Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Carrascosa, C.; Terrasa, A.; Fabregat, J.; Botti, V.
Gestión de Comportamientos en Agentes de Tiempo Real
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 9, núm. 25, 2005, pp. 39-48
Asociación Española para la Inteligencia Artificial
Valencia, España

ÁREA ARTÍCULO

Gestión de Comportamientos en Agentes de Tiempo Real

C. Carrascosa, A. Terrasa, J. Fabregat, V. Botti

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 - Valencia
España
{carrasco,aterrasa,jfabregat,vbotti}@dsic.upv.es

Resumen

De la misma forma que los seres vivos siguen diferentes comportamientos cuando se enfrentan a estímulos diversos, algunos sistemas computacionales son capaces de beneficiarse del uso de computaciones distintas para cada situación significativa que afrontan. Por ejemplo, esta estrategia se lleva a cabo como "modos de funcionamiento" en el dominio de los Sistemas de Tiempo Real y como "roles" en la teoría de Sistemas Multi-Agente.

Este artículo presenta una extensión de la arquitectura de agente ARTIS que le permite al agente gestionar de forma explícita diferentes comportamientos. La extensión tiene tres aspectos principales: (1) la definición de comportamientos alternativos para el agente, (2) la instrucción del agente en las condiciones particulares que deben producir un cambio de comportamiento, y (3) los mecanismos que producen el cambio. Además, siendo la de agente ARTIS una arquitectura pensada para entornos de tiempo real estricto, los tres aspectos tienen que ser consistentes con las restricciones temporales estrictas del agente.

En general, la inclusión de la gestión de comportamientos hace que la arquitectura de agente ARTIS sea más apta para tratar con entornos más complejos y cambiantes, mejorando de forma efectiva las capacidades adaptativas del agente, aunque conservando sus características temporales.

Palabras clave: Agentes, Comportamientos, Tiempo Real.

1. Introducción

Los seres vivos suelen seguir diferentes patrones de comportamiento cuando afrontan estímulos distintos. Es obvio que a mayor complejidad de la criatura, más complejos son así mismo los comportamientos. Además, la habilidad de los organismos de realizar distintas tareas cuando tratan con entornos diferentes se convierte en un rasgo adaptativo para ellos. Por ejemplo, un depredador típico merodea hasta encontrar la presa, en ese momento todos sus movimientos se transforman en sutiles y sigilosos; de repente, cuando considera que es capaz de alcanzarla, salta sobre la presa y empieza una carrera para cazarla. Sería un desperdicio de recursos para el depredador ser sigiloso todo el tiempo, o mantener su velocidad máxima cuando merodea. La adaptación de la conducta es, por lo tanto, un rasgo clave para

el depredador, como en la mayoría de seres vivos.

Un agente puede también beneficiarse del uso de diferentes comportamientos. Por ejemplo, en el modelo B-Robot [12], el módulo de Coordinación Motora es responsable de elegir un algoritmo motor apropiado en función de la situación actual del agente. En concreto, según la importancia de la tarea, las propiedades del entorno y la precisión requerida, el módulo puede elegir entre una navegación lenta pero precisa basada en mapas, o un rápido aunque burdo método de "dead reckoning".

Otro caso de cambio de comportamiento, en este caso en el dominio de los Agentes / Sistemas Multi-Agente, es el de los llamados *cambios de rol*. En un entorno típico de comercio electrónico, un agente puede tanto vender como comprar, y en cada transacción puede cambiar entre estos dos comportamientos, con fines opuestos y tareas probablemente contradictorias.

Finalmente, otro dominio de la Informática en el que un sistema es capaz de cambiar el conjunto de tareas que tiene configurado inicialmente para ejecutar es el de los Sistemas de Tiempo Real, en el cual se ha acuñado el término *cambio de modo* para denotar esta habilidad. Desde la perspectiva de este dominio, un cambio de modo es una petición en tiempo de ejecución de cambiar el conjunto de tareas que deben ser ejecutadas por el sistema por un conjunto diferente, con los diferentes conjuntos preconfigurados y su garantía asegurada.

El paradigma de Agente/Multi-Agente y el dominio de los Sistemas de Tiempo Real tienen diferentes razones, y diferentes objetivos, a propósito de este cambio de comportamiento/rol/modo. Los sistemas de Agente y Multi-Agente están orientados a la consecución de sistemas de computación que se caractericen por comportamientos arbitrariamente complejos, imitando las acciones de los seres vivos y las comunidades. En estos dominios, la habilidad de modificar dinámicamente el comportamiento (o rol) aumenta la adaptabilidad del sistema, aumentando sus opciones de conseguir una mejor solución. Por otro lado, en un sistema de tiempo real definido por un conjunto de tareas cada una de ellas caracterizada por unos requisitos temporales (plazo máximo de ejecución, período, tiempo de ejecución en el peor caso, ...) [10], la habilidad de cambiar el modo de ejecución no mejora la solución, sino que dedica los recursos computacionales sólo a las tareas que son requeridas en ese momento. Esta estrategia permite que el sistema garantice separadamente la ejecución de las tareas que se necesitan en cada modo (mientras que probablemente no sería capaz de garantizarlas todas a la vez). Además, el principal interés en un sistema de tiempo real respecto a un cambio de modo, es poder realizarlo sin dejar de mantener las restricciones temporales del sistema en su conjunto.

El foco principal del trabajo descrito en este artículo es la introducción de una gestión explícita de comportamientos en una arquitectura de agentes de tiempo real llamada ARTIS ($\mathcal{AA}$) [2]. Por lo tanto, este trabajo intenta unir todos los objetivos propuestos anteriormente, esto es: definir comportamientos alternativos para un Agente ARTIS ($\mathcal{AA}$), instruir al agente en cuáles son las mejores condiciones para ejecutar cada comportamiento y realizar el cambio de comportamiento de forma que las restricciones temporales estrictas del agente siempre se cumplan, es decir, que se asegure la viabilidad del conjunto de tareas que componen el nuevo comportamiento al que se va a cambiar. En general, estas extensiones harán al $\mathcal{AA}$ más adaptativo a cambios en las condiciones del problema, incrementando de forma efectiva sus habilidades para encontrar soluciones válidas en entornos complejos y no deterministas.

El resto del artículo está estructurado de la siguiente forma: la Sección 2 introduce brevemente los Sistemas de Tiempo Real, centrándose en el concepto de cambio de modo. La Sección 3 presenta la arquitectura ARTIS tal y como está definida ahora, tras la introducción de los comportamientos. La Sección 4 explica en más detalle como las capacidades de meta-razonamiento del agente se han empleado tanto para instruir al agente sobre cuando realizar un cambio de comportamiento como para hacer el cambio durante la ejecución. La Sección 5 ilustra la extensión de ARTIS mediante el ejemplo de un agente con varios comportamientos. Finalmente, la Sección 6 presenta las conclusiones y algunas futuras líneas de trabajo futuras.

2. Sistemas de Tiempo Real

2.1. Definición

Un Sistema de Tiempo Real (STR) es un sistema en el que la corrección depende no sólo del resultado de la computación, sino también del ins-

tante en el que este resultado es producido [10]. En un STR, algunas tareas tienen plazos que definen el mayor intervalo de tiempo en el que el sistema puede dar el resultado. Si un resultado se obtiene después de este instante, probablemente no será útil. La característica principal de un STR no es estar contínuamente interconectado o ser el sistema más rápido, sino que un STR tiene que garantizar sus restricciones temporales y, al mismo tiempo, intentar cumplir sus objetivos. Hay dos tipos de STR [10]:

- *Sistemas de Tiempo Real Acríticos* ("Soft Real-Time Systems"): Ejecutar una tarea después de su *plazo* únicamente conlleva una pérdida de la calidad del resultado.
- *Sistemas de Tiemo Real Estrictos* ("Hard Real-Time Systems"): Ejecutar una tarea fuera de su *plazo* es completamente inútil, incluso puede conllevar graves consecuencias. Por esta razón es necesaria una garantía temporal completa de la ejecución del sistema. Esto normalmente se consigue por medio de un análisis estático de viabilidad o garantía. Este análisis comprueba las propiedades temporales de las tareas del sistema contra los recursos computacionales en el escenario del caso peor. Éste es, por lo tanto, un análisis muy pesimista, pero asegura la capacidad de las tareas de cumplir sus plazos bajo cualquier circunstancia.

Un agente de tiempo real (ATR) es un agente con restricciones temporales. Según la clasificación previa, hay agentes de tiempo real estricto y de tiempo real no estricto, según sus restricciones temporales (críticas o no).

2.2. Cambio de Modo

Debido al pesimismo del análisis de viabilidad o garantía en los sistemas de tiempo real estricto, la cantidad de tareas que un sistema puede garantizar puede estar muy limitado. Este hecho hace difícil afrontar problemas de gran complejidad en los que son necesarias muchas tareas.

Algunos problemas requieren que todas las tareas estén en ejecución en todo momento, pero otros pueden tener diferentes conjuntos de tareas en instantes distintos. En estos casos, una aproximación consiste en clasificar las tareas del sistema en diferentes modos de trabajo, de forma que sólo uno de esos modos pueda estar activo en un momento determinado.

Si se usa más de un modo, sin embargo, hay otro problema: cómo hacer un cambio de modo.En [7] se define un cambio de modo como una transición entre dos modos de trabajo en un sistema, cada uno con su propio conjunto de tareas y perfil de las tareas dadas.

El requisito principal para un cambio de modo en un sistema de tiempo real es ser viable, esto es, garantizar que ninguna tarea pierde su plazo a causa del cambio de modo. Por lo tanto, un sistema de tiempo real estricto que incorpore cambios de modo debe realizar un análisis de la viabilidad o garantía de las tareas en cada modo y de todas las posibles *transiciones* entre ellos.

Un Protocolo de Cambio de Modo es un mecanismo que el sistema tiene para realizar el cambio de modo. Hay dos tipos de protocolos: *Síncronos* y *Asíncronos*. Los protocolos síncronos retrasan el comienzo de todas las tareas en el nuevo modo hasta que todas las tareas del modo previo han finalizado, mientras que los protocolos asíncronos permiten la coexistencia de tareas de ambos modos durante la transición. La diferencia principal entre ellos es que los protocolos asíncronos producen transiciones más rápidas, pero a cambio son más complicados de implementar y garantizar.

Llegado este punto, cabe destacar a la hora de hablar de sistemas de tiempo real, que el principal problema que suelen tener los sistemas de tiempo real tradicionales es la falta de un entorno de desarrollo que facilite, por medio de un conjunto de abstracciones adecuado, el desarrollo de estos sistemas. Esto es parte de lo que se consigue de forma general con la aplicación de metodologías basadas en agentes, y en particular, por ejemplo, con la inclusión del concepto de comportamiento para abstraer el concepto de modo de funcionamiento.

3. Gestión de Comportamientos

Teniendo en cuenta que, por definición, un agente está situado en un entorno, se define el concepto de *situación de un agente* como la conjunción del estado del entorno junto con el estado interno del mismo.

Frente a una situación, el agente presentará un comportamiento compuesto por las tareas que le deben permitir conseguir sus objetivos en la situación actual. Así, el agente dispondrá de un conjunto de comportamientos de entre los cuales tan sólo habrá uno activo en un instante dado, que será el adecuado a la situación actual.

De esta manera, se define una *situación significativa de cambio* como aquella situación que al darse obliga al agente a cambiar de comportamiento para enfrentarse a ella.

Todos los cambios de comportamiento deben especificarse en un autómata finito no determinista (AFND), cuyos estados representan los distintos comportamientos definidos para el agente y cuyas transiciones modelan los cambios disponibles entre comportamientos (correspondiendo a situaciones significativas de cambio). Un autómata finito no determinista es necesario porque el agente tiene que ser capaz de tratar con entornos no deterministas. En estos escenarios no se puede construir de forma estática una secuencia de comportamientos única. Por el contrario, cuando un agente está en un comportamiento, hay un gran número de situaciones alternativas que pueden suceder, cada una de las cuales puede llevarnos a un cambio de comportamiento (transición) potencialmente distinto.

Teniendo en cuenta que cada comportamiento supone un conjunto de tareas a realizar, un proceso de razonamiento diferente, toda la gestión de cambios de comportamiento forma parte del proceso de meta-razonamiento propio del agente.

Llegado a este punto cabe subrayar la diferencia existente entre el concepto de *comportamiento* y el de *plan precompilado*. Un *plan* es una secuencia de acciones que llevan a un objetivo, mientras que en el caso de un *comportamiento* no existe una secuencia, sino un conjunto de tareas a realizar de forma continuada, que puede que ni siquiera tengan una finalización establecida.

4. Agente ARTIS ($\mathcal{AA}$)

Esta sección proporciona una breve descripción de la arquitectura de Agente ARTIS ($\mathcal{AA}$), una arquitectura de Agente de Tiempo Real estricto (una definición más detallada puede encontrarse en [2] [9] [3]), prestando especial atención a la incorporación y gestión de la abstracción de **comportamiento** en la arquitectura (que es el foco principal de este artículo).

De acuerdo con las taxonomías de agente existentes [13], la arquitectura de $\mathcal{AA}$ podría ser etiquetada como vertical, híbrida y específicamente diseñada para trabajar en entornos de tiempo real estricto [2].

Esta arquitectura garantiza una respuesta que satisfaga todas las restricciones temporales estrictas del agente mientras intenta obtener la mejor respuesta para el estado actual del entorno.

La arquitectura de un $\mathcal{AA}$ puede ser vista desde dos perspectivas distintas: el modelo de usuario (modelo de alto nivel) [2] y el modelo de sistema (modelo de bajo nivel o de implementación) [11]. El modelo de usuario ofrece la vista del desarrollador de la arquitectura, mientras que el modelo de sistema es el marco de ejecución utilizado para construir la versión ejecutable final del agente.

Ha sido desarrollada una herramienta gráfica llamada *InSiDE* [9] para traducir de la especificación del modelo de usuario al modelo de sistema. Esta herramienta permite al desarrollador definir el modelo de usuario de un $\mathcal{AA}$ y convertir este modelo al correspodiente modelo de sistema de forma automática [5]. El resultado es un $\mathcal{AA}$ ejecutable que se ejecuta sobre un sistema operativo de tiempo real.

4.1. Modelo de Usuario

Desde el punto de vista del **modelo de usuario**, la arquitectura de $\mathcal{AA}$ es una extensión del modelo de pizarra [6] que ha sido adaptado para trabajar en entornos de tiempo real estricto. Este modelo está formado por los siguientes elementos:

1. Un conjunto de **sensores** y **actuadores** que permiten al agente interaccionar con el entorno. Debido a las restricciones del entorno, la percepción y la acción en estos entornos suele estar temporalmente acotada.

2. Un conjunto de **comportamientos**. Los comportamientos modelan los modos alternativos de afrontar el entorno que están disponibles para el agente. Durante la ejecución, un $\mathcal{AA}$ siempre está en un comportamiento, y sólo en uno, llamado comportamiento activo.

En un $\mathcal{AA}$, cada comportamiento está compuesto de un conjunto de **in-agents**, cada uno de los cuales trata con una parte del entorno (o resuelve una parte del problema) que el agente afronta. La razón principal para partir el método de resolución de problema en partes más pequeñas es proporcionar una abstracción que organice el conocimiento de resolución de problemas de forma modular y gradual.

Cada **in-agent** periódicamente realiza una actividad específica relacionada con la resolución de un subproblema particular, de tal forma que todos los **in-agents** (dentro de un comportamiento) cooperan para resolver el problema completo. La cooperación se consigue principalmente compartiendo los resultados calculados por los diferentes in-agents mediante una memoria global (explicada más adelante).

Los **in-agents** que conforman los distintos comportamientos de AA se clasifican en aquellos que se encargan de resolver las partes esenciales del problema del agente (y que aseguran alcanzar una solución de mínima calidad que mantiene el problema bajo control) y aquellos que son opcionales (que tratan de mejorar la respuesta conseguida por los esenciales o se encargan de partes del problema que no son estrictamente necesarias). Dentro de la arquitectura de AA se distingue entre estos dos tipos de **in-agents** según si tienen o no asociadas restricciones temporales estrictas, con lo que un **in-agent** que sea esencial se diseñará como un **in-agent** *crítico* y uno que sea opcional lo hará como uno *acrítico*. Cabe destacar que aunque los **in-agents** acríticos no tienen su ejecución garantizada, el agente intentará ejecutar tantos de ellos como sea posible con el objetivo de maximizar la calidad de la solución global.

Los **in-agents** exhiben algunas características temporales y funcionales que ahora se explicarán. Estas características no son las mismas en los **in-agents** críticos y acríticos.

Desde un punto de vista temporal, un **in-agent crítico** se caracteriza por su periodo y su plazo. De esta forma, el tiempo disponible para que un **in-agent** obtenga una respuesta válida está acotado de forma estricta entre el momento en el que es (periódicamente) lanzado y su plazo. En este intervalo, el **in-agent** ha de proporcionar una respuesta lo bastante buena a su subproblema, dada la situación actual del entorno.

Desde un punto de vista funcional, un **in-agent** crítico está formado por dos capas (ver Figura 4.1):

- La *capa refleja*: asegura una calidad mínima de la respuesta en un tiempo de ejecución garantizado. Tal y como se explicaba en un apartado previo de este artículo, un test de garantía estático asegura que la capa refleja de todos los in-agents críticos (en *cada uno* de los posibles comportamientos) tendrá suficientes recursos computacionales para completarse antes de su plazo. Las capas reflejas de todos los **in-agents** en el comportamiento actual conforman la capa obligatoria actual del $\mathcal{AA}$.
- La *capa deliberativa de tiempo real*: intenta mejorar la respuesta conseguida por la capa refleja. En ejecución, el agente usa todo el tiempo libre de procesador para ejecutar la capa deliberativa de tantos **in-agents** como pueda. Las capas deliberativas de tiempo real de todos los in-agents del comportamiento actual conforman la capa opcional actual del $\mathcal{AA}$.

Por otro lado, un **in-agent acrítico** está caracterizado temporalmente por un periodo, pero no tiene un plazo, Funcionalmente, este in-agent consiste solo en una capa deliberativa.

3. Un conjunto de **creencias** que comprende un modelo del mundo (con todo el conocimiento del dominio que es relevante al agente) y el estado interno, esto es, los estados mentales del agente. Este conjunto está almacenado en una pizarra basada en *frames* [1] que es accesible por todos los **in-agents**. Es posible especificar que algunos cambios significativos en una creencia pueden producir un evento en ejecución.

4. Un **módulo de control** que es responsable de la ejecución en tiempo real de los **in-agents** que pertenecen a la conducta actual del $\mathcal{AA}$. Los requisitos temporales de las dos capas en cada **in-agent** (refleja y deliberativa) son diferentes. Por lo tanto el Módulo de Control ha de emplear diferentes criterios de ejecución para

cada una. De hecho, el Módulo de Control está dividido en dos submódulos [3], *Servidor Reflejo (Reflex Server –RS–)* y *Servidor Deliberativo (Deliberative Server –DS–)*, a cargo de las partes reflejas y deliberativas del $\mathcal{AA}$ respectivamente. Las dos partes del Módulo de Control trabajan de forma coordinada y coherente con el análisis de viabilidad o garantía.

El Módulo de Control incorpora un proceso de *Meta-Razonamiento* [4] para adaptar el proceso de razonamiento del $\mathcal{AA}$ a situaciones cambiantes. La especificación de este proceso es la única parte del Módulo de Control que es dependiente de la aplicación.

La especificación está compuesta por un conjunto de meta-reglas escritas por el diseñador del $\mathcal{AA}$ en un lenguaje diseñado para esta función (que será explicado en la sección 5).

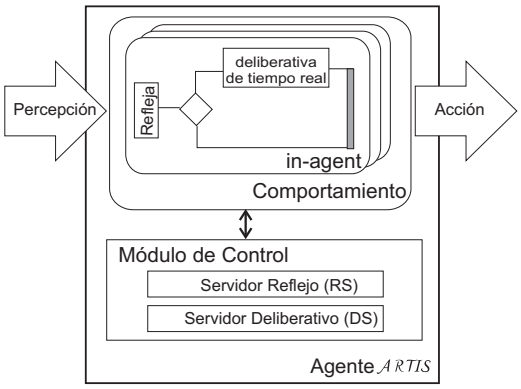

Figura 3.1. Arquitectura de Agente ARTIS: Modelo de Usuario

Es necesario destacar que una vez completada la especificación de los elementos antes mencionados, es necesario validarla para que sea considerada una especificación del modelo de usuario de un $\mathcal{AA}$. Esta validación debe garantizar las restricciones de tiempo real del agente. Esto se lleva a cabo por medio de un análisis estático de la viabilidad de la especificación (de cada uno de los diferentes comportamientos así como de sus transiciones). Este análisis, descrito en [5], está basado en técnicas de análisis de la viabilidad o garantía bien conocidas en la comunidad de Sistemas de Tiempo Real.

En este punto se han presentado las definiciones de los diferentes comportamientos del agente. La instrucción del agente acerca de qué condiciones deberían producir un cambio de comportamiento ha sido introducida (aunque se detallará en la sección 5). Las siguientes secciones introducirán los mecanismos reales que llevan a cabo este cambio.

4.2. Modelo del Sistema

El **modelo de sistema** proporciona la arquitectura software por la que el $\mathcal{AA}$ implementa todas las características de alto nivel expresadas en el modelo de usuario. Los principales elementos de este modelo (traduciendo las partes correspondientes del modelo de usuario) son [5]:

1. Una biblioteca de acceso a los dispositivos hardware y/o los propios dispositivos hardware. Esto corresponde a los sensores y actuadores expresados en el modelo de usuario.

2. Un conjunto de modos de funcionamiento que, junto con el Protocolo de Cambio de Modo, implementan la gestión de comportamientos expresada en el modelo de usuario.

   El concepto de modo de funcionamiento tiene que estar basado en un modelo concreto de tareas que garantice las restricciones temporales críticas del entorno. Por lo tanto, los **in-agents** del modelo de usuario se traducen en las **tareas** del modelo del sistema. Cada modo de funcionamiento tendrá las tareas correspondientes a los in-agents definidos en el comportamiento relacionado a ese modo.

   De acuerdo con este modelo de tareas, cada tarea puede tener tres partes:

   - Inicial: Está encargada de comprobar el estado del entorno respecto al subproblema que conoce. También calcula la primera respuesta a su problema, (probablemente con una calidad baja), pero en tiempo acotado. Por lo tanto, es la traducción de la capa refleja de su **in-agent** correspondiente.
   - Opcional: Incrementa la calidad de la respuesta calculada por la parte inicial, pero esta mejora puede usar métodos no acotados temporalmente. De esta forma, es la traducción de la capa deliberativa en tiempo real de su **in-agent** correspondiente.

- Final: Lleva a cabo la mejor respuesta calculada por las partes inicial u opcional.

Según este modelo, sólo las partes iniciales y finales de las tareas críticas (las que se traducen a partir de los **in-agents** críticos) tendrán restricciones estrictas de tiempo real y serán también las que se encarguen de la interacción con el entorno (por medio de los sensores y actuadores).

La implementación actual de la arquitectura de $\mathcal{AA}$ usa el **protocolo síncrono de cambio de modo** (definido en la sección 2.2). Este tipo de cambio de modo hace más fácil la transición entre modos porque las tareas pertenecientes a modos diferentes no pueden estar activas al mismo tiempo.

3. Una memoria compartida (implementación del modelo de pizarra) con todos los datos correspondientes a las diferentes creencias especificadas en el modelo de usuario. Esta memoria puede ser accedida por todas las tareas.

4. Los dos módulos del Módulo de Control del modelo de usuario son aquí dos entidades separadas [3]:

   - El *RS* incluye el planificador de tareas de tiempo real o Planificador de Primer Nivel (*First Level Scheduler* – FLS–). El FLS usa una política de planificación de tiempo real que, en ejecución, decide qué tarea es la siguiente que ha de ser ejecutada a continuación. Esta política es compatible con el test de garantía estático. Tener un planificador en tiempo de ejecución ayuda a que el $\mathcal{AA}$ se adapte a los cambios en el entorno, así como a beneficiarse de las tareas que usan menos tiempo que su tiempo en el caso peor.

     El FLS también implementa el protocolo de cambio de modo introducido al $\mathcal{AA}$, permitiendo que cambie de modo de funcionamiento según las transiciones especificadas en el modelo de usuario. Estos cambios son activados por mecanismos específicos en ejecución que están explicados en la sección 5.

     Un método de extracción de holgura (basado en una técnica específica de la comunidad de tiempo real llamada algoritmo de extracción de holgura *"slack stealing algorithm"*[8]) para calcular en tiempo de ejecución el tiempo disponible para que el DS ejecute partes opcionales pertenecientes al modo de funcionamiento actual.

   - El *DS* incluye al Planificador de Segundo Nivel (*Second Level Scheduler* – SLS–). Este planificador se encarga de ejecutar las partes opcionales de las tareas activas en el modo actual. De esta forma, en los intervalos de holgura, el SLS aplicará una política de planificación que atendiendo a criterios de calidad elegirá la siguiente parte opcional a ejecutar.

   La gestión del Meta-Razonamiento, que será comentada en la siguiente sección, también ha sido incluida en las obligaciones de los módulos *DS* y *RS*.

Aunque no tiene una contrapartida específica en el modelo de usuario, el modelo de sistema también incluye un análisis estático de la viabilidad o garantía de los diferentes modos de funcionamiento. Es importante destacar que este análisis no construye un plan que comprenda la secuencia de ejecución de las tareas. Por el contrario, el análisis solo asegura la capacidad del planificador en tiempo de ejecución (FLS) de ejecutar las tareas de tiempo real garantizando sus plazos.

5. Gestión del Meta - Razonamiento

5.1. Lenguaje de Meta-Reglas

Como se ha comentado en apartados anteriores, hay un proceso de Meta-Razonamiento integrado en el Módulo de Control del $\mathcal{AA}$. Las capacidades de Meta-Razonamiento del $\mathcal{AA}$ están especificadas por medio del llamado *Lenguaje de Meta-Reglas* en el modelo de usuario. Este lenguaje ha sido definido para instruir al $\mathcal{AA}$ en las condiciones particulares que deberían producir algunas acciones de meta-razonamiento.

De entre las diferentes acciones que componen el modelo de meta- razonamiento, cabe destacar el cambio de comportamiento. El conjunto de las

meta-reglas cuyas acciones comprenden un cambio de comportamiento forman las transiciones en el autómata finito no determinista de los comportamientos. En particular, el desarrollador debe especificar una meta-regla para cada una de las diferentes transiciones del autómata.

Las meta-reglas se agrupan en conjuntos llamados *Focos de Atención*, permitiendo la activación (y desactivación) de grupos de reglas en tiempo de ejecución.

La sintaxis de un foco de atención sería:

```
(AttentionFocus <AFName>
  (defMetaRule <Name> <Importance>
    (<EventType> <slotId>)
    (<condition>)*
    =>
    (<action>)+
  )+
)
```

El esquema de una meta-regla previo nos muestra que las meta-reglas se activan para responder a eventos significativos (normalmente asociados a cambios del conjunto de creencias). Además, la regla puede tener condiciones adicionales que han de ser satisfechas antes de ejecutar sus acciones asociadas.

Relacionado con la activación de las meta-reglas, aunque se especifica en el lenguaje de frames (creencias), cabe destacar que un evento tiene asociada una importancia, en base a la cual se ordenan para su respuesta. Además existe también una importancia asociada a cada meta-regla para priorizar entre meta-reglas activadas por el mismo evento y pertenecientes al mismo foco de atención.

Existe un grupo de acciones posibles de meta-razonamiento entre las que se encuentran la activación de un comportamiento y la activación y desactivación de un foco de atención. Cuando se activa un foco de atención se le asocia un grado de atención al foco activado.

5.2. Gestión de Eventos

Como parte del modelo de sistema, las meta-reglas son almacenadas en una memoria compartida accedida únicamente por el Módulo de Control.

El Módulo de Control puede dedicar parte de su tiempo a la gestión de eventos (ver [3] para más detalles), esto es, a responder a eventos pendientes en orden de importancia decreciente. Para cada uno de estos eventos pendientes, busca en el almacén de meta-reglas la más importante entre las meta-reglas del foco de atención activo con mayor grado de atención que este evento active y cuya condición sea también satisfecha.

Esta aproximación permite al diseñador especificar el proceso de meta-razonamiento en tiempo de diseño de forma que permita al $\mathcal{AA}$ adaptarse a cambios significativos en la situación, donde esta situación está compuesta no sólo del entorno sino también del estado interno.

6. Ejemplo: Robot Cartero

Se ha diseñado un Agente ARTIS para controlar un robot Pioneer2 que entrega el correo en un entorno de oficinas conocido. Esta funcionalidad permite que visite las habitaciones donde tiene una carta para repartir, teniendo en cuenta las urgentes y los plazos y tratando de escoger el mejor camino.

En la solución diseñada, el $\mathcal{AA}$ está compuesto de cuatro comportamientos útiles para la resolución del problema:

- *Espera de Objetivos (EdO).* Este comportamiento es el de inicialización, y también el empleado cuando se ha acabado de repartir todo el correo. Cuando está esperando correo nuevo, el agente sólo monitoriza el nivel de carga de baterías y espera a que lleguen nuevas órdenes.
- *Reparto de Cartas (RdC).* Es el comportamiento que sigue el $\mathcal{AA}$ cuando tiene correo que entregar. Planifica la ruta, visita las oficinas, evita los obstáculos y vigila el nivel de carga y el correo nuevo. Es el comportamiento más complejo del agente, y también el que se usa con más frecuencia.
- *Entrega de una Carta (EdC).* Este es el comportamiento a seguir cuando un agente llega a un despacho, el comportamiento a seguir es la entrega de correo. Si la puerta está cerrada, llama a la puerta chocando contra ella dos veces. El comportamiento es prácticamente el mismo que el anterior pero no esquivamos obstáculos, pues pretendemos chocar.

- *Vuelta a la base para Recargar baterías (VpR).* Si el nivel de carga de baterías baja por debajo de un umbral, deja todo el trabajo y vuelve a casa a recargar.

Todos estos comportamientos y los cambios entre ellos comprenden el AFND de la figura 6.

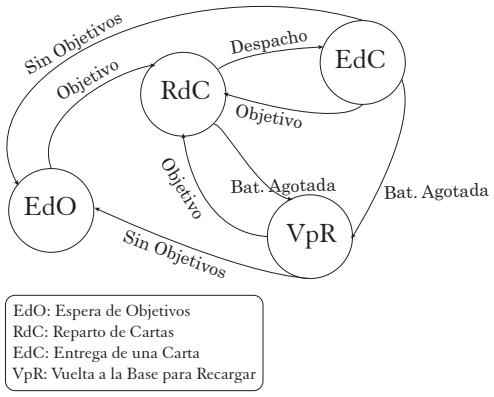

Figura 5. Autómata Finito No Determinista de Comportamientos

Como ejemplo del uso de las meta-reglas, la transición entre *Reparto de Cartas* y *Entrega de una Carta* hacia el estado de *Vuelta a la base para Recargar baterías* significa que, si durante la entrega o reparto el nivel de baterías baja por debajo de un umbral, el robot ha de volver a la base para recargar. La meta-regla asociada a estas transiciones se escribirá así:

```
(AttentionFocus Emergency
  (defMetaRule BatteryLevelChecker 0
    (MODIFICATION battery.current.level)
    (battery.current.level < 10)
    =>
    (DeActivateFocus Emergency)
    (SetBehaviour VpR)))
```

Esto significa que un Foco de Atención está definido para tratar con las situaciones de emergencia, en el interior del cual ponemos una regla. Tras el nombre de la regla (*BatteryLevelChecker*) viene la prioridad, cero, por lo tanto, máxima. El evento del KDM que la regla enlaza es la modificación del *slot* donde se almacena el nivel de batería. La parte izquierda de la regla comprueba si está por debajo del umbral. Si la regla satisface su parte izquierda, el foco de atención Emergency se desactiva y el comportamiento cambia a *Vuelta a la base para Recargar baterías.*

7. Conclusiones

En este artículo se ha presentado una extensión del $\mathcal{AA}$ para gestionar diferentes comportamientos. Esta extensión ha supuesto un gran cambio en toda la estructura y niveles de abstracción del $\mathcal{AA}$. Por lo tanto, cambia el modelo de usuario y la forma de diseñar el agente. Cambia también el modelo de sistema incorporando la gestión de cambios de modo.

La gestión de comportamientos se instruye por medio de un lenguaje de meta-reglas que permite al agente adaptarse a cambios significativos de la situación, siendo una forma de adaptación el cambio de la conducta actual del agente.

La extensión presentada ha catapultado la capacidad adaptativa de la arquitectura de $\mathcal{AA}$ a otro nivel, permitiéndole enfrentarse a problemas con un grado de complejidad imposible de abarcar con la aproximación anterior.

Por otro lado, aunque hoy en día las meta-reglas se escriben en tiempo de diseño, son traducidas a estructuras de datos preparadas para incorporar en el futuro un algoritmo de aprendizaje capaz de aprender (y olvidar) meta-reglas.

Agradecimientos

Este trabajo está parcialmente financiado por los proyectos DPI2002-04434-C04-02, TIC2003-07369-C02-01 y GV04B-513 del gobierno español.

Referencias

[1] F. Barber, V. Botti, E. Onaindía, and A. Crespo. Temporal reasoning in reakt: An environment for real-time knowledge-based systems. *AICOMM*, 7(3):175–202, 1994.

[2] V. Botti, C. Carrascosa, V. Julian, and J. Soler. Modelling agents in hard real-time environments. In *MAAMAW'99 Proceedings*, volume 1647 of *LNAI*, pages 63–76. Springer-Verlag, 1999.

[3] Carlos Carrascosa, Miguel Rebollo, Vicente Julián, and Vicente Botti. Deliberative server for real-time agents. In *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *LNAI*, pages 485–496. Springer, 2003.

[4] Carlos Carrascosa Casamayor. *Meta-Razonamiento en Agentes con Restricciones Temporaltes Críticas*. PhD thesis, Dept. Sistemas Informaticos y Computacion. Univ. Politecnica Valencia, 2004.

[5] A. Garcia-Fornes, A. Terrasa, V. Botti, and A. Crespo. Analyzing the schedulability of hard real-time artificial intelligence systems. *Engineering Applications of Artificial Intelligence*, pages 369–377, 1997.

[6] P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine*, pages 38–53, Summer 1986.

[7] J. V. Real. *Protocolos de Cambio de Modo Para Sistemas de Tiempo Real*. PhD thesis, Departamento de Informática de Sistemas y Computadores. Universidad Politécnica de Valencia, Enero 2000.

[8] Davis R.I, Tindell K.W., and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceedings of the Real-Time Systems Symposium*, pages 222–231. IEEE Computer Society Press, 1993.

[9] J. Soler, V. Julian, C. Carrascosa, and V. Botti. Applying the artis agent architecture to mobile robot control. In *Proceedings of IBERAMIA'2000. Atibaia, Sao Paulo, Brasil*, volume I, pages 359– 368. Springer Verlag, 2000.

[10] J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 12(10):10–19, 1988.

[11] A. Terrasa, A. García-Fornes, and V. Botti. Flexible real-time linux. *Real-Time Systems Journal*, 2:149–170, 2002.

[12] S. Uckun and B. Hayes-Roth. A control architecture for intelligent mobile robots. Technical Report KSL-93-09, Knowledge Systems Laboratory. Stanford University, January 1993.

[13] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.