. Dean, T. and McDermott, D. (1987). Temporal data base management. *Artificial Intelligence*, 32:1–55.

Dechter and Larkin, 2001. Dechter, R. and Larkin, D. (2001). Hybrid processing of beliefs and constraints. In *Proc. of Uncertainty in Artificial Intelligence (UAI '01)*, pages 112–119.

Dechter et al., 1991. Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraints networks. *Artificial Intelligence*, 49:61–95.

Dennet, 1987. Dennet, D. (1987). *The Intentional Stance*. The MIT Press.

d'Inverno et al., 1998. d'Inverno, M., Kinny, D., Luck, M., and Wooldridge, M. (1998). *A Formal Specification of dMars*, volume 1365 of *Intelligent Agents IV. Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Drummond, 1986. Drummond, M. (1986). *Reasoning About Actions and Plans*. Morgan Kauffman.

Dubois et al., 1996. Dubois, D., Fargier, H., and Prade, H. (1996). Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309.

Emerson, 1990. Emerson, E. (1990). *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*. Elsevier Science.

Emerson, 1995. Emerson, E. (1995). Automated temporal reasoning about reactive systems. In *Banff Higher Order Workshop*, pages 41–101.

Emerson et al., 1992. Emerson, E., Mok, A., Sistla, A., and Srinivasan, J. (1992). Quantitative temporal reasoning. *Real-Time Systems*, 4:331–352.

Emerson and Sistla, 1996. Emerson, E. and Sistla, A. (1996). Symmetry and model checking. *Formal Methods in System Design: An International Journal*, 9(1/2):105–131.

Emerson and Sistla, 1997. Emerson, E. and Sistla, A. (1997). Utilizing symmetry when model-checking under fairness assumptions: An automata-theoretic approach. *ACM Transactions on Programming Languages and Systems*, 19(4):617–638.

Emerson and Trefler, 1999. Emerson, E. and Trefler, R. (1999). Parametric quantitative temporal reasoning. In *Logic in Computer Science*, pages 336–343.

Etzioni, 1993. Etzioni, O. (1993). Intelligence without robots (a reply to brooks). *Artificial Intelligence*.

Fagin et al., 1995. Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (1995). *Reasoning About Knowledge*. The MIT Press, Cambridge, MA.

Ferguson, 1992. Ferguson, J. (1992). *TouringMachines: An architectures for dynamic, rational, mobile agents*. PhD thesis, TR 273, Univ. Cambridge, Cambridge, UK.

Francez, 1986. Francez, N. (1986). *Fairness*. Springer-Verlag.

García-Fornés, 1996. García-Fornés, A. (1996). *ARTIS: Un modelo y una arquitectura para sistemas de tiempo real inteligentes.* Tesis doctoral, Dept. Sistemas Informáticos y Computación. Univ. Politécnica Valencia.

García-Fornés et al., 1997. García-Fornés, A., Terrasa, A., Botti, V., and Crespo, A. (1997). Analyzing the schedulability of hard real-time artificial intelligence systems. *Engineering Applications of Artificial Intelligence*, pages 369–377.

Garrido and Barber, 2001. Garrido, A. and Barber, F. (2001). Integrating planning and scheduling. *Applied Artificial Intelligence.*

Garrido et al., 2001. Garrido, A., Onaindía, E., and Barber, F. (2001). Integración de planificación temporal y abstracción de recursos. In *Proc. de IX Conf. de la Asociación Española para la IA (CAEPIA'01).*

Georgeff and Lansky, 1987. Georgeff, M. and Lansky, A. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA.

Georgeff and Rao, 1991. Georgeff, M. and Rao, A. (1991). Modeling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, CA. Morgan Kaufmann.

Georgeff and Rao, 1995. Georgeff, M. and Rao, A. (1995). BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agents Systems (ICMAS-95)*, San Francisco, San Mateo, USA.

Gerevini et al., 1996. Gerevini, A., Perini, A., and Ricci, F. (1996). Incremental algorithms for managing temporal constraints. Tech. Rep. IRST-9605-07, IRST.

Godo and Vila, 1995. Godo, L. and Vila, L. (1995). Possibilistic temporal reasoning based on fuzzy temporal constraints. In *Proc. of IJCAI '95*, pages 1916–1922.

Goldman et al., 2001. Goldman, R., Musliner, D., and Krebsbach, K. (2001). Managing online self-adaptation in real-time environments. In *Proc. of Second International Workshop on Self Adaptive Software*, Balatonfured, Hungary.

Golumbic and Shamir, 1993. Golumbic, M. and Shamir, R. (1993). Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of ACM*, 40(5):1108–1133.

Graham, 2001. Graham, J. (2001). *Real-Time scheduling in distributed multiagent systems.* PhD thesis, Univ. Delaware, US.

Hanks et al., 1993. Hanks, S., Pollack, M., and Cohen, P. (1993). Benchmanks, testbeds, controlled experimentation and the design of agent architectures. *AI Magazine*, 14(4):17–42.

Haplern and Moses, 1992. Haplern, J. and Moses, Y. (1992). A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379.

Hart et al., 1968. Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minumun cost paths. *IEEE trans. Syst. Sci. Cybern.*, 4(2):100–107.

Hayes-Roth et al., 1983. Hayes-Roth, F., Waterman, D., and Lenat, D. (1983). *Building Expert Systems.* Addison-Wesley, New York.

Henao, 2001. Henao, M. (2001). *Consideraciones metodológicas para el desarrollo de sistemas inteligentes en tiempo real.* Tesis doctoral, Dept. Sistemas Informáticos y Computación. Univ. Politécnica de Valencia.

Hernández and Vivancos, 1998. Hernández, L. and Vivancos, E. (1998). Intelligent scheduling of production systems in a real-time architecture. In *Actas del II Congres Iberoamericano de Inteligencia Artificial (IBERAMIA '98)*, page 429.438.

Ishida, 1998. Ishida, T. (1998). Real-time search for autonomous agents and multiagent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(2):139–167.

Jonsson et al., 1999. Jonsson, P., Drakengren, T., and Bäckström, C. (1999). Computational complexity of relating time points with intervals. *Artificial Intelligence*, 109(1-2):273–295.

Julián, 2002. Julián, V. (2002). *Desarrollo de Sistemas Multiagente de Tiempo Real*. Tesis doctoral, Dept. Sistemas Informáticos y Computación. Univ. Politécnica de Valencia.

Julián and Botti, 2004. Julián, V. and Botti, V. (2004). Developing real-time multiagent systems. *Integrated Computer-Aided Engineering*, 11(2).

Julián et al., 2000. Julián, V., González, M., Rebollo, M., Carrascosa, C., and Botti, V. (2000). InSiDE: Una herramienta para el desarrollo de agentes ARTIS. In *Actas SEID 2000*, pages 79–88, Ourense.

Julián et al., 2002. Julián, V., Rebollo, M., Soler, J., Carrascosa, C., and Botti, V. (2002). SIMBA: An approach for real-time multiagent systems. In Springer-Verlag, editor, *Actas V Conferencia Catalana de Inteligencia Artificial (CCIA '02)*, Lecture Notes in Artificial Intelligence, No. 2504, pages 282–293, Castellón, Spain.

Kautz and Ladkin, 1991. Kautz, H. and Ladkin, P. (1991). Integrating metric and qualitative temporal reasoning. In *Proc. if AAAI-91*, pages 241–247, Anaheim, CA.

Korf, 1990. Korf, R. (1990). Real-time heuristic search. *Artificial Intelligence*, 42:189–211.

Kripke, 1963. Kripke, S. (1963). Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96.

Krokhin et al., 2001. Krokhin, A., Jeavons, P., and Jonsson, P. (2001). Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. Tech. Rep. PRG-RR-01-12, Computing Laboratory, Oxford University. Available from web.comlab.ox.ac.uk/oucl/publications/tr/rr-01-12.html.

Krokhin and Jonsson, 2002. Krokhin, A. and Jonsson, P. (2002). Extending the point algebra into the qualitative algebra. In *Proceedings of 9th International Symposium on Temporal Representation and Reasoning (TIME'02)*, pages 28–35, Manchester, UK.

Lamport, 1977. Lamport, L. (1977). Proving correctness of multiprocess programs. *IEEE Trans. on Software Engineering*, 2:125–143.

Lamport, 1984. Lamport, L. (1984). *Basic concepts*. Advance Course on Distributed Systemas. Lecture Notes in Computer Science. Springer-Verlag.

Lamport, 1993. Lamport, L. (1993). *Hybrid Systems in TLA+*. Hybrid Systems. Lecture Notes in Computer Science. Springer-Verlag.

Lamport, 1990. Lamport, L. (Apr., 1990). A Temporal Logic of Actions. Technical Report Research Report 79, DEC, Systems Research Center.

Ma and Knight, 2001. Ma, J. and Knight, B. (2001). Reified temporal logics: An overview. *Artificial Intelligence Review*, 15:189–217.

Manolios and Trefler, 2001. Manolios, P. and Trefler, R. (2001). Safety and liveness in branching time. In *Logic in Computer Science*, pages 366–.

McDermott, 1982. McDermott, D. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6(2):101–155.

Meiri, 1996. Meiri, I. (1996). Combining qualitative and quantitative constraints in temporal reasoning. In Dean, T. and McKeown, K., editors, *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 260–267, Menlo Park, California. AAAI Press.

Muller, 1996. Muller, J. (1996). *The Design of Intelligent Agents. A Layered Approach*. Lecture Notes in Artificial Intelligence, vol. 1177. Springer-Verlag, Berlin.

Musliner, 2002. Musliner, D. (2002). Safe learning in mission-critical domains: Time is of the essence. In *Working Notes of the AAAI Spring Symposium on Safe Learning Agents*, Stanford, California.

Newell, 1982. Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

Nii, 1986a. Nii, P. (1986a). Blackboard systems: Blackboard application systems, blackboard systems from a knowledge engineering perspective. *The AI Magazine*, pages 82–106.

Nii, 1986b. Nii, P. (1986b). Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine*, pages 38–53.

Nilsson, 1994. Nilsson, N. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158.

Nirkhe and Kraus, 1995. Nirkhe, M. and Kraus, S. (1995). Formal Real-Time Imagination. *Fundamenta Informaticae*, 23(2,3,4):371–390.

Nwana, 1996. Nwana, H. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244.

Onaindía, 1997. Onaindía, E. (1997). *Modelo de representación y razonamiento temporal para sistemas basados en el conocimiento de tiempo real*. Tesis doctoral, Dept. Sistemas Informáticos y Computación. Univ. Politécnica Valencia.

Onaindía and Rebollo, 1998. Onaindía, E. and Rebollo, M. (1998). Temporal representation and reasoning for dynamic environments. In Springer-Verlag, editor, *Proc. of IBERAMIA'98*, Lecture Notes in Artificial Intelligence, pages 207–218.

Ostroff, 1989. Ostroff, J. (1989). *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. Research Studies Press Limited (distributed by John Wiley and Sons), England.

Pearl, 1984. Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA.

Perlis et al., 1991. Perlis, D., Elgot-Drapkin, J., and Miller, M. (1991). Stop the World - I Want to Think. *International Journal of Intelligent Systems*, 6:443–456.

Pistore and Traverso, 2001. Pistore, M. and Traverso, P. (2001). Planning as model checking for extended goals in non-deterministic domains. In *IJCAI*, pages 479–486.

Pnueli, 1977. Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the Eighteenth Symposium on the Foundations of Computer Science*.

Pnueli, 1986. Pnueli, A. (1986). *Specification and development of reactive systems*. Information Processing. Elsevier Science, North Holland.

Real, 2000. Real, J. (2000). *Protocolos de cambio de modo para sistemas de tiempo real*. Tesis doctoral, Dept. de Informática de Sistemas y Computadores. Univ. Politécnica Valencia.

Rebollo and Onaindía, 1999. Rebollo, M. and Onaindía, E. (1999). Búsqueda limitada en el tiempo para razonamiento temporal. In García, A., Rizo, R., Moral, S., and Toledo, F., editors, *Proc. of CAEPIA'99*, volume 2, pages 44–53, Málaga, Spain.

Rebollo et al., 2003. Rebollo, M., Onaindía, E., and Botti, V. (2003). Formal modelling of dynamic environments for real-time agents. In *Conference Eastern Europe Multiagent Systems (CEEMAS '03)*, Lecture Notes in Artificial Intelligence, pages 475–484. Springer-Verlag.

Reichgelt, 1989. Reichgelt, H. (1989). A comparision of first-order and modal logic of time. In Jackson, P., Reichgelt, H., and van Harmelen, F., editors, *Logic-based Knowledge Representation*, pages 143–176. The MIT Press.

Rochenstein and Kaelbling, 1995. Rochenstein, S. and Kaelbling, I. (1995). A situated view of representation and control. *Artificial Intelligence*, 73:149–173.

Russell and Norvig, 1995. Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall International Editions.

Schreiber et al., 2000. Schreiber, A., Akkemans, J., Anjewierden, A., et al. (2000). *CommonKADS. Knowledge Engineering and Management*. MIT Press.

Shoham, 1987. Shoham, Y. (1987). *Reified Temporal Logics: Semantical and Ontological Considerations*. Advances in Artificial Intelligence - II. Elsevier Science Publishers, North Holland.

Shoham, 1988. Shoham, Y. (1988). *Reasoning about change*. The MIT Press.

Shoham, 1993. Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, 60:51–92.

Sistla, 1994. Sistla, A. (1994). Safety, liveness and fairness in temporal logics. *Formal Aspects of Computing*, 6(5):495–512.

Stankovic and Ramamritham, 1993. Stankovic, J. and Ramamritham, K. (1993). What is predictability for real time systems. Technical Report 1990-62, Dept. of Computer and Information Science. Univ. of Massachusetts.

Terrasa, 2000. Terrasa, A. (2000). *Flexible Real-Time Linux*. Tesis doctoral, Dept. Sistemas Informáticos y Computación. Univ. Politécnica Valencia.

Terrasa et al., 2002. Terrasa, A., García-Fornés, A., and Botti, V. (2002). Flexible real-time linux: A flexible hard real-time environment. *Real-Time Systems*, 22:151–173.

Thomson(Fr.) et al., 1990. Thomson(Fr.), Syseca(Fr.), CRIN(Fr.), GMV(Es.), Marconi(U.K.), Etnoteam(It.), and Computas-EX(No.) (1990). REAKT: Environment and Methodology for Real-Time Knowledge-Based Systems. European Community ESPRIT-II Project, n. 5146.

Thomson(Fr.) et al., 1993. Thomson(Fr.), Syseca(Fr.), CRIN(Fr.), GMV(Es.), Marconi(U.K.), Etnoteam(It.), and Computas-EX(No.) (1993). Knowledge data manager specification. Technical Report Deliverable D3.4.12, European Community ESPRIT-III Project, n. 7805.

Tormos et al., 2002. Tormos, P., Barber, F., and Lova, A. (2002). An integration model for planning and scheduling problems with constrained resources. In *Eighth International Workshop On Project Management And Scheduling (PMS '02)*, pages 354–358.

van Beek, 1990. van Beek, P. (1990). Reasoning about qualitative temporal information. In *Proc. of 8th National Conference on Artificial Intelligence*, volume 40, pages 728–734, Boston, Mass. AAAI Press.

van der Hoek and Wooldridge, 2002. van der Hoek, W. and Wooldridge, M. (2002). Tractable multiagent planning for epistemic goals. In *Proc. of the First International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 1167–1174, New York, USA. ACM Press.

van der Hoek and Wooldridge, 2003. van der Hoek, W. and Wooldridge, M. (2003). Cooperation, Knowledge and Time: Alternating-time Epistemic Logic and its Applications. *Studia Logica*, 75:125–157.

van Linder et al., 1998. van Linder, B., van der Hoek, W., and Meyer, J.-C. (1998). Formalizing abilities and opportunities of agents. *Fundamenta Informaticae*, 34(1-2):53–101.

Vardi, 1995. Vardi, M. (1995). An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266.

Vila, 1995. Vila, L. (1995). *On Temporal Representation and Reasoning in Knowledge-Based Systems.* Monografies de l'IIIA. Consell Superior d'Investigacions Científiques.

Vilain and Kautz, 1986. Vilain, M. and Kautz, H. (1986). Constraint propagation algorithms for temporal reasoning. In *Procc. of AAAI-86*, page 377.382, Philadelphia, PA.

Vilain et al., 1989. Vilain, M., Kautz, H., and van Beek, P. (1989). *Constraint propagation algorithms for temporal reasoning: A revised report.* Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann, San Mateo, Ca.

Weiss, 2000. Weiss, G., editor (2000). *Multiagent Systems. A modern approach to distributed A.I.* MIT Press.

Wooldridge, 1995. Wooldridge, M. (1995). Temporal belief logics for modelling distributed artificial intelligence systems. In O'Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence.* John Wiley & Sons.

Wooldridge and Jennings, 1995. Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.

Yampratoon and Allen, 1993. Yampratoon, E. and Allen, J. (1993). Performance of temporal reasoning systems. TRAINS Technical Note 93-1, Dept. of Computer Science, Rochester Univ.

Índice de autores

Giunchiglia, F. 68
GMV(Es.) 1, 61
Godo, L. 71
Goldman, R.P. 19
Golumbic, M.C. 75
Gomez, J. 28
González, M. 27
Graham, J.R. 22
Greenberg, M. 17, 65

Halpern, J.Y. 147
Hanks, S. 111
Haplern, J.Y. 64
Hart, D. 17, 65
Hart, P.E. 90
Hayes-Roth, F. 34
Henao, M. 1, 34, 148
Henzinger, T.A. 10, 66, 147
Hernández, L. 51, 61
Howe, A. 17, 65

Ishida, T. 90, 93
Israel, D.J. 29, 65

Jeavons, P. 62, 76
Jennings, N.R. 7
Jha, S. 50
Jonsson, P. 62, 69, 76
Julián, V. 1, 2, 7, 26–28, 50, 147

Kaelbling, I. 18
Katz, S. 30, 46
Kautz, H. 75, 76
Kearney, P.E. 28
Kinny, D. 65
Knight, B. 62
Korf, R.E. 90, 91
Kraus, S. VI
Krebsbach, K.D. 19
Kripke, S. 65
Krokhin, A. 62, 69, 76
Kupferman, O. 147

Ladkin, P.B. 76
Lamport, L. 30, 46, 47, 67, 69, 74
Lansky, A.L. 65
Larkin, D. 71
Leal, F. 28
Lenat, D.B. 34
Lova, A. 39
Luck, M. 65

Ma, J. 62
Manolios, P. 50, 51
Marconi(U.K.) 1, 61
Marín, R. 71
Massonet, P. 28
McDermott, D.V. 62, 73, 75, 77, 83
Meiri, I. 62, 75–78, 80, 81
Meyer, J.J.-Ch. 74
Miller, M. 73
Mira, J. 71
Mok, A.K. 68, 151
Moses, Y. 64, 147
Muller, J.P. 21, 29
Musliner, D. 19

Newell, A. 33
Nii, P. 2, 28
Nilsson, N. 15, 90
Nirkhe, M. VI
Norvig, P. 6, 35
Nwana, H.S. 8

Onaindía, E. 1, 11, 26, 39, 61, 70, 73, 79, 81, 95, 96, 98, 111, 121, 126, 155
Ostroff, J.S. 59

Patón, A. 71
Pavon, J. 28
Pearl, J. 75, 77, 91
Perini, A. 62
Perlis, D. 73
Pistore, M.o 68
Pnueli, A. 65
Pollack, M.E. 29, 65, 111
Prade, H. 71

Ramamritham, K. 2
Rao, A. 29, 65, 68
Raphael, B. 90
Real, J. 32
Rebollo, M. 7, 11, 26–28, 50, 61, 79, 81, 95, 96, 98, 111, 121, 147
Reichgelt, H. 71
Ricci, F. 62
Ripoll, I. 61
Rochenstein, S. 18
Roveri, M. 68

Russell, S. 6, 35

Schneider, F.B. 30, 46–48
Schreiber, A. 26, 34, 149
Shamir, R. 75
Shoham, Y. 6, 60, 62
Sistla, A.P. 47, 49–51, 68, 151
Soler, J. 2, 7, 26, 28, 147
Srinivasan, J. 68, 151
Stankovic, J.A. 2
Stark, J. 28
Sundermeyer, L. 84
Syseca(Fr.) 1, 61

Terrasa, A. 1, 26, 51
Thomson(Fr.) 1, 61
Tormos, P. 39
Traverso, P. 68
Trefler, R.J. 50, 51, 67, 69

van Beek, P. 75, 76, 89
van der Hoek, W. 68, 74, 147
van Linder, B. 74
Vardi, M.Y. 67, 68, 147
Vila, L. 69, 71
Vilain, M.B. 75, 76
Vivancos, E. 51

Waterman, D.A. 34
Wooldridge, M. 7, 64, 65, 68, 147

Yampratoon, E. 75

Índice alfabético