UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

TESIS DOCTORAL

Heurísticas para el control deliberativo en una arquitectura de agentes inteligentes de tiempo real

Luis Hernández López

Directores: Dr. Vicente J. Botti Navarro

Dra. Ana García Fornes

Valencia, abril de 2004.

A mis padres y hermanos.

”Efectuar elecciones viables se produce en un crisol de errores informativos. Así acepta la Inteligencia la fiabilidad. Y cuando no son conocidas las elecciones absolutas (infalibles), la Inteligencia corre sus riesgos con los datos limitados que posee en una arena donde los errores no sólo son posibles sino necesarios.
-Darwi Odrade”.
Casa capitular: Dune. Frank Herbert.

"La calidad nunca es un accidente;
siempre es el resultado de un esfuerzo
de la inteligencia."
John Ruskin

Reconocimientos

Quiero agradecer este trabajo en primer lugar a mis directores, Vicente Botti y Ana García Fornes, por su dedicación y ayuda para poder realizar este trabajo.

También mi agradecimiento a todos los miembros del grupo al que pertenezco, el grupo de Tecnología Informática e Inteligencia Artificial (GTI-IA). Especialmente quiero destacar a Andrés Terrasa e Ignacio Pachés por su trabajo en la arquitectura ARTIS sin el cual no podría haberse llevado a cabo este trabajo. También a Carlos Carrascosa por su trabajo en el módulo de control de la arquitectura ARTIS.

Resumen

El área de la Inteligencia Artificial está experimentando un gran avance en los últimos tiempos con su aplicación a un mayor número de campos diferentes. Uno de ellos es el de los problemas de tiempo real. Problemas donde no sólo es importante la lógica del cálculo de las soluciones, sino también el instante de tiempo en que son calculadas dichas soluciones.

Este acercamiento entre ambas áreas es, en principio, provechoso, pues la Inteligencia Artificial puede aportar nuevas posibilidades a los sistemas de tiempo real, como una mayor flexibilidad de adaptación a entornos complejos y dinámicos. Sin embargo esta aproximación ha presentado desde siempre importantes dificultades.

Principalmente los sistemas de tiempo real poseen unos requerimientos temporales (predecibilidad de los tiempos de respuesta principalmente) que no suelen ser habituales en las técnicas de Inteligencia Artificial.

Entre otras formas de abordar este problema, está el desarrollo de arquitecturas software para el diseño de agentes inteligentes para su uso en entornos de tiempo real. Estas arquitecturas poseen diferentes mecanismos para que los agentes construidos puedan trabajar en entornos de tiempo real ofreciendo comportamientos reactivos (para cumplir los requerimientos temporales) y deliberativos (que hacen uso de técnicas de Inteligencia Artificial para conseguir mejores prestaciones).

Una de estas arquitecturas es ARTIS. Esta arquitectura hace uso de una planificación de sus tareas a dos niveles para conseguir cumplir sus objetivos. Por un lado un planificador de primer nivel garantiza la obtención de respuestas dentro de límites temporales estrictos. Un planificador de segundo nivel se encarga del control de componentes que mejoran la calidad de los resultados.

El trabajo presentado se centra en este segundo planificador, desarrollando dos heurísticas, SSS (slack-slide scheduling, scheduling por desplazamiento del slack) y SSSM (SSS con Memoria) que utilizando de forma más precisa las características de la arquitectura y de los

componentes a planificar obtiene mejores calidades en las respuestas obtenidas que otras heurísticas aplicables.

Estas heurísticas son capaces de manejar los diferentes tipos de algoritmos que usa la arquitectura (de refinamiento progresivo y de métodos múltiples) y una de ellas, la SSSM, usa los recursos de la arquitectura para aprovechar resultados previos mejorando la eficiencia obtenida.

Finalmente se presenta pruebas estadísticas para verificar las características de las heurísticas así como dos aplicaciones reales para mostrar su viabilidad.

Resum

L'àrea de la Intel·ligència Artificial està experimentant un gran avanç als darrers anys amb la seua aplicació a un gran nombre de camps diferents. Un d'aquests camps és el dels problemes de temps real. Problemes on no solament és important la lògica del càlcul de les solucions, sinó també l'instant temporal on les solucions són calculades.

Aquest acostament entre les dues àrees és en principi profitós, ja que la Intel·ligència Artificial pot aportar noves possibilitats als sistemes de temps real, com per eixemple una major flexibilitat d'adaptació a entorns complexos i dinàmics. No obstant això, aquesta aproximació ha presentat des de sempre importants dificultats.

Principalment, els sistemes de temps real tenen uns requeriments temporals (predecibilitat dels temps de resposta principalment) que no acostumen a ser habituals en les tècniques d'Intel·ligència Artificial.

Entre altres formes d'abordar aquest problema, està el desenvolupament d'arquitectures software per al disseny d'agents intel·ligents pel seu ús en entorns de temps real. Aquestes arquitectures tenen diferents mecanismes per a que els agents construïts treballen en entorns de temps real oferint comportaments reactius (per a complir els requeriments temporals) i deliberatius (que fan ús de tècniques d'Intel·ligència Artificial per a aconseguir millors prestacions).

Una d'aquestes arquitectures es ARTIS. Aquesta arquitectura fa ús d'una planificació de les seves tasques a dos nivells per a aconseguir complir els seus objectius. Per una banda, el planificador de primer nivell garanteix l'obtenció de respostes dins de límits temporals estrictes. Un planificador de segon nivell s'encarrega del control de components que milloren la qualitat dels resultats.

El treball presentat es centra en aquest segon planificador, desenvolupant dues tècniques SSS (slack-slide scheduling, scheduling per desplaçament del slack) i SSSM (SSS amb Memòria) que utilitzant de forma més precisa les característiques de l'arquitectura i dels components a planificar obté

millors qualitats en les respostes obtingudes que altres heurístiques aplicables.

Aquestes heurístiques són capaces de manejar els diferents tipus d'algoritmes que emplea l'arquitectura (de raonament progressiu i de mètodes múltiples) i una d'elles, la SSSM, utilitza els recursos de l'arquitectura per a aprofitar resultats previs, millorant l'eficiència obtinguda.

Finalment, es presenten proves estadístiques per a verificar les característiques de les heurístiques així com dues aplicacions reals per a mostrar la seua viabilitat.

Abstract

Over the last years, the Artificial Intelligence area is undergoing a great advance. It is widely applied in many areas, one of which is the area of real-time problems. In these problems, the logic of the solution calculus is important, but also the time when the solutions are obtained.

The fusion of Artificial Intelligence and Real-Time areas is useful, because the Artificial Intelligence provides new possibilities to the Real-Time systems. New possibilities such as the flexibility for solving problems in complex and dynamic environments. However, this approach has shown important difficulties.

Mainly, the Real-Time systems have temporal requirements (they usually require predictable response times) that are not usual in Artificial Intelligence techniques.

One of the ways to solve this problem is the development of software architectures. These software architectures are used to design intelligent agents that work in real-time environments. These architectures have several mechanisms to allow to the build agents to work in real-time environments offering reactive behavior (to fulfill the temporal requirements) and deliberative behavior (that make use of Artificial Intelligence techniques to obtain better results).

One of these architectures is ARTIS. This architecture uses a two-level scheduling of its tasks to fulfill this objectives. A first-level scheduler guarantees to obtain the solutions within the hard-time limits. A second-level scheduler controls the components used to improve the result quality.

The methods shown in this work are centered in the second-level scheduler. Two new heuristics, SSS (Slack-slide scheduling) and SSSM (SSS with Memory) has been developed. These heuristics make a more precise use of the architecture characteristics and the components to be scheduled. This way, they can obtain better results qualities than other applicable heuristics.

These heuristics are able to handle several types of algorithms that the architecture ARTIS uses (progressive refinement and multiple methods). One of them, the SSSM heuristic , uses the resources of the architecture to take advantage of previous results to improve the system efficiency.

Finally, the memory shows statistical tests to verify the heuristic characteristics and two real application to show its viability.

Índice general

Índice de figuras

Índice de tablas

Índice de gráficas

Índice de fórmulas

1 INTRODUCCIÓN

El ámbito de aplicación de la Inteligencia Artificial (IA) es cada vez mayor. La IA es utilizada para la gestión de recursos, toma de decisiones en los negocios, control de procesos industriales, en el funcionamiento de electrodomésticos y hasta en los videojuegos. Se puede ver como una disciplina cada vez más consolidada y a la que se le presta una mayor atención, pero esto no ha sido siempre así. De unos comienzos excesivamente eufóricos, donde se veía a esta materia como la llave para resolver cualquier problema planteado, se paso a un largo periodo de frenazo por no alcanzar las perspectivas creadas.

Precisamente una de las causas visibles de este fracaso inicial fue el exceso de optimismo y las excesivas pretensiones iniciales [Rus96]. Se quería realizar programas de propósito casi universal, con una "inteligencia" equiparable a la humana. Un ejemplo de esto eran los proyectos de traducción automática, surgidos en los años de la guerra fría y al amparo de fuertes inversiones estatales. Las desmesuradas aspiraciones iniciales pronto se vinieron abajo. Las técnicas usadas, normalmente probadas con los llamados problemas juguete, fracasaban a la hora de ser aplicadas en entornos reales. La mayoría de estos proyectos fracasaron y con ello las subvenciones fueron retiradas. El problema mayor fue el desprestigio que adquirió el termino IA y con ello que la investigación en esta área quedase relegada a un segundo plano.

El resurgimiento de la Inteligencia Artificial vino entre otras cosas por el convencimiento de que había que afrontar los problemas y las posibilidades de esta disciplina de forma más realista. Se empezó a estudiar como aplicar las técnicas de Inteligencia Artificial existentes y a desarrollar otras nuevas a problemas concretos, que bien no resultaban factibles de ser solucionados de otra manera o bien su resolución por otros medios resultaba muy costosa.

Este nuevo enfoque tuvo éxito. La Inteligencia Artificial cobró de nuevo auge, siendo sus técnicas aplicadas a un número cada vez mayor de campos. Este aumento en el número campos de aplicación, ha hecho necesario que la Inteligencia Artificial evolucionase, mejorando sus técnicas existentes y surgiendo nuevas, que pudiesen dar respuesta a las demandas de una industria que produce sistemas cada vez más complejos.

1.1 MOTIVACIÓN

Una de las nuevas áreas de aplicación de la Inteligencia Artificial es el campo de los sistemas de tiempo real. Estos sistemas son requeridos en aquellos problemas donde se necesita una respuesta al problema en un determinado plazo de tiempo para poder garantizar el buen funcionamiento del sistema. Ejemplos de sistemas de este tipo pueden ser los destinados al control de procesos industriales, vigilancia en unidades de cuidados intensivos, etc. Muchos de estos problemas no son nuevos y ya existían técnicas para manejarlos sobre todo en el campo de control. Sin embargo las prestaciones cada vez mayores que se requieren a estos sistemas, ha llevado a la aplicación de técnicas de Inteligencia Artificial que puedan hacer frente a esta mayor complejidad, ya que los métodos tradicionales no son capaces de conseguir la sofisticación y flexibilidad necesarias.

El principal problema que presenta esta nueva aproximación es el de las diferentes características, en muchos casos contrapuestas, de los sistemas de tiempo real y las técnicas de Inteligencia Artificial. Como más adelante se comentará en la memoria, la necesaria predecibilidad temporal de los primeros se enfrenta a la frecuente variabilidad en los tiempos de cómputo de las segundas. Es necesario soluciones que alcancen un compromiso entre ambos mundos.

Una de las técnicas que en principio puede ser adecuada para su aplicación debido a la flexibilidad y potencia que aporta para la resolución de problemas complejos, es la de los sistemas basados en agentes. El concepto de agente, una entidad que percibe, razona y actúa en consecuencia, potencialmente colaborando con otros agentes, encaja bien en sistemas potencialmente distribuidos como los de control industrial. Cada agente haría uso entonces de técnicas particulares (incluyendo a su vez técnicas de Inteligencia Artificial) para resolver partes del problema global.

El problema se plantea entonces, tal y como se indicó anteriormente, en como conseguir el uso de agentes que utilicen estas técnicas en entornos de

tiempo real. Uno de los caminos planteados es el desarrollo de arquitecturas software específicas para resolver estos problemas, que basadas en mayor o menor medida en el concepto de agente, permitan la utilización de la Inteligencia Artificial en los sistemas de tiempo real.

Una de estas arquitecturas es ARTIS. ARTIS es una arquitectura software diseñada para desarrollar agentes que hagan uso de técnicas de Inteligencia Artificial dentro de entornos de tiempo real. Su diseño permite que el agente garantice las restricciones de tiempo real y busque soluciones de la mayor calidad posible, para lo que tendrá que hacer uso de técnicas de planificación (scheduling).

Las técnicas de planificación que se deberán usar para conseguir obtener la mejor calidad de resultados son diferentes a aquellas encaminadas a garantizar los tipos de respuesta. La diferencia, además del diferente objetivo, radica en el hecho de que los métodos a planificar hacen uso, tal y como se ha comentado, de técnicas de inteligencia artificial, por lo general muy diferentes a las usadas habitualmente en sistemas de tiempo real, sobre todo en cuanto al consumo de recursos, y a la predecibilidad de su comportamiento.

Aunque existen técnicas denominadas voraces, que comentaremos en la memoria, y que son capaces de trabajar con las técnicas de IA, se hace necesario utilizar nuevos métodos que haciendo mayor uso de la información de los métodos a planificar, consigan obtener mejores resultados.

Diferentes tipos de estas técnicas de planificación que de forma general podemos denominar como deliberativas, tales como computación flexible o métodos múltiples, ya producen resultados superiores a las anteriores. Sin embargo sus características hacen que sean aplicables a sistemas que cumplan determinadas características como por ejemplo que usen sólo algoritmos anytime. Nosotros requerimos técnicas que permitan la utilización de diferentes tipos de algoritmos, tal y como son usados en ARTIS.

1.2 OBJETIVOS

El objetivo de este trabajo será presentar una técnica de planificación de tareas inteligentes dentro de una arquitectura para agentes inteligentes como es ARTIS. Para ello se han estudiado tanto las técnicas ya existentes como diferentes arquitecturas diseñadas para el desarrollo de agentes en entornos

de tiempo real. Al ser la técnica a desarrollar de aplicación en la arquitectura ARTIS era necesario también estudiar a fondo las características de esta.

Los objetivos más concretos de este trabajo serían:

- Desarrollo de la técnica de planificación que debe permitir el uso de diferentes tipos de tareas como serán aquellas que mejoran incrementalmente una solución frente a las que presentan diferentes alternativas para calcularlas.

- La técnica debe estar encaminada a obtener la mejor calidad de resultados posible pero teniendo en cuenta que trabajará en un entorno de tiempo real, por lo que deberá respetar las restricciones de tiempo real existentes. Es decir dichos resultados sólo serán útiles si son calculados dentro de unos plazos determinados.

- Otro objetivo a perseguir, con el propósito de optimizar al máximo los recursos existentes, es decir el tiempo de CPU disponible, será el de reutilizar resultados previos con lo que el sistema tenga más tiempo para calcular respuestas a otros problemas o a mejorar la calidad de las respuestas ya obtenidas. La técnica de planificación deberá disponer de un mecanismo que tenga en cuenta esta recuperación de resultados y lo incorpore en sus decisiones.

- Validar dicha técnica frente a otras existentes y aplicables en la arquitectura ARTIS con el fin de comprobar la mejora de lo resultados.

1.3 ESTRUCTURA DE LA MEMORIA

En el capitulo 2 se introducirá la problemática del uso de la Inteligencia Artificial en entornos de tiempo real, así como diferentes propuestas existentes. En el capitulo 3 se comentará el concepto de agente y se hará un repaso a arquitecturas existentes que hacen uso del concepto de agente en entornos multiagente especialmente para su uso en entornos de tiempo real.

En el capitulo 4 se presentará la arquitectura de agente ARTIS, especialmente la parte dedicada al control y planificación de los componentes que la forman, puesto que es en esta arquitectura donde se van a aplicar las técnicas desarrolladas.

Los siguientes dos puntos presentan las dos aportaciones de este trabajo para mejorar los resultados. En el capitulo 5, una nueva técnica de planificación para utilizar mejor el tiempo disponible que las técnicas previas y posteriormente una modificación a la técnica anterior para la reutilización de resultados anteriores. El capitulo 6 presenta una serie de pruebas para evaluar dichas técnicas junto con los resultados obtenidos.

El capitulo 7 presenta las conclusiones obtenidas a partir de los resultados anteriores y se comenta el trabajo futuro.

Finalmente dos anexos presentan la aplicación del trabajo desarrollado en dos entornos concretos.

2 SISTEMAS INTELIGENTES EN TIEMPO REAL

Los problemas de tiempo real son aquellos en que en su resolución interviene el factor tiempo como requerimiento, en el sentido de la necesidad de obtener soluciones que además de ser correctas desde el punto de vista lógico, estén disponibles en un determinado momento temporal. Cada vez más complejos, en este tipo de problemas se requieren técnicas más potentes y flexibles para su manejo, como las proporcionadas por la Inteligencia Artificial.

En este capitulo se vera en primer lugar una breve introducción a los sistemas de tiempo real, haciendo una breve revisión de los conceptos básicos de dichos sistemas, para a continuación, ver la problemática que presenta la utilización de técnicas de IA en dichos sistemas. Finalmente se verán diversas técnicas que pretenden conseguir este objetivo, el uso de sistemas inteligentes en problemas de tiempo real.

2.1 SISTEMAS DE TIEMPO REAL

Un sistema de tiempo real se puede definir como un sistema cuyo buen funcionamiento depende no sólo de los resultados obtenidos sino también del momento en que dichos resultados son obtenidos [Sta88][Sta93].

Un sistema de tiempo real está formado por una serie de tareas con una serie de características que las definen. Estas características son:

- Deadline. El deadline o tiempo límite de una tarea, se define como el plazo máximo para que una tarea proporcione una respuesta. A partir

de esta definición se puede realizar una clasificación sobre los sistemas de tiempo real, en función del grado de cumplimiento que el sistema requiere sobre los deadlines. De esta manera, se puede hablar de dos tipos de sistemas de tiempo real:

- Sistemas de tiempo real estricto o hard real-time systems. Son aquellos que requieren que todas las respuestas estén disponibles antes de sus respectivos deadlines. De no ser así el problema podría quedar fuera de control con graves consecuencias.

- Sistemas de tiempo real no estricto o soft real-time systems. En estos casos el no cumplimiento del deadline, es decir la obtención del resultado una vez vencido este, únicamente produce una disminución en la utilidad del propio resultado para el sistema.

- Periodo. Se define como la frecuencia de activación de la tarea en el sistema. Frecuentemente el sistema tiene que tratar con eventos que debido a la naturaleza asíncrona del problema, ocurren de forma aperiódica. Sin embargo, este comportamiento haría imposible garantizar la asignación de tiempo de CPU para las tareas críticas, por lo que se opta por usar el concepto de máxima frecuencia de entre eventos. De esta manera, proporcionando un periodo a cada componente, se puede determinar el momento en que cada tarea debe activarse y hacer posibles los tests de garantía temporales que posteriormente se comentarán.

- Tiempo de ejecución en el caso peor. Para poder definir totalmente la tarea es necesario conocer su tiempo de ejecución en el caso peor. De esta manera los cálculos que intenten determinar si se tiene tiempo para cumplir el deadline de las tareas críticas, se harán con dicho tiempo.

Ya que el objetivo primordial es el que todas las tareas cumplan sus deadlines, especialmente en el caso de los sistemas de tiempo real estricto, se necesitan estrategias o paradigmas de planificación (scheduling) que garanticen esta finalidad. Por lo general y más aún si tenemos en cuenta la naturaleza crítica en la mayoría de los casos de estos sistemas, es necesaria una garantía a priori de que el sistema cumplirá todos los requerimientos temporales. Es lo que se conoce como viabilidad o planificabilidad (feasibility).

El logro de la viabilidad es obtenido a través de la predecibilidad de las acciones que el sistema puede llevar a cabo en tiempo de ejecución [Ter02]. Para conseguir dicha predecibilidad, la política de planificación se basa en tres aspectos:

- El modelo computacional, o características y restricciones que deben cumplir las tareas. Normalmente referidas a los elementos definitorios comentados previamente de deadline, periodo y tiempo de ejecución. En cuanto a las restricciones serían del tipo de que los agentes deben ser periódicos o en su defecto considerar la frecuencia máxima de llegada que comentábamos con anterioridad.

- La configuración pre-ejecución. Establece dependencias a las tareas que no pueden ser cambiadas en tiempo de ejecución. El grado de complejidad de estas configuraciones varía dependiendo de la política de planificación, desde generar un calendario o tabla explícito que será seguido de forma estricta en el denominado paradigma de ejecutivos cíclicos a no realizar ninguna restricción en políticas expulsivas de prioridad dinámica. Un punto intermedio sería el paradigma expulsivo de prioridad fija donde se establecen prioridades a las tareas.

- La política de planificación en tiempo de ejecución. Es decir como se determina en cada momento que tarea debe ser la ejecutada. En este aspecto, también varía la complejidad de la tarea en función del paradigma. El más simple sería el de ejecutivos cíclicos que simplemente sigue lo marcado por el calendario preestablecido. En las prioridades dinámicas se escogería la tarea con deadline absoluto más próximo y en prioridad fija el de mayor prioridad.

En cualquier caso para garantizar los requerimientos temporales, cada paradigma dispone de un test o análisis de planificabilidad, que basándose en las características de todas las tareas, y mediante cálculos aritméticos indica si todos los deadlines serán o no satisfechos.

2.1.1 Tiempo real flexible

Conforme los sistemas de tiempo real se han ido haciendo más complejos se ha visto la necesidad de dotarlos de mayor flexibilidad. Flexibilidad en términos de manejo de situaciones imprevistas; uso de métodos de resolución más potentes, pero sin las características temporales estrictas habituales en tiempo real (tiempo en el caso peor predecible y realista);

búsqueda de otros parámetros de evaluación además del cumplimiento de deadlines, como puede ser el aumento de la calidad de la respuesta.

Para tratar esta nueva realidad surgen los denominados sistemas de tiempo real flexible, termino que en realidad aglutina técnicas diferentes con objetivos en muchos casos distintos entre si. Se pueden encontrar por un lado técnicas que hacen uso de algoritmos que pueden utilizarse sin restricciones en cuanto al tiempo como los algoritmos anytime [Hor91], o que proporcionan diferentes aproximaciones como los métodos múltiples [Ken90][Gop90], estas técnicas son presentadas en el punto 2.3. También arquitecturas que pretenden optimizar el uso de diferentes técnicas y que están orientadas hacia este tipo de problemáticas como se presentará en el capitulo 3.

Los objetivos también pueden ser diferentes, mientras que algunas técnicas pretenden incorporar un control de tareas no tan estrictas (como acceso aplicaciones multimedia, etc.) dentro de un entorno donde también se mantienen tareas de tiempo real estricto, otras técnicas pretenden la flexibilización de las propias tareas críticas. Aunque con puntos en común, se trata de dos objetivos diferentes que requerirán soluciones también diferentes.

2.2 PROBLEMÁTICA

Nuestro propósito es utilizar sistemas inteligentes para resolver problemas en entornos de tiempo real. Al decir sistemas inteligentes nos referimos a que hagan uso de técnicas de Inteligencia Artificial para calcular las respuestas necesarias.

Desde un punto de vista de tiempo real un sistema de inteligencia artificial en tiempo real debería [Mus95]:

- Trabajar de forma continua sobre periodos extensos de tiempo.
- Interactuar con el entorno vía sensores y actuadores.
- Manejar con datos inciertos o incompletos.
- Focalizar recursos en los eventos más críticos.
- Manejar eventos síncronos y asíncronos de una forma predecible con tiempos de respuesta garantizados.

- Degradarse de forma controlada, para manejar sobrecargas y fallos.

Uno de los puntos primordiales es tener la garantía de que se dispondrá una respuesta a tiempo. Por ello lo que se requiere de los métodos que sean usados para obtener dichas respuestas no será rapidez (aunque evidentemente cuanto más eficiente sea el algoritmo mejor) sino que sea predecible en cuanto a coste temporal. Se necesita por tanto que pueda darse una estimación del coste temporal en el caso peor y evidentemente que dicho coste sea realista y asumible por el sistema.

Sin embargo las técnicas de Inteligencia Artificial simbólica suelen tener un punto común, la búsqueda en grandes espacios de soluciones, como por ejemplo la efectuada por los sistemas de producción. A menudo el coste de estas técnicas es impredecible o con grandes varianzas.

Hay que tener en cuenta que para poder determinar si un sistema cumple las restricciones temporales, se usan tests de planificabilidad tal y como se ha comentado anteriormente, como el empleado en el "Rate Monotonic Algorithm" [Liu73]. Este test utiliza los tiempos máximos de ejecución para el caso peor para indicar si el sistema cumplirá los plazos necesarios.

Sin embargo por las características indicadas anteriormente de los sistemas inteligentes, en muchos casos el test no puede aplicarse por que no puede predecirse un tiempo de coste máximo para el método a utilizar. En otros muchos casos el problema viene dado por la gran diferencia existente entre el coste máximo para el caso peor teórico del método y el coste promedio real o la gran varianza en el tiempo de ejecución. Aunque el sistema seguramente podría usar el método según los tiempos promedio, el test falla y no se puede garantizar el buen funcionamiento, o bien el sistema es planificable, pero la gran diferencia entre los costes teóricos y los reales hacen que estemos infrautilizando los recursos del sistema obteniendo un bajo rendimiento del mismo.

En general, para poder acercar el tiempo máximo con el promedio y disminuir la varianza, lo primero que se puede realizar es utilizar conocimiento sobre el problema para de esta manera reducir el espacio de búsqueda.

Sin embargo, en la mayoría de los casos esto resulta insuficiente. C. J. Paul indica en [Pau93] que es necesario estructurar el proceso de búsqueda de forma que las soluciones obtenidas estén disponibles a tiempo, aunque esta solución no sea la óptima (aunque sí lo suficientemente buena). Cualquier intento de mejorar estas variaciones consiste en usar lo que denomina grados de libertad. Estos grados de libertad que poseen (potencialmente)

los problemas permitirán usar el conocimiento sobre el propio problema para estructurar tanto el espacio como el propio proceso de búsqueda, produciendo soluciones de calidad dependientes de una función temporal. Según el autor, cualquier técnica existente encaminada a resolver este problema de los sistemas de tiempo real, estaría en realidad explotando uno o varios de los grados de libertad tipificados para poder conseguir la obtención de soluciones.

En este trabajo establece cuales serían estos grados de libertad:

- Poda: Consiste en utilizar conocimiento del problema para eliminar las zonas del espacio del problema que se sabe no contienen el estado meta. Esta técnica reduce el tiempo máximo de ejecución sin afectar a la búsqueda en cuanto a calidad de la solución obtenida.

- Ordenación: Con las técnicas que hagan uso de este concepto, lo que se realiza es determinar en que orden deben ser visitadas las ramas del espacio búsqueda. En este caso no se eliminan estados y tampoco se compromete la calidad de la solución.

- Aproximación: Las técnicas que hacen uso de este grado de libertad no garantizan encontrar la mejor solución. Esto es así puesto que se reduce la precisión con que usamos el conocimiento del problema. La precisión del resultado es menor, pero también lo es el tiempo en el peor caso y en muchas ocasiones también el tiempo promedio.

- Limitación del alcance: Normalmente cuando se desea elegir un operador para avanzar en el proceso de búsqueda se analiza las consecuencias de la decisión. De esta forma tendremos más información para realizar dicha decisión. Sin embargo puede suceder que el proceso de determinar que puede ocurrir, además de costoso, sea inútil, si durante el tiempo en que se determina, cambian las características del problema. En estos casos sería conveniente limitar el alcance de la anticipación, de esta forma se puede no tener tanta información sobre que puede ocurrir, pero al menos lo que se disponga puede ser más fiable y además obtenido con un menor coste.

2.3 TÉCNICAS DE RESOLUCIÓN DE PROBLEMAS EN TIEMPO REAL

La mayoría de las técnicas existentes para resolver un problema dado en tiempo real harían uso de uno o más de estos grados de libertad. A

continuación comentaremos algunas de estas técnicas. En [Garv94] puede verse una clasificación de estas técnicas y en [Hor01] nuevas aproximaciones a los problemas que algunas de ellas plantean.

- Razonamiento progresivo. Se realiza una búsqueda por niveles. Comenzando por la raíz del espacio de búsqueda se va aumentando el uso de la información según se desciende en la búsqueda hasta que se alcanza la solución óptima o el plazo máximo de ejecución [Win84]. De esta manera la precisión de la solución va aumentando. Los grados de libertad manejados serían los de ordenación, poda y limitación del alcance.

- Lógica de precisión variable. Se utiliza una lógica [Mic86] en la cual las reglas de producción disponen además del antecedente y el consecuente, de un tercer elemento que establece conclusiones que se establecen de forma excepcional. De esta manera sólo en el caso de que se disponga de suficientemente tiempo este nuevo elemento será evaluado. Se trata de una aproximación pues se reduce la precisión con que usamos el conocimiento.

- Algoritmos anytime [Bod94]. Reciben este nombre un tipo de algoritmo de refinamiento iterativo, es decir que va aumentando la calidad de la respuesta obtenida conforme se le proporciona más tiempo para ejecutarse y que además puede ser interrumpido en cualquier momento proporcionando una respuesta. Se basan en el uso de los denominados perfiles de ejecución que relacionan tiempo de ejecución con calidad de la respuesta obtenida, como por ejemplo el de la figura 2.1. Actuarían dentro de los grados de poda y ordenación.

  Dado que estos perfiles son en la mayoría de los casos solo aproximaciones, muchos de los sistemas que hacen uso de los algoritmos anytime necesitan monitorizar cuando el proceso realmente ha llegado a la calidad deseada. Esta monitorización produce interferencias en la propia ejecución del algoritmo, por lo que es necesario un equilibrio entre la precisión buscada y la monitorización. Algunos trabajos buscan como determinar el número óptimo de interrupciones [Fin01][Han01].

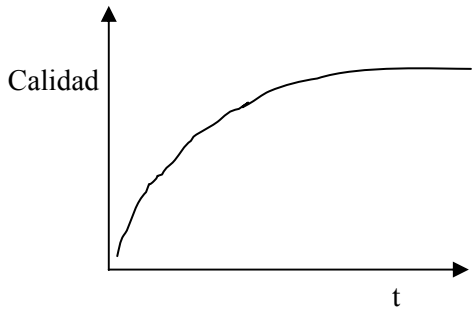


Figura 2.1. Perfil de ejecución de un algoritmo anytime

Existen diferentes trabajos que toman como base los algoritmos anytime. Son particularmente relevantes:

- Planificación deliberativa [Bod94]. Aplicada a problemas de planificación dependientes del tiempo, donde el tiempo disponible para dar una respuesta varía de una situación a otra. Su objetivo es realizar una asignación de tiempo a los algoritmos anytime que proporcione la mayor calidad de respuesta global posible.

- Computación flexible en resolución de problemas de teoría de la decisión. El trabajo de Horvitz [Hor88][Hor90][Hor91] en este campo estudia tres elementos: Cómo establecer los perfiles de ejecución a los algoritmos anytime, encontrar algoritmos anytime que sirvan para determinar la siguiente acción a realizar y finalmente como establecer el tiempo destinado a la ejecución de los algoritmos de resolución frente al tiempo destinado a la decisión.

- Compilación de algoritmos anytime. En este paradigma presentado por Zilberstein [Zil95], el cómputo es resultado de una serie de algoritmos anytime, donde la entrada de un algoritmo es la salida de otro. En este caso se trata de determinar cual es el tiempo asociado a cada algoritmo parcial teniendo en cuenta que el tiempo total será fijo. La ejecución debe llevarse a cabo totalmente para obtener un resultado, es decir se deben ejecutar todos los algoritmos parciales, aunque el porcentaje de ejecución de cada uno de ellos pueda variar.

- Computación imprecisa. [Chun90]. Similar a los algoritmos anytime, se distingue en que dispone de una parte inicial ininterrumpible que debe ser ejecutada en su totalidad para ofrecer una respuesta. Una

segunda parte opcional mejorará la calidad de la respuesta siempre y cuando se ejecute. En este caso el resultado ofrecido será el de la parte opcional, si esta no se completa se usará el resultado de la parte inicial. Al igual que en los casos anteriores se trabaja sobre la poda y la ordenación.

- Algoritmos de búsqueda en tiempo real. Se trata de adaptaciones de algoritmos de búsqueda usados en Inteligencia Artificial a las características de tiempo real. Estos algoritmos trabajarían sobre los grados de poda, ordenación, aproximación y limitación de alcance. Entre otros podríamos citar:

  - Técnicas de búsqueda para optimización en tiempo limitado. [Chu91]. En este caso se usa una función de aproximación para podar los nodos más alejados de la mejor solución encontrada hasta ese momento y de esta forma reducir el espacio de búsqueda. No se garantiza encontrar la mejor solución.

  - Búsqueda heurística en tiempo real. [Kor90]. En este caso se trata de realizar la búsqueda mediante un algoritmo anytime similar a los planteados por Zilberstein. Se pretende limitar el alcance de la anticipación asociando restricciones temporales a la expansión de cada nodo. Por tanto la profundidad de búsqueda se limita en función del tiempo disponible. No se garantiza encontrar la mejor solución pues el camino elegido dependerá de la heurística empleada.

- Métodos múltiples. Bajo esta denominación se engloban una serie de técnicas caracterizadas por el hecho de no presentar un único método para resolver un problema dado (tal y como si hacen los algoritmos anytime citados anteriormente). En su lugar se dispone de una serie de métodos que resuelven el mismo problema con diferente calidad de resultado y requerimientos de ejecución (normalmente tiempo de ejecución). Son técnicas que trabajan por tanto sobre el grado de aproximación.

  A continuación comentaremos algunas de estas técnicas que tienen como base el concepto de método múltiple:

  - Procesamiento aproximado. Recibe este nombre un conjunto de técnicas que realizan procesos de inferencia basados no en las lógicas exactas habituales, sino en lógicas aproximadas o probabilísticas para poder trabajar en condiciones de

incertidumbre. De esta manera, soluciones satisfactorias, es decir aceptables aunque no óptimas, pueden ser alcanzadas dentro de las limitaciones (normalmente temporales) del problema.

La aproximación actuaría en tres frentes: en la precisión, reduciendo el conjunto de valores que puede tomar algún dato; en completitud, ignorando alguna de las características de la solución; o en certeza, simplificando las relaciones entre las características del problema.

Ejemplos de esta técnica pueden verse en los trabajos de Lesser [Les88] y en [Dec90], donde Decker, Lesser y Whitehair realizan las modificaciones necesarias para que una arquitectura de pizarra pueda soportar este tipo de razonamiento.

- Planificación design-to-time. Se trata de una extensión de las técnicas anteriores. Se asume que existen métodos múltiples para diferentes tareas y el problema consiste en seleccionar que métodos se deben ejecutar para cada tarea, con el fin de obtener la mayor calidad posible. El principal problema de esta técnica se centra en el hecho de que calidad y coste temporal de cada método debe conocerse con precisión a priori. En [Gar93], Garvey y Lesser estudian precisamente las condiciones de predecibilidad que deben cumplir los métodos.

- Razonamiento restringido en el tiempo bajo incertidumbre. El objetivo de estudio en este caso es el de problemas de diagnóstico o clasificación dinámicos, es decir aquellos en los que alguno de los datos utilizados para realizar dicho diagnóstico puede variar antes de acabar el proceso.

  Bonissone y Halverson en [Bon90] presentan una técnica basada en disponer de planes precompilados de los cuales se hace una predicción de su utilidad. En tiempo de ejecución se selecciona uno de los planes, en función de las circunstancias del problema y del tiempo disponible uno de los planes, teniendo en cuenta además ejecuciones previas de los planes para poder estimar con mayor precisión los resultados que se pueden obtener.

- Heurística de utilidad marginal. Técnica propuesta por Etzioni [Etz91], está basada en conceptos de teoría económica. En concreto, define el coste de oportunidad de una acción llevada a cabo por un agente, como el máximo beneficio (calidad de

respuesta) que podría haber obtenido de haber ejecutado el agente otra acción, la cual no puede llevar a cabo por ejecutar la acción elegida. De esta forma la función de utilidad de una acción seria el beneficio de la acción elegida (su calidad) menos su coste de oportunidad. La heurística de utilidad marginal que será utilizada para elegir que acción llevar a cabo queda definida como la derivada de dicha función de utilidad respecto del tiempo (o bien otro recurso igualmente importante).

2.4 CONCLUSIONES

Realizando una visión de conjunto sobre las técnicas vistas en el punto anterior se puede concluir que la mayoría requieren a la aproximación con el fin de conseguir obtener soluciones en los plazos marcados. Esto hace que los resultados obtenidos no sean los óptimos, algo esperado pues su obtención es impensable por los costes requeridos.

Más importante para nuestros propósitos es el hecho de que estas técnicas no garantizan por si solas el cumplimiento de requerimientos de tiempo real estricto, por lo que serían más indicadas para sistemas de tiempo real soft o bien complementarlas para poder aplicarlas al primer tipo de sistemas.

Otra característica a destacar sería el hecho de estar pensadas para un tipo concreto de algoritmo, bien algoritmos anytime, bien métodos múltiples. Esto hace que limiten previamente las técnicas de IA que un sistema pueda utilizar para calcular las respuestas. En el caso de sistemas que requieran que diferentes aproximaciones convivan serían necesarias nuevas técnicas capaces de combinarlas.

3 AGENTES Y ARQUITECTURAS MULTIAGENTE

En el capítulo anterior se han mostrado diferentes técnicas que permiten utilizar métodos de inteligencia artificial en entornos de tiempo real. Una cuestión a resolver sería como estructurar dichos métodos, quién los ejecuta, qué modelo de arquitectura los soporta.

Una aproximación que cada vez es más utilizada es la de agente inteligente. Entidades autónomas que realizan tareas inteligentes, aunque como comentaremos más adelante no suelen enfrentarse a problemas de tiempo real, o si lo hacen, son de tiempo real no estricto. En este capítulo comentaremos el concepto de agente y haremos un repaso de algunas arquitecturas que hacen uso del mismo, como paso previo al capítulo siguiente donde se estudiará con más detalle una arquitectura específica, ARTIS, dentro de la cual se desarrolla el presente trabajo.

3.1 CONCEPTO DE AGENTE

Responder a la pregunta de qué es un agente es difícil, ya que en la literatura nos encontramos con diferentes definiciones, que hacen que lo que para algunos sea un agente, para otros no tenga la suficiente entidad para poder ser llamado así.

Rusell y Norvig en su libro “Inteligencia Artificial: Un enfoque moderno” usan el concepto de agente como nexo de unión para todo los temas, mostrando como dotar de contenidos y funcionalidad a un agente partiendo

de los temas clásicos de la inteligencia artificial como representación del conocimiento, razonamiento. De hecho el libro es presentado con el subtitulo de "el libro del agente inteligente".

La definición que Russell y Norvig dan de agente es la de aquel ente que percibe su ambiente mediante sensores y que responde o actúa en dicho ambiente mediante efectores. Su definición de agente inteligente o racional es obviamente, más restrictiva. Un agente inteligente es aquel que toma sus decisiones de actuación para favorecer lo máximo posible su medida de rendimiento. Es decir que el agente toma decisiones que le permitan desempeñar lo mejor posible sus objetivos.

Esta definición es vista como simplista por otros autores, que interpretan que un agente es más complejo. En muchos casos se confía no en una definición sino más bien en un conjunto de características que debe cumplir el agente para ser considerado como tal.

Un ejemplo de esta caracterización la podemos encontrar en [Woo95], donde Woodridge y Jennings definen un agente como un sistema informático hardware o más frecuentemente software, que posee las siguientes propiedades:

- Autonomía: los agentes actúan sin la intervención directa de humanos u otros agentes y tienen algún tipo de control sobre sus acciones y estado interno.

- Habilidad social: los agentes interactúan con otros agentes (e incluso con humanos) por medio de algún tipo de lenguaje de comunicación de agentes.

- Reactividad: un agente percibe su entorno y responde de forma apropiada en un tiempo razonable a los cambios que ocurren en él.

- Pro-actividad: los agentes no actúan simplemente en respuesta a su entorno, sino que también deben exhibir un comportamiento dirigido por objetivos tomando la iniciativa.

A estas propiedades otros autores como Franklin y Graesser [Fra96] y Nwana [Nwa96] incorporan algunos atributos más en sus definiciones de agente. Estos atributos serían:

- Adaptatividad: La capacidad de un agente para cambiar su comportamiento conforme al aprendizaje que el mismo realiza.

- Benevolencia: Un agente debe estar dispuesto a ayudar a otros agentes siempre que esto no le impida alcanzar sus propios objetivos.

- Continuidad temporal: Un agente se puede considerar como un proceso sin fin, que realiza sus funciones de forma continua.

- Movilidad: Si el agente puede trasladarse a través de una red telemática, para actuar en una maquina diferente a la de su propietario, por ejemplo, para buscar información en nombre de este.

- Racionalidad: Capacidad del agente para razonar a partir de los datos que recibe para obtener la mejor solución posible.

- Veracidad: Un agente no debe comunicar información errónea a propósito.

Cuáles son entre estas propiedades las que un agente debe cumplir, es una discusión no cerrada entre la comunidad que trabaja en estos temas. En opinión de H. Van Dyke Parunak, se puede realizar un paralelismo entre un agente y una navaja del ejército suizo. La definición básica de agente sería como sólo la navaja, y si el agente necesita alguna otra propiedad se le añade pero no es necesario llevar todo.

3.2 ARQUITECTURAS DE AGENTES

Bajo el concepto de arquitectura de agente subyace la descripción particular de los elementos que constituirán un agente concreto y como estos elementos interactúan entre si [Mae91][Igl98]. En este punto comentaremos primera una breve clasificación de agentes para a continuación presentar diferentes arquitecturas que hacen uso del concepto de agente en un sentido más o menos amplio. Algunas de ellas además están pensadas para el funcionamiento conjunto de agentes desarrollando lo que se conoce como arquitecturas multiagente. Se ha estudiado fundamentalmente aquellas que pretenden ser aplicadas a entornos de tiempo real o al menos con capacidad reactiva [Gol01][Mus02][Raj01].

Una primera división de las arquitecturas considera el acceso a los sensores y actuadores. En concreto si todas las capas de la arquitectura tienen acceso a dichos componentes se considera una arquitectura horizontal. Por el contrario, en las arquitecturas verticales, sólo la capa inferior tiene acceso a ellos.

Otra posible clasificación atendería al tipo de procesamiento empleado, distinguiendo en este caso las arquitecturas deliberativas, las reactivas y las híbridas.

Las arquitecturas deliberativas son aquellas que manejan símbolos físicos. En [Igl98] se indica que estas arquitecturas deben ser capaces de describir objetivos y como alcanzarlos. Normalmente hacen uso de técnicas de planificación para determinar que pasos realizar. Una subdivisión de este grupo, contemplaría las arquitecturas intencionales, aquellas en que los agentes pueden razonar sobre sus creencias e intenciones, y las sociales donde cada agente es capaz de mantener un modelo de otros agentes y razonar sobre ello. Normalmente presentan una estructura horizontal.

Las arquitecturas reactivas por el contrario no presentan un modelo de razonamiento simbólico complejo, sino un modelo estimulo-respuesta. En este caso las arquitecturas son verticales, donde las capas superiores especializadas manejan directamente los datos, inhibiendo las capas inferiores y obteniendo las acciones a realizar ante dichas entradas.

Finalmente las arquitecturas híbridas presentan varios subsistemas, unos deliberativos para resolver tareas que requieren un modelo simbólico y otros reactivos para responder ante estímulos que no requieren deliberación.

3.2.1 Arquitectura de sistemas inteligentes adaptativos

AIS Adaptative Intelligent Systems [Hay95] es una arquitectura pensada para el desarrollo de agentes inteligentes que trabajen en entornos de alta variabilidad. Se trata por tanto de una arquitectura que debería funcionar en entornos de tiempo real aunque no estricto.

Su nombre procede de la capacidad de adaptación que deben tener los agentes para poder trabajar en esos entornos de alta variabilidad. Para conseguirlo el agente podrá variar varias de sus características en función de la situación de cada momento. En concreto podrá variar su estrategia de percepción, el modo de funcionamiento del control y del meta-control, así como los métodos de razonamiento usados.

La arquitectura presenta tres componentes básicos que reflejan las operaciones básicas de cualquier agente, es decir, percibir, razonar y actuar. Estos componentes son:

Modulo de percepción. No se encarga únicamente de recoger datos del exterior sino también de filtrarlos y hacer un proceso de abstracción con ellos. El objetivo es reducir la cantidad de información a manejar por el agente, centrándose este en los datos más interesantes en cada momento.

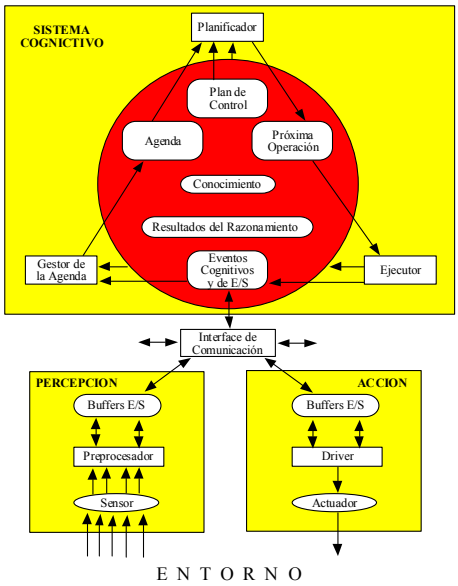


Figura 3.1. Arquitectura AIS

El subsistema cognitivo. Recibe e interpreta los datos que le manda el sistema de percepción (al cual también guía en el proceso de filtrado), realiza el razonamiento en función del conocimiento existente y elabora los planes que guían tanto su funcionamiento como el del resto de módulos. Utiliza para ello una arquitectura de pizarra que permite una planificación dinámica.

Subsistema de acción. Se encarga de ejecutar aquellas acciones externas calculadas por el subsistema anterior.

Como se puede ver por la funcionalidad de estos módulos los agentes operaran en función tanto de los datos externos como del proceso de razonamiento que es capaz de realizar, siendo su funcionamiento flexible para adaptarse a las circunstancias del entorno en cada momento.

Es de destacar la existencia de un agente denominado "Guardián" construido con esta arquitectura, encargado de la vigilancia de pacientes en unidades de cuidados intensivos.

3.2.2 ATLANTIS

Arquitectura diseñada para ser aplicada en tareas robóticas [Gat91], por lo que dispone de una alta reactividad, aunque incorporando mecanismos que le permitan ser lo bastante flexible para trabajar en entornos complejos.

Se basa en una estructura multicapa, donde las capas superiores controlan las inferiores hasta cierto punto, pues estas actúan en un modo semi-independiente.

La capa inferior representa la parte en contacto con el exterior, que realiza funciones de percepción y acción, leyendo de los sensores y mandando órdenes a los actuadores.

La capa intermedia o de secuenciación, señala las acciones a tomar en función de los datos recibidos. Esta operación la realiza de acuerdo a un mapping interno que puede ser cambiado en función de la evolución del sistema.

La capa superior o deliberativa, dispone de descripciones simbólicas del entorno, y se encarga de desarrollar planes para la obtención de los objetivos de alto nivel.

3.2.3 CIRCA

CIRCA [Mus92, Mus93, Mus95] son las siglas de Arquitectura de Control en tiempo Real Inteligente Cooperativo. Esta arquitectura forma parte del reducido grupo que pretende utilizar técnicas potentes de inteligencia artificial en entornos de tiempo real. Debido a la naturaleza impredecible de la mayoría de dichas técnicas, la apuesta de CIRCA para conseguir el uso de la inteligencia artificial en estos entornos, es la separación del sistema que se encarga de la parte de tiempo real y del que se encarga de las tareas inteligentes, ambos elementos están relacionados a través de un modulo de planificación.

Los elementos básicos que maneja esta arquitectura son los denominados pares test-acción (PTA). Estos pares constan, como su nombre indica, de un conjunto de precondiciones, que marcan cuando pueden ser aplicados y de las acciones ha realizar si se ejecutan. Además, y para poder ser

planificados, los PTA dispone de información acerca de los recursos que necesitan para su ejecución y el tiempo en el peor de los casos para evaluar las precondiciones y ser ejecutados.

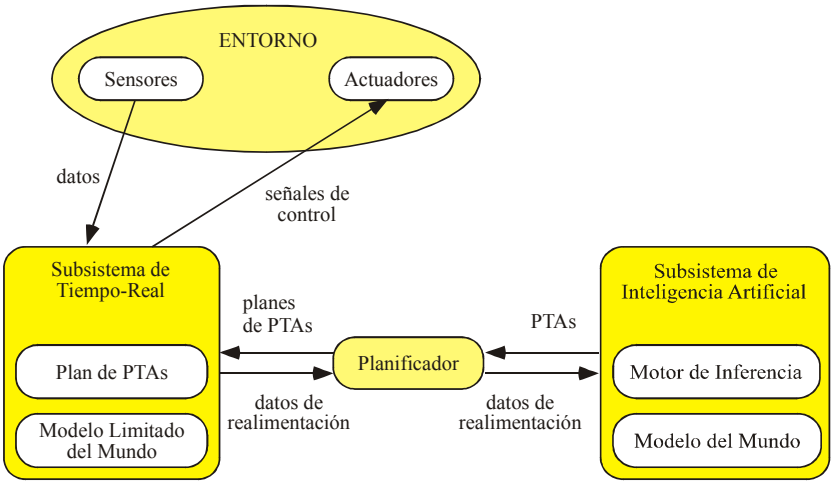


Figura 3.2. Arquitectura CIRCA

El subsistema de inteligencia artificial es el encargado de las tareas en principio más complejas, para ello descompone sus objetivos en elementos más simples, hasta llegar al nivel de PTA. Estos PTAs serán enviados al planificador para el estudio de su viabilidad.

El planificador construye planes con los PTAs usando la información que posee sobre los tiempos máximos de ejecución. Si la información recibida del sistema inteligente no es planificable, devolverá la petición a dicho sistema. Además detecta la existencia de tiempo excedente y lo comunica al sistema inteligente para que lo utilice en la ejecución de tareas no críticas.

El sistema de tiempo real se encarga de ejecutar tareas predecibles. Para ello comprueba las precondiciones y ejecuta las acciones según los planes de PTA que le informa el planificador.

3.2.4 HOMER

Modelo muy orientado a la utilización de técnicas de inteligencia artificial, pues se basa en el desarrollo de una arquitectura [Ver91] que disponga de elementos encargados de la planificación, módulos de gestión de memoria,

otros para los procesos reactivos incluso para interpretar el lenguaje natural y generación de mensajes al usuario.

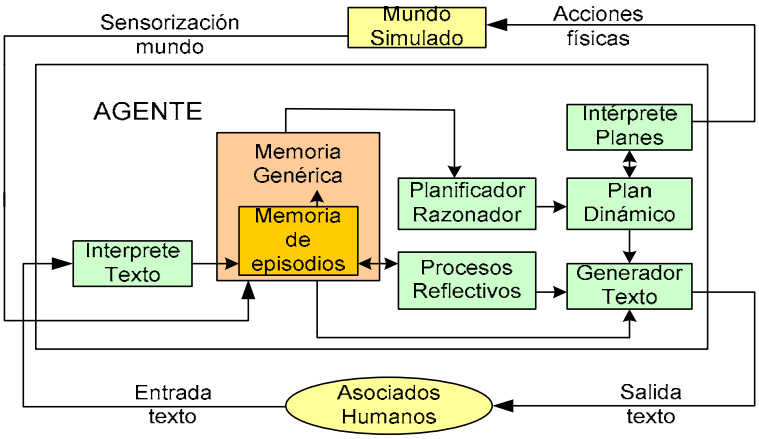


Figura 3.3. Arquitectura Homer

Su autonomía es limitada pues funciona bajo demanda del usuario. Así mismo el conocimiento que maneja es limitado pues se desarrolló para un ambiente determinado, la simulación de un entorno submarino.

3.2.5 ICARUS

Esta arquitectura presenta una estructura jerárquica donde vuelven a aparecer los elementos que hemos visto en otras arquitecturas. Dispone de los módulos de percepción, de acción y de planificación de las operaciones a realizar. Así como de un modulo de almacenamiento de la información.

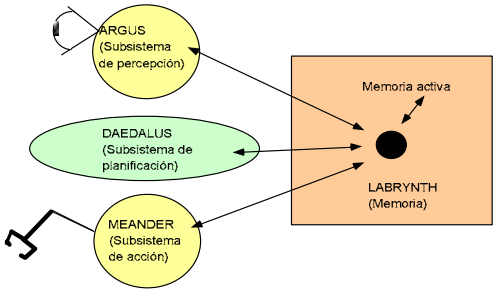


Figura 3.4. Arquitectura Icarus

El ciclo de control también es típico. El módulo de percepción toma información del exterior. Esta información junto a la que ya posee el

sistema es usada por el planificador para generar un plan. El tercer módulo se encarga tanto de ejecutar este plan como de monitorizarlo.

En esta arquitectura es importante el tratamiento que se hace de la información. Al trabajar en entornos dinámicos, permite por un lado el mantenimiento de información contradictoria y sobre todo el tratamiento incremental que hace de los planes. Estos no necesitan ser finalizados en su elaboración, para comenzar a ser ejecutados. La monitorización que realiza el módulo de acción permite su continuación completándolo de forma incremental o bien realizar una replanificación si el comportamiento se aleja de lo esperado.

3.2.6 MAX

La característica que hace especial esta arquitectura [Kuo91] es la atención que presta al meta-conocimiento. Los procesos de meta-razonamiento son los encargados de realizar las operaciones de planificación y aprendizaje, con la dificultad añadida de trabajar en entornos de tiempo real.

Para poder trabajar de forma uniforme, todo el conocimiento, tanto el de bajo nivel como sería los datos percibidos, o las ordenes sobre los actuadores, como el conocimiento de alto nivel, es decir el que es utilizado por el meta-razonamiento, es expresado en un formato homogéneo.

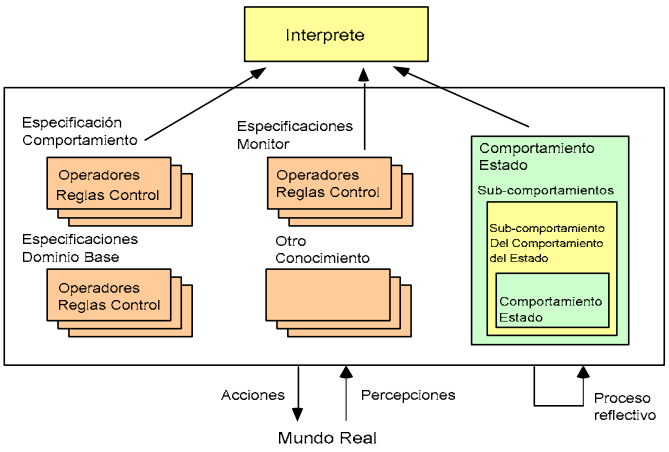


Figura 3.5. Arquitectura MAX

Otra innovación es la de presentar un control distribuido, con diferentes comportamientos, cada uno implementado por un módulo, que pueden ser llamados directamente.

Similar a la arquitectura anterior también se realiza un seguimiento de los resultados a través de módulos de monitorización.

3.2.7 Motor de reducción de entropía

Esta arquitectura [Dru91] hace uso de uno de los conceptos que mencionábamos en el capitulo anterior para integrar la inteligencia artificial en entornos de tiempo real: la computación anytime. Y lo hace no en el sentido de que aplique algoritmos anytime a las tareas a resolver, sino que la propia planificación que el sistema realiza es anytime. Esto le permite ejecutar planes no finalizados en su totalidad y que puede ir completando según las restricciones en cada momento.

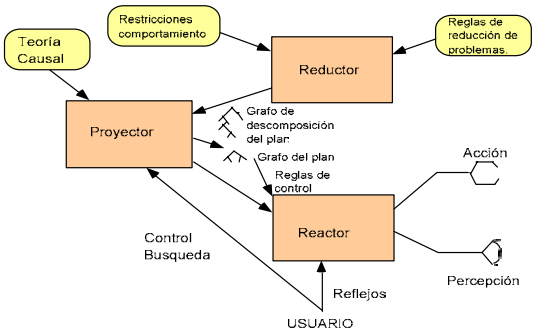


Figura 3.6. Arquitectura MRE

De nuevo aparece la división de tareas que hemos visto en arquitecturas anteriores. Aquí se trata de tres módulos, encargados de las operaciones de planificación, scheduling y control.

El primero llamado reductor sería el módulo cognitivo de alto nivel, encargado de elaborar estrategias para resolver el problema actual. Estas estrategias las generará basándose en estrategias ya existentes y en su propia experiencia.

El segundo módulo recibe el nombre de prospector. En base a las estrategias generadas por el módulo anterior planificará y asignará las acciones a realizar por el tercer módulo.

El tercer módulo, el ejecutor sería el componente de acción existente en otras estrategias, controlando su ejecución

3.2.8 PHOENIX

Este proyecto [How90] tiene como objetivo el diseño de agentes autónomos, que operen en entornos complejos y que combine componentes reactivos junto a componentes deliberativos.

La arquitectura que presenta PHOENIX para el desarrollo de un agente propone cuatro componentes diferenciados. Por una parte están los típicos módulos de relación con el exterior, por un lado el de los sensores y por otro el de actuadores.

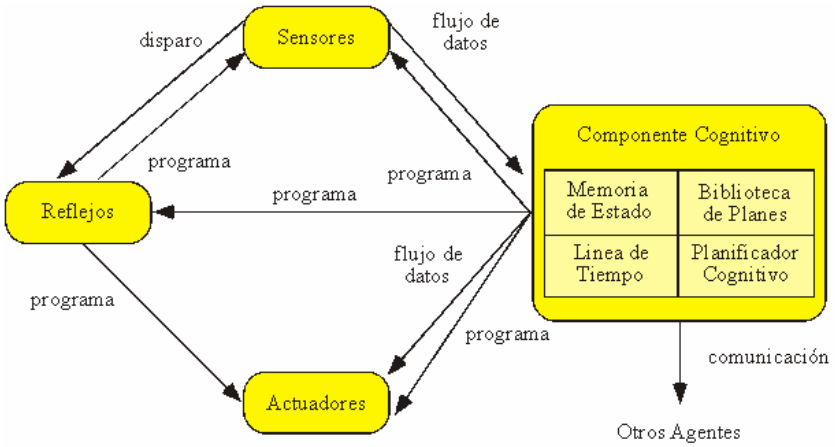


Figura 3.7. Arquitectura PHOENIX

A continuación se encuentran los módulos que generan la respuesta. Existen para ello dos módulos. El primero es el de acción refleja, compuesto de pares estímulo respuesta, cuya acción se calcula directamente con la llegada del estímulo del entorno y que se utiliza cuando no se dispone de suficiente tiempo para que entre el modulo de más alto nivel.

El módulo de alto nivel, o módulo cognitivo se encarga de calcular respuestas más complejas (y costosas), así como de realizar planificación, coordinación entre agentes, análisis de la percepción, monitorización de las acciones, etc. Es de destacar que PHOENIX presenta no sólo la arquitectura para construir los agentes, sino las herramientas para construir un entorno simulado (en concreto de control de extinción de bosques) para la validación de los agentes. Esto es especialmente importante debido a la

complejidad de los agentes es necesario disponer de entornos de simulación lo suficientemente ricos para realizar pruebas con garantías.

3.2.9 PRODIGY

Esta arquitectura [Vel95] es de tipo modular y se basa en la utilización de la lógica de primer orden como mecanismo de representación del conocimiento. La lógica es usada para representar las reglas de inferencia y de control, el meta-conocimiento y los hechos del problema.

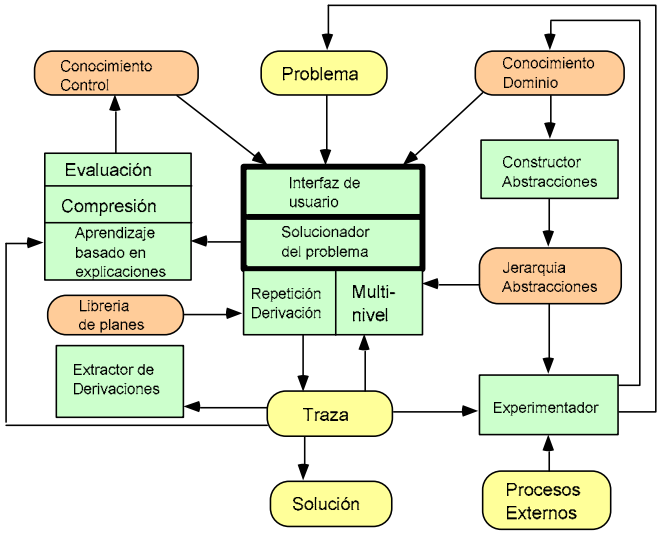


Figura 3.8. Arquitectura Prodigy

El control es bastante simple, ya que emplea una estrategia de búsqueda, donde a partir del conocimiento que dispone selecciona un nodo que expande y aplica todos los operadores posibles hasta llegar a la solución.

Esta arquitectura hace especial énfasis en el aprendizaje, pues permite aplicar simultáneamente diferentes técnicas, cada una llevada a cabo por un módulo de aprendizaje diferente y que no interfieren entre si.

3.2.10 PRS

PRS son las siglas de “Procedural Reasoning System” [Ing90, Ing92, Ing96] un sistema de razonamiento general para entornos de tiempo real.

Un agente PRS consta de una serie de elementos para realizar dicho razonamiento. Estos elementos son: una base de datos, con las creencias que tiene el agente sobre el estado actual del entorno; un conjunto de objetivos que deben ser alcanzados por el agente; una biblioteca de procedimientos denominados áreas de conocimiento que describen cómo y cuando actuar; y una estructura de intenciones, que consiste en un conjunto parcialmente ordenado de planes que indican que elemento elegir para ejecutar.

Otro de los elementos es el intérprete, este debe ser visto como un mecanismo de inferencia, que maneja los componentes anteriores, seleccionando planes en función de los datos existentes y de los objetivos, modifica la estructura de intenciones, y ejecuta los planes.

Las áreas de conocimiento describen, tal y como se ha indicado previamente, las acciones a llevar a cabo para resolver el problema. En realidad estas áreas no solo describen las operaciones que realizan sino también que condiciones deben darse en el estado actual y que eventos deben haberse producido para que dicha área pueda ejecutarse.

Además existen meta-áreas de conocimiento, que funcionan de forma similar a las anteriores pero cuyo objetivo es guiar el proceso de solución indicando al agente como debe manejar sus objetivos, creencias, etc., en función de la situación.

Finalmente indicar que la necesaria conexión con el exterior es efectuada en este caso a través de la propia base de datos, que dispone de sistemas de mantenimiento de la consistencia para mantener en un estado fiable las creencias que se tiene sobre el entorno.

3.2.11 RALPH-MEA

Esta arquitectura [Oga91] trabaja a varios niveles, siempre utilizando como factor principal en la búsqueda de soluciones el maximizar la utilidad esperada de la acción a tomar.

En un nivel superior, el sistema dispone de diferentes métodos de resolución de problemas, denominados por los autores arquitecturas de ejecución. Cada uno de estos módulos maneja un tipo de conocimiento diferente y presentan diferentes soluciones. Un módulo de arbitraje elegirá en función de la utilidad, cuál de las respuestas es la que interesa.

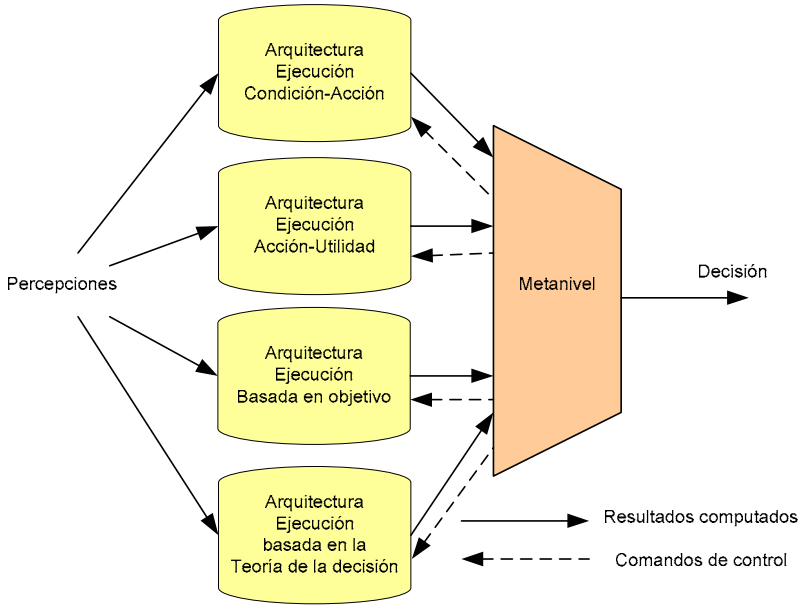


Figura 3.9. Arquitectura RALPH-MEA

A un nivel inferior, dentro de cada uno de estos módulos, la búsqueda de una solución se realiza en función de la situación del problema y del meta-conocimiento del que se dispone, guiándose también por la utilidad.

Esta arquitectura presenta también propiedades de tiempo real, pues uno de los módulos, denominado de condición-acción, proporciona acciones inmediatas en función del estado actual.

3.2.12 REAKT

REAKT [Men93, Ker94] (“Entorno y metodología para sistemas basados en el conocimiento de tiempo real”) es un sistema multiagente construido sobre una pizarra temporal que sirve como medio de comunicación entre los diferentes agentes.

Los agentes son instancias ejecutables de fuentes de conocimiento, que cooperaran para resolver los objetivos del sistema, y que poseen conocimiento en formato procedural o basado en reglas.

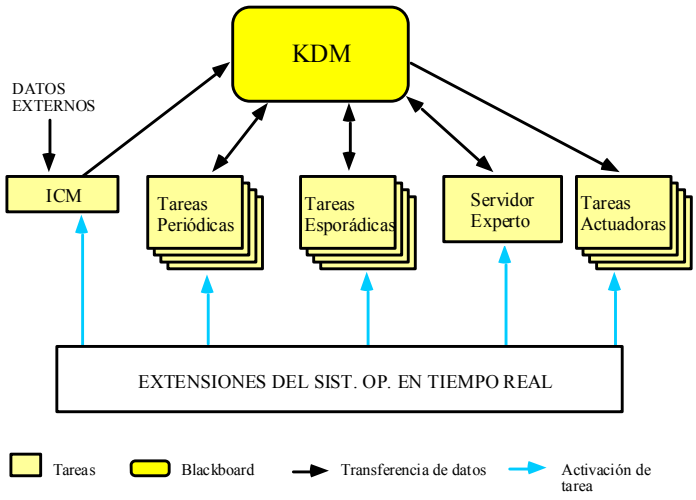


Figura 3.10. Arquitectura REAKT

Esta arquitectura presenta además de los agentes otros componentes para facilitar la integración de dichos agentes en el sistema global. Estos componentes son:

- El KDM, o gestor del conocimiento. Está diseñado como una pizarra o blackboard, donde se almacenan datos tanto estáticos como dinámicos. Sirve como medio de intercambio de información entre agentes. No se trata de un elemento estático, mero contenedor de datos. Dispone de dos módulos, el TMM ("Time Map Manager") y el RMS ("Reason Maintenance System"), encargados respectivamente de mantener las relaciones temporales y la coherencia lógica de los datos que almacena.

- Una serie de tareas periódicas que realizan operaciones determinadas cada cierto tiempo, como el ICM o gestor inteligente de comunicaciones, encargado de tomar los datos del exterior y filtrarlos, antes de ser depositados en el KDM y por tanto estar disponibles para los agentes.

- Tareas de actuación. Igual que con la lectura de los datos de los sensores, las operaciones con los actuadores no son realizadas por los agentes en si, sino a través de estos módulos. El objetivo de realizarlo así, es tener un mayor control sobre las operaciones del sistema y con llevar mejorar las prestaciones para su aplicación en problemas de tiempo real.

- Tareas esporádicas. Encargadas de resolver de forma inmediata eventos inesperados pero que requieren una respuesta rápida.

- Modulo de extensiones del sistema operativo en tiempo real, para permitir la comunicación entre los diferentes elementos.

- Servidor experto. Va a ser el encargado de la gestión de los agentes. Para ello dispondrá de un control que en función de los eventos del sistema creará y planificará intenciones. Las intenciones describen que agentes deben participar a la hora de resolver un problema en cuestión

3.2.13 SOAR

Esta arquitectura esta basada en la existencia de diferentes estrategias de control [Lai87, Lai93, Lai94], que aunque dirigidas por el objetivo, presentan diferentes alternativas a la hora de tratar los datos (que toman la forma de pares atributo-valor) y por tanto de obtener los resultados.

Entre las estrategias se encuentran las que están basadas en ciclos de decisiones secuenciales o en paralelo, las guiadas por interrupciones, que le permiten un comportamiento más reactivo, o las basadas por la no existencia de valores.

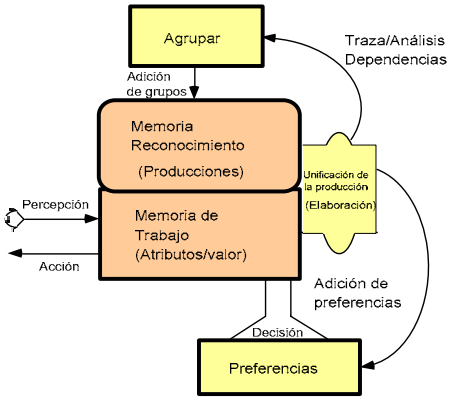


Figura 3.11. Arquitectura SOAR

La estrategia basada en la no existencia de valores basa su funcionamiento en completar aquellos atributos sin valor. Para ello si no consigue determinar un valor para un atributo, por ejemplo por la existencia de ambigüedades, el sistema dividirá el objetivo en subobjetivos, de forma que sean estos los

que guíen ahora el proceso de búsqueda, en principio de una forma más efectiva al ser más específicos.

3.2.14 Arquitectura de Sumsunción

Arquitectura [Bro86, Bro91] desarrollada para trabajar en entornos de tiempo real se caracteriza por tener una alta reactividad aunque a costa de perder flexibilidad, pues parte de comportamientos prefijados.

La estructura que presenta esta arquitectura es la de una división en niveles según la tarea a resolver. Las capas inferiores proporcionan comportamientos simples, comportamientos que aumentan de complejidad según se sube de nivel. Cada capa presenta un comportamiento prefijado, de acuerdo a la complejidad del nivel.

El control no está centralizado sino distribuido entre las diferentes capas, capas que consumen pocos recursos individualmente pero que trabajarán de forma conjunta, como en el ejemplo de la figura 3.12.

En principio no dispone de una representación explicita del conocimiento, sino que este se encuentra imbuido en las propias reglas de comportamiento. Esto tiene la ventaja de no tener que preocuparse de actualizar dicho conocimiento lo cual es costoso, con lo cual se mejora su reactividad. El principal inconveniente es que lo hace muy poco flexible, y excesivamente dependiente de lo que recibe del entorno.

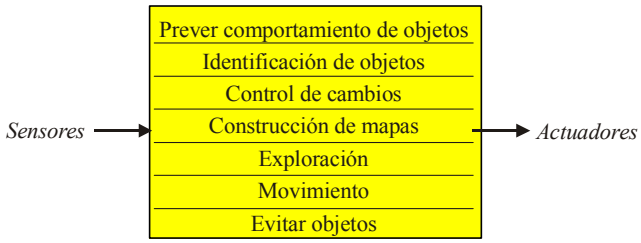


Figura 3.12. Ejemplo de capas en la arquitectura de sumsunción

Para aumentar la flexibilidad se han propuesto modificaciones, como la arquitectura basada en el comportamiento [Mat91, Mat92], que presenta como incorporar pequeñas representaciones de entorno sin perder la reactividad. Además se incorporan mecanismos de aprendizaje aplicados a las relaciones entre las diferentes capas.

3.2.15 TETON

Esta arquitectura está basada en buscar la mayor simplicidad en el funcionamiento para así mejorar sus prestaciones [Van91].

El ciclo de ejecución es muy simple, consta básicamente de dos pasos: determinar cual es la tarea a ejecutar en ese momento, decisión que toma en base a heurísticas y ejecutarla. Similar a arquitecturas anteriores, presenta un modo de funcionamiento donde ni siquiera esa pequeña planificación previa es realizada, de manera que la reactividad sea mayor. También incorpora el mecanismo de subdividir el objetivo en subobjetivos cuando no puede tratar directamente el primero.

En cuanto a la información, también busca simplificar su almacenamiento, por ello mantiene dos memorias. La primera mantiene la información que los agentes usan para el funcionamiento normal. La segunda, de largo plazo, contiene reglas objetivos, y meta-conocimiento derivado del aprendizaje como determinar aquellos procesos que le son más útiles.

3.2.16 THEO

Esta arquitectura está pensada fundamentalmente para el aprendizaje [Mit91], pues permite incorporar diferentes métodos para realizarlo. Por ejemplo, tiene un módulo que realiza aprendizaje estadístico inferencial aplicado a los resultados que obtienen los diferentes métodos de resolución de los que dispone. De esta manera puede ordenar los métodos en función de su eficacia para poder utilizar posteriormente primero aquellos que se espera obtengan mejores resultados.

En cuanto a la forma de resolver el problema hay que tener en cuenta que utiliza una representación del conocimiento basada en frames, por lo que sigue una estrategia usual en este tipo de representación y ya vista en arquitecturas anteriores, como es la de buscar valores para aquellos datos que carezcan de ellos o ya no sean validos. La gran cantidad de datos que tiene que manejar es su principal problema, y le hacen necesario el uso de un sistema de mantenimiento de la razón para mantener la validez de la información que almacena.

3.2.17 CELLO

Arquitectura pensada para el desarrollo de agentes usados en el control de sistemas de tiempo real y de robots móviles [Occ98].

El modelo que propone presenta un agente con capacidades tanto reactivas como deliberativas, basadas estas últimas en el paradigma de percepción/decisión/razonamiento/acción.

La integración de este comportamiento deliberativo con el reactivo es lograda a través del paralelismo en la propia estructura de los agentes. En concreto la arquitectura muestra un conjunto de módulos, independientes entre si, que funcionan de forma asíncrona y que permiten muchos de ellos trabajar directamente sobre el entorno. La organización de todos estos módulos está basada en un modelo de blackboard especialmente diseñado para sistemas de tiempo real embebidos. Los cambios en dicho blackboard provocaran la reacción de los diferentes módulos.

3.3 CONCLUSIONES

Se puede observar en los puntos anteriores el rango que existe a la hora de diseñar arquitecturas para agentes. Mientras que en la mayoría de ellas se dispone de elementos que permitan la percepción y actuación en el entorno, tal y como aparece en la práctica totalidad de definiciones de agentes, la existencia de elementos más complejos, como el aprendizaje o la relación entre agentes varía de unas arquitecturas a otras.

Es interesante, dado nuestro propósito de aplicar tareas inteligentes en tiempo real, como tratan este tema las diferentes arquitecturas que hemos descrito. Muchas de ellas meramente incorporan un componente reactivo, con poca o ninguna relación con el comportamiento inteligente y en cualquier caso no dejan de ser un modo de respuesta rápido, algo que por si solo no puede considerarse de tiempo real. Otras arquitecturas ya presentan ciertas características de tiempo real, aunque en la mayoría de los casos se trata de lo que se considera como tiempo real no estricto, en el sentido de que el estricto cumplimiento de los tiempos límites no es obligatorio para el funcionamiento del sistema.

4 LA ARQUITECTURA ARTIS

ARTIS [Gar95] es una arquitectura para el diseño de agentes inteligentes con capacidad para actuar en entornos de tiempo real estricto. El objetivo es disponer de agentes que no sólo sean capaces de garantizar respuestas dentro de los límites temporales requeridos por un entorno de tiempo real [Bot99], sino además que dichas respuestas presenten la mayor calidad posible. Las técnicas de planificación basadas en la utilidad desarrolladas en este trabajo serán aplicadas a esta arquitectura, por lo que este capítulo tratará de describirla haciendo hincapié en los aspectos que más afectarán a dicha planificación.

4.1 IDEAS GENERALES

ARTIS es una arquitectura diseñada como una extensión del modelo de blackboard [Nii86a], adaptado para trabajar en entornos de tiempo real estricto. El sistema garantiza cumplir los requerimientos de tiempo real estricto cumpliendo los deadlines y a su vez proporciona la flexibilidad para aplicar técnicas de IA que permitan calcular respuestas de mayor calidad.

El agente construido con ARTIS será capaz de percibir información del entorno a través de sensores, calcular respuestas (de forma reactiva o con un mayor o menor grado de deliberación) y finalmente actuar en el exterior conforme a la respuesta usando actuadores o bien transmitiendo dichas respuestas a otros agentes.

A la hora de describir la arquitectura se hará en primer lugar viendo como sería su modelo de alto nivel tanto en un plano funcional como estructural.

Este modelo de alto nivel se traduce en un modelo de entidades de bajo nivel, que es el que se implementa y que también describiremos.

4.2 MODELO DE ENTIDADES DE ALTO NIVEL

En primer lugar vamos a presentar una descripción formal del agente ARTIS y de sus componentes.

Definición 1. Un agente ARTIS es una estructura:

$$AA = (\text{Behav}, G, B, \beta)$$

Donde:

- Behav, es el conjunto de diferentes comportamientos del agente para enfrentarse a diferentes situaciones.
- $G = \{g_1^{[t_1,t_2]}, g_2^{[t_2,t_3]}, ..., g_n^{[t_n,t_m]}\}$, un conjunto de objetivos a alcanzar por el agente. Cada objetivo es un estado deseable y puede estar acotado temporalmente, en cuyo caso se denotaría el intervalo temporal en el cual el objetivo debe ser alcanzado.
- B es el conjunto de creencias del agente en el instante actual. Cada dato dispone de una etiqueta temporal que define su validez y que permite su manejo usando una lógica temporal [Cre94].
- $\beta : G \times B \rightarrow \text{Behav}$, es una función de selección que determina el comportamiento actual del agente en función de sus objetivos y creencias actuales.

Definición 2. Cada comportamiento beh $\in$ Behav, es un conjunto de in-agents o agentes internos:

$$\text{beh} = \{a_j\}$$

De esta manera el esquema básico de un agente ARTIS se puede ver en la siguiente figura.

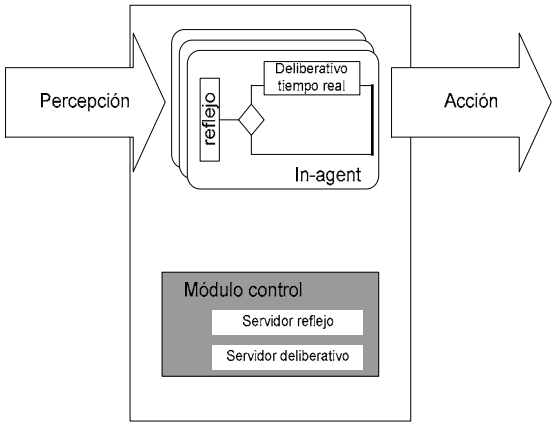


Figura 4.1. Arquitectura de un agente ARTIS

En esta figura se puede ver como el agente contiene un conjunto de in-agents, cada uno de ellos realizando operaciones de percepción, cognitivas y de acción, y por otro lado dispone de un módulo de control. Dicho módulo será el que implemente la función β comentada anteriormente. Dicho modulo dispondrá de dos componentes, el servidor reflejo y el deliberativo encargados cada uno de ellos de la gestión de los diferentes componentes que forman parte de un in-agent y que se comentan a continuación.

Definición 3. Un in-agent es una estructura

$$a_j = \langle \rho_j, f\rho_{sel}, \sigma_j, f\sigma_{sel}, B_j, D_j, T_j \rangle$$

donde:

- ρ_j es el conjunto de acciones reflejas que puede ejecutar el in-agent
- $f\rho_{sel}: B_j \times (D_j, T_j) \rightarrow \rho_j$, es una función de selección que determina la acción refleja a ejecutar por el in-agent más conveniente de acuerdo a su estado actual y a sus restricciones temporales.
- σ_j es el conjunto de acciones cognitivas que puede ejecutar el in-agent.
- $f\sigma_{sel}: B_j \times (D_j, T_j) \rightarrow \sigma_j$, es una función de selección que determina la acción deliberativa a ejecutar por el in-agent más conveniente de acuerdo a su estado actual y a sus restricciones temporales.

- $B_j \subset B$, es el conjunto de creencias que representan tanto el estado del entorno como el estado interno del in-agent. Estas creencias forman parte del conjunto de creencias (B) del agente ARTIS al que el in-agent pertenezca.

- D_j es el deadline o plazo máximo de ejecución del in-agent.

- T_j es el periodo de activación del in-agent. Al comienzo de cada periodo el in-agent leerá los nuevos datos de entrada y con dichos datos se actualizará su conjunto B_j.

Cada in-agent da una solución para un problema particular del agente ARTIS y será un elemento primordial de esta arquitectura. Reciben este nombre debido al hecho de que se comportan como agentes (aunque de funcionalidad limitada). Un agente ARTIS tendrá por tanto diferentes in-agents que trabajaran de forma colaborativa para actuar según el comportamiento dominante en el momento actual. Cada in-agent percibirá del entorno y calculará acciones de manera refleja y/o deliberativa.

El comportamiento de un in-agent viene reflejado por el siguiente ciclo de funcionamiento que se repetirá cada periodo:

Percepción(t) ← Leer(Entorno, t)

$B_j \leftarrow B_j \cup$ Percepción(t)

(*B*j, Acción_refleja) ← $f\rho_{sel}(B_j, \rho_j)$

(*B*j, Acción_ deliberativa) ← $f\sigma_{sel}(B_j, \sigma_j$, Acción_refleja, $D_j)$

if (Acción_ deliberativa = no_completada)

Ejecutar(Acción_refleja)

sino Ejecutar(Acción_ deliberativa)

Como se puede apreciar en la descripción anterior, el in-agent calcula primero una acción refleja basándose en su conocimiento actual y posteriormente intenta calcular una acción deliberativa teniendo en cuenta además de su estado la acción refleja calculada previamente. Si el in-agent no consigue calcular dicha acción deliberativa, hará uso de la reactiva calculada previamente como acción a ejecutar.

Para calcular la acción refleja o deliberativa, el in-agent utiliza las funciones de selección f_{sel} correspondientes. Estas funciones están implementadas por

los denominados niveles. Un nivel puede ser código procedural, reglas de producción, redes neuronales, etc.

Definición 4. Un nivel reactivo $\mathcal{L}r^i_j$ es una estructura

$\mathcal{L}r^i_j = (\theta^i_j, Tw_{ij}, q_{ij})$

donde:

- θ^i_j es la representación del conocimiento que calcula las acciones reactivas del nivel.
- Tw_{ij} es el tiempo de ejecución en el peor de los casos de θ^i_j.
- $q_{ij} \in]0, 1]$ es la calidad de la respuesta calculada por el nivel.

Definición 5. Un nivel deliberativo $\mathcal{L}d^i_j$ es una estructura

$\mathcal{L}d^i_j = (\theta^i_j, Tm_{ij}, q_{ij})$

donde:

- θ^i_j es la representación del conocimiento que calcula las acciones deliberativas del nivel.
- Tm_{ij} es el tiempo medio de ejecución de θ^i_j.
- $q_{ij} \in]0, 1]$ es la calidad de la respuesta calculada por el nivel.

De esta manera un nivel, ya sea reactivo o cognitivo, será la unidad mínima de ejecución del sistema. Cuando se ejecuta un nivel este produce un resultado que lleva asociada una calidad. Esta calidad será utilizada como medida de la eficiencia en el desempeño de su función del agente ARTIS.

La diferencia en cuanto a los tiempos utilizados (tiempo medio para deliberativos y en el peor de los casos para reactivos), viene dada por su propia naturaleza y en como se utilizarán, tal y como se verá en el apartado dedicado a la planificación.

Básicamente, los niveles reactivos tienen que tener garantizada su ejecución en un tiempo determinado, ya que pueden ser usados para operaciones críticas del sistema, con lo cual el conjunto de todos ellos deben satisfacer

un test de planificabilidad como los indicados en el capitulo anterior. Esto solo puede hacerse conociendo su tiempo de ejecución en el caso peor.

Por el contrario, los niveles deliberativos, tal y como se ha visto en el ciclo de ejecución de un in-agent, van a ser utilizados para mejorar la solución del reactivo por lo que su ejecución no es obligatoria. Sin embargo estos niveles suelen hacer uso de técnicas inteligentes que en la mayoría de los casos implican procesos de búsqueda. En estas técnicas un tiempo en el peor de los casos no puede ser calculado, o dicho tiempo no resulta representativo en la práctica, por lo que se utiliza el tiempo medio.

Las diferentes estructuras vistas hasta el momento (agente, in-agent, nivel) conformarían la jerarquía básica de la arquitectura ARTIS. Sin embargo desde el punto de vista estructural aparece un nuevo elemento: el MKS o fuente de conocimiento múltiple. Este elemento surge para agrupar aquellos niveles de un in-agent que intentan resolver el mismo subproblema, pero con una calidad de resultado y unos requerimientos diferentes. De esta manera se proporciona un modelo computacional que permite integrar técnicas de IA, en general con tiempos de cómputo no acotados para el caso peor, en sistemas de tiempo real, proporcionando una respuesta de calidad mínima acotada en el tiempo, que puede ser mejorada, si hay tiempo, e interrumpida en cualquier instante.

Definición 6. Un MKS perteneciente a un in-agent a_j es una estructura

$MKS_{ij} = (Lr_{ij}, \cup Ld_{ij}, I_{ij})$

donde:

- Lr_{ij} es un nivel reactivo correspondiente al conjunto de acciones reactivas del agente j. Pueden existir MKS sin nivel reactivo.

- $\cup Ld_{ij}$ es un subconjunto de niveles deliberativos (Ld^1_{ij}, Ld^2_{ij}, …, Ld^n_{ij}) pertenecientes al conjunto de niveles deliberativos del agente j. Todos estos niveles resuelven el mismo problema que el nivel reactivo Lr_{ij}. Este conjunto puede ser vacío.

- I_{ij} es la importancia del MKS. Es una medida de la relevancia del MKS y es utilizada para calcular la calidad total del sistema. La calidad proporcionada al ejecutar un nivel será finalmente la calidad de dicho nivel multiplicada por la importancia del MKS al que pertenece.

De esta manera un in-agent estará compuesto de uno o varios MKS, cada uno de ellos encargados de resolver un subproblema de entre los que puede ser descompuesto el problema que resuelve el in-agent al que pertenecen. Para calcular dicha respuesta dispondrá de un nivel reactivo y/o varios deliberativos.

Existen diferentes tipos de MKS en función además de diferentes criterios. Una primera clasificación sería en cuanto a la limitación o no del tiempo en que puede proporcionar la respuesta requerida. De esta manera tendríamos:

- MKS limitados. Aquellos con un tiempo de cómputo para el caso peor predecible y realista. Dispondrán de un nivel reactivo (nivel-0) que garantizará el cumplimiento de dicho deadline y puede tener además de forma opcional de un conjunto de niveles deliberativos (nivel-1, nivel-2, …, nivel-n), estos niveles no tienen porque disponer de un tiempo predecible o realista.

- MKS no limitados. No tienen tiempo en el caso peor predecible o bien este no es realista. No disponen de nivel reactivo, y si del conjunto de niveles deliberativos.

Muy relacionada con esta clasificación estaría otra que divide los MKS en críticos y no críticos:

- MKS críticos. Deben proporcionar obligatoriamente una respuesta antes del deadline del in-agent al que pertenecen. Por ello serán MKS limitados que proporcionarán dicha respuesta en primer lugar mediante su nivel reactivo, el cual debe garantizarse que se ejecutará totalmente. En el caso de disponer más tiempo se ejecutarán uno o más niveles deliberativos, siendo el resultado proporcionado por el MKS el perteneciente a su nivel deliberativo que proporcione mayor calidad y que se haya ejecutado en su totalidad. En caso de no ejecutar ninguno, sería el proporcionado por el nivel reactivo.

- MKS acríticos. Aquellos encargados de operaciones cuya no realización proporcionará una disminución en la calidad del sistema pero no supone riesgo sobre la integridad del sistema. Pueden ser tanto MKS limitados o no limitados, y cualquier nivel (incluido el 0 puede ser interrumpido).

Atendiendo al papel que lleva a cabo el MKS dentro del in-agent nos encontraríamos con:

- MKS de percepción. Encargados de percibir del entorno la información que le interesa al in-agent. Únicamente MKS limitados pueden ser de este tipo.

- MKS cognitivos. Encargados de calcular las respuestas, tanto reflejas como deliberativas. Estos MKS pueden ser tanto limitados como no limitados.

- MKS de acción. Aquellos que realizan las acciones calculadas sobre el entorno. Los MKS de acción serán siempre MKS limitados con un sólo nivel el nivel-0.

Otra clasificación de los MKS vendría dada por el tipo de relación existente entre los diferentes niveles deliberativos del MKS. Esta relación es muy importante pues condiciona de gran manera los métodos de planificación que pueden ser utilizados para seleccionar el nivel a ejecutar. Dos serían los tipos existentes en la arquitectura:

- MKS métodos múltiples.

Estos MKS responden a la idea que se comento en el capitulo 2 de métodos múltiples. En este caso todos los niveles que forman el MKS son métodos alternativos para resolver el mismo problema, con diferentes consumos de recursos (es decir tiempo de cpu) y diferentes calidades de resultado y sobre todo sin ninguna relación entre un nivel y otro. Es decir no existen dependencias en cuanto a utilización de resultados o de otro tipo entre diferentes niveles. Por ello la ejecución de un nivel es totalmente independiente de otros niveles.

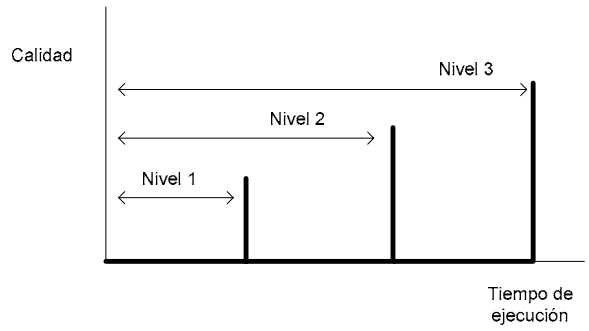


Figura 4.2. Perfil de ejecución de MKS métodos múltiples

Como se aprecia en el ejemplo de la figura 4.2 existen para ese MKS 4 niveles con diferentes tiempos de ejecución (eje x) y diferentes calidades de resultados. En cualquier caso cada nivel se debe ejecutar en su totalidad para producir un resultado. Si formalizamos el perfil de ejecución del ejemplo tendremos:

Sea *f*(t) la calidad resultante del MKS, que dispone de los siguientes niveles:

L1: $<t_1\ q_1>$, L2: $<t_2\ q_2>$, L3: $<t_3\ q_3>$, L4: $<t_4\ q_4>$, siendo cada t el tiempo medio de ejecución de la tarea y q su calidad, entonces:

$$f(t) = \begin{cases} 0 & 0 < t < t_1 \\ q_1 & t_1 \le t < t_2 \\ q_2 & t_2 \le t < t_3 \\ q_3 & t_3 \le t < t_4 \\ q_4 & t_4 \le t \end{cases}$$

- MKS de refinamiento progresivo.

  Están basados en el formalismo de Progressive Deepening [Mou92]. En este caso los niveles que lo forman también resuelven el mismo problema con consumos de recursos (cpu) y calidad resultante diferentes. La diferencia con los MKS anteriores estriba en el hecho de que en este caso si existe una relación de dependencia entre niveles, de manera que un nivel utiliza los resultados del nivel anterior (o al menos una parte de ellos) como un subconjunto de sus datos de entrada.

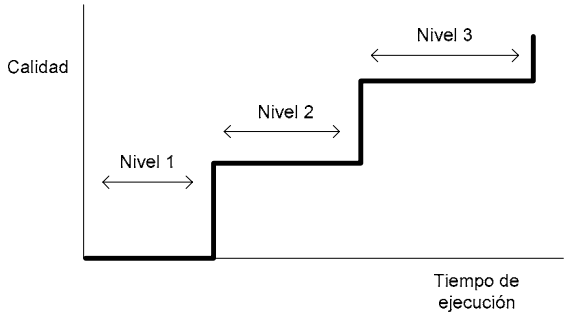


Figura 4.3. Perfil de ejecución de MKS por refinamiento progresivo

En este caso el conjunto de niveles de un MKS estará ordenado, de manera que un nivel sólo se ejecutará si previamente se ha ejecutado el nivel anterior. Primero se ejecutarán aquellos niveles que proporcionan una solución de menor calidad y posteriormente los niveles que necesitan utilizar los resultados previos y que aumentarán la calidad resultante, como se observa en la figura 4.3.

Este tipo de MKS puede verse como una variación de los algoritmos anytime en el sentido de que al igual que en ellos la calidad aumenta cuanto más tiempo se concede para la ejecución y además el resultado se va refinando. La diferencia consiste básicamente en el hecho de que el perfil del algoritmo anytime es continuo, con lo cual en todo momento puede ser interrumpido una vez se ha ejecutado el nivel 0, y el resultado calculado en el momento previo a la interrupción será valido. En el refinamiento progresivo, además de tratarse de varios métodos y no uno sólo, el perfil es escalonado, con lo cual si se interrumpe, el tiempo en que ha estado ejecutándose el nivel actual que no ha llegado a su terminación se puede considerar tiempo perdido, y el resultado valido será el calculado por el nivel precedente que si ha podido ejecutarse en su totalidad.

La ventaja frente a los anytime consiste en la disponibilidad de estos métodos que es mucho mayor, pues los anytime a pesar de sus bondades para su utilización en entornos de tiempo real tienen la tara de ser difíciles de construir para muchos problemas.

Si formalizamos el perfil de ejecución del ejemplo tendremos:

Sea $f(t)$ la calidad resultante del MKS, que dispone de los siguientes niveles:

L1: $<t_1\ q_1>$, L2: $<t_2\ q_2>$, L3: $<t_3\ q_3>$, L4: $<t_4\ q_4>$, siendo cada t el tiempo medio de ejecución de la tarea y q su calidad, entonces:

$$f(t) = \begin{cases} 0 & 0 < t < t_1 \\ q_1 & t_1 \le t < t_1 + t_2 \\ q_1 + q_2 & t_1 + t_2 \le t < t_1 + t_2 + t_3 \\ q_1 + q_2 + q_3 & t_1 + t_2 + t_3 \le t < t_1 + t_2 + t_3 + t_4 \\ q_1 + q_2 + q_3 + q_4 & t_1 + t_2 + t_3 + t_4 \le t \end{cases}$$

En este caso puede observarse que los tiempos son acumulativos, pues para ejecutar el tercer nivel se deben haber ejecutado los dos anteriores. La calidad también es acumulativa respecto a los niveles anteriores.

4.3 MODELO DE ENTIDADES DE BAJO NIVEL.

A la hora de implementar el modelo de agente ARTIS en un sistema operativo real, dicho modelo se traduce a un modelo que denominaremos de bajo nivel. Dicho modelo está basado en el modelo de pizarra (blackboard) que suele estar compuesto por tres elementos [Nii86a] [Nii86b]:

- Blackboard. Es una memoria global, que contiene objetos del espacio de soluciones. Estos objetos están estructurados de forma jerárquica en niveles y pueden estar enlazados entre si. Es el único medio de comunicación entre agentes. Cuando en el blackboard se producen cambios significativos se producen eventos para informar al control.

- Agentes. Son representaciones de resolución de problemas. Cada agente es independiente, especializado en una tarea y capaz de resolver un subproblema del problema global. Los agentes leerán y escribirán en el blackboard, de esta manera la solución se construye incrementalmente y de forma cooperativa entre todos los agentes.

- Control. Encargado de seleccionar que secuencia de agentes es la apropiada para solucionar el problema actual. También puede encargarse de dirigir la cooperación entre agentes.

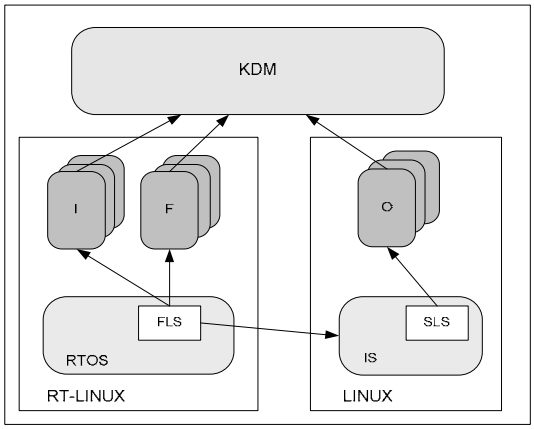


Figura 4.4. Arquitectura de bajo nivel de un agente ARTIS

Tomando como base el modelo general de blackboard aparece el modelo de agente ARTIS de bajo nivel. Dicho modelo estará constituido (Figura 4.4) por un conjunto de tareas, traducción de los in-agents, y que serán los componentes a ser planificados.

Existen además otra serie de componentes como el RTOS, el servidor inteligente, encargado de la planificación de los niveles opcionales y el blackboard o memoria global, que mantiene el conocimiento del agente. Todos estos componentes serán comentados a continuación.

Un elemento muy importante a la hora de comprender esta arquitectura, es el hecho de que esta no se ejecuta sobre un único sistema operativo, sino sobre dos simultáneamente. El sistema se está ejecutando en una maquina bajo el control de RT-Linux. La utilización de este sistema operativo, diseñado especialmente para aplicaciones de tiempo real, permitirá garantizar (junto a otros componentes) el cumplimiento de los requerimientos temporales. Como veremos más adelante, en este sistema se ejecutarán los componentes críticos del agente.

Para ser usado en ARTIS, RT-Linux ha sido extendido para que pueda proporcionar más servicios. Este modelo es llamado Flexible RT-Linux [Ter02].

En el sistema también esta funcionando otro sistema operativo, Linux, este recibirá del RTOS el control de la máquina cuando se trate de llevar a cabo las operaciones no críticas del sistema, como encargarse del comportamiento deliberativo del agente y otras operaciones.

4.3.1 Modelo de tarea

En la descripción de alto nivel hemos visto como cada in-agent está compuesto de una serie de niveles agrupados en MKS. En el modelo de bajo nivel, cada uno de estos in-agents definidos por el usuario es compilado, produciendo como resultado un conjunto de tareas de bajo nivel que deberán ser planificadas para cumplir los requerimientos del sistema y la mejor calidad de resultado global.

La tarea resultante de la compilación de un in-agent tiene el mismo periodo y deadline que éste, y está compuesto por tres partes diferenciadas:

- Parte obligatoria inicial. Compuesta por los niveles 0 de los MKS críticos encargados de la percepción o cognición del in-agent.

- Parte opcional. Compuesta por todos los niveles de los MKS no críticos y por los niveles 1 o superiores de MKS críticos, en ambos casos también de componentes de percepción o cognición.

- Parte obligatoria final. Formada por los niveles 0 de los MKS de acción (los únicos niveles que tienen estos MKS).

Las partes inicial y final de cada tarea corresponden con los elementos críticos del in-agent, cuya ejecución debe garantizarse para cumplir los requerimientos. Para ello deberán cumplir una serie de características como se comentarán en el apartado de planificación.

La parte inicial de una tarea tenderá a ejecutarse lo antes posible dentro de su periodo, y la parte final ajustará su terminación a su deadline. El tiempo disponible entre la ejecución de ambas partes, será el tiempo que puede ser aprovechado para ejecutar los diferentes niveles de que puede constar la parte opcional de la tarea y refinar las soluciones calculadas por los niveles 0 antes de ejecutar las acciones que constituyen las soluciones.

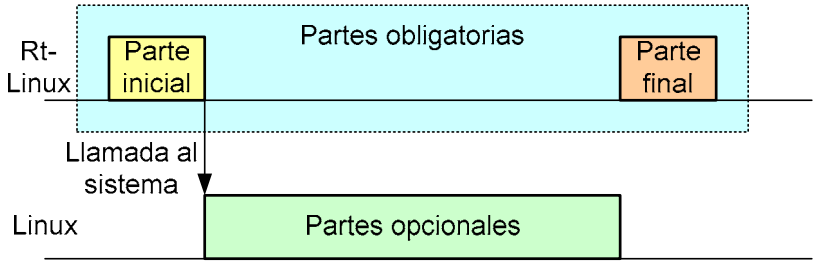


Figura 4.5. Componentes de una tarea

Con el fin de ilustrar la relación existente entre los in-agents y las tareas de bajo nivel, la figura 4.6 muestra un ejemplo simple de la traducción desde el modelo de alto nivel al de bajo nivel. En el ejemplo, un in-agent debe controlar que un depósito de aguas residuales mantenga su contenido dentro de unos límites determinados de temperatura y pH.

Para ello, el in-agent dispone de un MKS de percepción, que dispone de un nivel crítico que lee los valores actuales de ambos parámetros. Para calcular las respuestas, dispone de dos MKS de cognición. El primero controla la temperatura, calculando la cantidad de agua fría o caliente que debe ser añadida. Este MKS dispone de tres niveles, un nivel 0, que calcula la respuesta en un tiempo limitado, y dos niveles opcionales, que refinan esta solución. El otro MKS calcula la cantidad de componentes

químicos a añadir para controlar el pH, y tiene una distribución en niveles similar al MKS anterior. Finalmente un MKS de acción abrirá o cerrará válvulas de agua o componentes químicos de acuerdo a las respuestas calculadas por los MKS previos.

En el modelo de bajo nivel resultante de la traducción del in-agent, el componente crítico tiene una parte inicial compuesta por los niveles 0 de los MKS de percepción, temperatura y pH. La parte final esta compuesta por el nivel 0 del MKS de acción. El componente opcional está compuesto por cuatro niveles, los niveles opcionales de los MKS de temperatura y pH. Los niveles mantienen relaciones de precedencia entre si: En la parte inicial, el nivel1_0 debe ejecutarse antes que el nivel2_0 y el nivel3_0. También hay relaciones de precedencia entre el nivel2_1 y el nivel2_2 y entre el nivel3_1 y el nivel3_2.

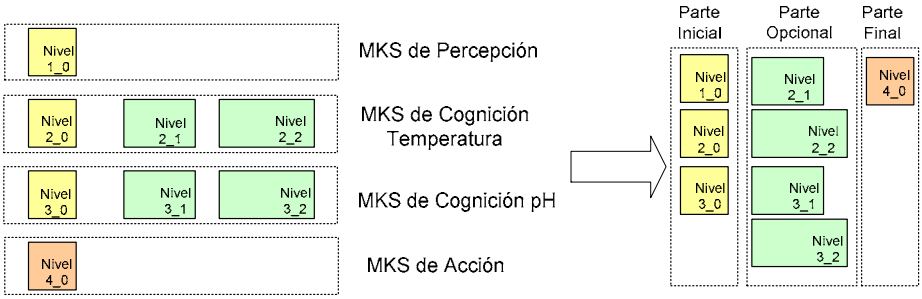


Figura 4.6. Ejemplo de traducción de un in-agent al modelo de tareas de bajo nivel

4.3.2 RTOS

El RTOS o Sistema Operativo de Tiempo Real es un módulo encargado de proporcionar llamadas al sistema para crear procesos internos, comunicarlos y sincronizarlos. Además incluye parte de las funcionalidades que haría el modulo de control en una arquitectura de blackboard. En concreto uno de sus componentes es el FLS (First Level Scheduler) o planificador de primer nivel. Este planificador será el encargado de la planificación de las partes iniciales y finales de las tareas, y del servidor inteligente. En el punto de planificación se comentará como realiza su función.

Este modulo se encarga por tanto de las tareas críticas y por ello está implementado bajo RT-Linux.

4.3.3 Servidor inteligente

El servidor inteligente (IS) es otro módulo del agente ARTIS, en este caso funcionando en la parte LINUX del sistema. Aunque esta encargado de diversas operaciones la más interesante desde nuestro punto de vista es la que realiza uno de sus componentes, el planificador de segundo nivel o SLS (Second Level Scheduler).

El SLS formará junto al FLS el módulo de control de un sistema de blackboard [Hay90]. En este caso el SLS se encargará de planificar las partes opcionales de las tareas, también se ocupará de la posible carga aperiódica, tareas que no están sujetas a un periodo de activación y que en algún momento pueden activarse y precisar de respuesta. La carga aperiódica en ningún caso puede ser crítica, pues en ese caso sería necesaria la garantía de su ejecución y para ello el sistema necesitará que fuesen tratadas de forma periódica.

En cualquier caso, el SLS se va a encargar de la planificación de todos aquellos componentes que no son críticos, por ello el SLS no va a garantizar su ejecución, sino que tal y como veremos en el apartado de la planificación va buscar la secuencia de ejecución de estos niveles que proporciona la mayor calidad de salida posible.

4.3.4 Blackboard

El blackboard o KDM (Knowledge Data Manager)[Bar94a][Bot93][Bot95], es una memoria global estructurada que contiene los objetos del espacio de la solución. Constituye además el medio de comunicación entre in-agents, a través de la cual se pasarán la información que requieran unos de otros.

El blackboard no es un elemento pasivo dentro de la arquitectura. Aunque es un almacén de datos, además dispone de mecanismos de control sobre estos datos. Esto es debido al hecho de que gran parte de la información que mantenga va a ser información temporal. Hay que tener en cuenta que el agente final va a trabajar en sistemas de tiempo real, resolviendo problemas de naturaleza dinámica. Los hechos que mantendrá el blackboard serán ciertos solo durante ciertos intervalos de tiempo, tras los cuales nuevos valores los sustituirán. Para que el agente, (y por tanto los in-agents a través de los MKS) sean capaz de razonar con esta información, estos datos deberán indicar cual es su plazo de validez.

El hecho de manejar información temporal no afecta únicamente a los MKS, sino que el sistema deberá mantener la coherencia temporal de los datos del blackboard [Bar94b][Bot98]. Esto se realiza por medio de un módulo del propio blackboard denominado TMM (Time Map Manager), encargado de gestionar estas relaciones temporales entre datos.

Además el sistema incluye un mecanismo por el que liga determinados datos con los in-agents que están interesados en dichos datos. De esta manera, si el dato es modificado se informará a los in-agents relacionados. Este mecanismo puede ser utilizado para realizar operaciones de mantenimiento de la razón, disparo de reglas de meta-razonamiento o incluso para la propia planificación, tal y como se propondrá en este trabajo.

4.4 PLANIFICACIÓN

Como se ha comentado anteriormente el objetivo del sistema es conseguir que el agente sea capaz de trabajar en un entorno de tiempo real, ofreciendo las respuestas en el tiempo adecuado y procurando conseguir la mejor calidad de las respuestas. Como las tareas que forman el agente tienen que competir por el tiempo de procesador, se hace necesario que un sistema de control planifique la ejecución de los componentes de las tareas. Esta planificación se lleva a cabo a dos niveles.

Dado que el conseguir cumplir con todas las tareas a tiempo se consigue ejecutando los componentes inicial y final de cada tarea dentro de los límites temporales, existirá una planificación que denominaremos de primer nivel, encargada de la planificación de estos niveles. Este nivel se encargará por tanto de garantizar que el agente cumple con los requerimientos de tiempo real.

El cumplimiento de la tarea anterior deja tiempo de procesador no ocupado. Este tiempo será utilizado por un módulo del control, el servidor inteligente, para ejecutar las partes opcionales de las tareas (es decir sus niveles superiores) y la carga aperiódica. Para ello el servidor inteligente lleva a cabo una planificación que denominaremos de segundo nivel y que estará encaminada a obtener la mayor calidad global de resultado en el tiempo disponible.

4.4.1 Planificación de primer nivel

Esta planificación debe asegurar que todos los componentes críticos deben ejecutarse cumpliendo sus restricciones temporales (es decir deben

ejecutarse antes de que venza su deadline, respetar el periodo, posibles desplazamientos iniciales u offset, ...), o de lo contrario el agente no cumpliría los requerimientos de tiempo real. Además se debe tener en cuenta la división entre parte inicial y final de cada tarea, para luego poder planificar la parte opcional entre ambas.

El sistema utiliza un planificador por prioridades expulsivas, con asignación de prioridades fijas a las tareas [Aud93][Bur93]. Esta asignación de prioridades se puede realiza usando el algoritmo del deadline monotonic, es decir prioridad al más urgente. Para garantizar el cumplimiento de las restricciones temporales de los componentes críticos, se aplica el test de planificabilidad denominado análisis RMA [Kle93], usando para ello los siguientes atributos de las tareas: Ti (periodo), Di (plazo de ejecución), Ci (suma de los tiempos de ejecución para el caso peor de las partes inicial y final).

En tiempo de ejecución, se escoge para ejecutar la tarea preparada más prioritaria. Cuando finaliza la ejecución, bien de una parte inicial, bien de una parte final, el sistema consulta la cantidad de holgura disponible para que en tal caso entre en ejecución el Servidor Inteligente que dará servicio a la parte opcional.

Se utiliza el algoritmo extractor de holgura aproximado dinámico definido en [Dav93a][Dav3b][Gar97] para calcular la cantidad de tiempo que puede utilizarse para ejecutar componentes opcionales sin comprometer los requisitos temporales de las tareas críticas (holgura).

El Servidor Inteligente se puede considerar una tarea que se ejecuta con la prioridad más alta pero sólo en determinados intervalos de tiempo, el tiempo de holgura o slack.

Como consecuencia de esta planificación, entre la parte inicial y final de una misma tarea, únicamente se pueden ejecutar tareas de mayor prioridad, y la parte final de la tarea de mayor prioridad debe finalizar antes que la de las tareas de menor prioridad. En la figura 4.7 se puede observar un ejemplo de esta consecuencia, donde existen tres tareas siendo la número 1 la de mayor prioridad y las tres la tarea de menor prioridad. Como se ve las tareas 1 y 2 se ejecutan entre las partes inicial y final de la tres, la de menor prioridad. Lo mismo ocurre entre la tarea 1 y la 2.

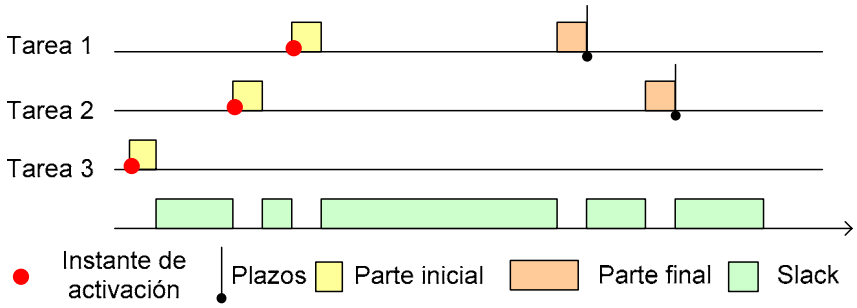


Figura 4.7. Planificación de primer nivel

Como el planificador de primer nivel conoce los instantes de activación de las partes críticas calcula el intervalo de tiempo que denominaremos también hueco o tiempo de slack. El hueco de slack es el intervalo temporal existente entre dos partes críticas consecutivas. En la figura 4.7, puede verse la existencia de hasta 5 huecos de slack diferentes. Estos huecos de slack serán los usados para ejecutar las partes opcionales de las tareas y las tareas aperiódicas. De esta manera otra tarea del planificador de primer nivel será la de informar al planificador de segundo nivel de los huecos de slack.

El núcleo del sistema garantizará que el tiempo de slack que se ofrece para ejecutar dichos componentes no críticos no comprometerá en ningún caso la ejecución de los componentes críticos.

4.4.2 Planificación de segundo nivel

El planificador de segundo nivel (SLS) tiene como objetivo utilizar el tiempo sobrante de la ejecución de los componentes críticos, para ejecutar la parte opcional de los MKS y conseguir la mayor calidad en los resultados, de manera que el comportamiento global del agente sea el mejor posible.

En principio encontramos una diferencia fundamental con el planificador de primer nivel, mientras que este último busca como objetivo ejecutar todas las partes críticas antes de su tiempo límite o deadline, el objetivo del planificador de segundo nivel es el de la calidad, aunque para ello pueda no ejecutar partes opcionales de un MKS y si todas las de otro, etc.

Esto último no significa que en la planificación de segundo nivel no se tengan en cuenta las restricciones temporales, al contrario estas siguen existiendo. Las partes opcionales de una tarea no pueden ser ejecutadas después de la

parte final de la tarea a la que pertenecen, y esta parte está fijada temporalmente según hemos visto.

Ahora bien, lo que si que se debe tener en cuenta es que no hay una presión por ejecutar esas parte opcionales, ya que en cualquier caso son métodos que proporcionan una solución de mayor calidad a la ya calculada por el componente crítico. Si ningún elemento opcional de una tarea es ejecutado, el sistema dispone de la solución calculada por el componente de nivel 0, con lo cual se dispone de la respuesta dentro del plazo de su tiempo límite.

Por tanto tenemos que tras ejecutarse la parte inicial o final de una tarea, el RTOS mandará a ejecutarse al servidor inteligente, pasándole una serie de información, entre ella que partes opcionales activar y el tiempo de slack existente hasta la siguiente parte inicial o final. El servidor inteligente activa al SLS que realizará su planificación. Esta planificación está basada en el bucle de control de un sistema de blackboard y su algoritmo (que se repetiría cada vez que entra el servidor inteligente llamado por el RTOS) sería:

```
Recepción de mensajes

Triggering

Condition Testing

Rating

Mientras Scheduling!= Null

      Interpret
```

Figura 4.8. Bucle de control

A continuación pasamos a comentar cada una de estas fases:

- Recepción de mensajes.

En primer lugar el planificador de segundo nivel recibe una serie de mensajes. Estos mensajes son enviados por el planificador de primer nivel, y en ellos se informa de la duración del slack actual, calculada por el FLS, es decir cuanto tiempo dispone el SLS antes de que entre otra vez el RTOS.

La duración del slack le sirve al planificador para saber cuanto tiempo dispone para poder ejecutar partes opcionales y para sus propios cálculos, pues no hay que olvidar que el planificador también se ejecuta en tiempo de slack.

También se informa de la identidad de las tareas que han ejecutado su parte inicial o final desde la última vez que se ejecutó el planificador de segundo nivel.

La identidad de las tareas que han ejecutado su parte inicial servirá al planificador para conocer nuevas partes opcionales candidatas a ser ejecutadas, pues pertenecerán a estas tareas. No serán las únicas, ya que partes opcionales correspondientes a entradas anteriores del planificador y que no fueron ejecutadas en su momento, pueden serlo ahora, siempre y cuando no haya vencido su deadline correspondiente. También componentes de tareas aperiódicas podrán ser ejecutadas.

La identidad de las tareas cuya parte final se ha ejecutado servirá para eliminar partes opcionales del conjunto de candidatos, ya que la parte opcional de una tarea no puede ser ejecutada si ya lo ha sido su parte final correspondiente.

- Triggering

La siguiente fase es la de triggering o disparo de la parte opcional de los in-agents de los que se ha recibido un mensaje en la fase anterior. Para ello se inserta en una lista los MKS pertenecientes a las tareas anteriores y que representan la parte opcional que puede ser ejecutada.

Esta lista representa por tanto, a las tareas cuya parte inicial se ha ejecutado en el periodo actual.

- Condition Testing

Esta fase recibe la lista anterior y comprueba las posibles precondiciones que puedan tener ciertas tareas para ser ejecutadas. Únicamente aquellas tareas que cumplan las precondiciones permanecerán en esta lista, serán las tareas activas.

- Rating

Esta fase es la más importante del proceso pues es la que realiza la planificación en si. Recibe una lista de tareas y determina el orden en que los niveles pertenecientes a dichas tareas pueden ser ejecutados. Este orden

dependerá de la estrategia de planificación utilizada. En cualquier caso debe respetar el hecho de que cuando se trata de niveles pertenecientes a MKS de refinamiento progresivo, sólo se puede ejecutar un nivel determinado si el nivel anterior del mismo MKS ya ha sido ejecutado.

La lista de tareas que recibe no está formada únicamente por aquellas que formaban parte de la lista que pasó la fase de "condition testing". Además son añadidas aquellas tareas de activaciones anteriores (es decir cuya activación se produjo en otra entrada del control por un hueco de slack anterior), de las cuales aún existen niveles sin ejecutar de calidad superior a los ya ejecutados y para los que todavía no ha vencido su deadline.

- Schedule

Este paso elige el nivel que ha quedado mejor posicionado en la fase anterior y si existe tiempo suficiente para su ejecución lo manda ejecutar, sino pasa al inmediato siguiente según la fase anterior.

El nivel no escogido no es eliminado, pues puede ser elegido en un hueco de slack posterior en el que se disponga de más tiempo (siempre y cuando no haya vencido su deadline).

- Interpret

En este punto se manda ejecutar el nivel elegido en el paso anterior. Aunque la elección la realiza el sistema de alto nivel, el mensaje es enviado al sistema de bajo nivel que es el que realmente controla la ejecución.

Si el nivel sobrepasa el tiempo de ejecución indicado (dentro de unos rangos), el sistema termina con la ejecución del nivel informando al control de la finalización incorrecta.

El sistema de bajo nivel informa no sólo de la finalización correcta o no del nivel, sino que además le proporciona al control otros datos que este puede necesitar para la planificación o la mejora de ésta, como puede ser el tiempo real de ejecución. Este tiempo puede ser utilizado para realizar un aprendizaje con el cual ajustar en mayor medida los perfiles de ejecución.

Además en función del método de planificación escogido, puede llevar a cabo una replanificación, por ejemplo en aquellos métodos que eligen en cada momento según las circunstancias y no generan planes generales.

4.4.3 Métodos de planificación

En la fase de rating se realiza una clasificación de los MKS utilizando un método o estrategia de planificación. La estrategia elegida determinará la eficiencia del agente. A grandes rasgos existen dos grandes bloques de métodos, los que se podrían determinar de fuerza bruta, y los deliberativos.

- Los métodos de fuerza bruta son aquellos que hacen poco uso de la información del problema, y sobre todo no generan planes, sino que en cada momento indican el paso siguiente a hacer. Estos métodos elegirían entre los niveles disponibles de las tareas activas. Dichos niveles serían todos en el caso de MKS múltiples y el nivel más bajo de los no ejecutados para los de refinamiento. Una vez ejecutado el nivel elegido puede ser necesaria una reordenación de los niveles, si el nivel ejecutado pertenecía a un MKS por refinamiento, para tener en cuenta el nivel superior de ese MKS que ahora si puede ser ejecutado.

  Algunos de estos métodos que han sido utilizados en ARTIS son:

  - DM. Deadline Monotonic. Esta política prima aquellos niveles pertenecientes a tareas que tienen deadlines muy restrictivos. Para ello simplemente ordena los niveles en función de su plazo máximo de ejecución, es decir del intervalo que tiene completar sus niveles.

    Esta estrategia tiende a ejecutar el mayor número posible de niveles y no suele proporcionar buenas calidades de resultado.

  - BIF. Best Importance First. Ordena los niveles en función de la importancia estimada de la tarea a la que pertenecen. Es una primera aproximación al concepto de calidad, aunque bastante sencillo al tratarla a nivel de tarea.

    Al no distinguir entre las diferentes calidades que poseen los niveles de un mismo MKS, tiende a ejecutar muchos niveles de un subgrupo determinado de MKS, con ello aunque la calidad resultante es mejor que en la anterior, sigue siendo baja.

  - EDF. Earliest Deadline First. En este caso la política de ordenación también se fija en el deadline, aunque no de forma absoluta como en el DM, ya que no considera la duración del intervalo sino cuando vencerá el deadline teniendo en cuenta cuando se activó la tarea.

Produce mejores resultados que el DM, al postergar aquellos niveles cuyo vencimiento no está próximo, pero sigue siendo más orientado a ejecutar un número alto de niveles, más que a obtener altas calidades.

- ELDF. Earliest Level Deadline First. Variación de la anterior, en este caso se considera el deadline no en términos de tarea sino de nivel. Para ello se considera el deadline de la tarea y la duración estimada de cada nivel. El objetivo es hacer una planificación que aproveche más el formato de niveles de los MKS.

  Supone una mejora de nuevo en cuanto a ejecución de número de niveles y no respecto a calidades.

- HQF. High Quality First. En este caso se trata de una ordenación de niveles en función de la calidad asociada a cada uno de ellos sin tener en cuenta la premura de su ejecución o no por el vencimiento de su deadline.

  Mejora del BIF al tratar de forma individual los niveles, con lo cual la calidad resultante suele ser mayor que en el caso anterior, pero al no tener en cuenta el deadline puede dejar por ejecutar niveles que con otra ordenación podrían ejecutarse y aumentar la calidad.

- HSF. High Slope First. Esta política hace una ordenación de los niveles en función de la relación calidad/tiempo de ejecución de cada nivel. Se intenta por tanto fusionar las dos tendencias anteriores, la de calidad y la de los deadlines, aunque de una forma sencilla pues únicamente debe de calcular el equivalente a las pendientes de los perfiles de ejecución de cada MKS.

  Mejora los resultados de las anteriores por el hecho de incluir los dos criterios.

- Las técnicas deliberativas vistas en el capitulo 2 (planificación deliberativa, computación flexible, etc.) hacen en cambio uso de mayor información, para intentar aprovechar al máximo el tiempo disponible. Para ello generan planes indicando que niveles y de que MKS deben ejecutarse, y en que momento. Para el caso de los MKS de refinamiento considerará también diferentes niveles y no sólo uno. El problema con este tipo de métodos es que pueden depender mucho de una tipología determinada de algoritmos y por ello no ser aplicables a una arquitectura determinada o necesitar de una fuerte adaptación que en muchos casos hace que pierdan o vean disminuidas considerablemente sus ventajas.

La conclusión en este punto es la necesidad de desarrollar una técnica deliberativa que haga uso de la información del sistema y que sea capaz de permitir la utilización de los dos tipos de MKS (refinamiento y métodos múltiples) que el sistema utiliza. Esta técnica será presentada en el capítulo siguiente.

5 HEURÍSTICA SSS

En este capítulo se va a presentar una heurística que recoge las necesidades de la arquitectura ARTIS, tanto a nivel de los algoritmos que va a manejar como de las características del sistema.

En primer lugar se presentará cuales son los requerimientos que hacen necesaria una nueva heurística frente a las ya existentes y comentadas en capítulos anteriores. Esta heurística se denomina SSS, Slack-Slide Scheduling, en referencia a una de sus propiedades el desplazamiento de los diferentes huecos de slack existentes. A continuación se presentará la nueva heurística en dos versiones, la segunda de las cuales, denominada SSSM (SSS con Memoria) es una ampliación de la primera que pretende mejorar más el rendimiento aprovechando otra característica de la arquitectura, como es el saber que datos cambian de una activación a otra de un in-agent.

5.1 PROBLEMÁTICA

El propósito de diseñar una nueva heurística surge por la problemática que presentan las heurísticas preexistentes (voraces o deliberativas) y que se han citado previamente. Estas heurísticas ya se han comentado con anterioridad por lo que ahora se remarcará las características que hacen que no se puedan aplicar a ARTIS o que sean mejorables.

En relación a las heurísticas voraces, su simplicidad hace que sí sean aplicables a la arquitectura. Sin embargo el hecho de que no generen planes hace que no puedan sacar partido de todas las características del sistema (como el tener particionado el slack en diferentes partes) con lo cual su rendimiento puede ser mejorado. Otra característica es que en cada momento comienzan de cero sin tener en cuenta resultados previos que puedan mejorar su comportamiento.

En cuanto a las técnicas deliberativas comentadas (planificación deliberativa, técnicas design-to-time, etc) mucho más eficientes (algunas de ellas óptimas) presentan algunos inconvenientes a la hora de ser aplicadas a ARTIS:

- Diferencia entre algoritmos anytime y algoritmos de refinamiento progresivo. Aunque los algoritmos de refinamiento progresivo (usados en ARTIS) están basados en los anytime, con la idea de mejorar la solución cuanto más tiempo disponen partiendo de resultados previos, son diferentes a estos últimos. Los anytime pueden ser interrumpidos en cualquier momento utilizando el resultado que acaba de calcular sin tiempo malgastado, con lo cual un algoritmo anytime puede ser "troceado" como se necesite.

  Por el contrario los algoritmos de refinamiento progresivo presentan un comportamiento discreto, si bien también puede ser interrumpido, el resultado del que se dispone en ese caso no tiene, por lo general, que haber sido obtenido en el momento anterior, sino que existirá un intervalo de tiempo entre que se obtuvo el resultado y se ha interrumpido. Este tiempo será tiempo perdido pues los cálculos realizados no pueden ser utilizados al no haber finalizado la ejecución del nivel correspondiente.

  Por todo ello aunque algunas de las ideas usadas en los métodos de planificación de algoritmos anytime pueden ser usadas para los que usan refinamiento progresivo (como se comentará al presentar la nueva heurística), los métodos en si no son directamente aplicables.

- Existencia de dos tipos diferentes de métodos a planificar. La mayoría de las técnicas existentes están diseñadas para tratar con un único tipo de algoritmos, bien los anytime como los citados en el punto anterior, bien del tipo métodos múltiples como el "design-to-time". Sin embargo ARTIS maneja dos tipos de MKS, los de refinamiento progresivo y los métodos múltiples simultáneamente. Es decir no es que se tenga la posibilidad simplemente de que el diseñador de la aplicación se decante por uno u otro, sino que pueden coexistir, y en un momento determinado, cuando el planificador tenga que elegir se encuentre con que dispone de MKSs activos tanto de un tipo como de otro pertenecientes al mismo o diferentes in-agents. Es necesario por tanto que la heurística maneje de forma conjunta ambos tipos de métodos.

- Naturaleza del slack. Debido a la política de planificación de primer nivel, que ya se comentó en el capítulo anterior, el tiempo del que dispone un in-

agent para ejecutar sus partes opcionales, es decir el tiempo entre la ejecución de su parte inicial y el instante en que está planificada la ejecución de su parte final, no es continuo. Al contrario, al poder entrar durante ese tiempo la ejecución de partes iniciales de otros in-agents más prioritarios, dicho tiempo se ve troceado en diferentes huecos de slack. Esto contrasta con la aproximación que usan la mayoría de las técnicas, que tratan con un único hueco de slack, aunque con diferentes deadlines para los diferentes elementos a ejecutar.

Es necesaria por tanto, una heurística que maneje simultáneamente dos tipos de MKS, los de refinamiento progresivo y los tipo métodos múltiples. Además esta heurística deberá tratar con diferentes huecos de slack para cada in-agent y tener esto en cuenta a la hora de distribuir los niveles.

5.2 HEURÍSTICA SSS

5.2.1 Consideraciones previas

Antes de entrar a describir el nuevo algoritmo, es necesario comentar con más detalle la planificación hecha por el planificador de primer nivel, ya que sobre esa base actuará el planificador de segundo nivel.

Como se ha comentado previamente cuando un in-agent ha ejecutado su parte inicial y hasta que se ejecute su parte final, únicamente in-agents de mayor prioridad pueden ser activados, y su parte final debe ejecutarse antes de la del in-agent de menor prioridad. Esto produce escenarios como el visto en la figura 4.7.

Analizando dicho escenario se puede establecer tres tipos de intervalos diferentes en cuanto a la información que dispone el planificador de segundo nivel para llevar a cabo su labor, en función del componente que ha ejecutado el primer nivel antes de ceder el control al segundo nivel y/o cual va a ser el componente que se ejecutará después (figura 5.1):

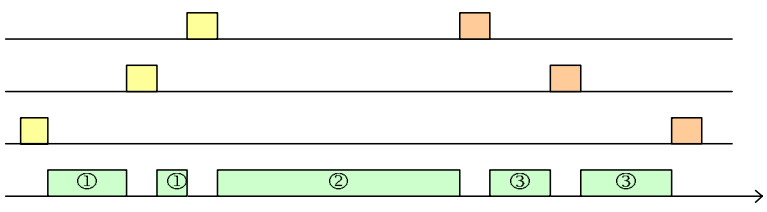


Figura 5.1. Tipos de intervalos

- Intervalos finalizados por una parte inicial. Estos intervalos (tipo 1 en figura 5.1) se caracterizan porque al finalizar con una parte inicial, la información que dispone se limita a la de las tareas activas en ese momento, conjunto formado por las tareas cuya parte inicial ha marcado el comienzo del intervalo, más todas aquellas tareas provenientes de intervalos anteriores y cuyo deadline todavía no ha vencido (aunque realmente sólo interesan aquellas que además no han ejecutado todos sus niveles en el caso de MKS de refinamiento progresivo o no han ejecutado el nivel de mayor calidad en MKS por métodos múltiples).

  Además, el planificador de segundo nivel no puede modificar la longitud del intervalo, ni adelantando ni retrasando la ejecución de la parte inicial. En el primer caso se debe respetar los periodos de activación marcados por el diseñador del sistema, y en el segundo, retrasar el tiempo límite marcado por el planificador de primer nivel para la ejecución de un componente crítico (como es una parte inicial), puede provocar que el sistema no cumpla los deadlines de éste o posteriores in-agents.

- Intervalos con parte inicial como límite inferior y parte final como superior (tipo 2 en figura 5.1) En este caso el sistema dispone además de la información sobre las tareas activas en dicho intervalo, de datos sobre algunos intervalos siguientes. Este intervalo pertenece a la tarea activa más prioritaria en ese momento, pues acaba en una parte final y por tanto sólo puede tratarse de la parte final de la más prioritaria. En este punto el sistema conoce para todas las tareas activas cuál será su deadline de ejecución (es decir el instante de tiempo en que la tarea debe haber finalizado, que por la dinámica del sistema puede ser menor que el deadline especificado por el diseñador). Estos deadlines serán, siempre y cuando no entre una tarea de más prioridad, los que marquen la duración de los intervalos siguientes. También si se cumple la condición anterior, el sistema conocerá que tareas estarán activas en los intervalos siguientes, al no entrar nuevas tareas.

  En este caso se cumple además que el planificador puede decidir un adelantamiento de la parte final del intervalo, ya que dicho adelantamiento no viola ninguna de las especificaciones del problema ni pone en peligro la planificación crítica.

- Intervalos delimitados por dos partes finales. En este caso el intervalo (tipo 3 en figura 5.1) no introduce nuevas tareas, sino que trabajará

con las existentes de intervalos anteriores. Se trataría de los intervalos cuyas características son conocidas en los intervalos del tipo anterior.

También en este caso se puede adelantar su límite superior, por las mismas razones que en el caso anterior.

Teniendo en cuenta los diferentes tipos de intervalo comentados, se establecen dos tipos de zonas diferentes en la planificación resultado del trabajo del planificador de primer nivel. Estas zonas (figura 5.2) son:

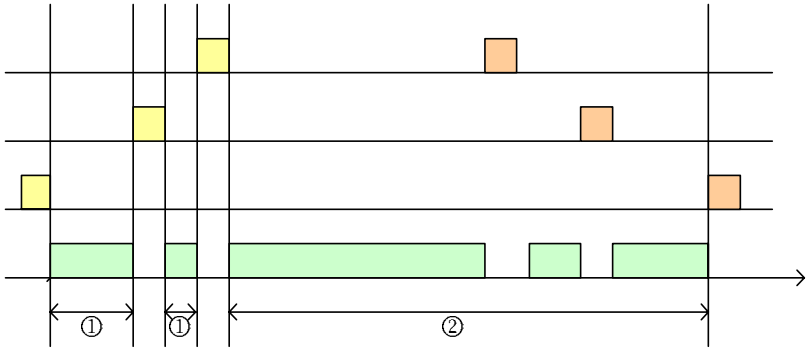


Figura 5.2. Zonas de planificación

- Zona 1. Cada zona de tipo 1 corresponderá con un intervalo del primer tipo comentado previamente, es decir de aquellos cuyo final viene marcado por la ejecución de una parte inicial. Por tanto en una zona 1 únicamente se tiene información sobre las tareas activas en dicha zona, y sobre la duración de la misma. No se puede variar el instante que determina el final de la zona.

- Zona 2. Una zona de tipo 2, estará formada por un intervalo del segundo tipo (aquellos que comienzan con una parte inicial y finalizan con una final) y por todos aquellos del tercer tipo (que comienzan y acaban con una parte final) que le sigan, hasta llegar a una zona de tipo 1 (o un intervalo sin MKS activos). En este caso la zona no engloba un único intervalo sino varios, ya que desde el primer intervalo de la zona se conoce las características (tareas activas y duración) del resto. Además los límites de los intervalos pueden ser modificados, ya que al ser partes finales pueden ser adelantados.

Cada zona va a necesitar un tratamiento diferente. En el caso de las zonas de tipo 1, al estar constituidas por un único intervalo, el tratamiento será más limitado. Sin embargo, las zonas tipo 2, al estar constituidas cada una de

ellas por varios intervalos, se va a intentar realizar una planificación conjunta de todos ellos, esto va a suponer una ventaja frente a muchas de las otras técnicas que sólo consideran el hueco actual y guarda más similitudes con tratamientos como el de la planificación deliberativa de Dean y Boddy, aunque en aquel caso se tratase de un único hueco pero con diferentes deadlines, y aquí de varios huecos, pues los intervalos están separados por la ejecución de las partes críticas.

5.2.2 Algoritmo

La heurística SSS se va a aplicar en dos fases, por un lado en la fase de rating donde se decidirá la mayoría de niveles que van a ejecutarse y la de scheduling donde se decide el orden y adicionalmente la ejecución de algún nivel más.

- Rating

La fase de rating utiliza como base el ratio calidad vs tiempo medio de ejecución, igual que la técnica HSF, pues este criterio se ha comprobado el mejor cuando no se dispone de más información, como es el caso en la zona 1.

```
Datos
    Sea ℒ la lista de MKS activos
    I, nuevas tareas activas
    ℒℋ, lista de huecos de los cuales el control conoce su duración
    Slack, hueco_actual
    ℒℙℱ, lista partes finales
    LA, lista de MKS

Begin
    Si slack.tipo == sección_1
            asigna_niveles (slack.duración, ℒ)
            guarda_huecos( ℒℋ, I)
    Si slack.tipo == sección_2 /*sólo primer intervalo de esta sección*/
            guarda_huecos(ℒℋ, I)
            para todo s ∈ ℒℋ desde final(ℒℋ) hasta principio(ℒℋ)
                    asigna_niveles_seccion( s, ℒA)
            para todo s ∈ ℒℋ desde inicial(ℒℋ) hasta final(ℒℋ)
                    asigna_niveles_seccion(s, ℒA)
                    marca_adelantamiento_PF (s, ℒℋ, ℒℙℱ)
End
```

Figura 5.3. Algoritmo SSS

En la zona 2 HSF también servirá de base pero en un esquema similar al utilizado en la planificación deliberativa de Dean y Boddy con las correspondientes diferencias. Es decir se comenzará a planificar desde el último intervalo (en la planificación deliberativa era el ultimo deadline) pero no se podrá completar completamente cada intervalo, al no ser algoritmos anytime que pueden ser troceados. Se necesitará un proceso posterior de ajuste, así como manipular de la forma correcta los dos tipos de MKS.

En primer lugar, como se puede observar, los datos que maneja el algoritmo son:

- Por un lado la lista de los MKS activos, $\mathcal{L}$, tanto si se han activado en ese momento debido a que se acaba de ejecutar su parte inicial, como si ya estaban activados de invocaciones anteriores al planificador de segundo nivel. Es la lista donde se va a marcar que niveles van a ser ejecutados y cuales no. Esta lista proviene de las fases anteriores del control de segundo nivel y se mantiene y actualiza entre diferentes llamadas a dicho control.

- Además el algoritmo necesita recibir en otra lista separada, I, el conjunto de tareas que acaban de ser activadas en la última entrada del scheduler de primer nivel (es decir aquellas para los que se ha ejecutado su parte inicial). Esta lista proviene del control de primer nivel.

- Una lista $\mathcal{LH}$ formada por una descripción de los huecos de slack siguientes al actual de los cuales se conoce su extensión. Cada uno de los nodos de esta lista, describe un hueco de slack, indicando su punto de inicio y final. Servirá para hacer el scheduling por adelantado de la zona 2. Posteriormente se comentará su construcción y manejo. Esta lista, interna al scheduler de segundo nivel vendrá de activaciones anteriores del propio scheduler.

- Slack actual. El scheduler de primer nivel comunica la duración del slack actual, y su tipo, de los comentados en el punto anterior (intervalos de tipo 1, 2 o 3) que será usado para conocer que operaciones poder realizar.

- La lista de partes finales adelantadas, $\mathcal{LPF}$, que indicará donde decide el scheduler de segundo nivel que deben acabar los intervalos.

- $\mathcal{LA}$. Esta lista formada por MKS, se utilizará cuando se recorra los intervalos en orden inverso, de forma que para cada hueco reflejará los MKS que seguirán activos en los huecos posteriores al actual.

Ahora pasaremos a comentar cada una de las partes del algoritmo, así como las funciones en que se descompone. Hay que tener en cuenta la zona en que se encuentra, pues el tratamiento es diferente:

❖ Zona de tipo 1. Si se encuentra en la primera zona, el tratamiento es más sencillo ya que únicamente deberá considerar el hueco de slack actual. En dicho hueco se aplicará como ya se ha comentado el criterio del ratio calidad vs tiempo de ejecución.

Para ello se dispondrá de la función de asignar niveles que recibirá como datos de entrada la duración del slack actual y la lista de MKS activos. Posteriormente se ejecutará la función de guardar_huecos que almacenará información para ser utilizada posteriormente.

Asignar_niveles. La función asignar_niveles (Figura 5.4), escoge niveles para el slack actual que se pasa como parámetro, hasta que la duración total de todos los niveles escogidos iguala o supera la duración de dicho slack. En cada paso la función elige el nivel seleccionable con mejor ratio de la lista de MKS activos ($\mathcal{L}$).

Por seleccionable entendemos en el caso de MKS de refinamiento progresivo, al nivel de orden inmediatamente superior al último elegido del mismo MKS, si no existe ninguno elegido al último ejecutado (dentro de la misma activación de su tarea) del mismo MKS, y si tampoco ocurre esta circunstancia a su primer nivel.

En el caso de métodos múltiples, serán niveles seleccionables de un MKS todos aquellos cuya calidad absoluta sea mayor que el último elegido de dicho MKS, o si no hay elegido, del último ejecutado (en la misma activación) de dicho MKS. Si no hay ninguno elegido ni ejecutado, cualquier nivel de dicho MKS puede ser seleccionado.

Entrada

Slack: hueco actual.
$\mathcal{L}$: lista de MKS activos

Datos

T: tiempo sin asignar del slack
N, N_{act}: Niveles
M: MKS
$\mathcal{L}_{pend}$: Lista de MKS descartados hasta siguiente hueco slack

Begin

```
T = Slack
Hacer
  N = Nivel seleccionable con mejor ratio de ℒ
  M = MKS al que pertenece N
  Si M es de tipo refinamiento progresivo
       Entonces si t_ejec(N) <= T          /* t_ejec = tiempo ejecución*/
                      Entonces   Marcar N como ejecutable
                                 Si existen niveles seleccionables de M
                                    Entonces insertar M en ℒ
                                 T = T – t_ejec(N)
                       Si no     Insertar M en ℒpend
  Si M es de tipo métodos múltiples
       Entonces sea Nant el nivel de M ya asignado
                  Si t_ejec(N)<= T + t_ejec(Nant)
                      Entonces  Marcar N como ejecutable
                                Si ∃ niveles seleccionables de M
                                       Entonces insertar M en ℒ
                                T = T – t_ejec(N) + t_ejec(Nant)
                       Sino     Insertar M en ℒpend
Mientras T>0 y ∃ nivel seleccionable
```

End

Figura 5.4. Función asignar_niveles

El nivel seleccionado sólo será marcado como ejecutable si existe hueco disponible para ejecutarlo en función del tiempo que aún quede disponible del slack y de su tiempo de ejecución previsto. En realidad, la decisión de escoger un nivel únicamente si cabe dentro del tiempo no es estrictamente necesaria, puesto que el nivel podría seguir ejecutándose en el siguiente hueco de slack. Esta decisión se debe al hecho de que desconocemos

cuales van a ser las condiciones del siguiente hueco (duración, nuevos MKS de in-agents recién activados), que podrían hacer que el nivel interrumpido no se reanudara en el slack siguiente, perdiendo el tiempo utilizado en él. Se prefiere por tanto utilizar dicho slack para ejecutar aquellos niveles que van a entrar completamente y no van a dejar su resultado pendiente de situaciones futuras de las que desconocemos completamente cualquier característica. Si no puede ser ejecutado, el MKS se introduce en una lista de pendientes que se usará en el siguiente hueco de slack en la fase de trigger para unirse junto con los MKS recien activados para formar la nueva lista a planificar.

Cuando se comprueba si el nivel seleccionado puede entrar en el slack disponible, se tiene en cuenta dicho tiempo y el tiempo de ejecución del nivel seleccionado en el caso de los MKS de refinamiento progresivo. Sin embargo, esto no es así en el caso de métodos múltiples. En este caso, el tiempo que se va a comparar con el disponible no va a ser siempre el tiempo de ejecución del nivel. Únicamente en el caso de que no se haya elegido en un paso anterior un nivel del mismo MKS se usará el tiempo de ejecución. Pero si ya se ha elegido, hay que tener en cuenta que el nuevo nivel no va a complementar al anterior, puesto que los métodos múltiples son métodos alternativos, sino a sustituirle. Por eso, el tiempo que se considerará será la diferencia entre el tiempo de ejecución entre el nivel en cuestión y el previamente escogido, pues es esa diferencia de tiempo la que realmente va a necesitar de más el nuevo nivel, puesto que si finalmente se escoge, el nivel anterior se elimina del plan. La eliminación del nivel "viejo" se debe al hecho de que el nuevo proporciona una solución de mayor calidad que no necesita de la ejecución del otro nivel.

El nivel seleccionado queda marcado para poder ser elegido por la fase de schedule, que como vimos anteriormente va sacando uno a uno el nivel a ejecutarse. El tiempo disponible se reduce según el tiempo de ejecución del nivel elegido (o del tiempo extra en el caso de métodos múltiples como se ha comentado).

Esta asignación de niveles termina en un primer momento cuando ya no pueden ser elegidos más niveles ya que su duración sobrepasa el tiempo disponible. Normalmente es muy difícil que el tiempo disponible llegue exactamente a 0, siempre puede quedar cierta cantidad de tiempo. En este caso si que se añade un último nivel (de nuevo según el ratio calidad – tiempo de ejecución) aún a sabiendas que no va a poder ejecutarse en su totalidad y necesitará ser completado en el siguiente slack. Anteriormente se expusieron la razones para no realizar esto de forma sistemática y preferir otros niveles con menor ratio pero cuya ejecución pudiese ser

completada en el slack, pero ahora hay que tener en cuenta que ya no existe ningún otro nivel que entre en dicho tiempo, y de no asignar nada sería tiempo desperdiciado. Hay que recordar que estamos en un intervalo del tipo 1, por lo que el final del intervalo, una parte inicial, no puede ser adelantado esa cantidad de tiempo no asignada. Por eso se decide elegir ese último nivel, aunque hay que tener en cuenta que la parte de método que quede sin ejecutarse no entrará directamente a ejecución en el siguiente slack, sino que deberá competir con el resto de niveles, eso si teniendo en cuenta sólo el tiempo que le resta para completarse.

Guardar_huecos. La segunda función que se ejecuta en la zona 1, una vez ha finalizado la de asignar_niveles es guardar huecos. Esta función en realidad va guardando información que será necesaria posteriormente para la planificación de la zona 2.

Esta función (Figura 5.5) recibe como entrada la lista I de tareas que se han activado desde la última ejecución del planificador de segundo nivel, los cuales normalmente corresponderán a aquellos cuya parte inicial marcaba el comienzo del intervalo actual.

El segundo dato que recibe la función es la lista de huecos $\mathcal{LH}$. Esta lista es creada por la propia función, de manera que en una activación recibe la lista que se ha ido construyendo en activaciones anteriores por ella misma, para ir añadiendo más información. En esta lista se va almacenando información relativa a los intervalos de la zona 2 que seguirán a los intervalos de zona 1 cuando estos finalicen. Hay que tener en cuenta que cuando comienza un intervalo de tipo 1 (al haberse ejecutado una parte inicial), conocemos también cuando será su deadline de ejecución, con lo cual podemos conocer la duración del intervalo de tipo 3 que su parte final provoca.

En concreto cuando la función es llamada y la lista de huecos está vacía, se crea un nuevo nodo que representa un intervalo de tipo 3, y cuyo limite superior viene marcado por el deadline de ejecución del in-agent cuya activación es la parte inicial del intervalo actual. El límite inferior de dicho intervalo queda indeterminado, pues aún no tenemos suficiente información para determinarlo. En otro campo denominado “activo”, se almacena la referencia al in-agent cuya parte final cierra dicho intervalo. En otras palabras, dicho hueco será el último donde dicho in-agent estará activo.

Entrada

$\mathcal{LI}$: lista de tareas activadas en este hueco de slack
$\mathcal{LH}$: lista de huecos
Slack: hueco actual

Datos
I: In-agent
H, H_{ant}: huecos

Begin
$\forall I \in \mathcal{LI}$
si Slack es de tipo 1
entonces Si vacio($\mathcal{LH}$)
entonces crea_nodo(H)
Limite_superior(H) = Deadline_ejecución(I)
Activo(H) = I
Sino H_{ant} = primero($\mathcal{LH}$)
Limite_inferior(H_{ant}) = Deadline_ejecución(I) + T_ejec(Parte_final(I))
crea_nodo(H)
Limite_superior(H) = Deadline_ejecución(I)
Activo(H) = I
Inserta H en $\mathcal{LH}$
si Slack es de tipo 2
entonces Si vacio($\mathcal{LH}$)
entonces crea_nodo(H)
Limite_superior(H) = Deadline_ejecución(I)
Limite_inferior(H) = Inicio_slack_actual
Activo(H) = I
Sino H_{ant} = primero($\mathcal{LH}$)
Limite_inferior(H_{ant}) = Deadline_ejecución(I) + T_ejec(Parte_final(I))
crea_nodo(H)
Limite_superior(H) = Deadline_ejecución(I)
Limite_inferior(H) = Inicio_slack_actual
Activo(H) = I
Inserta H en $\mathcal{LH}$
End

Figura 5.5. Función guardar_huecos

Cuando la función es llamada y sí existe una lista creada, la función coge el primer nodo de dicha lista, que es el último nodo que se introdujo en la lista y representa el intervalo que hasta ese momento antes iba a ocurrir. Hay que recordar que entre la parte inicial y final de una tarea, sólo tareas

de mayor prioridad pueden interrumpir, y que en ese caso la parte final del más prioritario debe finalizar antes que la del de menor prioridad. Debido a esto, la lista de huecos se va creando en orden inverso a cómo dichos huecos se sucederán posteriormente en el tiempo, ya que el primer deadline de ejecución que se conocerá es la de la tarea que primero aparece, es decir la de la menos prioritaria y por tanto será la última parte final que se ejecute. El nodo que escoge por tanto la función es el creado en la activación anterior y es un nodo no cerrado, es decir del cual no se conocía todavía su límite inferior. Este límite inferior está marcado precisamente por el deadline de ejecución de la tarea que se ha activado en este intervalo (más el tiempo de ejecución de su parte final), con lo cual se guardará dicha información y el nodo quedará cerrado. Posteriormente, se añadirá un nuevo nodo, similar al introducido cuando la lista está vacía. Es decir que cada parte final delimita el límite superior de un nuevo intervalo abierto y completa la constitución de un intervalo señalando su límite inferior.

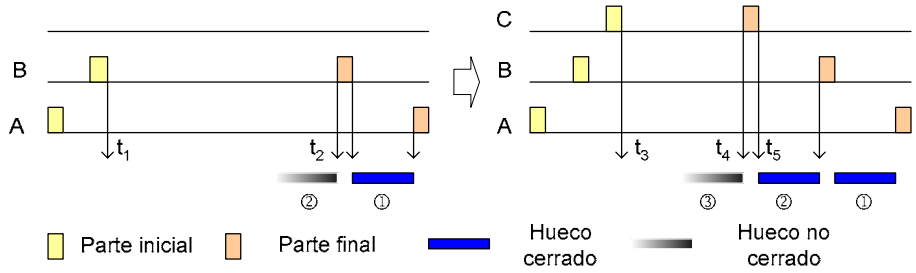


Figura 5.6. Ejemplo creación lista de huecos.

En la figura 5.4 podemos observar un ejemplo de cómo se va creando esta lista de huecos. La situación de la izquierda corresponde al instante t1, en ese momento se ha construido una lista de huecos que contiene el intervalo ① delimitado por las partes finales de las tareas A y B (siendo B la más prioritaria). El intervalo ② permanece sin cerrar, ya que desconocemos su límite inferior, pues esta previsto que se activarán más tareas que producirán nuevos intervalos.

La situación de la derecha corresponde al momento t_3, se ha ejecutado la parte inicial de una nueva tarea, la C (la más prioritaria de los tres). Corresponde a un intervalo de tipo 1 ya que se conoce que posteriormente va a entrar otra tarea más prioritaria (que no aparece en el dibujo ya que para el paso actual no afecta). La función de guardar huecos, recibe la lista con los intervalos ② y ① (en ese orden, ya que se crean en orden inverso a su ubicación temporal). En el punto t_3, se conoce cual es el deadline de

ejecución de la tarea B, que corresponde con el punto t_4. Si en t_4 se ejecuta la parte final de B, se puede cerrar el intervalo ② por su extremo inferior. Dicho extremo será t_5, que corresponde al instante t_4 más el tiempo de ejecución de la parte final de B. Así mismo se crea un nuevo intervalo, el ③, con limite superior precisamente t_4, y con el límite inferior aun por concretar hasta otra entrada de la función.

- Zona de tipo 2. Cuando el algoritmo es llamado desde un punto correspondiente a la zona 2, el tratamiento que va a realizar es distinto al anterior. Hay que tener en cuenta que va a considerar toda la zona 2 en su globalidad, y esta incluye varios intervalos como ya se ha comentado. De hecho, en la fase de rating, el algoritmo sólo va a entrar en el primer intervalo de la zona, el intervalo de tipo 2, correspondiente a la tarea activa más prioritaria, y en este punto generará el plan con los niveles que tiene que ejecutar. En el resto intervalos de la zona (intervalos de tipo 3) usará las asignaciones ya hechas. Por tanto todas las operaciones indicadas a continuación son realizadas al comienzo del primer intervalo de la zona.

Guardar_huecos. La primera operación que se realiza es llamar a la función guardar_huecos comentada con anterioridad. En este caso el funcionamiento es ligeramente diferente. Igual que en el caso anterior la función va a cerrar el último hueco insertado usando su información sobre su parte final. Sin embargo, a diferencia de las activaciones anteriores, la función no va a insertar un nuevo nodo que refleje un intervalo sin cerrar. Sí que insertará un nuevo nodo, pero en este caso cerrado, es decir totalmente definido. La razón se debe a que estamos en un intervalo de tipo 2, es decir aquel que comienza y acaba con las partes inicial y final de la misma tarea, por lo tanto se trata de un intervalo que no va a sufrir ninguna división por la llegada de otra tarea, con lo cual el intervalo a guardar en la lista coincide con el slack actual. Este nodo será el último introducido en la lista, con lo que ésta estará preparada para ser usada.

Asignar_niveles_por_sección. La siguiente operación a realizar va a ser la de asignar niveles a cada uno de los intervalos cuya descripción ha sido guardada en la lista de huecos comentada. Esta operación va a ser realizada por la función asignar_niveles_por_sección (Figura 5.7). Esta función, tal y como se puede observar en la descripción del algoritmo de la figura 5.3, no es llamada una vez sino varias, ya que esta dentro de un bucle que se repite tantas veces como intervalos tenga la lista. Cómo ya se ha comentado se sigue una estrategia similar a la utilizada en la planificación deliberativa de Dean y Boddy, consistente en empezar por el último hueco, es decir el intervalo más alejado del momento actual, ya que en ese

intervalo sólo estará activo el in-agent menos prioritario. Las asignaciones que se hagan en ese intervalo no necesitan considerar las otras tareas que en ese intervalo ya no estarán activas por haber vencido su deadline de ejecución y haberse ejecutado su parte final. De esta manera se asignan niveles que ya no entrarán en conflicto por la CPU en huecos anteriores. De esta manera se va retrocediendo intervalo por intervalo, cada uno de ellos con mayor número de tareas en conflicto, hasta llegar al primero de la lista, el correspondiente a la tarea más prioritaria. En este intervalo estarán activas todas las tareas con las que se ha ido construyendo la lista de intervalos. Es por tanto donde el conflicto entre niveles sería mayor, pero por haber hecho la planificación hacia atrás todos los niveles ya asignados en intervalos posteriores no tendrán que ser tenidos en cuenta en este intervalo.

La función recibe en cada paso el nodo relativo al hueco que va a tratar y una lista con los MKS que permanecerán activos en los huecos posteriores.

Con la duración del intervalo y los MKS activos en él (los que finalizan en este hueco y los que finalizan en huecos posteriores y están en $\mathcal{LA}$), se realiza una asignación de niveles al intervalo utilizando un proceso similar al de la función asignar_niveles. Sin embargo, a diferencia de esta, cuando se va a asignar el nivel seleccionado, si este no cabe en el intervalo libre, no se descarta y se pasa a buscar el siguiente nivel con mejor ratio, en su lugar la función acaba dejando un espacio libre. La razón de este comportamiento se debe al hecho de que aquí si se podrán tratar de forma más o menos conjunta estos espacios que van sobrando, como se comentará con posterioridad. Cuando se asigna un nivel se reduce la duración del intervalo en la lista de huecos.

Otra diferencia radica en el tratamiento de los métodos múltiples. Cuando se selecciona un nivel de este tipo, que mejora la calidad de otro nivel de su propio MKS, y que por tanto debe sustituirlo, si el nivel antiguo está asignado en otro intervalo ya tratado, para saber si se puede escoger el nuevo nivel tendremos en cuenta no sólo el tiempo que dejaría libre el nivel antiguo, y el tiempo disponible en el intervalo que ahora se está tratando, sino además el tiempo que quedó libre en el intervalo donde estaba asignado el nivel antiguo. De realizarse la sustitución se verían reducidos el intervalo anterior, y si fuese necesario el intervalo que se está tratando en la duración que faltase por asignar.

```
Entrada
    H: hueco (de la lista ℒℋ)
    ℒ𝒜: Lista MKS considerados en huecos siguientes

Datos
    I: In-agent
    ℒℳ: Lista MKS
    M: MKS
    T: tiempo_restante
    Final: Boolean

Begin
    I=activo(H)
    ℒℳ=MKS de I
    ℒ𝒜 =añadir ℒℳ a ℒ𝒜
    T= Tiempo sin asignar de H
    Final = false
    Hacer
       N = Nivel seleccionable con mejor ratio de ℒ𝒜
       M = MKS al que pertenece N
       si M es de tipo refinamiento progresivo
            entonces
                si t_ejec(N) <= T       /* t_ejec = tiempo ejecución*/
                  Entonces   Marcar N como ejecutable
                             Si existen niveles seleccionables de M
                                    Entonces insertar M en ℒ
                             T = T – t_ejec(N)
                si no Final = true
      si M es de tipo métodos múltiples
            entonces sea N_ant el nivel de M ya asignado
                     T_ant = t_ejec(N_ant)
                     Tl_ant=T_libre(Hueco(N_ant))
                              /*tiempo libre en el hueco en que se asigno N_ant*/
                     si t_ejec(N)<= T + t_ejec(N_ant)+T_ant
                          entonces   Marcar N como ejecutable
                                     si ∃ niveles seleccionables de M
                                          entonces insertar M en ℒ
                                     si Tl_ant >=t-ejec(N)-t_ejec(N_ant)
                                          entonces T_libre(Hueco(N_ant))=
                                                   Tl_ant – t-ejec(N)-t_ejec(N_ant)
                                          sino Tl_ant = 0
                                     T_libre(H)= t-ejec(N)-t_ejec(N_ant)-Tl_ant
                                Sino       Final=true
    Mientras T>0 y Final==False
End
```

Figura 5.7. Función asignar_niveles_por_sección

Un ejemplo del resultado de las asignaciones de esta función puede verse en la figura 5.8.

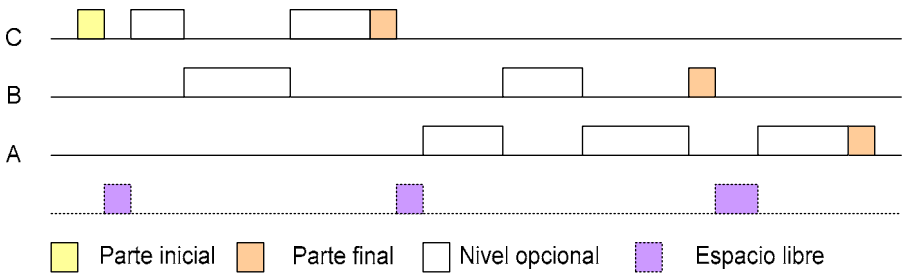


Figura 5.8. Asignación de niveles en zona 2

Como se puede apreciar en la figura anterior pueden quedar pequeñas porciones de los intervalos sin utilizar. Para poder aprovecharlos de una forma no costosa se va a utilizar la posibilidad de adelantar las partes finales de un intervalo. Para ello se va a volver a recorrer la lista de huecos (en su situación actual, es decir con las asignaciones realizadas), pero ahora hacia delante, es decir desde el intervalo actual hasta el más alejado temporalmente.

En cada paso de este segundo bucle que aparece en el algoritmo (en el tratamiento de la zona de tipo 2) se van a llevar a cabo dos operaciones: realizar nuevas asignaciones y un desplazamiento de la parte final que marca el límite superior del intervalo, adelantándolo.

Asignar_niveles_por_sección. La primera operación es decir la asignación de niveles, se va a realizar de nuevo con la función anterior asignar_niveles_por_sección que actuará de la forma ya descrita. Si va a conseguir asignar más niveles de los obtenidos en las activaciones previas, se debe al hecho de que los intervalos van a ver modificado el espacio que aún mantienen libre gracias al tratamiento que va a realizar la siguiente función.

Marcar_adelantamiento_partes_finales. Esta función (Figura 5.9) recibe como entrada el hueco actual, la lista de huecos, y una lista, *LPF*, que almacena cuales serán los nuevos límites superiores de cada intervalo (esta lista estará vacía en la primera invocación a esta función dentro del bucle).

```
Entrada

    H: Hueco actual
    ℒℋ: Lista huecos
    ℒℙℱ: Lista partes finales

Datos
    t_duración: tiempo asignado al hueco
    I: Tarea
    PF, PF_ant: Nodos lista parte finales
Begin
    t_duración = 0
    ∀I ∈final(H)  /* tarea cuya parte final delimita H */
            t_duración = t_duración + tiempo_asignado(I)
    Si vacia(ℒℙℱ)
      entonces crea_nodo(PF)
                Limite_inferior(PF)=Inicio_hueco(H)
                Limite_superior(PF)= Inicio_hueco(H)+t_duración
      sino
               PFant=ultimo(ℒℙℱ)
               crea_nodo(PF)
               Limite_inferior(PF)=Limite_superior(PF_ant)+t_final(PF_ant)
               Limite_superior(PF)= Limite_inferior(PF)+t_duración
     Insertar PF en ℒℙℱ
End
```

Figura 5.9. Función marcar_adelantamiento_partes_finales

Cuando se llega a esta función dentro de un paso del bucle que representa a un hueco determinado, es debido a que en dicho hueco ya se han realizado todas las asignaciones de niveles que entran de forma completa en el, de forma que no es posible realizar ninguna asignación de niveles que permitan su finalización antes de que finalice el intervalo. Para no planear ejecuciones parciales de niveles que deberían ser completadas en intervalos posteriores a riesgo de no finalizarlas, esta función adelanta la ejecución de la parte final (algo totalmente posible como ya se ha comentado) de forma que el espacio libre sin asignar pasa al hueco siguiente aumentándolo y permitiendo que la ejecución de la función asignar_niveles_por_sección, pueda aprovechar el nuevo espacio (de mayor tamaño del que disponía) para hacer nuevas asignaciones.

Aunque el espacio que se transmite de un intervalo al siguiente, es el tiempo que en el intervalo permanecía sin asignar, la parte final no va a ser adelantada en la misma cantidad de tiempo sino en igual o mayor medida. En el intervalo que se está tratando, el algoritmo ha ido asignando

diferentes niveles. Estos niveles corresponden tanto a la tarea cuya parte final marca el final del intervalo que se está estudiando como a tareas cuyo deadline de ejecución es más tardío (es decir se dará en intervalos posteriores). Realmente los niveles que tienen que ser ejecutados obligatoriamente en ese intervalo (es decir que no pueden ser postergados a intervalos siguientes) son los primeros, es decir aquellos cuya parte final acaba el intervalo. Esta función tendrá en cuenta sólo la duración de estos y marcará a su fin el momento en que se debe adelantar la parte final del intervalo, guardando este dato en la lista de límites superiores que va creando. Los niveles de otras tareas que estaban planeadas en este intervalo pasan al hueco siguiente. Este adelantamiento provoca, que el hueco siguiente se vea aumentado una cantidad de tiempo igual a la suma del tiempo sobrante que tenía el hueco tratado y del tiempo que necesitan los niveles desplazados, por lo que el nuevo hueco resultante dispone de tiempo para ejecutarlos.

El adelantar una parte final para ejecutar un componente opcional en un mismo hueco, en lugar de ejecutar parte del nivel antes de la parte final y una vez ejecutada ésta, continuar con la ejecución proporciona dos ventajas adicionales al hecho de facilitar los cálculos. Por un lado permite la aplicación del algoritmo en situaciones donde la ejecución de un nivel no puede ser interrumpida y después reanudada desde el punto donde se dejó, debido a la naturaleza del nivel o del sistema operativo bajo el que se opere. Por otro lado, de esta forma se evitan operaciones de cambio de contexto lo que hace al sistema más eficiente.

El hecho de que finalmente sólo los niveles cuya parte final cierra un intervalo sean los ejecutados en éste y se adelante hasta ese punto la ejecución de la parte final de la tarea, permite que en cuanto se dispone de los resultados de mayor calidad que se prevé se van a disponer para un in-agent no se retrase su aplicación (sea esta la realización de acciones externas, utilización por otras tareas, etc.) que normalmente es llevada a cabo por la parte final. Permitir la ejecución de niveles de otras tareas que pueden ser ejecutados sin ningún problema posteriormente no proporciona ninguna ventaja, y por el contrario se considera aconsejable utilizar un resultado en cuanto este esté disponible.

En la siguiente figura aparece como sería el resultado de aplicar esta función a la situación que aparecía en la figura 5.8 (sólo la primera ejecución del bucle).

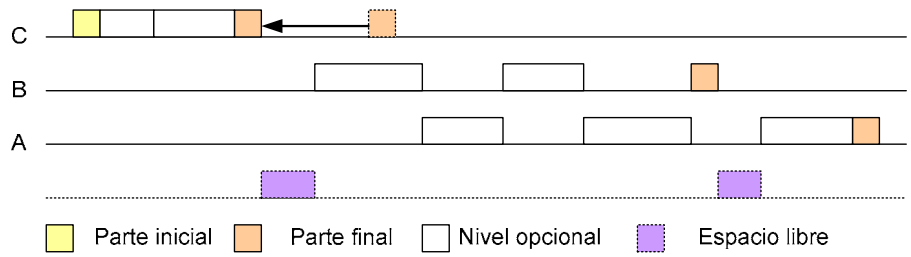


Figura 5.10. Adelantamiento partes finales

En la figura puede verse como la parte final de la tarea A es adelantada desde la posición final (rectángulo punteado) a la nueva posición (indicada por la flecha). Si se compara con la situación previa de la figura 5.6, se puede observar como se han unido los huecos libres existentes. Al ser este espacio mayor la función de asignar niveles_por_secciones podría insertar un nivel que antes no pudo. En cualquier caso se repetirá el proceso con el hueco resultante pasándolo al intervalo siguiente mediante la aplicación de la función de marcar_adelantamiento_partes_ finales.

En la última ejecución de esta función dentro de una zona determinada, se produce como resultado una lista de los instantes temporales en que cada parte final debe ser ejecutada, lista que será utilizada en siguientes fases del control como se comentará. Además se borra la lista de huecos ya que no va a ser utilizada, de manera que en la próxima zona se donde deba construir esta lista esté inicialmente vacía.

Como resumen podemos indicar que cuando acaba la fase de rating, en la zona 2, el algoritmo ha marcado en la lista de MKS activos cuales son los niveles que deben de ejecutarse y una lista que indica el momento en que debe ejecutarse la parte final de los intervalos siguientes.

- Scheduling

En este paso del bucle de control el sistema elige el nivel a ejecutar. También en esta fase el comportamiento es diferente según la zona en que nos encontremos.

Si el control está dentro de un intervalo de la zona 1, se escogerá un nivel de los marcados como ejecutables en la fase de rating. En dicha zona podrán ser elegidos niveles de diferentes tareas no sólo de los que se han activado en ese momento. Una vez elegido se ejecutará y volverá a realizarse esta operación para elegir al siguiente.

Como ya se ha determinado que niveles deben ser ejecutados, la función schedule únicamente debe ir eligiendo aquellos niveles marcados excepto por el hecho de que se debe respetar que en el caso de MKS de refinamiento progresivo un nivel no puede ejecutarse hasta que lo hayan hecho los niveles inferiores del mismo MKS. Únicamente el nivel que se había planificado al final porque su ejecución no cabía en su totalidad, es el que debe ejecutarse en último lugar.

En el caso de la zona 2, independientemente del tipo de intervalo en que se encuentre, la elección del nivel va a depender de los niveles planificados y del intervalo concreto en que se encuentre. De esta manera se elegirá un nivel marcado en la fase de rating pero sólo si corresponde a una tarea cuya parte final cierra el intervalo. Con ello se lleva a cabo la propiedad comentada anteriormente de finalizar una tarea en cuanto no hay más resultados para ella. De nuevo la elección debe tener en cuenta el orden interno de los niveles de un MKS de refinamiento progresivo.

Aunque este sería el comportamiento normal del proceso de scheduling (elegir los niveles previamente marcados por el algoritmo en la fase de rating), hay que tener en cuenta que el tiempo de ejecución real de los niveles puede desviarse del previsto, que es el utilizado para decidir los niveles al ejecutar. Debido a esta razón es necesario llevar a cabo unas medidas correctoras tanto en el control de ejecución como en el propio proceso de scheduling.

En primer lugar se quiere controlar que un nivel no se desvíe en demasía de su tiempo previsto. Por ello el control establece un margen máximo de exceso del tiempo de ejecución (fijado en un 15%). Si la ejecución sobrepasa dicho límite el nivel es interrumpido, ya que se considera que puede afectar muy negativamente al resto de niveles planeados. En cualquier caso esta desviación es tenida en cuenta para actualizar el tiempo promedio del nivel que será usado en siguientes entradas del algoritmo de planificación.

El exceso de tiempo por un nivel puede provocar que la ejecución de ciertos niveles planeados exceda el tiempo previsto de finalización del intervalo por el planificador, sin embargo el control acepta este hecho siempre y cuando no sea más allá del deadline de ejecución que si debe ser respetado, ya que ello no provoca más que una ligera desviación en cuanto a lo planeado, pero no un mal funcionamiento al no violarse restricciones críticas. Además dicho aumento de tiempo puede además ser compensado por ejecuciones de menor duración que la prevista de otros niveles.

Puede ocurrir que la suma de los tiempos en exceso y por defecto de los niveles que no se han ejecutado en el tiempo previsto resulte negativa. Es decir que el instante en el que se ha acabado de ejecutar todos los niveles que han sido planeados para un intervalo dado sea menor que el instante planeado y guardado en la lista $\mathcal{LPF}$ que mantenía cuando debían de ejecutarse las partes finales. En este caso se dispone de un tiempo extra que el planificador no tuvo en cuenta. En este momento el scheduler puede decidir escoger para ejecución un nivel no planificado de la tarea cuya parte final cierra el intervalo. La elección tendrá en cuenta la calidad que dicho nivel puede proporcionar, ya que el tiempo que puede conseguirse de esta manera raramente permitirá la ejecución de más niveles, y por tanto no es aconsejable (ni habrá mucho tiempo para ello) el hacer un análisis mayor para hacer una planificación mejor de dicho espacio sobrante. Si en dicho hueco no puede ejecutarse nada más, simplemente se adelanta definitivamente la parte final, con lo que el tiempo sobrante pasa al intervalo siguiente donde podrá ser usado para una situación similar o compensar posibles excesos de tiempo.

5.2.3 Algoritmo SSSM

Cuando se estudia el comportamiento de un problema de tiempo real, puede contemplarse cómo este puede pasar por dos situaciones diferentes en cuanto a la variabilidad de los indicadores (lecturas de los sensores, etc.) que lo caracterizan. Por un lado puede presentarse un comportamiento más dinámico, con una fuerte variación en los datos de entrada para las mismas variables. En cambio, en otras fases el problema tiende a estabilizarse, resultando en escasas variaciones en las variables de entrada o por lo menos en algunas de ellas.

Cuando los datos de entrada no varían de forma significativa, es lógico que los resultados de aquellos componentes que deben proporcionar una solución tampoco varíen. La consecuencia inmediata sería utilizar otra vez los resultados calculados con anterioridad en lugar de repetir los cálculos para llegar a obtener las mismas soluciones.

Una primera solución para conseguir este funcionamiento sería implementar un sistema de mantenimiento de la razón que mantuviese constancia de que datos siguen siendo validos. Este sistema debería ser lo suficientemente ágil para ser usado en un entorno de tiempo real.

La modificación que planteamos en la heurística presentada y que denominamos SSSM (Slack Slide Scheduling with Memory – SSS con

Memoria-), pretende ir un paso más allá y utilizar los resultados previos no sólo para disponer de la solución sino como punto de partida para calcular mejores soluciones. Es decir transportar la mejora de resultados que proporcionan los diferentes niveles de un MKS a diferentes activaciones de una misma tarea y no sólo dentro de una misma activación cómo se hacía hasta ahora.

El primer paso será conocer si los resultados siguen siendo validos o no. Para este propósito se utiliza un recurso que proporciona la arquitectura ARTIS. Como ya se ha comentado uno de los componentes de esta arquitectura es el KDM o memoria global. En esta memoria se guardan toda la información que el agente recoge, y además se utiliza como centro de intercambio de información entre tareas. El KDM no es una memoria pasiva, pues dispone de diferentes mecanismos para realizar operaciones de mantenimiento de la coherencia lógica de los datos, procesos de meta-razonamiento con los datos, etc. Uno de estos mecanismos vincula subconjuntos de datos a in-agents que están interesados en ellos y asocia a cada dato qué se considera como cambio significativo.

Cuando se produce una modificación de un dato (que entre dentro de lo que se ha definido como significativo para ese dato), el KDM informa a los in-agents que están interesados en este dato para que realicen posibles operaciones asociadas a este cambio. De esta manera se puede vincular un in-agent con sus datos de entrada, de manera que el in-agent puede conocer si se ha producido alguna modificación en alguno de ellos desde su última activación.

Cada in-agent almacena de forma local los resultados que calcula. Para poder utilizar el SSSM además guardará cual fue el nivel de mayor rango que ejecutó completamente para cada MKS en la última activación. También tendrá un campo que marcará si ha sufrido o no la modificación de sus datos de entrada (campo que se modificará por el mensaje recibido del KDM si lo ha habido).

Las fases de rating y scheduling que proporcionaban el algoritmo SSS no se ven modificadas con el nuevo algoritmo. El SSSM si incluye su participación en una fase previa del bucle de control, la de triggering. Como se comentó en el capítulo anterior, en esta fase se insertan en una lista todos aquellos MKS pertenecientes a tareas (si los hubiera) que se han activado en la parte critica que marca el inicio del intervalo actual. Cuando se insertan, se hace incluyendo todos sus niveles para que estos entren en competencia. Además, en la lista se incluyen MKS de tareas que se activaron en intervalos anteriores y que todavía no ha vencido sus

deadlines. Para estos últimos sólo se incluyen aquellos niveles todavía no ejecutados (en el caso de métodos múltiples, además sólo aquellos de mayor calidad al ya ejecutado).

```
Datos
    Sea ℒ la lista de MKS activos

Begin
    ∀ i ∈Conjunto tareas recién activadas
      Si cambios_entrada( i ) == False
            Introducir en ℒ todos los MKS de i  con todos sus niveles
      sino   ∀ m∈ Conjunto de MKS de i
            Introducir nivel restauración de m
            Introducir niveles de m de rango superior al último ejecutado
End
```

Figura 5.11. Algoritmo SSSM (fase de trigger)

El algoritmo SSSM va a provocar el siguiente cambio. Cuando el planificador de segundo nivel recibe la lista de tareas que acaban de ser activados, consultará para cada uno de ellos si ha habido modificación o no mirando el campo mencionado. En el caso de haber habido modificación, los resultados previos no son útiles, con lo cual se insertará en la lista todos los MKS en su forma original (es decir con todos sus niveles elegibles) como se hacía hasta ahora.

Sin embargo, si se comprueba que no ha habido cambios, el SSSM va a insertar para cada MKS del in-agent dos elementos diferentes. Por un lado un nivel que denominamos de restauración y por otro lado aquellos niveles del MKS de rango superior a aquel que la propia tarea guarda como el último ejecutado (en el caso de métodos múltiples los de calidad superior a dicho nivel).

El nivel de restauración, es un nivel especial en cuanto que no realiza ningún tipo de cálculo relacionado con el problema a resolver. Este nivel que es introducido en la lista como un nivel más, únicamente tiene como objetivo volcar los resultados anteriores a la memoria global (resultados que el in-agent tiene para el MKS almacenados localmente desde su activación anterior). Los resultados deben ser volcados, ya que aunque los datos de entrada no hayan sido modificados, alguna otra tarea si puede haber escrito sobre dichos datos de salida en el KDM.

Este nivel tendrá grandes posibilidades de ser ejecutado, puesto que su ratio será bastante elevado, ya que su calidad será la que proporcionaba el nivel que calculó las respuestas que el restaura, pero por el contrario su tiempo de ejecución será muy pequeño ya que no tiene que hacer prácticamente cálculos.

De esta manera el sistema puede aprovechar los cálculos ya realizados para disponer de una solución opcional sin apenas coste, y utilizar el tiempo disponible para mejorar la solución. En el caso de los MKS de refinamiento progresivo, el nivel de restauración sustituye al nivel ejecutado de mayor rango, de manera que si aquel era el nivel n servirá para que se pueda ejecutar (si se decide así) el nivel n+1 pues se debe seguir cumpliendo la restricción de no ejecutar un nivel, si antes no lo ha hecho el de rango inferior.

En el caso de los métodos múltiples, simplemente evita ejecutar aquellos que van a proporcionar una calidad inferior.

En ambos tipos de MKS si no existen niveles que mejoren la calidad de los resultados existentes sólo se incluirán en la lista los niveles de restauración.

De esta manera el SSSM en regimenes estables de funcionamiento, cuando las entradas permanecen en mayor o menor grado invariables, permite que el sistema tienda a conseguir unas respuestas de mayor calidad, aprovechando los huecos de slack que el mismo tiene que manejar para ejecutar niveles que en activaciones previas no pudieron llegar a ejecutarse por falta de tiempo y/o competencia de otros niveles con mejor ratio.

5.3 COSTES

Un aspecto a tener en cuenta es el coste del algoritmo de planificación, pues este va a ejecutarse en tiempo de slack y quitaría tiempo para ejecutar niveles. Cómo va a comprobarse a continuación el orden del coste del SSS (posteriormente se verá el del SSSM) no es elevado y además es comparable al del HSF, que era el método voraz que siendo aplicable a nuestro entorno (no como los deliberativos por las razones comentadas) mejores resultados proporcionaba.

Heurística HSF. En primer lugar por su simplicidad, calcularemos el coste para el caso peor del HSF. El caso peor ocurre cuando al principio de un

intervalo de slack tenemos activas todas las tareas del problema y no se ha ejecutado ningún nivel de ningún MKS.

Vamos a considerar el problema desde el punto de vista de los MKS y sus niveles, ya que en la planificación la lista a manipular está compuesta por los MKS y estos por los niveles, sin distinción de si pertenecen o no a un mismo in-agent. De esta manera denotamos por M el número de MKS total del problema (es decir todos los MKS de todas las tareas) y N el número total de niveles.

En primer lugar el algoritmo debería tener una lista ordenada de MKS, orden dado por el ratio calidad vs tiempo de ejecución del primero de sus niveles. El primer nivel es el nivel 1 en los de refinamiento progresivo, en el caso de métodos múltiples el de mejor ratio dentro del propio MKS. Puesto que la identidad de los MKS le va llegando uno a uno al planificador de segundo nivel desde el de primer nivel, lo más eficiente sería ir construyendo la lista de forma ordenada. Utilizando inserción ordenada tendríamos un coste de $O(M*\log_2 M)$ (ya que el tamaño de la lista ordenada es M, pues se tiene en cuenta sólo un nivel por MKS).

Cada vez que se ejecuta un nivel, se debería realizar una reordenación, pues el MKS puede variar su posición en la lista, pues ahora vendrá determinada por otro nivel que tendrá diferente ratio que su predecesor. Sin embargo como sólo varía un MKS, no hace falta una reordenación total, sino que basta una inserción directa del MKS en su nueva posición. El coste de esta inserción será de $O(\log_2 M)$. Como esta operación habría que repetirla tantas veces como niveles puedan ser ejecutados, el coste total sería de $O(N*\log_2 M)$.

El coste total de esta heurística sería de $O((N + M) * \log_2 M)$.

Heurística SSS. En este caso habría que tener en cuenta tanto la fase de rating como la de scheduling, sin embargo como veremos es realmente la primera la que va a aportar mayor peso al coste.

En el rating cuando se trata la zona 1 el proceso es similar al realizado en la heurística HSF, con la diferencia de que en el HSF se ejecutaba directamente el nivel elegido, mientras que en el SSS se marca como planificable, pero el proceso de elección por niveles, insertando de forma ordenada para considerar el siguiente nivel, es idéntico. Por lo tanto el coste en el caso peor sería del mismo orden que el SSS.

En la zona 2, el coste vendría en el primer intervalo de dicha zona, que es el momento donde se genera el plan para dicho intervalo y los siguientes que pertenecen a la misma zona. La diferencia principal entre este problema y el del HSF va a consistir en la existencia de varios intervalos en lugar de uno sólo y los dos recorridos (desde el final al principio y al revés que se realizan). Sin embargo si nos fijamos en la naturaleza del proceso, podemos observar que las operaciones son la mismas y que en toda la extensión de la zona, independientemente que esta esté dividida vamos asignando nivel por nivel (un nivel sólo una vez) y cada vez que hacemos esta asignación volvemos a insertar el MKS con el siguiente nivel. Por tanto el proceso de ordenaciones por cada inserción de nivel es de la misma magnitud, siendo por tanto el mismo coste de $O((N + M) \log_2 M)$.

Se podría plantear el coste conjunto de las zonas 1 y 2 pues ambas zonas representan la activación y finalización de una serie de tareas, pero precisamente por ello, aquellos niveles ya ejecutados en una zona 1, no serán planificados en subsecuentes zonas 1 y en la zona 2 posterior aunque la tarea esté activa. No sólo es el hecho de que no sean planificados por haberlo hecho ya, sino además no aparecen en la propia lista a planificar, es decir no hay que realizar operaciones con dichos niveles en la propia planificación y no producen coste en esta. Por ello el total de las operaciones de ordenación (que al final cómo se comentará son las que mayor aporte hacen al coste) incluso considerando ambas zonas conjuntamente sigue siendo de $O((N + M) \log_2 M)$.

En el algoritmo SSS hay algunas operaciones adicionales. Una de ellas es la creación de la lista de huecos, pero el coste de esta operación es en el caso peor de I (siendo I el número de tareas), cuando por cada tarea del problema se creara un intervalo de forma anidada. Este coste es despreciable frente al anterior. La creación de la lista que marca el adelantamiento de partes finales también tendría coste de orden I por la misma razón.

En la fase de scheduling simplemente se van eligiendo para ejecutar los niveles que han sido marcados, el coste de operación sería N si se eligieran todos los niveles, también es un coste de orden menor al anterior.

Por tanto el coste del algoritmo SSS es del mismo orden al HSF, es decir: $O((N + M)\log_2 M)$.

Si hablamos del SSSM, tenemos que distinguir dos partes: la fase de triggering donde actúa el algoritmo en si, y el proceso externo de comprobación de cambios. En la fase de scheduling (que es el coste

propiamente dicho del algoritmo SSSM), para cada tarea se comprueba en su marcador si ha recibido o no comunicación de cambios. Su coste será por tanto proporcional al numero de tareas I. Por cada MKS sería posible insertar un nivel de restauración, luego tendría un coste adicional de M. Ambos costes siguen siendo despreciables frente al calculado previamente. En la fase de rating y scheduling, aparecen nuevos niveles a planificar y ejecutar (los de restauración), pero cada uno de ellos sustituye al menos a un nivel de los ejecutados con anterioridad (si no estos niveles de restauración no se insertarían), con lo que el número de niveles a planificar y ejecutar será igual o menor. Seguimos por tanto con coste $O((N + M) \log_2 M)$.

El otro aspecto es la comunicación a cada in-agent de si se han producido cambios o no. Esta cuestión es externa al SSSM, pues no la controla el mismo y por lo tanto el coste dependerá del mecanismo utilizado para realizarlo. En nuestro caso el proceso se realiza en la propia actualización de los datos (cuando se actualiza el dato se marca el cambio en el in-agent interesado) y depende del número de datos de entrada en los que está interesado cada in_agent.

6 PRUEBAS

Una vez desarrollados los algoritmos SSS y SSSM es necesario comparar su comportamiento frente al de las otras heurísticas ya comentadas, en concreto frente a las voraces puesto que aunque menos eficientes en cuanto a la calidad global de resultados obtenida, si son implementables en la arquitectura ARTIS frente a las deliberativas, que aunque más eficientes respecto a dicha calidad (algunas como la planificación deliberativa de Dean y Boddy son óptimas), no son implementables en nuestro sistema por diversas razones como se justifica en el capítulo 5.

En este punto empezaremos describiendo el simulador donde se han llevado a cabo las pruebas y el porqué de su utilización. Posteriormente se describirán las pruebas realizadas y los resultados obtenidos para cada uno de los dos algoritmos.

6.1 SIMULADOR

Cuando se pretende comparar el comportamiento de cada heurística, es evidente que es necesario llevar a cabo un gran número de pruebas con el objetivo de que los resultados no dependan de las características particulares de una prueba determinada.

Para poder realizar las pruebas se ha desarrollado un simulador que forma parte de una herramienta denominada InSide [Sol00] desarrollada para facilitar el diseño de agentes ARTIS. La razón para utilizar un simulador para las pruebas y no directamente aplicaciones concretas que usen agentes ARTIS se debe fundamentalmente al gran número de pruebas necesarias. En el caso de utilizar aplicaciones reales, sería necesario desarrollar para cada prueba in-agents diferentes, cada uno de ellos con MKS que a su vez deben contar con niveles que implementarán código real, diferentes para cada prueba puesto que deben resolver el problema que plantea la aplicación real.

Esto presenta dos grandes problemas. Por un lado cada aplicación real, por sus propias características limita los in-agents que se necesitan para resolverla, en cuanto a cuales van a ser sus periodos y deadlines, el número, tiempo de computo o tipo de los niveles que van a componer sus MKS. Una aplicación real va a permitir pocas variaciones en cuanto a dichas características, por lo que para realizar el alto número de pruebas que queremos deberíamos disponer de un gran número de aplicaciones. Este planteamiento resulta inviable si hablamos de cientos de pruebas, más aún si tenemos en cuenta que debería implementarse código real (que resolviese las tareas de los in-agents) para cada nivel de cada prueba. Además las pruebas no gozarían además de la suficiente aleatoriedad para considerarse independientes.

El segundo problema, relacionado con el primero, en cuanto a que estaríamos usando código real, sería en relación al tiempo necesario para realizar las pruebas. En el simulador el tiempo es también simulado, pero en una aplicación real no. El tiempo necesario para ejecutar cientos de pruebas con aplicaciones reales sería inviable.

El simulador por el contrario nos permite diseñar un gran número de pruebas, con características diferentes entre si para poder comparar diferentes circunstancias y con unos tiempos de ejecución que permiten llevarlas a cabo en unos tiempos razonables. El hecho de usar simulaciones no afecta a la comparación entre heurísticas que es lo que realmente nos interesa. En cualquier caso, ejemplos de la aplicabilidad de los métodos propuestos a aplicaciones reales, se presentan posteriormente en los anexos A y B.

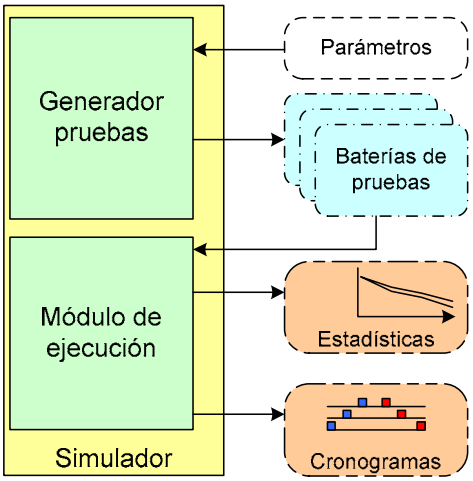


Figura 6.1. Estructura del simulador

El simulador está compuesto por dos partes (figura 6.1): el generador de pruebas aleatorias y el módulo de ejecución. El generador de pruebas va a generar baterías de pruebas, cada una de ellas compuesta por un número determinado de pruebas básicas y de acuerdo con unos parámetros especificados. El generador creará las pruebas de manera que cada característica que define la prueba estará dentro de un rango delimitado por los parámetros proporcionados.

Estas baterías de pruebas servirán de entrada al módulo de ejecución que simulará la ejecución de un agente ARTIS y proporcionará las estadísticas de los resultados obtenidos. Además, para cada prueba concreta, también se puede obtener un cronograma que refleja la evolución del sistema, mostrando que elementos se han ido ejecutando, en que momento y durante cuanto tiempo.

Las características que definen una prueba ya han sido comentadas cuando se describieron los componentes de un agente ARTIS y son:

- Número de in-agents del agente. Cada prueba puede tener un número diferente de in-agents.

- Periodos y deadlines de cada in-agent. Los in-agents dentro de una misma prueba tendrán diferentes periodos y deadlines entre si.

- Número de MKS por cada in-agent. Igual que ocurre con la característica anterior, el número de MKS de cada in-agent dentro de una misma prueba no tiene porqué ser el mismo.

- Número de niveles opcionales e importancia de cada MKS. Cada MKS tendrá un valor numérico, la importancia, que reflejará su aportación a la calidad del sistema. Este valor junto al número de niveles por MKS también variará por cada MKS de la prueba.

- Tiempos de ejecución y calidad de los niveles. Cada nivel estará también definido por su tiempo de ejecución y su calidad, de manera que la conjunción de estos parámetros para los niveles de un MKS determinado constituye su perfil de ejecución.

Además de estos parámetros ya comentados, aparecen una serie de parámetros adicionales que definirán las pruebas:

- Carga crítica. Representa que porcentaje del tiempo total de ejecución del sistema es consumido por los componentes críticos del agente.

- Carga opcional. Representa el porcentaje de tiempo que sería necesario para ejecutar todos los niveles opcionales del agente frente al tiempo total de ejecución del sistema.

  Este porcentaje puede ser mayor del 100%, lo que se denomina saturación y representa el hecho de que no se dispone de tiempo suficiente para ejecutar todos los niveles opcionales. Este hecho no debe considerarse como erróneo y debido a un mal diseño pensando que simplemente sobran los niveles más altos que no serían nunca ejecutados. Al contrario, esta situación suele ser la habitual y la más interesante (en situaciones donde todo puede ser ejecutado la planificación no resulta problemática). Hay que tener en cuenta que el hecho de que no se disponga de tiempo suficiente para ejecutar todos los niveles opcionales en todas las activaciones de sus in-agents correspondientes, no significa que siempre vayan a ser los niveles más altos los descartados, pues dependerá en cada momento de los in-agents que estén activos y de sus respectivas calidades. Además incluso en una saturación del 100%, que podría considerarse límite, no se va a conseguir la ejecución de todos los niveles para todas las activaciones. Hay que tener en cuenta que este porcentaje es una relación entre el total del tiempo necesario y el total del disponible, pero dependiendo de las diferentes activaciones de los in-agents tendremos intervalos donde entran en conflicto diferentes in-agents no existiendo tiempo para la ejecución de todos sus niveles mientras que en otros intervalos existirá tiempo sobrante que no puede ser utilizado porque no hay más niveles que puedan activarse.

- Relación entre deadline y periodos. En el sistema los periodos de los in-agents puede coincidir con sus deadlines (con lo cual en todo momento hay in-agents activos) o pueden existir deadlines menores que su periodo. Este último caso es más restrictivo en cuanto a tiempo, y además puede producir la existencia de intervalos temporales durante los cuales ningún in-agent esté activo.

6.2 CRITERIOS DE COMPARACIÓN

A la hora de comparar las heurísticas, y puesto que el objetivo es obtener el mejor comportamiento global del in-agent, el parámetro a considerar será la calidad obtenida dentro del hiperperiodo. El hiperperiodo es el intervalo de tiempo calculado como el mínimo común múltiplo del periodo de todos los in-agents y es importante ya que una vez se ha llegado a este límite se vuelve al estado inicial en cuanto a que las relaciones entre activaciones de

los diferentes in-agents se van a repetir produciendo las mismas situaciones. Por ello basta con estudiar el comportamiento de cada heurística (es decir la calidad que se obtiene) dentro del hiperperiodo que define cada prueba.

Para poder realizar las comparaciones necesitamos antes definir una serie de variables con que construir las formulas que nos proporcionen el índice de comparación. Estas variables son:

- N, el número de in-agents.
- MR_i, el número de MKS de refinamiento progresivo del in-agent i.
- MM_i, el número de MKS de tipo método multiple del in-agent i.
- ACT_i, el número de activaciones del in-agent i.
- I^m_i, la importancia del MKS m del in-agent i.
- L^m_i, el número de niveles del MKS m del in-agent i.
- q^l_{mi}, la calidad del nivel l del MKS m del in-agent i.
- q^m_{max}, la máxima calidad entre los niveles del MKS m (del tipo métodos múltiples).
- e^l_{mi}, número de ejecuciones del nivel l del MKS m del in-agent i.
- ef^l_{mi}, número de ejecuciones efectivas del nivel l del MKS m del in-agent i, siendo MKS del tipo métodos múltiples. Se considera efectiva a la ejecución de un nivel correspondiente a un MKS de tipo método múltiple, si en la misma activación de su MKS no se llega a ejecutar ningún otro nivel del mismo MKS que proporcione más calidad. Es decir si en una misma activación de un MKS de tipo métodos múltiples se ejecutan varios de su nivel, solo se contabilizará como efectiva la ejecución del que proporcione mayor calidad, ya que son alternativos entre si y no acumulan calidad.

El criterio básico de comparación se definiría como la calidad real absoluta obtenida en el hiperperiodo o CRA (Fórmula 6.1). Este valor sería la suma de las calidades proporcionadas por cada nivel ejecutado multiplicadas por la importancia del MKS al que pertenecen en el caso de MKS de

refinamiento progresivo. Para los MKS de tipo métodos múltiples se consideran sólo las ejecuciones efectivas.

$$CRA = \sum_{i=1}^{N}\left(\sum_{m=1}^{MR_i}\left(I_m^i * \sum_{l=1}^{L_m^i}\left(q_l^{m,i} * e_l^{m,i}\right)\right) + \sum_{m=1}^{MM_i}\left(I_m^i * \sum_{l=1}^{L_m^i}\left(q_l^{m,i} * ef_l^{m,i}\right)\right)\right)$$

Fórmula 6.1. Calidad real absoluta (CRA)

El problema con usar este índice es que sólo podríamos comparar las heurísticas prueba a prueba. Si pretendemos hacer cálculos estadísticos como medias tendríamos que combinar datos de diferentes pruebas, pero las pruebas producen resultados tan dispares entre si como sus características. No tiene sentido hacer medias de unas CRA cuyos valores sean por ejemplo de 15000 para una heurística y 15500 para otra frente a valores de 50 y 40 para las mismas heurísticas en otra prueba, cuando las causas de la diferencia de magnitudes es debida por ejemplo, a la diferencia en la duración de los periodos de cada prueba. Evidentemente los valores altos anularían a los más bajos, cuando éstos pueden ser igual o más relevantes que los primeros.

Para evitar este problema se plantea usar en lugar de la CRA otra medida que denominaremos calidad real relativa o CRR (Fórmula 6.2). Esta se define como el cociente entre CRA y la calidad óptima o CO. El índice CO sería la calidad que se obtendría en una prueba con las mismas características que la que se evalúa excepto con una carga opcional del 100% (es decir justo en el limite de la saturación) y suponiendo que se hiciera una planificación óptima, es decir aquella que con las características del sistema proporcionaría los mejores resultados.

$$CRR = \frac{CRA}{CO}$$

Fórmula 6.2. Calidad real relativa (CRR)

El utilizar la saturación del 100% y no la real para el cálculo de la óptima no afecta a nuestro objetivo de comparar las heurísticas. Si usáramos la saturación real el porcentaje indicado por CRA variaría. Normalmente como usaremos saturaciones superiores al 100%, la CRA sería menor ya que si los niveles duran más tiempo y disponemos del mismo tiempo total, ejecutaremos menos niveles. Lo mismo ocurre con la calidad óptima. El resultado sería que la CRR permanecería más o menos constante para pruebas similares donde solo cambia la carga opcional. Por el contrario al

usar la prueba con saturación del 100% para la calidad óptima, tendremos que la calidad absoluta (el numerador) va disminuyendo al aumentar la carga opcional, pero no la óptima (el denominador) con lo cual el cociente va disminuyendo.

Pero a nosotros nos interesa la comparación entre heurísticas y no el valor en si de la calidad para cada heurística. De esta manera la disminución en la calidad relativa al usar la saturación del 100% afecta por igual a todas las heurísticas y puede ser usada para la comparación. Por el contrario al utilizar esta saturación para la calidad óptima obtenemos otra información que se perdería de no ser calculada de esta manera. Se trata de poder observar como afecta el aumento de la carga opcional en los resultados del sistema, es decir en la disminución de la calidad obtenida. Como hemos comentado anteriormente, la calidad absoluta si disminuye, pero de usar la calidad óptima calculada con la saturación real no podríamos percatarnos de ello con el estudio de la calidad relativa, ya que la disminución de los dos componentes que se usan para su calculo hace que se compensen. El uso de la calidad óptima tal y como se ha definido si que muestra esta disminución de los resultados.

El valor CRR sería entonces una medida que indicaría el porcentaje respecto del resultado óptimo que es alcanzado en una prueba en particular, este valor ya permitiría comparar resultados de pruebas diferentes y hacer medias entre ellos.

$$CRR = \frac{CRA}{CSC}$$

Fórmula 6.3. Corrección de la calidad real relativa (CRR)

El problema surge a la hora de calcular CO. Calcular el valor óptimo haría necesario disponer de un método de planificación que calculase en un tiempo razonable la planificación óptima, pero este es un problema NP-completo. Por ello la calidad relativa se va a modificar (Formula 6.3) utilizando para su calculo no la calidad óptima sino una cota superior (CSC) de ésta que se puede calcular más fácilmente (Fórmula 6.4).

$$CSC = \sum_{i=1}^{N} \left(ACT_i * \left(\sum_{m=1}^{MR_i} \left(I_m^i * \sum_{l=1}^{L_m^i} q_l^{m,i} \right) + \sum_{m=1}^{MM_i} \left(I_m^i * q_{\max}^m \right) \right) \right)$$

Fórmula 6.4. Cota superior de la calidad (CSC)

Para su cálculo se considera que se pudieran ejecutar todos los niveles de cada in-agent en cada una de sus activaciones. Igual que ocurría con el hecho de utilizar una saturación del 100%, el hecho de no obtener la calidad relativa real sino una aproximación por usar una cota superior, no representa ningún problema, ya que como se ha explicado nos interesa la comparación entre heurísticas y el uso de la cota superior afecta a todas las heurísticas por igual.

6.3 PRUEBAS

6.3.1 Heurísticas voraces.

El primer objetivo ha sido comparar entre sí las diferentes heurísticas voraces que comentamos en el capítulo 4, es decir BIF, EDF, DM, HQF y HSF. De esta manera obtener cuales producen mejores resultados y posteriormente comparar la nueva heurística solo con las mejores. Para ello se generó una batería de 500 pruebas con valores para los parámetros según se refleja en la tabla 6.1.

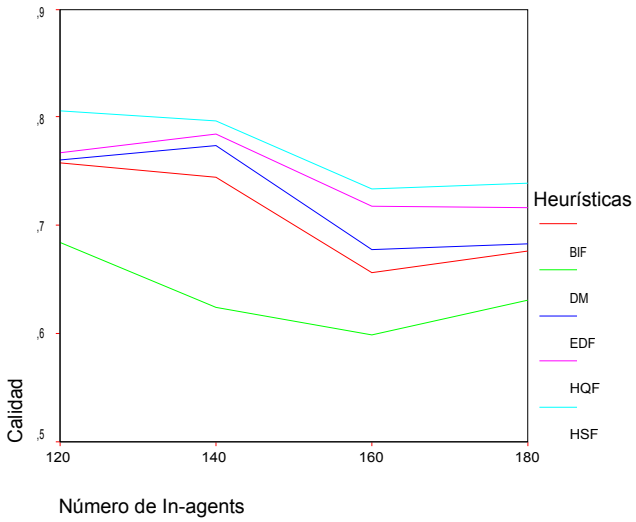
DATO	RANGO DE SELECCIÓN	FORMA DE SELECCIÓN
Número de in-agents	(3, 6, 9, 12)	Uno de los valores
Número de MKS	(1, 2, 3, 4)	Uno de los valores
Número de niveles por MKS	(1, 2, 3, 4)	Uno de los valores
Importancia	(1, 2, 3)	Uno de los valores
Período de cada in-agent	[1000 – 100000]	Un valor del intervalo
Deadline de cada in-agent[1]	[1000 – 100000]	Un valor del intervalo
Deadline de cada in-agent[2]	(70%, 80%, 90%, 100%)	Uno de los valores
Saturación opcional	[100% – 200%]	Uno de los valores

[1] : Deadlines igual al Período;

[2] : Deadlines menor en un porcentaje al período

Tabla 6.1. Parámetros de generación de las baterías de pruebas

Todas las pruebas fueron realizadas para cada una de las cinco heurísticas anteriores y los resultados se reflejan en la gráfica 6.1.



Gráfica 6.1. Comparación entre heurísticas

Como se pude observar en esta gráfica el resultado indica que la heurística HSF, es decir aquella que utiliza el ratio calidad vs. tiempo de ejecución es la que produce los mejores resultados.

Este resultado era el esperado y de hecho este criterio es usado como base para otras planificaciones como la deliberativa de Dean y Boddy, la de la heurística marginal de Etzioni y también es utilizada en las heurísticas desarrolladas en este trabajo. Por ello a partir de este momento se utilizará los resultados de la heurística HSF como base de comparación.

6.3.2 Heurística SSS

Para probar la heurística SSS, se han generado 8 baterías de pruebas, cada una de ellas de 500 pruebas individuales. Los parámetros de cada una de ellas se distribuyen aleatoriamente según las condiciones indicadas en la tabla 6.1. El hecho de diferenciar 8 baterías o escenarios diferentes ha sido para comprobar si ciertas características del problema podían hacer diferenciar los resultados de una u otra heurística.

En concreto por un lado se quería ver la influencia del rango de variación del periodo, usando baterías donde los periodos de los in-agents de cada prueba varían poco entre si frente a baterías donde en cada prueba hay mayor variación entre los periodos.

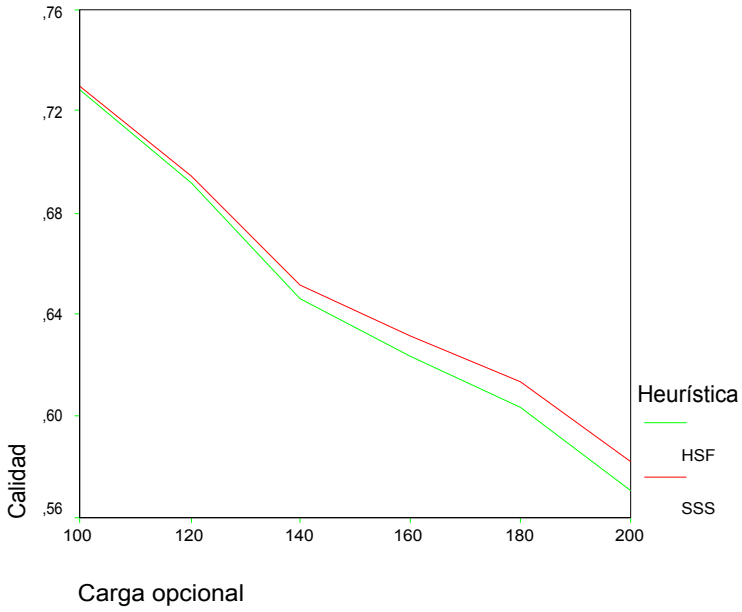
El otro factor que se quería comprobar su posible influencia es la relación deadline vs. periodo de cada in-agent. Por un lado en unos escenarios los in-agents tendrán periodos iguales a su deadline y en otras cada deadline puede suponer del 70% al 100% de su periodo correspondiente, tal y como figura en la tabla 6.1. Los 8 escenarios creados tendrían las características reflejadas en la tabla 6.2.

RANGO DE LOS PERIODOS	**RELACIÓN DEADLINE VS. PERIODO**	
	100%	70% - 100%
1000 – 1300	Escenario 1.	Escenario 5.
1000 – 5000	Escenario 2.	Escenario 6.
1000 – 15000	Escenario 3.	Escenario 7.
1000 - 100000	Escenario 4.	Escenario 8.

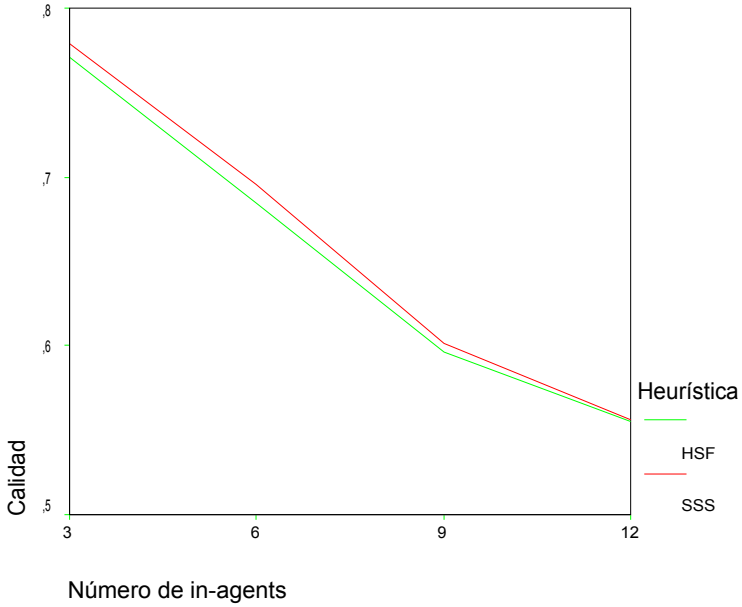
Tabla 6.2. Características de los escenarios de prueba

Para cada escenario se va a representar la CSC media de todas las pruebas en función de dos parámetros, por un lado el número de in-agents de cada prueba y por otro de la carga opcional de cada prueba.

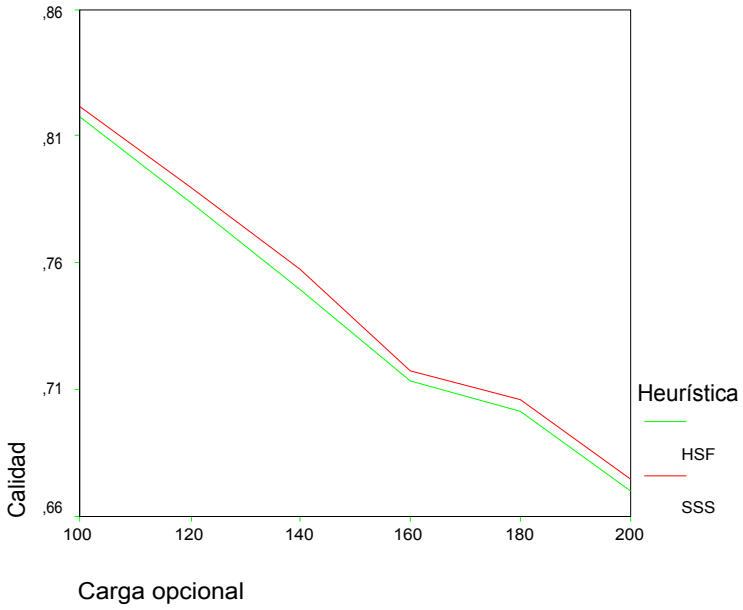
Los resultados se presentan en las gráficas que se presentan a continuación. Para cada escenario por tanto dispondremos de dos gráficas, una para cada factor comentado anteriormente.



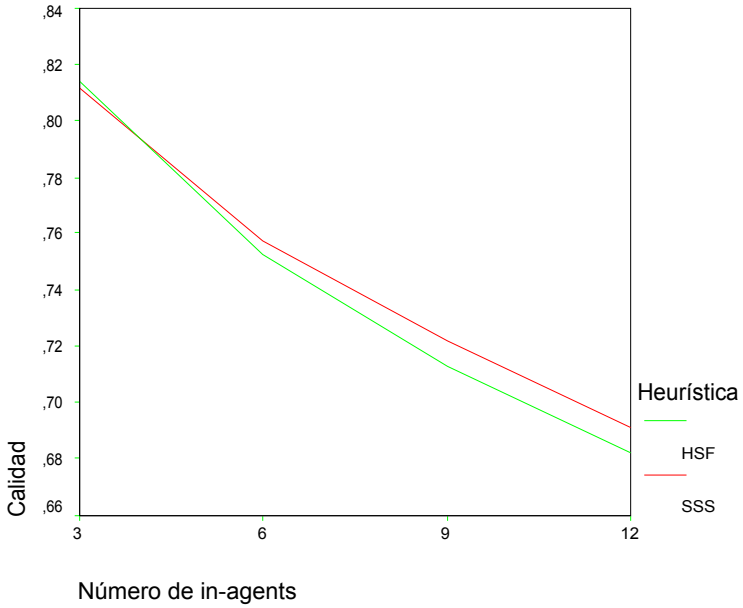
Gráfica 6.2. Escenario 1. Calidad vs. carga opcional



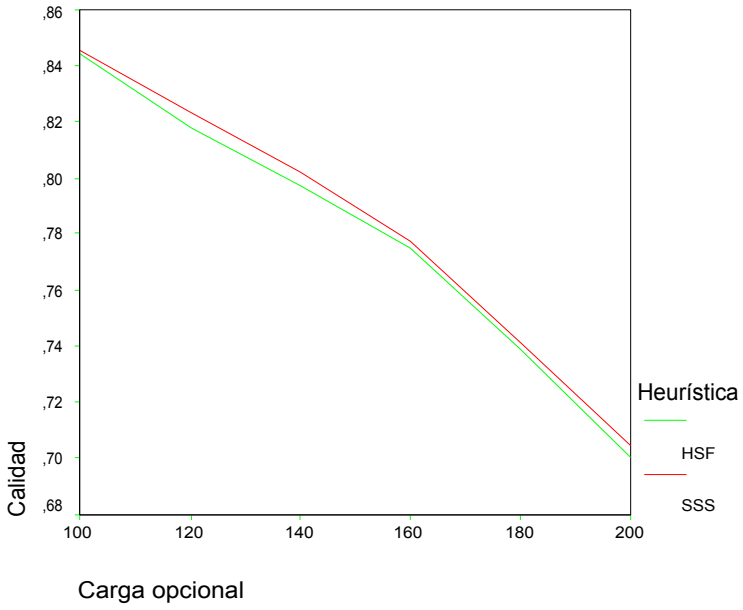
Gráfica 6.3. Escenario 1. Calidad vs. número de in-agents



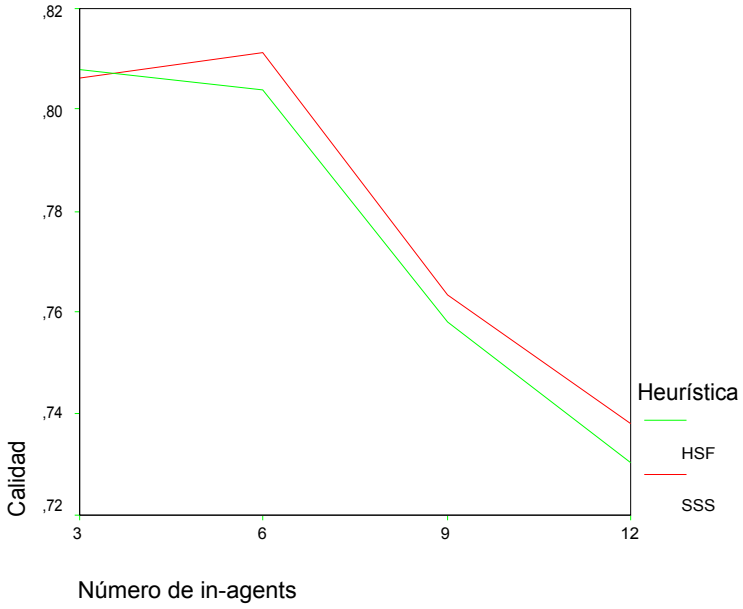
Gráfica 6.4. Escenario 2. Calidad vs. carga opcional



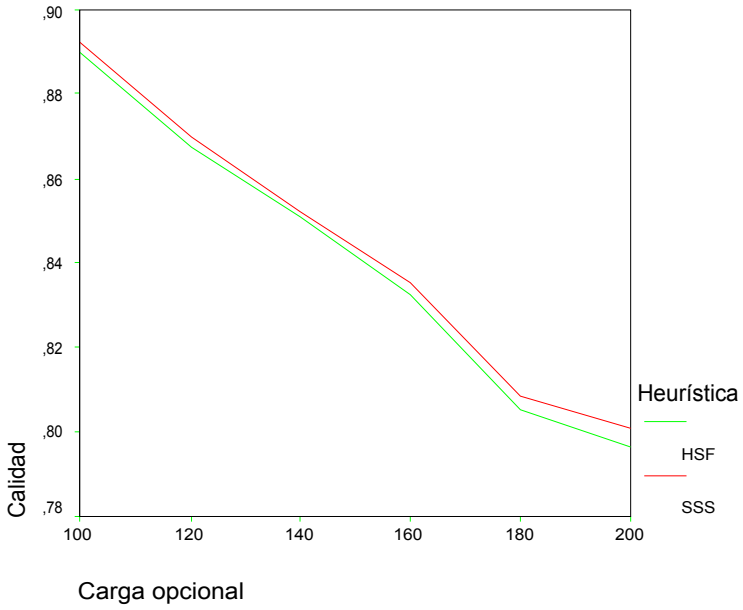
Gráfica 6.5. Escenario 2. Calidad vs. número de in-agents



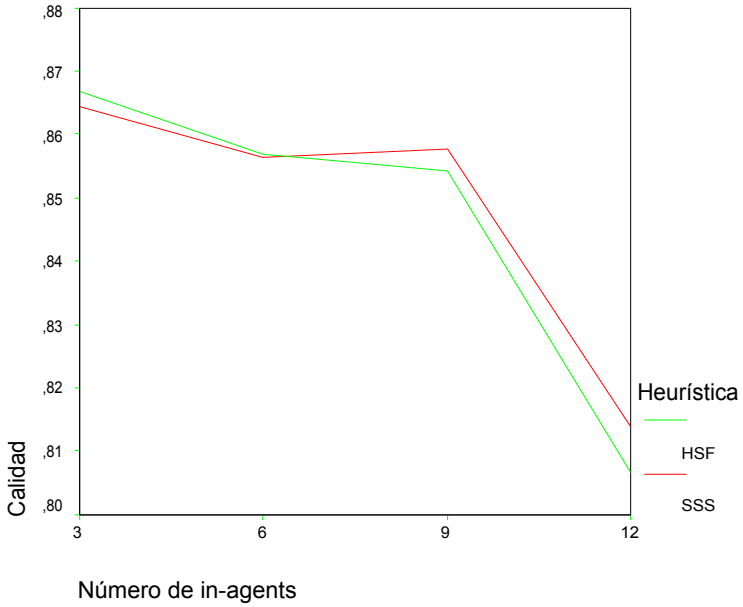
Gráfica 6.6. Escenario 3. Calidad vs. carga opcional



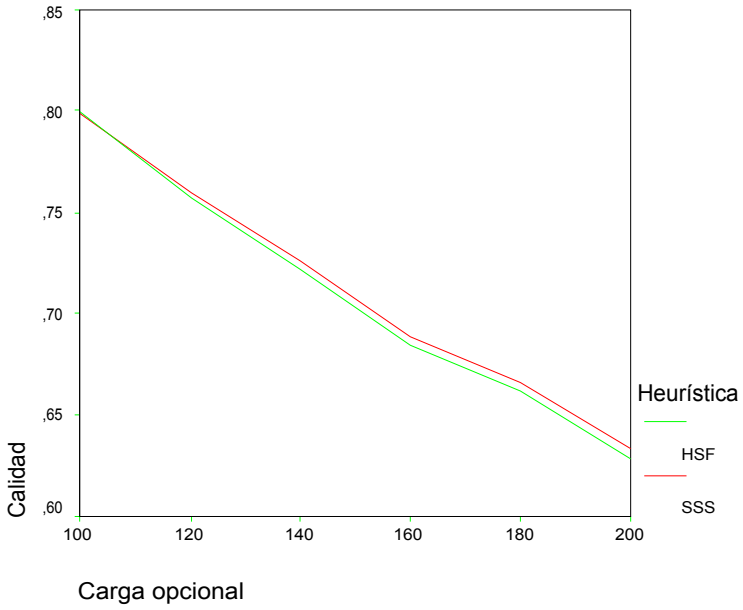
Gráfica 6.7. Escenario 3. Calidad vs. número de in-agents



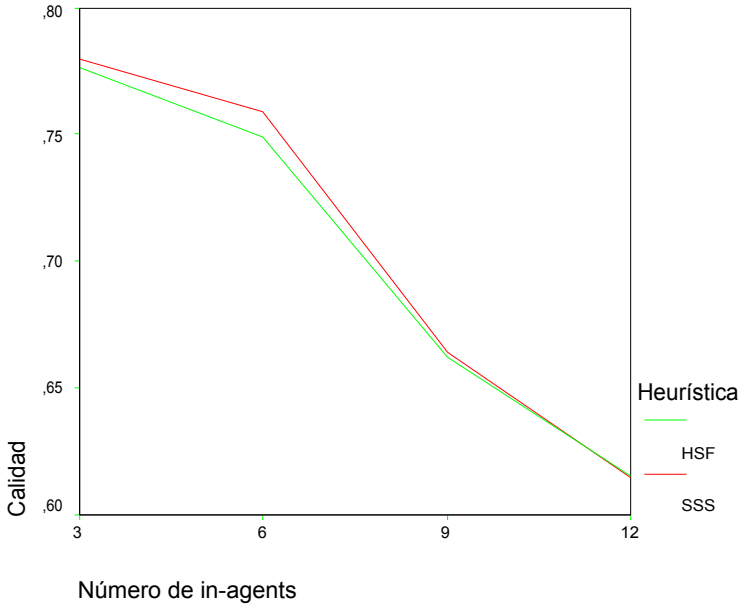
Gráfica 6.8. Escenario 4. Calidad vs. carga opcional



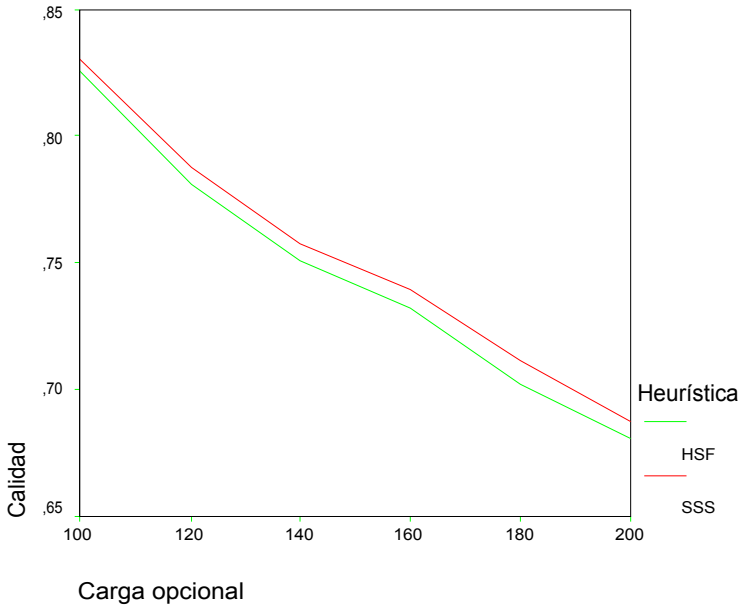
Gráfica 6.9. Escenario 4. Calidad vs. número de in-agents



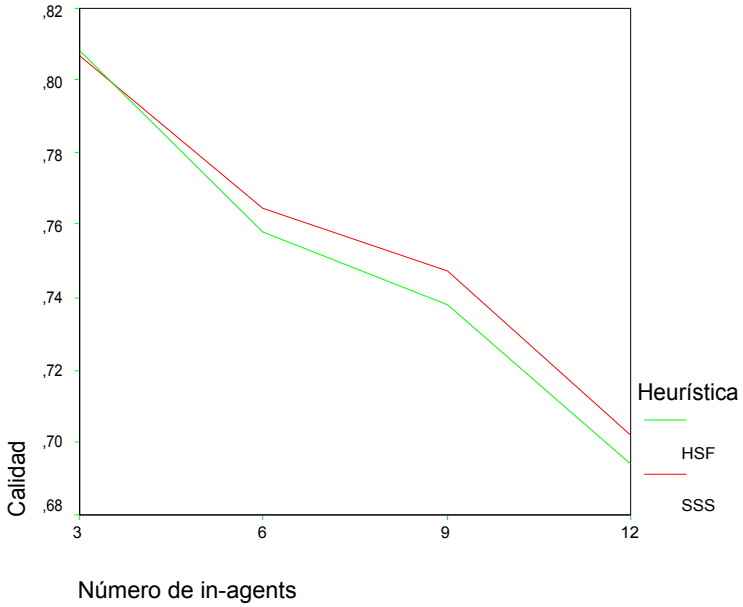
Gráfica 6.10. Escenario 5. Calidad vs. carga opcional



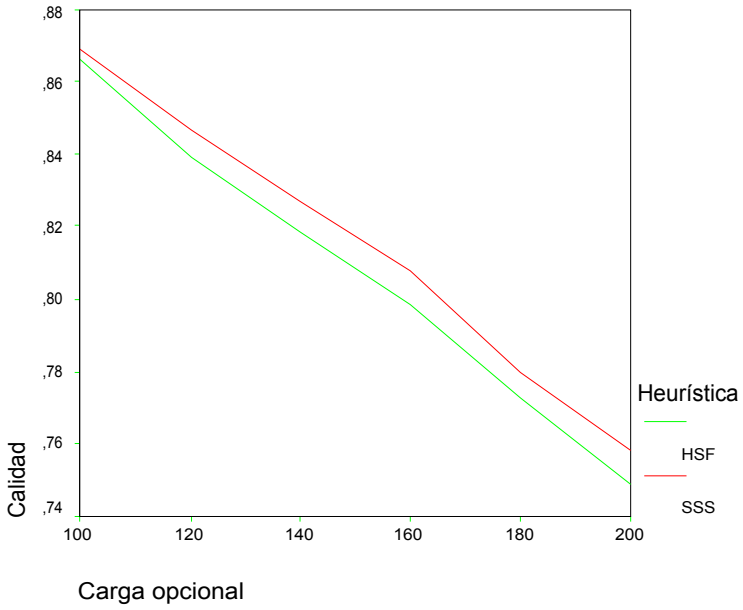
Gráfica 6.11. Escenario 5. Calidad vs. número de in-agents



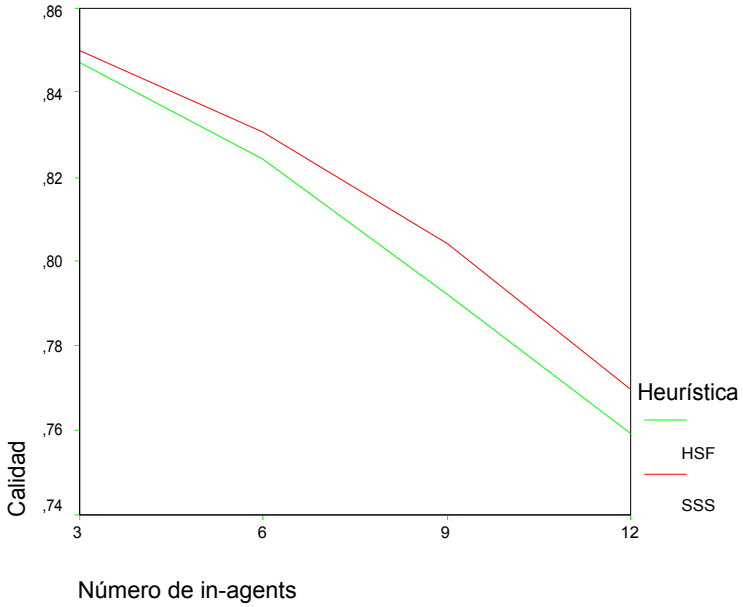
Gráfica 6.12. Escenario 6. Calidad vs. carga opcional



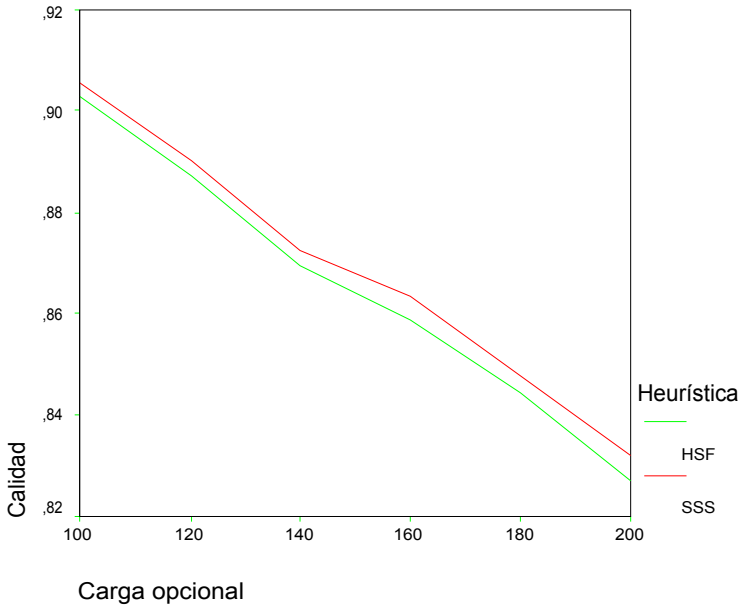
Gráfica 6.13. Escenario 6. Calidad vs. número de in-agents



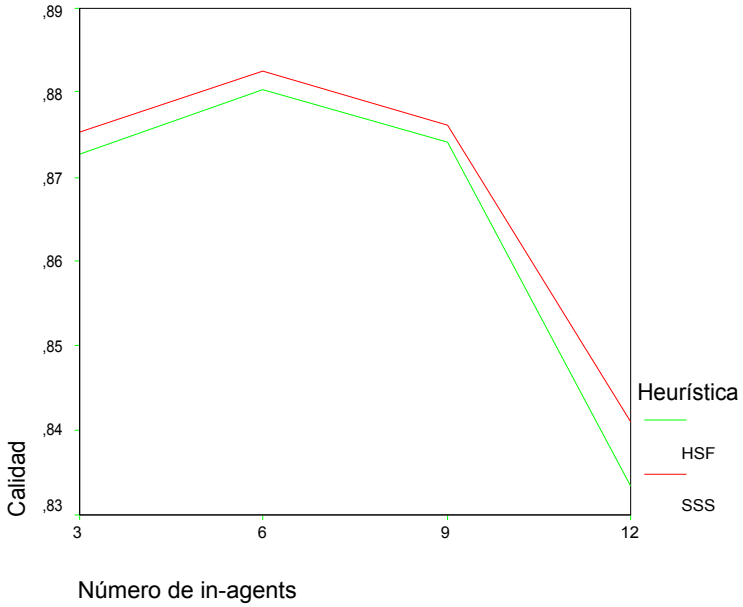
Gráfica 6.14. Escenario 7. Calidad vs. carga opcional



Gráfica 6.15. Escenario 7. Calidad vs. número de in-agents



Gráfica 6.16. Escenario 8. Calidad vs. carga opcional

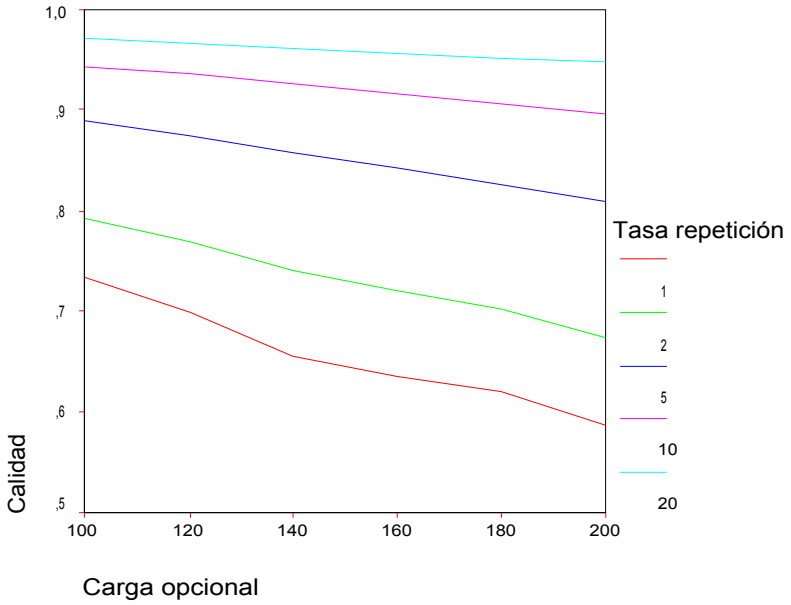


Gráfica 6.17. Escenario 8. Calidad vs. número de in-agents

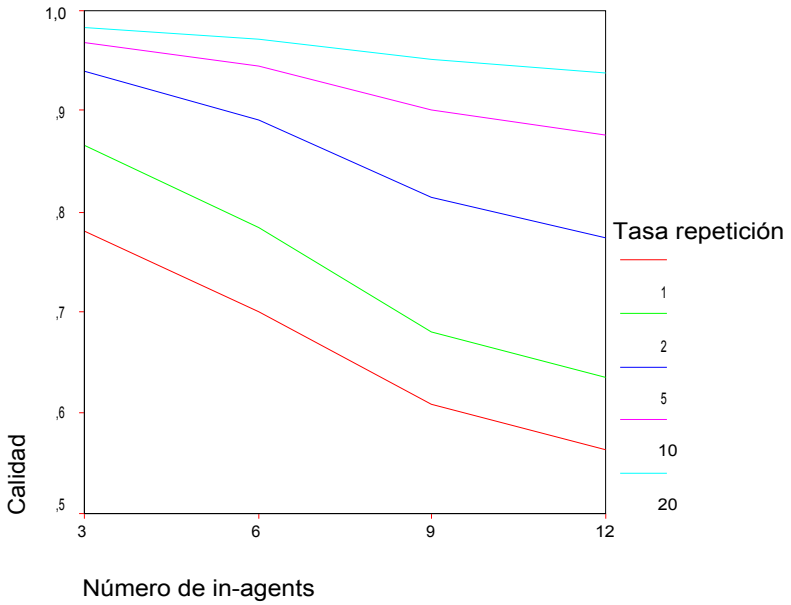
A la hora de evaluar estos resultados, la conclusión es la mejora que se obtiene al utilizar SSS frente al HSF en la mayoría de los casos. Mejora que aunque porcentualmente leve, es constante en la mayoría de las pruebas. Esta mejora aparece en todos los escenarios cuando el criterio de agrupación es la carga opcional. Únicamente se aprecia ciertos casos donde HSF aparece por encima. En concreto los escenarios 2, 3 y 4, con el criterio de número de in-agents y usando un número de in-agents bajo, pues en los mismos escenarios cuando aumenta el número de in-agents vuelve a aparecer por encima SSS. La posible interpretación a estos resultados puede verse en el hecho de que la heurística SSS ha sido diseñada para mejorar los resultados en problemas donde el slack está fragmentado, cosa que ocurre en nuestra arquitectura por la activación de diferentes in-agents con distintos periodos, la ejecución de sus partes críticas, etc. En estos escenarios con un número de in-agents bajo, esta fragmentación es mínima, con lo cual no se aprovechan las cualidades de la nueva heurística. Precisamente los escenarios 6 al 8, equivalentes a los anteriores excepto por el hecho de que utilizan deadlines menores que los periodos, no presentan esta ventaja del HSF. En estos casos, el utilizar deadlines menores hace que el slack aparezca más fragmentado, con lo que de nuevo se aprovechan las características del algoritmo.

6.3.3 Heurística SSSM

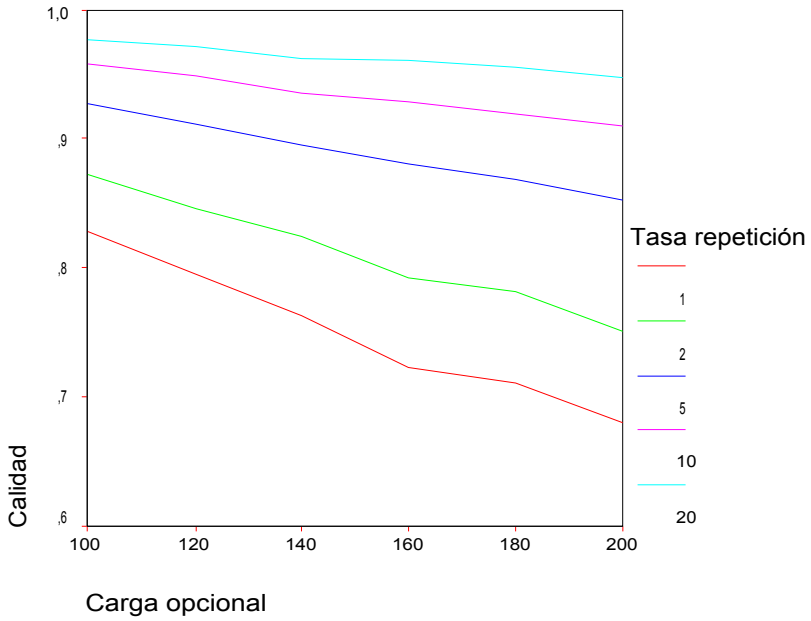
A la hora de evaluar la heurística SSSM, se van a utilizar de nuevo los 8 escenarios utilizados en el punto anterior. Ahora en cada escenario se va comparar los resultados de la heurística SSS con los de la SSSM. Como la heurística SSSM depende del régimen más o menos estable del sistema, cada prueba se repetirá en diferentes condiciones de estabilidad. Estas condiciones se definen como la tasa de modificación de los datos de entrada en relación con el periodo del in-agent con menor periodo. De esta manera si se define que el valor de dicha tasa es n, indicamos que cada n activaciones del in-agent con periodo menor, los datos de entrada cambian significativamente, estos cambios aunque expresados con relación al in-agent con menor periodo, afectarán a todos los in-agents. De esta manera se consideran tasas de 2, 5, 10 y 20. La tasa 1 realmente equivale a un régimen con modificaciones constantes con lo cual la heurística aplicada con esta tasa es la SSS y lo más importante para la comparación sería el resultado que se obtendría si usáramos SSS aunque el sistema fuera más estable, ya que el SSS no lo aprovecharía. Por tanto lo que realmente estaremos comparando es el resultado que proporciona SSS independientemente de la tasa de repetición (que en las gráficas aparece como tasa 1) y los resultados de SSSM con diferentes tasas.



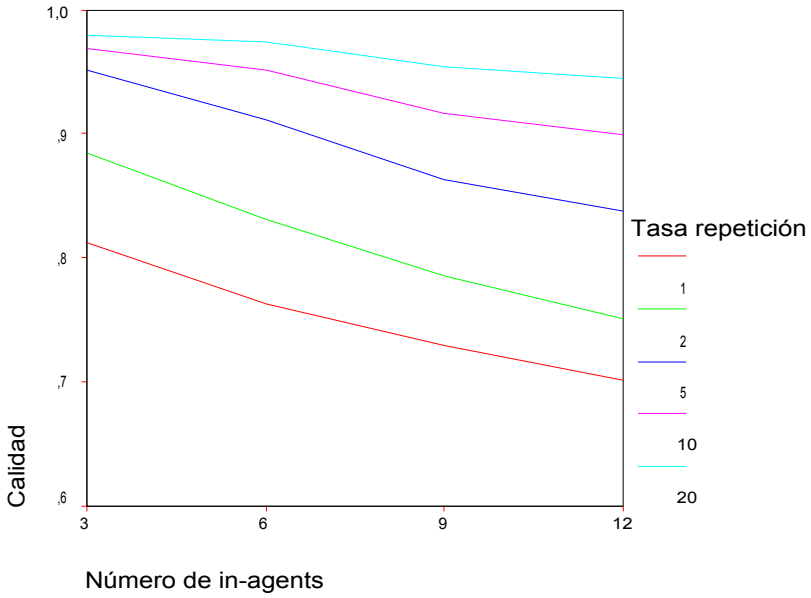
Gráfica 6.18. Escenario 1. Calidad vs. carga opcional



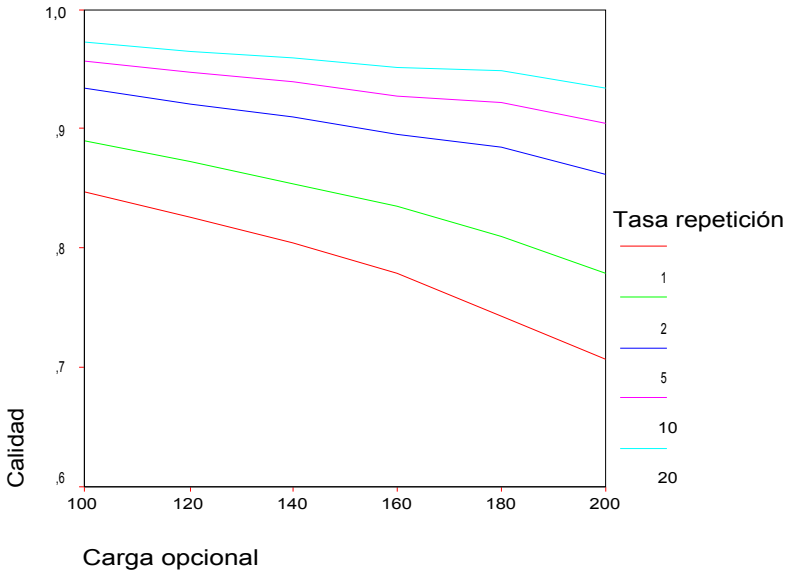
Gráfica 6.19. Escenario 1. Calidad vs. número de in-agents



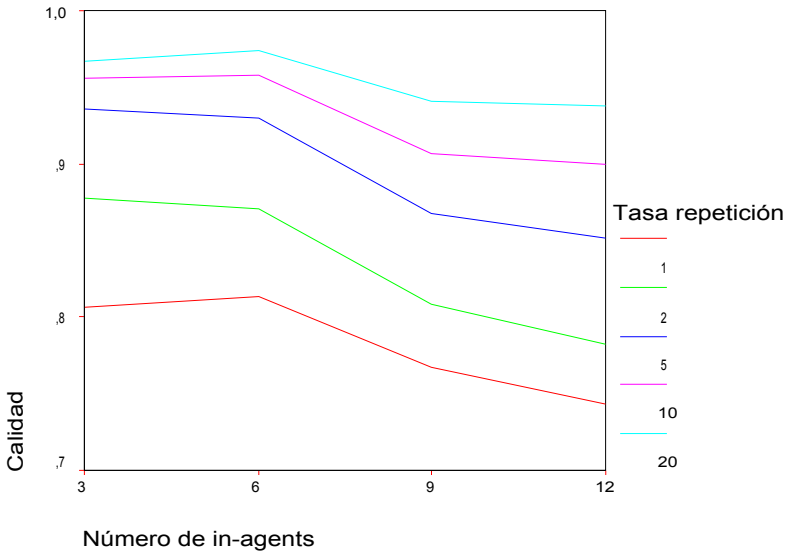
Gráfica 6.20. Escenario 2. Calidad vs. carga opcional



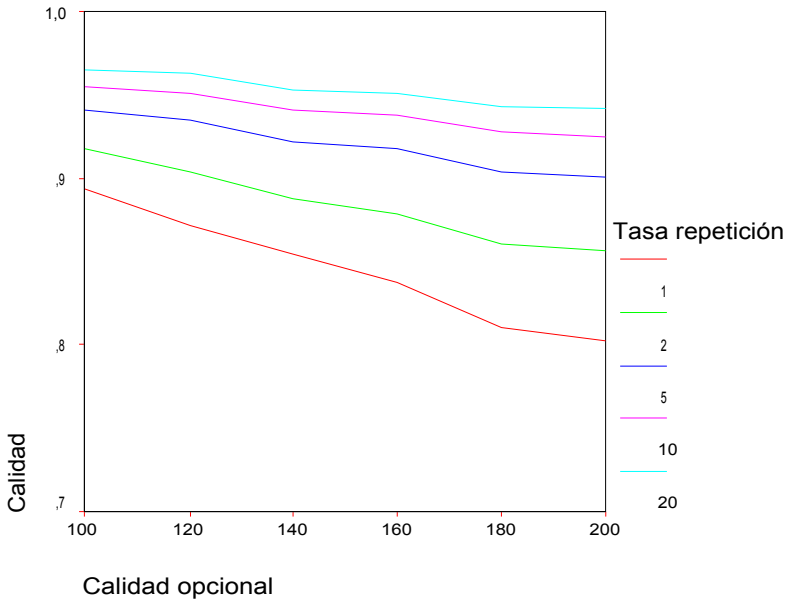
Gráfica 6.21. Escenario 2. Calidad vs. número de in-agents



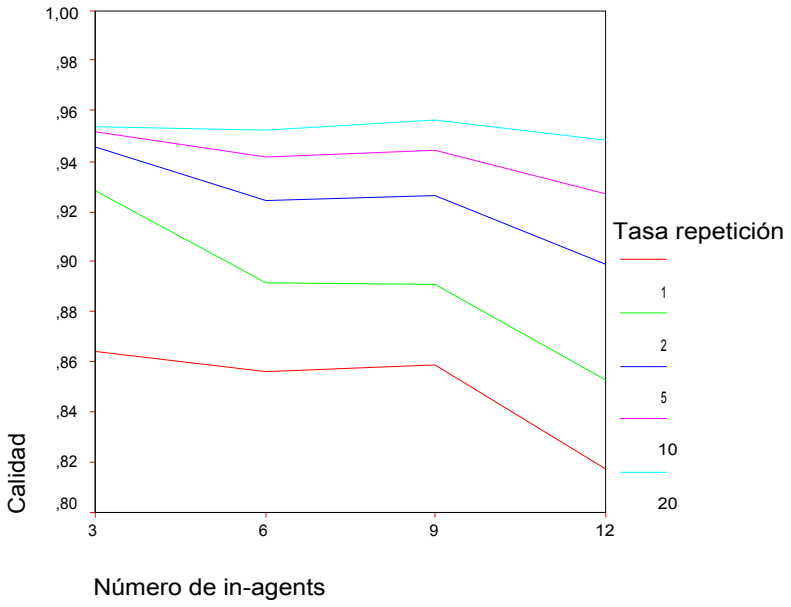
Gráfica 6.22. Escenario 3. Calidad vs. carga opcional



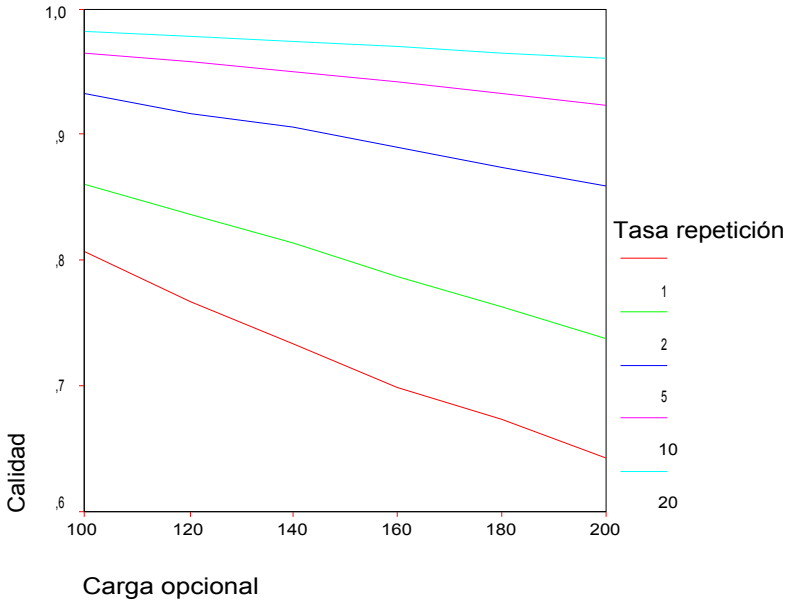
Gráfica 6.23. Escenario 3. Calidad vs. número de in-agents



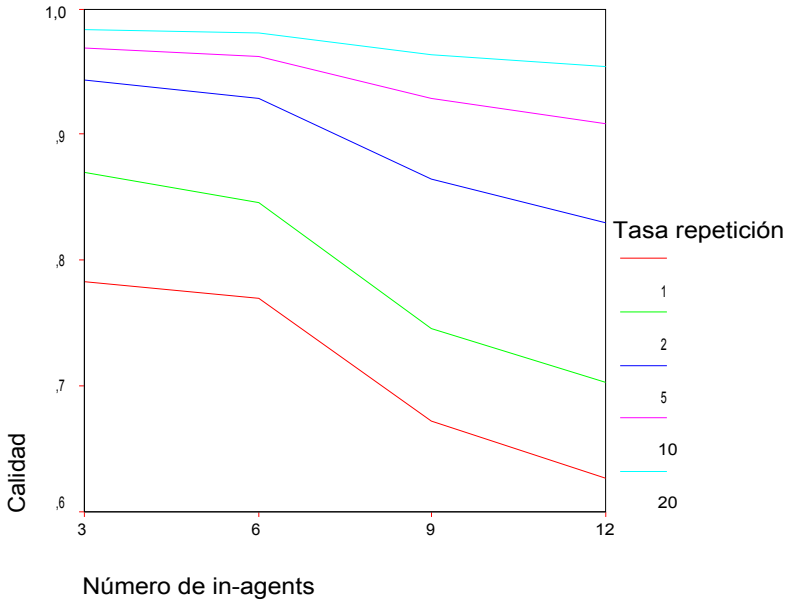
Gráfica 6.24. Escenario 4. Calidad vs. carga opcional



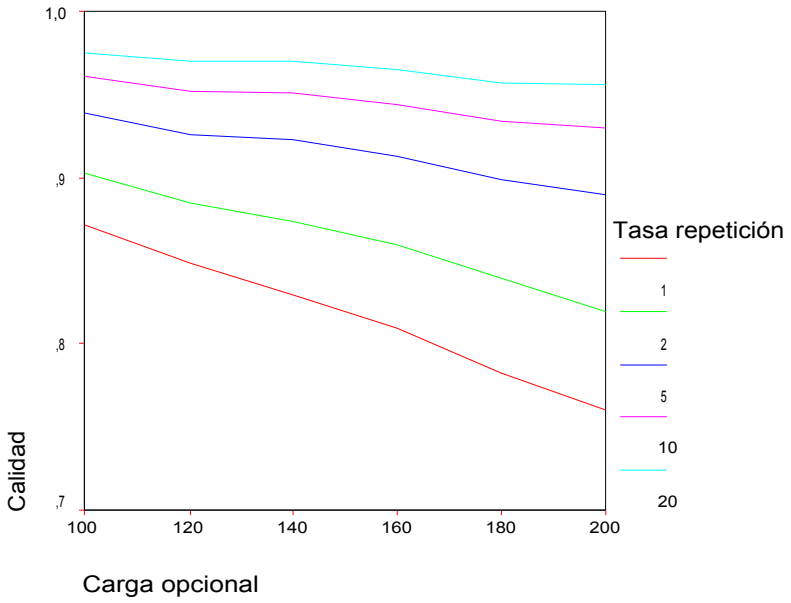
Gráfica 6.25. Escenario 4. Calidad vs. número de in-agents



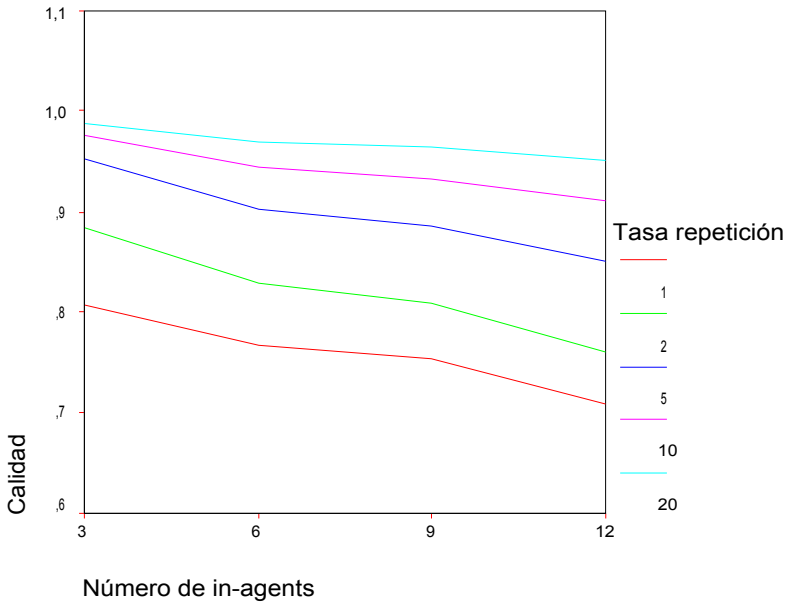
Gráfica 6.26. Escenario 5. Calidad vs. carga opcional



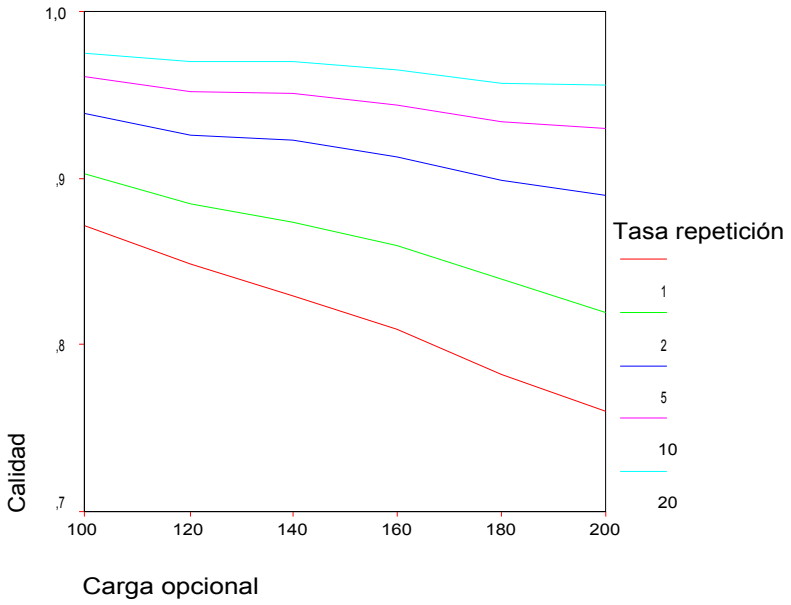
Gráfica 6.27. Escenario 5. Calidad vs. número de in-agents



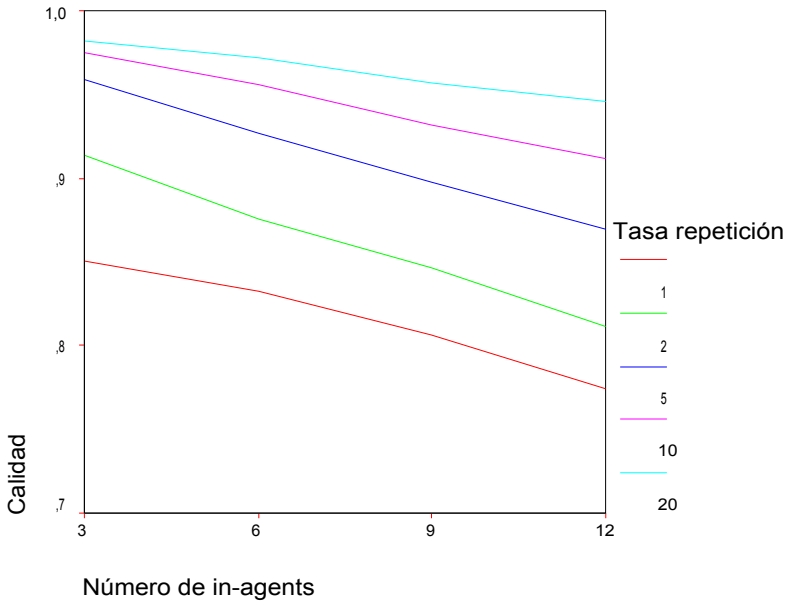
Gráfica 6.28. Escenario 6. Calidad vs. carga opcional



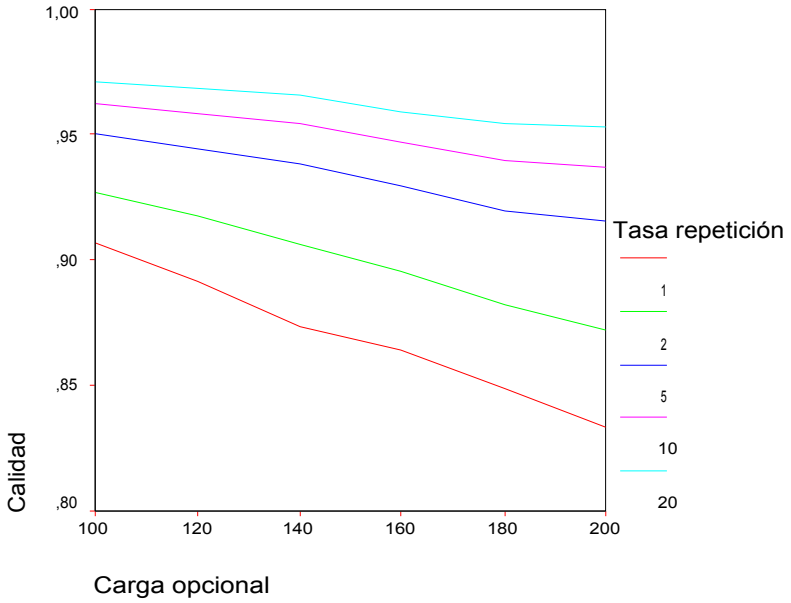
Gráfica 6.29. Escenario 6. Calidad vs. número de in-agents



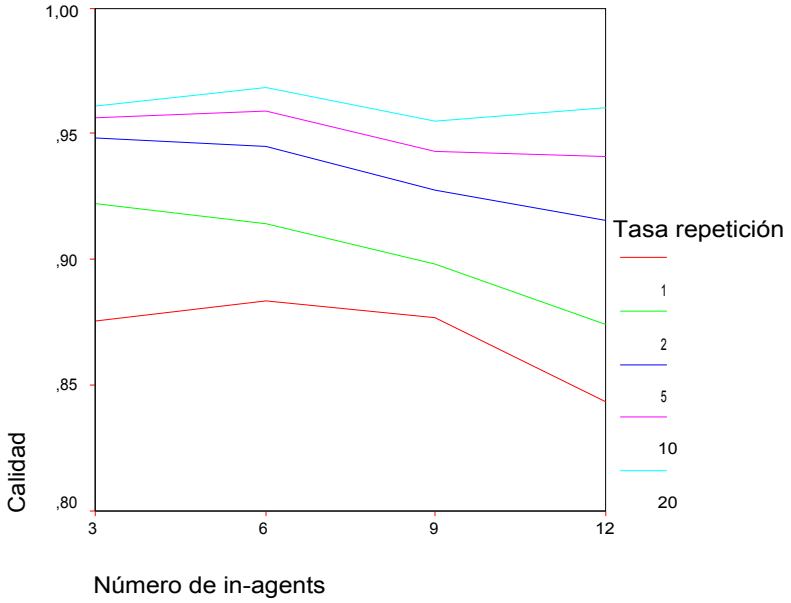
Gráfica 6.30. Escenario 7. Calidad vs. carga opcional



Gráfica 6.31. Escenario 7. Calidad vs. número de in-agents



Gráfica 6.32. Escenario 8. Calidad vs. carga opcional



Gráfica 6.33. Escenario 8. Calidad vs. número de in-agents

Si vemos las gráficas correspondientes a los 8 escenarios tanto en su vertiente de número de in-agents vs. calidad como en el caso de carga opcional podemos extraer las mismas conclusiones. Independientemente de diferencias en cuanto a la forma concreta de cada gráfica, debido a las diferentes características de cada una, se puede observar claras similitudes.

La primera y más importante la clara mejora que se obtiene usando el SSSM ya que como se puede observar en todos los casos queda por encima de los resultados obtenidos con tasa 1, que como comentamos realmente son los obtenidos con el SSS, pues para esta heurística es indiferente la tasa. Esta mejora además es independiente tanto de los criterios de agrupación (número de in-agents o carga opcional) como de las definitorias de los escenarios (relación entre periodos y entre deadline y periodo).

Además también se puede observar como la mejora va disminuyendo en magnitud conforme aumenta la tasa de repetición. En las pruebas la tasa va aumentando de 2 a 5 después a 10 y finalmente a 20. Si observamos las gráficas veremos que la calidad entre diferentes tasas también aumenta aunque no con la misma progresión. Esto es lógico, ya que la calidad máxima esta limitada, nunca podremos obtener más calidad que la proporcionada en unas condiciones como las indicadas para el cálculo de CSC, es decir donde para cada activación de un in-agent se ejecutan todos sus niveles. Y esto (obtener el 100% de la calidad) difícilmente ocurrirá incluso en situaciones (por otro lado extrañas) de nula variabilidad de entrada, porque incluso en estos casos necesitará el agente de algún tiempo para poder llegar a ejecutar los niveles máximos. Siempre habrá alguna situación donde el óptimo necesitaría la ejecución de un nivel que no podría llevarse a cabo en la realidad, con la consecuente perdida de calidad (aunque mínima) frente al óptimo.

En cualquier caso el hecho de que en los primeros aumentos en la tasa de repetición se consigan los mejores aumentos en cuanto a la calidad, es claramente beneficioso, ya que tampoco debe esperarse circunstancias con situaciones de estabilidad permanentes en exceso y con muy pocos cambios.

7 CONCLUSIONES

En este capítulo final recordaremos cuales eran los objetivos del presente trabajo, el desarrollo realizado y los resultados alcanzados. Finalmente también comentaremos trabajo futuro a desarrollar.

7.1 OBJETIVOS

El objetivo principal del presente trabajo era desarrollar métodos basados en la utilidad para ser utilizados en la arquitectura para la construcción de agentes inteligentes de tiempo real ARTIS. Estos métodos constituirían la política de planificación de segundo nivel para mejorar la calidad de los resultados, coexistiendo con un planificador de primer nivel que garantiza cumplir las restricciones de tiempo real.

Los objetivos particulares que debía conseguir dicha heurística eran:

- Trabajar con un método de resolución de problemas, los algoritmos de refinamiento progresivo, que aunque basados en la idea de mejora progresiva de la solución como los algoritmos anytime, requieren un tratamiento diferente.

- Manejo simultaneo de dos tipos de métodos de resolución de problemas, los algoritmos de refinamiento progresivo comentados en el punto anterior y los métodos múltiples. Al ser ambos tipos de métodos diferentes (los primeros van mejorando la solución previa utilizándola como punto de partida, mientras que los segundos presentan métodos de resolución alternativos) se requiere que la heurística sea capaz de tratar con ambos a la vez de manera que en un mismo plan puedan elegirse para ciertos subproblemas algoritmos del primer tipo y para otros subproblemas del segundo.

- Tratamiento de intervalos temporales, el tiempo de CPU que el planificador tiene que repartir entre los diferentes métodos, que no son continuos, sino fragmentados en diferentes subintervalos. La heurística debe tratar con varios subintervalos de forma conjunta para un mejor aprovechamiento de este tiempo.

- Reutilización de resultados previos cuando el sistema alcanza regímenes de funcionamiento estables. Cuando el sistema alcanza un régimen estable, las entradas al agente son similares, con diferencias no significativas. Esto hace que las respuestas calculadas no variarían. Una primera aproximación haría que no se calcularan de nuevo esas respuestas sino simplemente utilizar la calculada en el momento previo pues va ser igual a la calculada. La nueva heurística no solo debe proporcionar esta característica, sino que además debe utilizar los resultados previos como punto de partida para calcular soluciones mejores, planificando sólo métodos que proporcionan mejores calidades que las obtenidas en los periodos anteriores. Para ello la heurística deberá manejar el conocimiento sobre que resultados se obtuvieron con anterioridad y que métodos se utilizaron para ello.

- Utilización de los resultados tan pronto como estos sean obtenidos y no se prevea posible la mejora de los mismos. De esta manera evitamos retrasar la parte de actuación o comunicación de los resultados durante intervalos de tiempo que no van a aportar nada a las soluciones.

7.2 DESARROLLO

El desarrollo de la presente memoria ha presentado aspectos relativos al estado del arte, descripción del entorno donde iba a aplicarse la heurística desarrollada, la presentación de la propia heurística y resultados. Con más detalle el desarrollo sería:

- Descripción de arquitecturas existentes para trabajar en sistema de tiempo real más o menos estricto.

- Descripción de la arquitectura ARTIS, pensada para el desarrollo de agentes inteligentes en entornos de tiempo real estricto. Es en esta arquitectura donde van a aplicarse los métodos a desarrollar, por lo que se hace un especial hincapié en la descripción de los componentes del sistema, en concreto del control y de la estructuración de los métodos que un agente dispone para resolver un problema, que

constituyen al final los elementos que la heurística tendrá que planificar.

- Descripción de técnicas de planificación ya existentes. Por un lado las denominadas voraces, que utilizan poca información, y por otro lado las deliberativas, que utilizan más información del sistema y producen mejores resultados. Además se indican las razones por la que estas técnicas deliberativas no son aplicables a ARTIS (tipo de algoritmos que manejan, etc.) y la necesidad del desarrollo de una nueva heurística.

- Presentación de la nueva heurística SSS (Slack Slide Scheduling) que incorpora en sus características los elementos para alcanzar la mayoría de los objetivos planteados.

- Presentación de la heurística SSSM (SSS con Memoria), heurística que amplía la anterior, y que incorpora la característica de usar resultados previos para ser mejorados.

- Desarrollo de un conjunto de pruebas para ser usadas en un simulador con el fin de verificar el comportamiento de las heurísticas realizadas. Ejecución de las pruebas, recogida de resultados y análisis de estos.

- Desarrollo de dos pruebas empíricas en escenarios prácticos para comprobar la viabilidad del sistema (Anexos A y B).

7.3 RESULTADOS

El presente trabajo ha presentado una nueva heurística denominada SSS (Slack Slide Scheduling). Esta heurística cumple con la mayoría de los objetivos deseados que han sido comentados en el punto 1 de este capitulo.

Realizando un mejor aprovechamiento de las características del sistema (tipo de los algoritmos a planificar, espacio a planificar dividido en diferentes intervalos, etc.), SSS consigue mejores resultados (calidad total de los resultados obtenidos) que la heurística voraz que mejores resultados obtenía, la HSF. Se usa esta heurística como referencia ya que las heurísticas deliberativas que suelen proporcionar mejores resultados que las voraces no son aplicables a nuestra arquitectura por diversas razones, especialmente por el tipo de algoritmos que usan. Estos resultados han sido verificados como se ha comentado en el punto 2 utilizando un simulador

que permite realizar un gran número de pruebas con diferentes parámetros que definen cada prueba.

En segundo lugar se ha presentado otra heurística, la SSSM (SSS con Memoria), heurística que representa una extensión de la anterior y que amplía sus características añadiéndole la capacidad de cumplir el objetivo citado en el punto 1 y que no era cubierto por SSS, es decir la reutilización de resultados previos. Esta heurística, haciendo uso de las capacidades de la arquitectura ARTIS para indicar cuando ha habido cambios significativos en los datos de entrada (es decir en el comportamiento del problema), permite utilizar resultados previos para ser mejorados. Esto permite aumentar el rendimiento del sistema cuando se alcanzan regimenes estables de funcionamiento al permitir usar el tiempo del sistema en mejorar resultados previos en vez de calcularlos de nuevo o simplemente volver a utilizarlos sin mejorarlos.

7.4 TRABAJO FUTURO

Diversas líneas de investigación pueden citarse como trabajo futuro, directa o indirectamente relacionadas con el trabajo presentado:

Mejora de las técnicas de aprendizaje de los in-agents. Cada in-agent incorpora técnicas de aprendizaje para conocer con más precisión características propias, como los tiempos de ejecución y la calidad de los resultados de los métodos que utiliza para resolver los problemas. Como estos métodos son los que la heurística SSS tiene que planificar y precisamente usa dicha información para realizar la planificación, disponer de información más precisa mejoraría la planificación. Se trataría de aplicar técnicas de aprendizaje que hicieran más precisa dicha información sobre los propios in-agents, como las usadas para refinar procesos de búsqueda en tiempo real [Shi03]

Otras heurísticas de selección. Otra línea de investigación sería el estudio de otras heurísticas que puedan mejorar las calidades obtenidas, si bien no en todas las situaciones, si bajo determinadas situaciones, con lo que sería necesario además establecer que condiciones serían las indicadas para utilizar cada heurística. En este sentido actualmente se están siguiendo diferentes aproximaciones, destacando el desarrollo de heurísticas basadas en métodos de negociación.

Incorporar habilidades de meta-control a ARTIS. Otra vía importante es proveer a la arquitectura ARTIS de mecanismos para incorporar un

sistema de metacontrol. Este sistema incluiría un lenguaje de metacontrol que permitiría guiar el comportamiento del componente de control ARTIS en tiempo de ejecución. De esta manera el sistema podría variar el funcionamiento del control para que prestará más importancia a unos eventos que a otros, guiara el comportamiento del in-agent, cambiar el tiempo dedicado a ejecutar partes opcionales o en relación al presente trabajo y a la línea de investigación comentada anteriormente de desarrollar otras heurísticas, que el control fuera capaz de decidir que heurística de selección aplicar en cada momento en función de las circunstancias del momento.

7.5 PUBLICACIONES RELACIONADAS

A lo largo de este proyecto y como consecuencia de los distintos trabajos desarrollados relacionados con él, se han realizados una serie de publicaciones que se detallan a continuación en orden cronológico inverso:

- L. Hernández, V. Botti, A. García-Fornes, M. González. A Quality-based Heuristic for Real-Time Scheduling. Artificial Intelligence Research and Development. Frontiers in Artificial Intelligence Research and Applications, Volume 100, IOS Press, Noviembre 2003. Congreso Catalán de Inteligencia Artificial CCIA'03.

- L. Hernández, E. Vivancos, V. Botti. Intelligent Scheduling of Production Systems in a Real-Time Architecture. Actas del VI Congreso Iberoamericano de Inteligencia Artificial, IBERAMIA'98, Lisboa, Portugal, octubre 1998.

- V. Botti, L. Hernández. Control in Real-Time Multiagent Systems. Proceedings of the II Iberoamerican Workshop on D.A.I. and M.A.S. Toledo, octubre 1998.

- E. Vivancos, L. Hernández, V, Botti. Inteligencia artificial distribuida en entornos de tiempo real. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, nº 6, 1998. ISSN 1137-3601

- L. Hernández, E. Vivancos. Aplicación de técnicas de planificación de agentes guiadas por la utilidad en entornos multiagente de tiempo real. Actas de la VII Conferencia de la asociación española para la inteligencia artificial, CAEPIA'97, Torremolinos, Málaga, Noviembre 1997.

- E. Vivancos, L. Hernández, V, Botti. Técnicas y arquitecturas de inteligencia artificial en tiempo real. Informática y Automática, Vol. 30, nº4, 1997. ISSN 0214-932X

- L. Hernández, E. Vivancos, V. Botti. Una aproximación a la inteligencia artificial distribuida. Novática, Vol. 127, 1997. ISBN 0211-2124

Anexo A. Ejemplo de aplicación: Agente ARTIS para el control de una planta de aguas residuales

El propósito del presente anexo es el de mostrar una aplicación de la arquitectura ARTIS donde el módulo de control hace uso de las heurísticas desarrolladas. No se pretende hacer estudios comparativos con las otras heurísticas pues por las características de la aplicación sería difícil reproducir exactamente las condiciones del problema en diferentes pruebas, condición necesaria para poder comparar de forma rigurosa. Además existe el hecho de que los resultados no serían significativos para realizar comparaciones pues serían necesarias gran número de pruebas, y es por ello por lo que se ha utilizado el simulador para las comparaciones.

El objetivo de estas pruebas es simplemente mostrar como un agente construido bajo la arquitectura ARTIS puede trabajar en un problema real mientras hace uso de los algoritmos desarrollados.

El problema que se plantea para ser controlado y que luego describiremos con mayor precisión, es el de una serie de depósitos de agua residual interconectados, con diferentes entradas y salidas de flujos de agua y cuyos volúmenes de líquido almacenado se pretende que estén dentro de los límites que un usuario marque en cada momento y sin desbordamientos.

Este proceso está simulado mediante el programa LabView. Labview™ 7 es un programa generador de instrumentación virtual desarrollado por Nacional Instruments que permite la simulación de procesos físicos y el control del proceso en tiempo real. Aunque se trata de un proceso simulado

y no de un proceso físico real, no debe verse como una prueba similar a las realizadas para hacer las comparaciones. Aquí se está simulando un sistema físico real y sobre todo se trata de algo totalmente externo al agente ARTIS, de hecho se ejecuta en un ordenador diferente al que ejecuta ARTIS. Es decir, para el agente el proceso que está controlando es un proceso real al que está conectado, en este caso vía puerto serie, del que recibe lecturas de sensores y sobre el que puede actuar abriendo y cerrando grifos y válvulas, tal y como comentaremos.

A.1 DESCRIPCIÓN PROBLEMA

El sistema que se pretende controlar está formado por tres depósitos de agua (figura A.1). Dos de ellos, los depósitos A y B de capacidad 5000 litros y el tercero, C, de capacidad 10000. Cada depósito cuenta con cinco sensores que controlan si la cantidad de agua en su depósito respectivo ha superado un determinado porcentaje. En concreto controlan si la cantidad almacenada ha superado el 20%, el 40%, el 60%, el 80% o el 100% del volumen total.

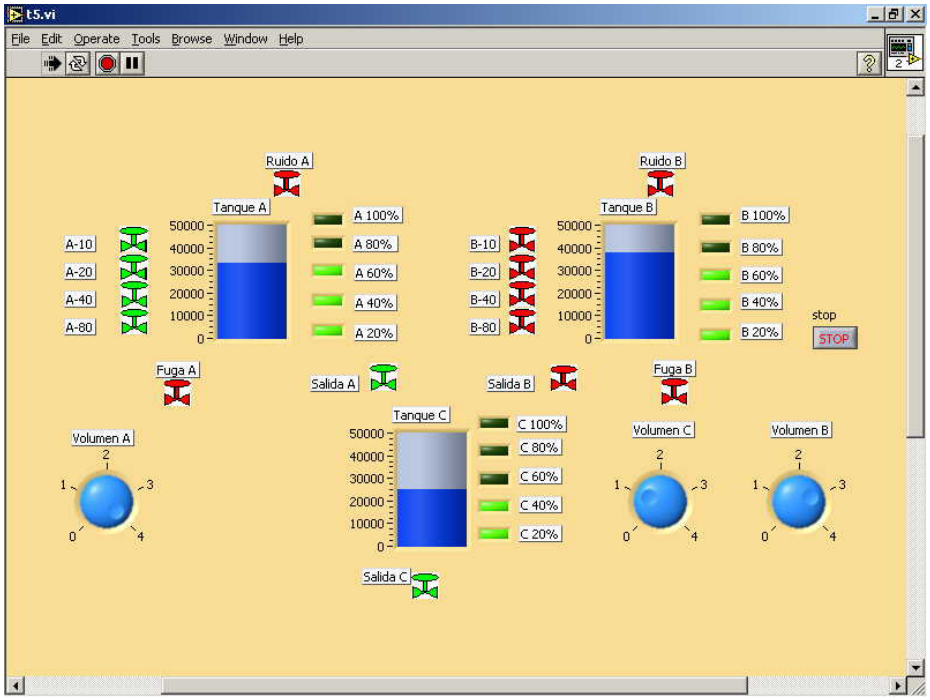


Figura A.1. Componentes del proceso a controlar

Los depósitos A y B poseen cada uno de ellos 4 grifos de entrada que pueden ser controlados (cada uno de ellos sólo permite ser abierto o cerrado, no controlar su caudal). Cada grifo tiene un caudal diferente, siendo estos de 10, 20, 40 y 80 litros por segundo. Cada uno de estos depósitos dispone de una salida con una válvula que permite su vaciado vertiendo el agua sobre el depósito C, de manera que éste dispone de estas dos válvulas como entradas controladas, cada una con caudal de salida de 95 litros segundo. Finalmente C tiene una única salida para ser vaciado (de caudal 180 litros segundo), siendo controlada también por una válvula.

Para introducir perturbaciones en el sistema se han incluido una serie de entradas y salidas adicionales que no serán controladas automáticamente, sino que servirán para modificar de forma manual la cantidad de líquido que entra a un depósito o la que sale. En concreto cada depósito A y B dispone de un grifo de entrada (de caudal 20 litros segundo) denominados ruido y un grifo de salida (de caudal 10 litros segundo) denominados fuga, no vertiendo estas últimas al C como si ocurría con los controlados automáticamente.

Además posee unos controles que sirven para fijar el intervalo deseado de volumen almacenado. Estos controles están representados por mandos giratorios para cada depósito con el significado de la tabla 1.

POSICIÓN	VALOR MÍNIMO	VALOR MÁXIMO
0	0%	20%
1	20%	40%
2	40%	60%
3	60%	80%
4	80%	100%

Tabla A.1. Controladores de rango

A.2 MODELO DE PROCESO

El sistema desarrollado se ejecutará bajo dos ordenadores (Figura A.2). El primero de ellos ejecutará el proceso de los depósitos con la aplicación LabView, es decir representará el sistema real a controlar.

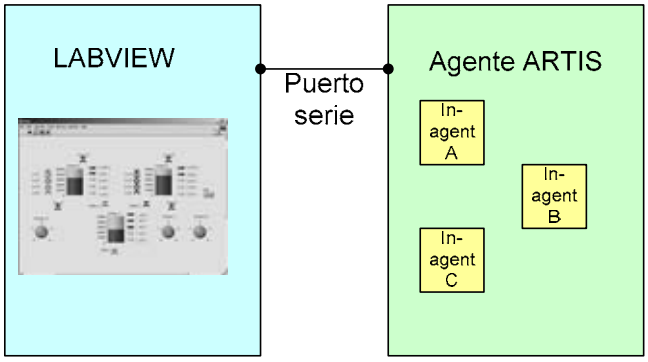


Figura A.2. Modelo del proceso

Este ordenador estará conectado vía serie con el ordenador que ejecuta el agente ARTIS. El agente ARTIS estará formado por tres in-agents, cada uno de ellos encargado de uno de los depósitos. Estos in-agents son descritos en el punto siguiente.

A.3 AGENTE ARTIS

El agente ARTIS diseñado para controlar todo el sistema está formado por tres in-agents cada uno de ellos está encargado de uno de los depósitos. Por dicha razón los in-agents encargados de los depósitos A y B serán idénticos.

Cada in-agent está formado por tres MKSs:

El primero es el de sensorización y se encarga de leer los sensores que indican el nivel de llenado de su depósito respectivo. También se encarga de leer los pulsadores de elección de nivel deseado, tanto los que se encuentran en el panel de control, como los de emergencia en el propio simulador. Este MKS dispone de un solo nivel, en este caso nivel 0 al ser un MKS crítico.

El segundo MKS calcula las acciones a realizar, que en este caso será la apertura o cierre de los grifos correspondientes. Es un MKS crítico que dispone de tres niveles, es decir además del nivel 0 crítico posee 2 opcionales. En el ejemplo se han añadido dos niveles opcionales extras a cada in-agent. Estos niveles consumen tiempo aunque no realizan ninguna operación significativa. Su inclusión se debe únicamente a motivos de verificación de como trabajan las heurísticas.

En el caso del nivel 0 el algoritmo calcula una respuesta rápida aunque de baja calidad. Para ello lo que hace es simplemente mandar abrir todos los grifos de entrada y cerrar la válvula de salida del depósito correspondiente en el caso de que el volumen almacenado esté por debajo de lo solicitado. Si está por encima realiza lo contrario, manda cerrar todos los grifos de entrada y abrir la válvula de salida.

```
Si Rango_Solicitado(A) > Rango_Actual(A)
        Entonces AbreGrifos(A10, A20, A40, A80)
        Sino si Rango_Solicitado < Rango_Actual
                Entonces CierraGrifos(A10, A20, A40, A80)
```

Figura A.3. Nivel 0 cognitivo para depósito A

El nivel 1 (figura A.4) realiza un cálculo más preciso, ya que se fija en la diferencia entre lo solicitado y lo existente. De esta manera no abre o cierra todos los grifos. Por ejemplo, si esta por debajo del rango solicitado pero no demasiado permitirá el paso de liquido por los grifos de poco caudal. Por el contrario si falta mucho para llenarse abrirá los grifos de mayor caudal.

```
Diferencia = Rango_Solicitado(A) – Rango_Actual(A)
    si diferencia >0 calculo_abre_grifos(A, diferencia)
        sino si diferencia < 0 calculo_cierra_grifos(A, diferencia)
```

Figura A.4. Nivel 1 cognitivo para depósito A

Las funciones calculo_abre_grifos y calculo_cierra_grifos serían las que calcularían que grifos en concreto se deben abrir o cerrar en función de la diferencia entre el nivel deseado y el real, teniendo en cuenta que hay cinco niveles posibles para cada depósito.

```
abre_grifos(deposito, diferencia)

    si diferencia > 4 AbreGrifos(deposito10, deposito20, deposito40, deposito80)
    sino si diferencia >3 AbreGrifos(deposito20, deposito40, deposito80))
    sino si diferencia >2 AbreGrifos(deposito40, deposito80)
    sino AbreGrifos(deposito80)
```

Figura A.5. Función Abre_grifos

El nivel 2 es el que realiza un mejor calculo, para ello utiliza la respuesta calculada en el nivel anterior (se trata pues de un método de refinamiento progresivo) y puede modificarla teniendo en cuenta la situación de los otros depósitos. Hay que tener en cuenta por ejemplo, que la válvula de salida del depósito A es el de entrada a C. Cerrar la salida de A supone no permitir la entrada a C. Este nivel intenta alcanzar, si es posible, un compromiso entre las acciones de los tres depósitos. Por ejemplo si el depósito A requería llenarse la acción a realizar sería entre otras cerrar la válvula de salida, pero si se quiere también que C se llene entonces se requeriría abrirlo. Mientras tanto se desea que el depósito B permanezca en el intervalo actual, una solución sería abrir las entradas de B y su salida de manera que permanezca su volumen constante y permita que sea el único que abastece a C.

El tercer MKS es el MKS de acción. Este tipo de MKS siempre dispone de un solo nivel que es crítico. Su labor es llevar a cabo las acciones calculadas por el MKS anterior, es decir, abrir o cerrar grifos y válvulas. Para ello mandará las órdenes oportunas al simulador.

A.4 EJEMPLO DE EJECUCIÓN

Para ilustrar el funcionamiento de la aplicación se muestra un ejemplo de ejecución.

En primer lugar se presentan los datos significativos de los diferentes in-agents y niveles.

In-agent	Periodo	Deadline
A	2500 mseg	2500 mseg
B	2700 mseg	2700 mseg
C	2900 mseg	2900 mseg

Tabla A.2. Datos de los In-agents

A continuación los datos para los diferentes niveles, teniendo en cuenta que para los niveles críticos (los número 0) el tiempo hace referencia al tiempo de ejecución en el caso peor, y en los opcionales (1 al 4) al tiempo medio de ejecución. En cuanto a la calidad en los niveles opcionales, al tratarse

de MKS por refinamiento sucesivo, se trata de incrementos sobre la calidad aportada por el nivel anterior. Por ejemplo, si el nivel 1 cognitivo del In-Agent A proporciona 0.4 de calidad, si se ejecuta completamente el nivel 2 obtendremos 0.6 de calidad (0.4 acumulada más el 0.2 propio).

In-Agent	**MKS**	**Importancia**	**Nivel**	**Tiempo** (mseg)	**Calidad**
A	Percepción	0	0	170	0
	Cognición	3	0	30	0.05
			1	205	0.4
			2	240	0.2
			3	265	0.15
			4	285	0.1
	Acción	0	0	200	0
B	Percepción	0	0	170	0
	Cognición	3	0	30	0.05
			1	170	0.4
			2	285	0.2
			3	320	0.15
			4	340	0.1
	Acción	0	0	200	0
C	Percepción	0	0	170	0
	Cognición	2	0	30	0.05
			1	170	0.4
			2	205	0.2
			3	340	0.15
			4	385	0.15
	Acción	0	0	200	0

Tabla A.3. Datos de MKS y niveles

En este ejemplo (figura A.6) se presenta el comportamiento del depósito A durante 6 minutos, donde el eje X representa milisegundos y el Y el volumen en litros que mantiene el depósito en cada momento.

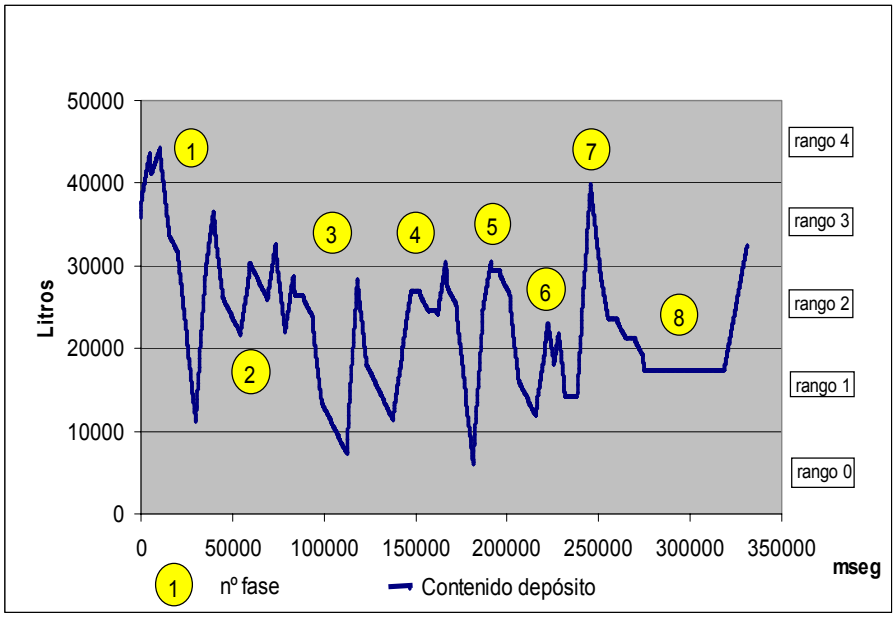
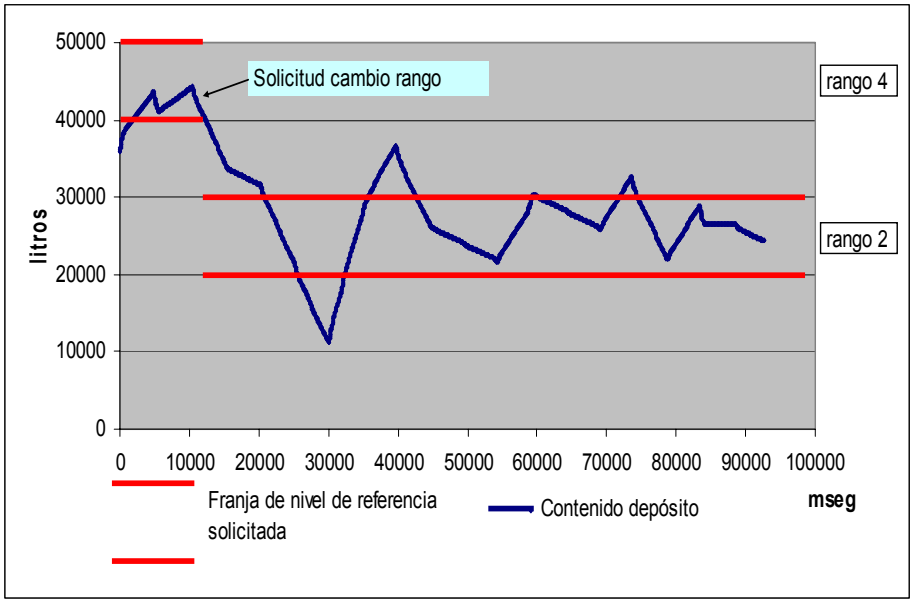


Figura A.6. Evolución del contenido del depósito A

En la figura pueden verse diferentes fases (① a ⑧) por las que pasa el depósito A como consecuencia de cambios en el rango de referencia solicitado (niveles del 0 al 4 que corresponden a los indicados en la tabla A.1) o entradas extraordinarias de líquido. Estas fases son comentadas a continuación.

En un primer momento, en la fase ①, el sistema permanece estable dentro de la franja de referencia 4 (4000 - 5000 litros) hasta que desde el panel del simulador (mediante el mando giratorio correspondiente) se solicita que se establezca una nueva referencia para el depósito A, en concreto la petición establece que el depósito se sitúe dentro de la franja de referencia 2 (2000 - 3000 litros), tal y como se puede ver en la ampliación de las zonas (① y ②) presentada en la figura A.7.



A.7 Reacción del sistema ante cambio de referencia

En la fase ②, el sistema se estabiliza en dicha franja, fluctuando debido a su propio comportamiento y a las interferencias que le suponen los otros depósitos, especialmente el C. Hay que tener en cuenta que el liquido que el depósito A elimina para poder establecerse en la nueva franja de referencia, inferior a la previa, pasa al deposito C, que en un primer momento cerrará su entrada (es decir el grifo que también sirve de salida del depósito A). Por ello el depósito A que había conseguido bajar su nivel simplemente abriendo su salida y moderando su entrada, sufre un repunte de su nivel hasta que reacciona y consigue estabilizarse en su nuevo rango.

El depósito permanece en su franja de referencia hasta que un cambio en el depósito C (se pide que se llene) produce que el depósito A se desestabilice (fase ③) hasta que el sistema reacciona y vuelve a su rango de referencia (fase ④).

En este punto se abre momentáneamente el grifo que provoca una fuga en el depósito A, el nivel baja hasta que el sistema reacciona para volver al su rango (fase ⑤).

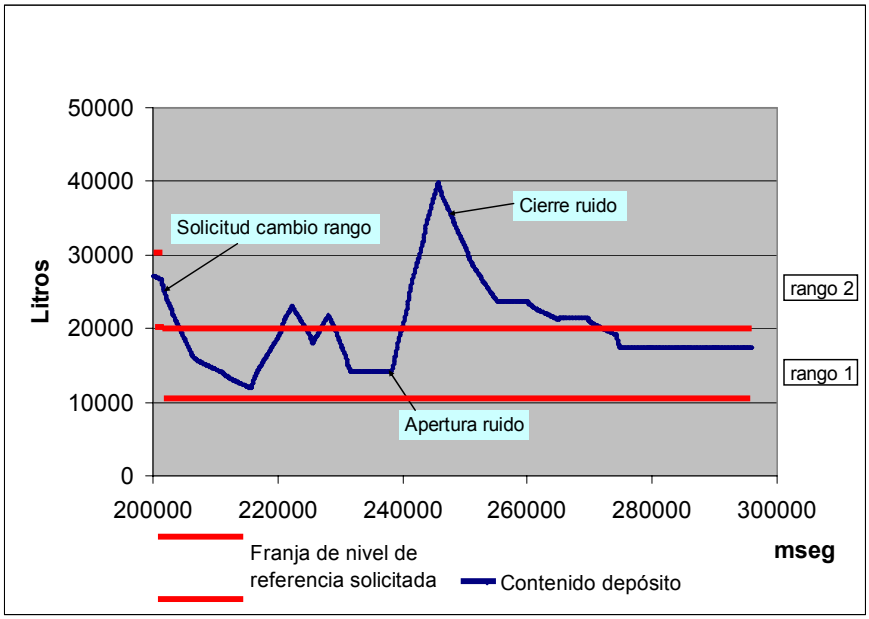


Figura A.8. Reacción del sistema ante entrada no controlada

En la figura A.8 puede contemplarse otro cambio en el rango de referencia solicitado (fase ⑥ figura A.5), en este caso el rango pasa de ser del rango 2 (2000 – 3000 litros) al rango 1 (1000 – 2000 litros). El sistema se estabiliza tras una breve oscilación. En ese momento se produce la apertura del grifo de ruido de A (fase ⑦ figura A.5), es decir se produce una entrada no controlada de liquido. Se puede observar como el sistema tiende a volver a su rango de referencia antes del cese de las perturbaciones (debido al cierre del grifo que lo provoca).

Se puede observar como finalmente (fase ⑧) el sistema tiende a estabilizarse quedando a un rango fijo y contrarrestando en su totalidad el aporte extraordinario de líquido que había sufrido en la fase anterior.

En general y observando las figuras A.5 a la A.7 en conjunto, se observa como el nivel del depósito varía para alcanzar los rangos de referencia vigentes en cada momento, reaccionando a los cambios de rango de referencia pedidos por el usuario en diferentes momentos y también para compensar posibles fugas o entradas no controladas en el sistema y que en un primer momento hacen que el depósito pierda su referencia. El sistema siempre consigue que el depósito vuelva a su rango de referencia sin que se llegue en ningún momento a la capacidad máxima.

En las páginas siguientes se muestran dos cronogramas que representan un mismo instante en la ejecución del sistema con dos heurísticas diferentes, la HSF (Figura A.9) y la SSS (Figura A.10). Las ejecuciones se han llevado a cabo en condiciones equivalentes para que ambas situaciones fueran lo más similares posibles (en la medida en que esto era posible, ya que el experimento no es posible reproducirlo con exactitud). En cualquier caso el propósito de mostrar ambos cronogramas es meramente ilustrativo de como varía la selección y el comportamiento del sistema con diferentes heurísticas, y no para realizar una comparación estricta, ya que ello no es posible en este tipo de pruebas, razón por la que se desarrollaron las pruebas en el simulador, tal y como ya se ha comentado.

En estos cronogramas (que son obtenidas de trazas de funcionamiento del sistema mediante el programa Kiwi), las líneas marcadas como In-agent 1, 2 y 3, representan los componentes críticos de cada in-agent (partes iniciales y finales). Las líneas marcadas como T6 a T9 representan los componentes opcionales del In-agent A (del 1 al 4), T10 a T13 los del in-agent B, y T14 a T17 los del In-agent C.

En las líneas de los componentes críticos, el círculo representa el comienzo del periodo de dicho in-agent y la flecha invertida el deadline. En este ejemplo ambos símbolos coinciden al ser iguales los periodos que los deadlines para cada in-agent.

La línea marcada como Kernel hace referencia al tiempo utilizado por el RT-Linux, la línea marcada como DS el tiempo gastado por el servidor inteligente, que entre otros elementos incluye el planificador de segundo nivel. Finalmente la línea marcada como Linux, refleja tiempo en el que el sistema está bajo el S.O. Linux y no se ésta realizando ninguna operación.

Al observar los cronogramas se puede percibir el efecto que produce el adelantamiento de partes finales, no sólo en el propio intervalo en el que se adelanta, sino en posteriores intervalos cambiando la distribución de los huecos de slack (El in-agent B ve ampliado su slack de forma significativa al adelantarse la parte final del in-agent C como se ve el la figura B.10 respecto a lo que ocurre en la figura B.9). Por ello parte de los trabajos que se están llevando a cabo a partir del actual es estudiar en que condiciones puede resultar interesante cambiar estos intervalos, el tiempo dedicado a ejecutar partes opcionales, etc., todo bajo control de unas reglas que dirigirían el comportamiento bajo un control de más alto nivel o metarazonamiento.

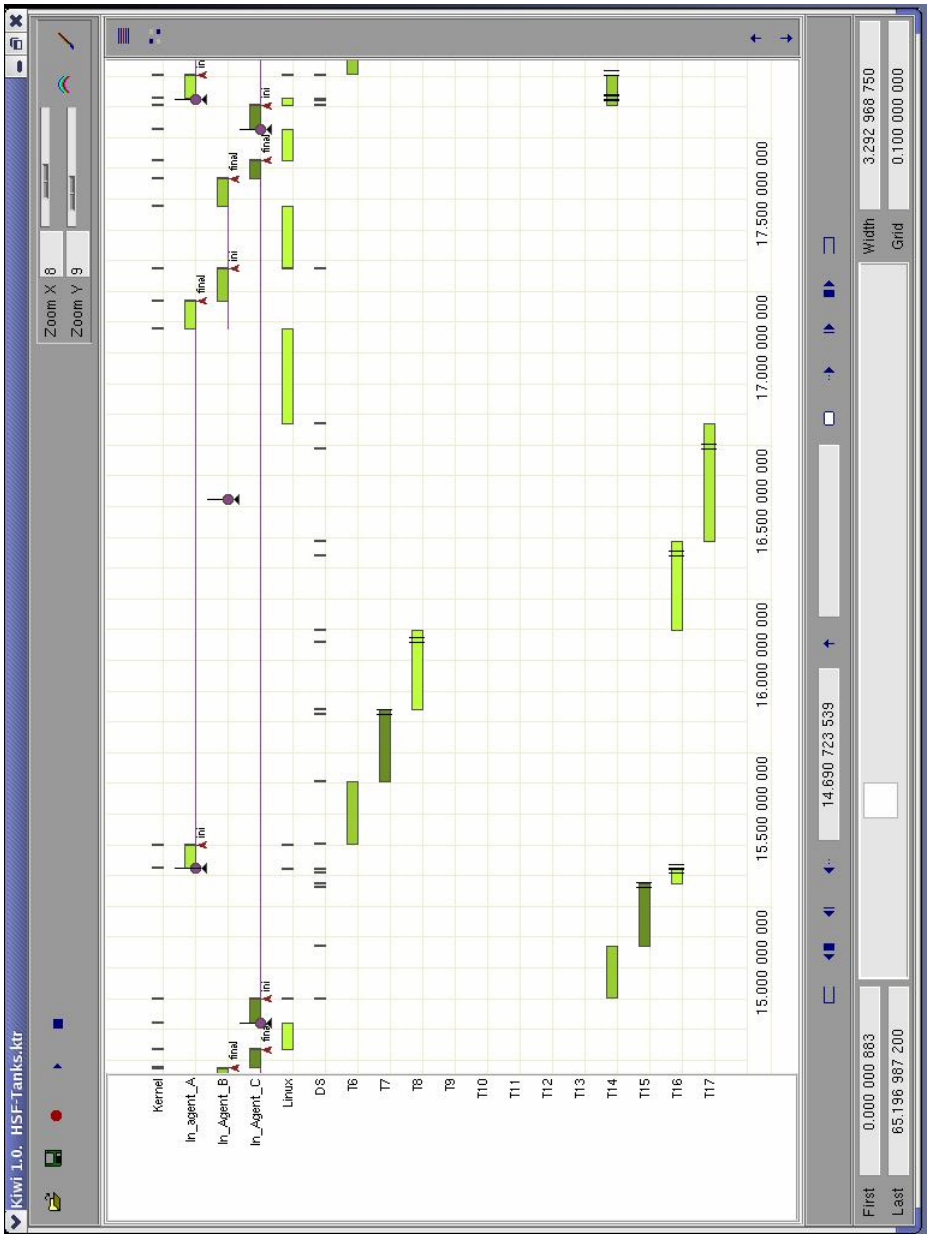


Figura A.9. Ejemplo de traza con la heurística HSF

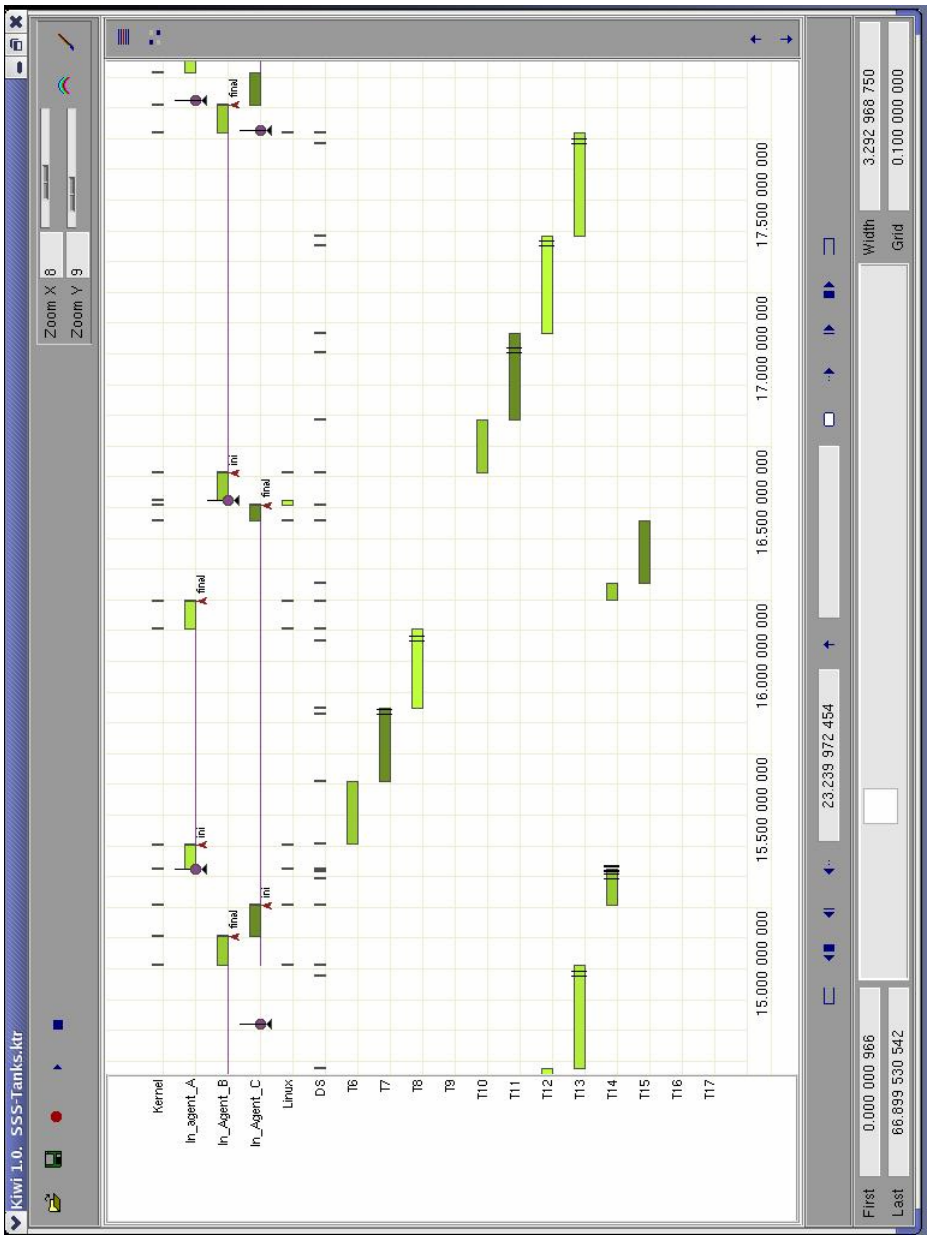
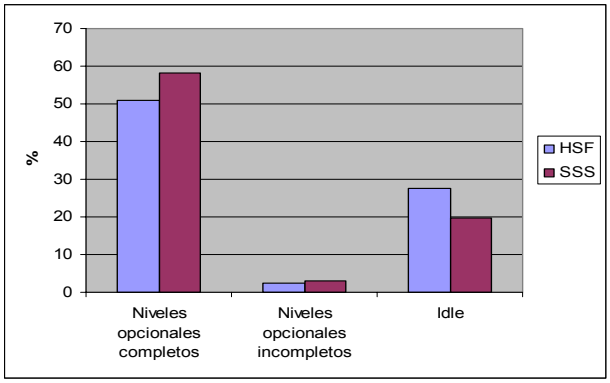


Figura A.10. Ejemplo de traza con la heurística SSS

En la gráfica A.1 se muestra el porcentaje de tiempo usado para ejecutar niveles opcionales respecto del tiempo total. Pertenecen los datos a las ejecuciones anteriores pero considerando no sólo los seis minutos presentes en la figura A.6 sino todo el hiperperiodo (1957,5 segundos). Se presentan los porcentajes (respecto al tiempo total) de tiempo dedicado a ejecutar niveles que han completado totalmente su ejecución, de tiempo gastado en niveles que no han conseguido acabarse, y de tiempo idle o tiempo que no ha podido aprovecharse para realizar operaciones, que como se ve ha sido disminuido con la heurística SSS. Finalmente indicar que las calidades relativas obtenidas al final del hiperperiodo fueron de 0.53 para HSF y 0.61 para SSS.



Gráfica A.1. Porcentajes de utilización del tiempo total en los ejemplos.

A.5 CONCLUSIONES

La aplicación desarrollada, en la cual el agente ARTIS hace uso de la heurística SSSM para la planificación de las partes opcionales, muestra como es factible su aplicación en un entorno de tiempo real. En este caso el sistema se mantenía estable durante los periodos de tiempo en que se mantenía en funcionamiento, no produciéndose un mal funcionamiento del sistema ni desbordamientos.

Además para comprobar cómo se comportaba el sistema se probó diferentes escenarios. En ciertos escenarios el sistema debía reaccionar a los cambios en el rango de llenado cuando un usuario los demandaba, variando el nivel de referencia en el panel de control de forma brusca. En otros casos se producía una alteración provocada por los grifos especialmente diseñados para ello, abriendo los grifos para que entrase agua de forma no controlada, o también para que hubiese pérdidas.

Anexo B. Ejemplo de aplicación: Agente ARTIS para el control de una maqueta de trenes

El objetivo de este anexo es similar al anterior, es decir presentar una aplicación de un agente desarrollado bajo la arquitectura ARTIS a un problema real utilizando las heurísticas desarrolladas. En este caso el sistema a controlar si es real tratándose de una maqueta de trenes.

En concreto se desea mantener en funcionamiento constante un par de trenes moviéndose dentro de un circuito evitando las colisiones. El sistema deberá detectar las posiciones de los trenes, y controlar las velocidades de cada uno de ellos así como el sentido de los cruces.

B.1 DESCRIPCIÓN DEL PROBLEMA

El sistema a controlar es el mostrado en la figura B.1. Se trata de una maqueta de trenes formada por dos grandes anillos uno interno y otro externo que comparten un tramo en común. En el problema a controlar se manejan dos trenes, cada uno de ellos circulará en uno de los dos anillos y el sistema debe encargarse de evitar problemas en la zona común.

La maqueta ha sido construida utilizando trenes de escala N y tecnología digital de la casa Marklin que permite el control de velocidad individual de cada tren. Los cambios de vías son controlados mediante circuitería

construida específicamente para dicho fin y que se conecta mediante tarjetas de E/S al ordenador. Para controlar las posiciones de los trenes se dispone de tarjetas de sensorización que detectan el paso de cada tren mediante leds, identificando cada tren mediante el led emisor que cada tren dispone.

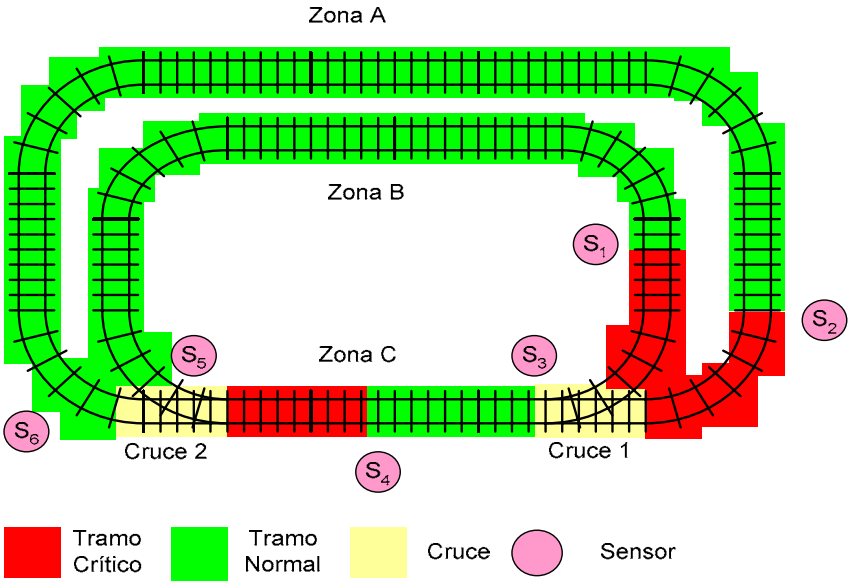


Figura B.1. Circuito de la maqueta de tren

El circuito ha sido dividido en tres zonas A, B y C, siendo esta última la zona en común. Las zonas están delimitadas por los dos cruces. Además cada zona dispone de un tramo de funcionamiento normal, donde el tren puede moverse sin ninguna restricción y un tramo de funcionamiento crítico. En estos tramos situados en la parte final de cada zona, se tiene que tomar decisiones sobre el comportamiento del tren y de la dirección del cambio de vía.

Precisamente los sensores de posición estarán situados en el punto de paso a los tramos críticos para conocer cuando llegan a dichos tramos los trenes. También habrá sensores al principio de las zonas de funcionamiento normal para detectar cuando el tren deja las zonas críticas.

Aunque existen dos cruces sólo uno, el cruce 2, necesita ser controlado para elegir a que tramo el A o el B debe ir el tren que venga del tramo C. Esto es debido a que los trenes siempre girarán en el sentido de las agujas del reloj.

Finalmente comentar que el tren que circula por la zona A se considerará el más prioritario, de manera que en el caso de que ambos trenes estén en igualdad de condiciones en los tramos críticos de las zonas A y B, será el tren de la zona A quien pasará a la zona C.

B.2 MODELO DE PROCESO

El sistema completo esta formado por varios componentes (figura B.2):

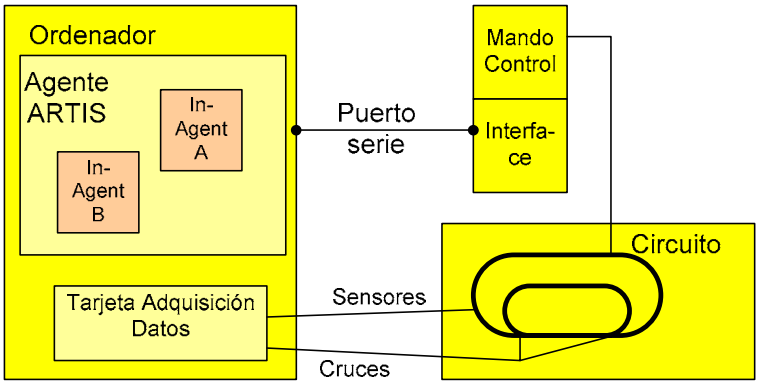


Figura B.2. Componentes del sistema

En primer lugar la maqueta formada por las vías, los trenes, los cambios de vías y los sensores de posición.

El mando de control de los trenes. Este mando dispone de un teclado que permite elegir el tren a controlar, y un mando giratorio que permite elegir la velocidad y sentido del tren. Este mando permite además la parada de todo el sistema en caso de emergencia. El mando de control está conectado a las vías del tren para transmitir la energía a las maquinas así como las ordenes. También está conectado a un interface que permite su conexión al ordenador. De esta manera el ordenador (el agente ARTIS) dará órdenes que serán recibidas por el interface, traducidas a código ASCII (que es el que internamente usan los trenes) y pasadas al mando de control que las transmite a la maqueta. En el mensaje se indica el código del tren (cada tren dispone de uno diferente) y la nueva velocidad. En nuestro problema sólo consideraremos 2 velocidades para cada tren.

Tarjeta de adquisición de datos, a esta tarjeta estarán conectadas tanto las tarjetas de cambios de vía como las de los sensores de posición.

Un ordenador donde se ejecuta el agente ARTIS, formado en este caso por dos in-agents cada uno de ellos encargados de un tren.

B.3 AGENTE ARTIS

El agente ARTIS diseñado para controlar el sistema está formado por dos in-agents cada uno de ellos encargados de un tren, siendo equivalentes excepto por el hecho de que el tren del circuito A tiene mayor prioridad.

Cada in-agent está formado por tres MKSs:

El primer MKS es el de sensorización y se encarga periódicamente de leer la información de los sensores de posición. La velocidad de los trenes y el estado del cambio de vía no pueden ser leídos, por lo que deberá mantenerse de forma interna dicha información. Cuando al leer un sensor este identifica al tren en dicho momento, se considera que el tren está en la zona correspondiente. Si ningún sensor da positivo, se considera que el tren estaba en la última zona conocida.

El segundo MKS calcula las acciones a realizar que en este caso será determinar la velocidad del tren y si es necesario cambiar la orientación del cruce de vías. Este MKS dispone de tres niveles, el nivel crítico y dos opcionales.

El nivel 0 calcula una respuesta rápida consistente en pararse si el tren entra en un tramo crítico y la siguiente zona está ocupada. En caso contrario el tren sigue a la misma velocidad actual y cambiará el sentido del cruce si es necesario. En caso de que se trate del cruce 1 y el tren B, también debe controlar si el tren A está en el tramo crítico de A. En la figura B.3 aparece el esquema básico del nivel 0 para el tren B cuando entra al cruce 1.

```
Si posición trenB = zona_criticaB
    Si posición tren A = zona C o tramo crítico A
        parar tren
```

Figura B.3. Tratamiento tramo crítico B para segundo tren

Los otros 2 niveles pretenden obtener respuestas no tan extremas. En concreto el primer nivel no para directamente el tren en la circunstancia anterior sino que baja su velocidad y espera un intervalo de tiempo antes de tomar la decisión de parar.

El segundo nivel realiza un cálculo más preciso teniendo en cuenta las velocidades respectivas de los trenes y puede decidir no parar o sólo disminuir la velocidad si ello permite el paso de los trenes sin colisiones.

Los datos significativos de los diferentes in-agents y niveles se muestran en la siguiente tabla.

In-agent	Periodo	Deadline
A	100 mseg	100 mseg
B	105 mseg	105 mseg

Tabla B.1. Datos de los In-agents

A continuación los datos para los diferentes niveles, teniendo en cuenta que para los niveles críticos (los número 0) el tiempo hace referencia al tiempo de ejecución en el caso peor, y en los opcionales (1 al 2) al tiempo medio de ejecución.

In-Agent	MKS	Importancia	Nivel	Tiempo (mseg)	Calidad
A	Percepción	0	0	10	0
	Cognición	1	0	5	0.1
			1	10	0.5
			2	20	0.4
	Acción	0	0	10	0
B	Percepción	0	0	10	0
	Cognición	1	0	5	0.1
			1	10	0.5
			2	285	0.4
	Acción	0	0	200	0

Tabla B.2. Datos de MKS y niveles

Las calidades de los niveles opcionales son acumulativas, es decir si la ejecución del primer proporciona una calidad de 0.5, la ejecución del segundo aportara 0.4 adicionales, es decir un total de 0.9.

B.4 CONCLUSIONES

Igual que en el caso anterior el agente ARTIS es capaz de llevar el control del sistema dentro de los parámetros establecidos, en este caso permitir que los trenes se mantengan en funcionamiento continuo parando los trenes cuando es necesario y reanudando su marcha sin que se produzcan colisiones entre ambos trenes, aunque el periodo de funcionamiento sea prolongado.

El agente ARTIS hace uso de las heurísticas diseñadas para elegir los niveles opcionales, siendo en este caso un ejemplo de uso de algoritmos de tipo métodos múltiples, seleccionando en cada caso el nivel 1 o 2 según las circunstancias temporales.

Para poder comprobar el funcionamiento en diferentes situaciones en las diferentes pruebas realizadas se procedió a variar las velocidades de referencia (tanto la lenta como la rápida) de los trenes, por un lado haciendo que cada tren tuviese velocidades diferentes, y por otro aumentando en general ambas velocidades para hacer más crítico el funcionamiento del sistema.

Bibliografía

[Aud93] Audsley, N. C.; Burns, A.; Davis, I. R.; Tindell, K. W.; Wellings, A. J. "Fixed Priority Pre-emptive Scheduling: A Historical Perspective". Real-Time Systems, 8, 173-198. 1993.

[Bar94] Barber, F.; Botti, V.; Crespo, A.; Gallardo, D.; Onaindía, E. "A Temporal Blackboard for Real-time Process Control". Engineering Applications of Artificial Intelligence, 7, 255-321, 1994.

[Bar94b] Barber, F.; Botti, V.; Onaindía, E.; Crespo, A. "Temporal Reasoning in REAKT: An environment for Real-Time Knowledge-Based Systems". AICOMM, 3, 175-202, 1994.

[Bod89] Boddy, M.; Dean, T. "Solving Time-Dependent Planning Problems". Proceedings of the 11th International Joint Conference on Artificial Intelligence. 1989

[Bod94] Boddy, M.; Dean, T. "Deliberation Scheduling for Problem Solving in Time-Constrained Environments". Artificial Intelligence, 67:245-285. 1994.

[Bon90] Bonissone, P.; Halverson, P.C. "Time-Constrained Reasoning under Uncertainty". Journal of Real-Time Systems, 2(1/2), 25-45. 1990.

[Bot93] Botti, V.; Barber, F.; Crespo, A.; Onaindía, E.; Ripoll, I.; Gallardo, D.; Hernandez, L. "Sharing Temporal Knowledge by Multiple Agents". Database and Expert Systems Applications. Lecture Notes in Computer Science, 720, 470-473, Springer-Verlag. 1993.

[Bot95] Botti, V.; Barber, F.; Crespo, A.; Onaindía, E.; García-Fornes, A.; Ripoll, I.; Gallardo, D.; Hernandez, L. "A Temporal Blackboard for a Multi-Agent Environment". Data and Knowledge Engineering, 15 (3). 1995.

[Bot98] Botti, V.; Barber, F.; Crespo, A. "Towards a Temporal Coherence Management in Real-time Knowledge-based Systems". Data & Knowledge Engineering, 25, 247-266. 1998.

[Bot99] Botti, V.; Carrascosa, C.; Julian, V.; Soler, J. "Modelling Agents in Hard Real-Time Environments". Lectures Notes in Artificial Intelligence, 1647, 63-76. 1999.

[Bro86] Brooks, R.A. "A Robust Layered Control System for a Mobile Robot". IEEE Journal of Robotics and Automation, RA-2, 14-23. 1986.

[Bro91] Brooks, R.A. "Integrated Systems Based on Behaviors". SIGART Bulletin, 2 (4): 46-50. 1991.

[Bur93] Burns, A. "Preemptive Priority Based Scheduling: an Appropiate Engineering Approach." Real-Time Systems Research Groep. Department of Computer Science. University of York, UK. Report number YCS214, 1993.

[Chu91] Chu, L.C; Wash, B. "Optimization in Real-Time". Proceedings of the 12th Real-time Systems Symposu¡ium. IEEE. 150-159. 1991.

[Chun90] Chung, J.Y.; Liu, J.W.S.; Lin, K.L. "Scheduling Periodic Jobs that Allow Imprecise Results". IEEE Transactions on Computers, 39, 1156-1173. 1990

[Cre94] Crespo, A.; Botti, V.; Gallardo, D.; Onaindia, E. "A temporal blackboard for real-time process control". Engineering Applications of Artificial Intelligence. Pergamon Press Ltd., 225-256. 1994.

[Dav93a] Davis, R. I. "Approximate Slack Stealing Algorithms for Fixed Priority Pre-emptive Systems". Technical report YCS217, Department of Computer Science, University of York, 1993.

[Dav93b] Davis, R.I.; Tindell, K.W.; Burns, A. "Scheduling Slack Time in Fixed Priority Preemptive Systems". In Proceedings of Real-Time Systems Symposium, 222-231. Raleigh-Durham, North Carolina, IEEE Computer Society Press. 1993.

[Dea88] Dean, T.; Boddy, M. "An Analysis of Time-Dependent Planning". Proceedings of the 7th National Conference on Artificial Intelligence. 49-54. 1988

[Dec90] Decker, K.S.; Lesser, V.R.; Whitehair, R.C. "Extending a Blackboard Architecture for Approximate Processing". Journal of Real-Time Systems, 2(1/2):47-79. 1990.

[Dru91] Drummond, M.; Bresina, J.; Kedar, S. "The Entropy Redution Engine: Integrating Planning, Scheduling and Control". SIGART bulletin 2, 61-65. 1991.

[Etz91] Etzioni, O. "Embedding Decision-Analityc Control in a Learning Architecture." Artificial Intelligence. 49:129-159, 1991.

[Fin01] Finkelstein, L.; Markovitch, S. "Optimal schedules for monitoring anytime algorithms". Artificial Intelligence 126, 63-108. 2001.

[Fra96] Franklin, S,; Graesser, A. "Is it an Agent, or just a Program?: A taxonomy for Autonomous Agents". Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag. 1996.

[Gar93] Garvey, A. Lesser, V. "Design-to-time Real_time Scheduling". IEEE Transactions on Systems, Man and Cybernetics, 23(6):1491-1502. 1993.

[Gar95] García-Fornés, A.; Botti, V. "ARTIS: Una Arquitectura para Sistemas de Tiempo Real Inteligentes." VI Conferencia de la Asociación Española para la Inteligencia Artificial, 161-174, 1995.

[Gar97] García-Fornes, A.; Terrasa, A.; Botti, V.; Crespo, A. "Analyzing the Schedulability of Hard Real-time Artificial Intelligent Systems". Engineering Applications of Artificial Intelligence, 10, 369-377. 1997.

[Garv94] Garvey, A.; Lesser, V. A "Survey of Research in Deliberative Real-Time Artificial Intelligence". Journal of Real-Time Systems, 6(3), 317-347. 1994.

[Gat91] Gat. E. "Integrating Planning and Reacting in a Heterogeneus Asynchronous Architecture for mobile robots". SIGART bulletin 2, 71-74. 1991.

[Gol01] Goldmann, R. P.; Musliner, D. J.; Krebsbach, K. D. "Managing online self-adaptation in real-time environments". In Proceedings of Second International Workshop on Self Adaptive Software. 2001.

[Gop90] Gopinath, P.; Gupta, R. "Applying Compiler Techniques to Scheduling in Real time Systems". In Proceedings of the 11th IEEE Real-Time Systems Symposium, 247-256. Orlando, Florida. 1990.

[Han01] Hansen, E.A.; Zilberstein, S. "Monitoring and control of anytime algorithms: A dynamic programming approach". Artificial Intelligence 126, 139-157. 2001.

[Hay90] Hayes-Roth, B. "Architectural Foundations for Real-Time Performance in Intelligent Agents". The Journal of Real-Time Systems, 2, 99-125. 1990.

[Hay95] Hayes-Roth, Barbara. "An Architecture for Adaptative Intelligent Systems". Artificial Intelligence, 72:329-365. 1995.

[Hor01] Horwitz, E.; Zilberstein, S. (Eds.) Special Issue: Computational tradeoffs under bounded resources. Artificial Intelligence, 126, vol 1-2. 2001.

[Hor88] Horvitz, E.J. "Reasoning under Varying and Uncertain Resource Constraints". Proceedings of the 7th National Conference on Artificial Intelligence, 111-116. 1988

[Hor90] Horvitz, E.J.; Breese, J.S. "Ideal Partition of Resources for Meta-reasoning". Technical Report KSL-90-26, Knowledge Systems Laboratory, Standford University. 1990.

[Hor91] Horvitz, E.J.; Rutledge, G. "Time-Dependent Utility and Action under Uncertainty". In proceedings of the 6th Conference on Umcertainty in Artificial Intelligence. 1991.

[How91] Howe, A. E.; Hart, D. M.; Cohen, P.R. "Addressing Real-Time Constraints in the Design of Autonomous Agents". Journal of Real-Time Systems, 2(1/2):81-97. 1990.

[Igl98] Iglesias Fernández, C. A. "Definición de una Metodología para el Desarrollo de Sistemas Multiagente". Tesis Doctoral. Departamento de Sistemas Telemáticos. Universidad Politécnica de Madrid. Enero 1998.

[Ing90] Ingrand F. F.; Georgeff M. P. "Managing Deliberation and Reasoning in Real-Time AI Systems". In Proceedings of the Workshop Innovative Approaches to Planning, Scheduling and Control, 284-291. 1990.

[Ing92] Ingrand, F.F.; Georgeff, M.P.; Rao, A.S. “An Architecture for Real-Time Reasoning and System Control”. IEEE Expert, 7 (6):34-44. 1992.

[Ing96] Ingrand, F.F.; Chatila, R; Alami, R. Robert, F. “PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots”. Proc. of the IEEE International Conference on Robotics and Automation. 1996.

[Ken90] Kenney, K.; Lin, K.J. “Structuring Real-Time Systems with Performance Polymorphism”. In Proceedings of the 11th. Real-Time Systems Symposium, 238-246. Los Alamitos, California: IEEE Computer Society Press. 1990.

[Ken91] Kenny, K.B.; Lin, K.J. “Building Flexible Real-Time Systems Using the Flex Language”. IEEE Computer, 24 (5):70-78. 1991

[Ker94] Kersual, D.; Mensch, A. “REAKT: A Real-Time Architecture for Knowledge Based Systems”. IFAC Artificial Intelligence in Real-Time Control. 513-518. 1994.

[Kle93] Klein, M.H.; Ralya, T.; Pollak, B.; Obenza, R.; González Harbour, M. “A Practitioner’s Handbook for Real-Time Análisis: Guide to Rate Monotonic Análisis for Real-Time Systems”. Kluwer Academic Publishers. 1993.

[Kor90a] Korf, R.E. “Real-Time Heuristic Search”. Artificial Intelligence, 42 (2-3):189-211. 1990.

[Kor90b] Korf, R.E. “Depth-Limited Dearch for Real-Time Problem Solving”. Journal of Real-Time Systems, 2(1/2): 7-24. 1990.

[Kuo91] Kuokka, D.R. “MAX: A meta-reasoning architecture for X”. SIGART bulletin 2, 93-97. 1991.

[Lai87] Laird, J.E.; Newell, A.; Rosenbloom, P.S. “SOAR : An Architecture for General Intelligence”. Artificial Intelligence, 33, 77-84. 1987.

[Lai93] Laird, J.E.; Yager, E.S.; Hucka, M.; Tuck, C.M. “Robo-Soar: An integration of external interaction, planning and learning using SOAR”. In Walter Van De Velde (Eds.), Towards Learning Robots, 113-130. 1993.

[Lai94] Land. J.E.; Rosenbloom, P.S. “The Evolution of the Soar Cognitive Architecture”. Technical report CSE-TR-219-94. 1994.

[Les88] Lesser, V.R.; Pavlin, J; Durfee, E. “Approximate Processing in Real-Time Problem Solving”. AI Magazine, 9 (1):49-61. 1988

[Liu73] Liu, C.L.; Layland, J. W. "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment." Journal of the Association for Computing Machinery *20*, 1, 40-61. 1973.

[Liu91] Liu, J.W.S.; Lin, K.J.; Shih, W.K.; Yu, A.C.S. “Algorithms for Scheduling Imprecise Computations”. IEEE Computer, 24(5). 1991.

[Mae91] Maes, P. “The agent network architecture (ANA)”.SIGART bulletin, 2(4), 115-120, 1991.

[Mat91] Mataric, M.J. “Behavioral Synergy Without Explicit Integration”. SIGART Bulletin 2 (4): 130-133. 1991

[Mat92] Mataric, M.J. “Behavior-Based Control: Main Properties and Implications”. Proc. IEEE International Conference on Robotics and Automation, 46-54. 1992.

[Men93] Mensch, A.; Kersual, D.; Crespo, A.; Charpillet, F. “REAKT Architecture”. Workshop on Integration Technology for Real-Time Intelligent Control Systems. Madrid 1993.

[Mic86] Michalski, R.S.; Winston, P.H. “Variable Precision Logic”. Artificial Intelligence 29 (2): 121-146. 1986

[Mit91] Mitchell, T.M.; Allen, J.; Chalasani, P.; Cheng, J.; Etzioni, O.; Ringuette, M.; Schlimmer, J.C.; “Theo: A framework for self-improving systems”. Architectures for Intelligence, K. VanLehn (ed.), 323-355, Lawrence Eribaim Associates. 1991.

[Mou92] Mouabdib, A.; Charpillet, F.; Haton, J.P. “approximation and progressive reasoning”. Procc. AAAI Workshop on Approximation and Abstraction of Computational Theories, San José, Julio 1992.

[Mus02] Musliner, D. J. “Safe learning in mission-critical domains: Time is of the essence”. In Working Notes of the AAAI Spring Symposium on Safe Learning Agents. 2002.

[Mus92] Musliner, D.J.; Durfee, E.H.; Shin, K.G. “Reasoning about Bounded Reactivity to Achieve Real-Time Guarantees”. Working Notes of the AAAI Spring Symp. in Selective Perception, 104-107. 1992.

[Mus93] Musliner, D.J.; Durfee, E.H.; Shin, K.G. "CIRCA: A Cooperative Intelligent Real-Time Control Architecture". IEEE Transactions on Systems, Man and Cybernetics, 23 (6): 1561-1574. 1993.

[Mus95] Musliner D.J.; Hendler J.A.; Agrakala A.K.; Durfee E.H.; Strosnider J.K.; Paul C.J. "The Challenges of Real-Time AI". Computer IEEE, January, 58-66. 1995.

[Mus95] Musliner, D.J.; Durfee, E.H.; Shin, K.G. "World Modelling for the Dynamic Construction of Real-Time Control Plans". AI Journal, 17(1):83-127. 1995.

[Nii86a] Nii, H. P. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures". The AI Magazine, Summer: 38-53. 1986.

[Nii86b] Nii, H. P. "Blackboard Systems: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective". The AI Magazine, Summer: 82-106. 1986.

[Nwa96] Nwana, H. S. "Software Agents: An Overview". Intelligent Systems Research. AA&T, BT Laboratories, Ipswich, UK. 1996.

[Occ98] Occello, M.; Demazeau, Y. "Modelling decisión making systems using agent satisfying real time constraints". IFAC Proceedings of 3rd IFAC Symposium on Intelligent Autonomous Vehicles, 51-56, Vol. 1, march 25-27, Madrid, Spain, 1998.

[Oga91] Ogasawara, G.H. "A distributed, decision-theoric control system for a mobile robot". SIGART Bulletin, 2:140-145. 1991.

[Pau93] Paul, C.J. "A Structured Approach to Real-Time Problem Solving". Tesis doctoral. Carnegie Mellon University, 1993.

[Raj01] Raja, A.; Lesser, V. "Real-time meta-level control in multi agent systems". In Proceedings of Multi-Agent Systems and Applications - ACAI 2001 and EASSS 2001 Student Sessions. Also Adaptability and Embodiment Using Multi-Agent Systems: AEMAS 2001 Workshop. Prague, Czech Republic. 2001.

[Rus91] Russell, S.J.; Zilberstein, S. "Composing Real-Time Systems". In proceedings of the 12th IJCAI. Sydeney, Australia. 1991.

[Rus96] Russell, S; Norvig, P. “Inteligencia Artificial: Un enfoque moderno”. Prentice-Hall. 1996.

[Shi03] Shimbo, M.; Ishida, T. “Controlling the learning process of real-time heuristic search”. Artificial Intelligence, 146, 1-41. 2003.

[Shi91] Shih, W.K.; Liu, J.W.S.; Chung, J.Y. “Algorithms for Scheduling Imprecise Computations with Timing Constraints”. SIAM Journal on Computing, 20 (3): 537-552. 1991.

[Sol00] Soler, J.; Julian, V.; Carrascosa, C.; Botti, V. “Applying the ARTIS agent architecture to mobile robot control”. In Proceedings of IBERAMIA’2000, volume I, 359-368. Springer Verlag. Atibaia, Sao Paulo, Brasil. 2000.

[Sta88] Stankovic, J. A. “Misconceptions about real-time computing”. IEEE Computer, 12(10): 10-19. 1998.

[Sta93] Stankovic, J.A.; Ramamritham, K. “EDITORIAL: What is Predictability for Real-Time Systems?”, TechnicalReport 1990-62, Dept. of Computer and Information Science. University of Massachusetts. Amherst, Mass. July, 1993.

[Ter02] Terrasa, A.; García-Fornes, A.; Botti, V. “Flexible Real-Time Linux”. Real-Time Systems Journal, 2, 149-170. 2002.

[Van91] VanLehn, K.; Ball, W. “Goal Reconstruction: How Teton Blends Situated Action and Planned Action”. Architectures for Intelligence, K. VanLehn (ed.), 147-188, Lawrence Eribaim Associates. 1991.

[Vel95] Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Furk, E.; Blythe, J. “Integrating Planning ad Learning: The Prodigy architecture”. Journal of Experimental anf Theoretical Artificial Intelligence 7 (1). 1995.

[Ver91] Vere, S. “Organization of the Basic Agent”. SIGART bulletin 2, 151-155. 1991.

[Win84] Winston, P.H. “Artificial Intelligence”. Second Edition. Addison-Wesley Publishing Compàny. 1984.

[Woo95] Woodridge, M.; Jennings, N. R. “Agent Theories, Architectures, and Languages: a Survey," Intelligent Agents, Wooldridge and Jennings Eds, Springer-Verlag, Berlin, 1-22, 1995.

[Zil93] Zilberstein, S. “Operational Rationality through Compilation of Anytime Algorithms”. Ph. D. Dissertation, Department of Computer Science, University of California at Berkeley. 1993

[Zil95] Zilberstein S. “Operational Rationality through Compilation of Anytime Algorithms”. THE AI MAGAZINE, 16 (2), 79-80. (1995).

[Zil96] Zilberstein, S. “Using Anytime Algorithms in Intelligent Systems”. AI Magazine, 17(3): 73-83. 1996.