UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Dept. Sistemes Informàtics i Computació

TESIS DOCTORAL

PROGRAMA DE DOCTORADO EN INFORMÁTICA

Computación Distribuida basada en Objetivos

Autor:
Javier Palanca Cámara

Supervisado por:
Dra. Ana García Fornes
Dr. Vicente Julián Inglada

AUTOR: **JAVIER PALANCA CÁMARA**

DIRECTORES: **DR. VICENTE J. JULIAN INGLADA**
DRA. ANA M. GARCÍA FORNES

TÍTULO: **COMPUTACIÓN DISTRIBUIDA BASADA EN OBJETIVOS**

DEPARTAMENTO: **SISTEMAS INFORMÁTICOS Y COMPUTACIÓN**

UNIVERSIDAD: **UNIVERSIDAD POLITÉCNICA DE VALENCIA**

GRADO: **DOCTOR** MES: **OCTUBRE** AÑO: **2012**

Grupo de Tecnología Informática - Inteligencia Artificial
Departament de Sistemes Informàtics i Computació
UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Camino de Vera, s/n
46020 València, Spain

A mi madre y a mi padre.

Prólogo

Cuando Javier me pidió que le escribiera el prólogo lo primero que me vino a la mente fue la música. No entendí por qué fue mi primer pensamiento hasta que empecé a leer el libro. Fue la intuición de aquello que me iba a encontrar entre las páginas de aquel documento sin encuadernar y lleno de ilusión que me llegó a las manos.

La música es muy importante en todas las fases de la vida. Hay miles de estilos, de sonidos, que marcan nuestro estado de ánimo. Podemos ir desde la calma provocada por un adagio interpretado por una orquesta hasta el enérgico furor de una canción punk. Cada estilo tiene sus instrumentos, las secciones de cuerda en la música barroca, los vientos en las formaciones de blues, la guitarra en el rock, etc, pero todos los estilos tienen algo en común, algo que los une y que es fundamental a la hora de interpretar cualquier melodía, la percusión. Cada uno de los músicos tiene un objetivo dentro de la formación pero todos, unidos, persiguen un objetivo común, que la melodía suene, y suene correctamente. Dentro de esta organización el percusionista debe de crear los huecos donde los demás interpretes tienen que enlazar sus notas, si la percusión se acelera o disminuye su cadencia todo so-

nará diferente. Da igual el estilo, da igual el resto de instrumentos, sin percusión todo es más caótico.

A medida que avancé en la lectura del manuscrito apreció que Javier, como buen percusionista, ha sabido plasmar cómo entidades con diferentes objetivos pueden trabajar conjuntamente por un bien mayor o, simplemente, para que la música *suene bien*. El sistema creado aporta al usuario de forma transparente mecanismos que gestionan eficientemente el trabajo de las entidades, teniendo en cuenta que cada una de ellas dispondrá de sus propios objetivos, ordenando el plan de ejecución de las entidades, dando paso a aquellas más prioritarias, determinando cual es su *tempo* correcto, y finalmente marcando un ritmo armónico a cada ejecución de cada una de ellas.

Su trabajo aporta un nuevo paradigma de diseño y programación que facilita la creación y gestión de sistemas multi-agentes de forma trasparente al usuario final, y ofrece un sistema de planificación de tareas basada en los objetivos que se desean alcanzar y no en los servicios que se ofrecen. Busca los huecos donde deben sonar cada uno de los instrumentos para que el objetivo, la melodía, sea correctamente alcanzado.

Puede que un sistema con varias entidades heterogéneas trabajando conjuntamente sea capaz de encontrar soluciones complejas sin necesidad de una mano que las dirija, incluso el caos en algún momento tiende al orden, pero de lo que estoy seguro es que con un percusionista, todo irá mejor.

Dr. Martí Navarro

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Índice de listados

Resumen

Para los sistemas de computación actuales, la habilidad de utilizar dinámicamente los recursos que se alojan en la red se ha convertido en un factor clave de éxito. Conforme la red sigue creciendo se hace cada vez más difícil encontrar soluciones a los problemas que los usuarios plantean al sistema computacional. Los usuarios saben habitualmente *qué* quieren hacer, pero no *cómo* hacerlo. Si el usuario conoce el objetivo que desea alcanzar es más sencillo ayudarle mediante el uso de una aproximación diferente. La cantidad de software desarrollado y la complejidad del mismo ha crecido dramáticamente en los últimos años, llevándonos a observar que los paradigmas clásicos de desarrollo de software pueden no ser suficientes para desarrollar aplicaciones muy complejas a la velocidad demandada por la industria. Es por esta razón por la que existe un constante trabajo en el desarrollo de nuevos paradigmas que aumenten el nivel de abstracción utilizado para desarrollar las cada vez más complejas aplicaciones. Entre estos paradigmas podemos destacar el de la Computación Orientada a Servicios (SOC) y los Sistemas Multi-Agente.

La Computación Orientada a Servicios es un paradigma donde el

componente fundamental para el desarrollo de aplicaciones es el servicio. Mediante el uso de servicios sencillo o de composiciones de servicios es posible alcanzar soluciones a los problemas de una forma descentralizada y con un alto grado de adaptabilidad. Este paradigma, unido al de la Computación en la Nube, están adquiriendo cada día más importancia debido a que ambos nos permiten desarrollar aplicaciones distribuidas, independientes de la plataforma y de bajo coste de los elementos computacionales. El uso de la Computación Orientada a Servicios en sistemas multi-agente viene avalado por propuestas como la de buscar y alcanzar los objetivos de los agentes por medio de la invocación y composición de conjuntos de servicios que se encuentren disponibles para el sistema multi-agente.

El análisis de la evolución de los sistemas operativos actuales, poniendo especial atención en las abstracciones utilizadas, nos ha revelado el poco uso de estos paradigmas en su diseño, manteniendo una gran distancia entre las aplicaciones diseñadas mediante estos modernos paradigmas y los diseños de los SO. Por consiguiente, en este trabajo proponemos centrarse en los principales retos actuales de las ciencias de la computación que todavía no son resueltos por los sistemas operativos (SO): la presencia en la red, la orientación a servicios y, por supuesto, los tres principales factores de diseño de todo sistema operativo: rendimiento, seguridad y fiabilidad. Para todo ello, nuestra propuesta se orienta en incrementar los niveles de abstracción proporcionados por el sistema operativo y sus servicios. Esto nos permitirá ofrecer una capa de ejecución del sistema operativo perfectamente integrada en la red y con mecanismos de seguridad y fiabilidad que no pueden estar disponibles en niveles más bajos de abstracción de los sistemas operativos actuales.

Esta evolución comienza por cambiar el paradigma utilizado en el

diseño del SO. Cambiar las abstracciones que el SO utiliza está íntimamente relacionado con el paradigma utilizado, y al cambiar este paradigma necesitamos proponer un nuevo módulo de ejecución para el SO que soporte dicho paradigma. Este módulo de ejecución es presentado también en este trabajo.

Para solventar estos problemas en este trabajo presentamos un nuevo paradigma de computación basado en objetivos. Este paradigma es el de Computación Distribuida basada en Objetivos. Para implementar el paradigma se ha diseñado una arquitectura de sistema operativo orientado a objetivos. En este SO el usuario expresa sus objetivos y el SO se encarga de ayudar al usuario a alcanzar sus objetivos por medio de una aproximación orientada a servicios. Además se presenta un simulador que da soporte a este paradigma, siguiendo los requisitos planteados en este trabajo. Algunos de estos requisitos son parámetros que comprenden cómo definirlas propiedades de un objetivo o los que definen la bondad de un plan que cumple dichos objetivos. Algunos de los parámetros que están implicados en la creación y selección de planes son el *tiempo* y la *confianza*. Para ello el sistema operativo presenta además un planificador de tareas con predicción del tiempo de ejecución y razonamiento basado en beneficios. Finalmente se han realizado un conjunto de experimentos para analizar las ventajas de la propuesta y se ha diseñado un caso de uso para validar la propuesta.

Resum

Per als sistemes de computació actuals, l'habilitat d'utilitzar dinàmicament els recursos que s'allotgen a la xarxa s'ha convertit en un factor clau d'èxit. Conforme la xarxa segueix creixent es fa cada vegada més difícil trobar solucions als problemes que els usuaris plantegen al sistema computacional. Els usuaris saben habitualment *què* volen fer, però no *com* fer-ho. Si l'usuari coneix l'objectiu que vol assolir és més senzill ajudar mitjançant l'ús d'una aproximació diferent. La quantitat de programari desenvolupat i la complexitat del mateix ha crescut dramàticament en els darrers anys, portant-nos a observar que els paradigmes clàssics de desenvolupament de programari poden no ser suficients per crear aplicacions molt complexes a la velocitat demandada per la indústria. És per aquesta raó per la qual existeix un constant treball en el desenvolupament de nous paradigmes que augmentin el nivell d'abstracció utilitzat per desenvolupar les cada vegada més complexes aplicacions. Entre aquests paradigmes podem destacar el de la Computació Orientada a Serveis (SOC) i els Sistemes Multi-Agent.

La Computació Orientada a Serveis és un paradigma on el com-

ponent fonamental per al desenvolupament d'aplicacions és el servei. Mitjançant l'ús de serveis atòmics o de composicions de serveis és possible assolir solucions als problemes d'una manera descentralitzada i amb un alt grau d'adaptabilitat. Aquest paradigma, unit al de la Computació a la Núvol, estan adquirint cada dia més importància pel fet que tots dos ens permeten desenvolupar aplicacions distribuïdes, independents de la plataforma i de baix cost dels elements computacionals. L'ús de la Computació Orientada a Serveis en sistemes multi-agent ve avalat per propostes com la de buscar i assolir els objectius dels agents per mitjà de la invocació i composició de conjunts de serveis que estiguin disponibles per al sistema multi-agent.

L'anàlisi de l'evolució dels sistemes operatius actuals, posant especial atenció en les abstraccions utilitzades, ens ha revelat el poc ús d'aquests paradigmes en el seu disseny, mantenint una gran distància entre les aplicacions dissenyades mitjançant els moderns paradigmes i els dissenys dels SO. Per tant, en aquest treball proposem centrar-se en els principals reptes actuals de les ciències de la computació que encara no són resolts pels sistemes operatius (SO): la presència a la xarxa, l'orientació a serveis i, per descomptat, els tres principals factors de disseny de qualsevol sistema operatiu: rendiment, seguretat i fiabilitat. Per tot això, la nostra proposta s'orienta a incrementar els nivells d'abstracció proporcionats pel sistema operatiu i els seus serveis. Això ens permetrà oferir una capa d'execució del sistema operatiu perfectament integrada a la xarxa i amb mecanismes de seguretat i fiabilitat que no poden estar disponibles en nivells més baixos d'abstracció dels sistemes operatius actuals.

Aquesta evolució comença per canviar el paradigma utilitzat en el disseny del SO. Canviar les abstraccions que el sistema operatiu utilitza està íntimament relacionat amb el paradigma utilitzat, i en canviar

aquest paradigma necessitem proposar un nou mòdul d'execució per al SO que suporte el paradigma. Aquest mòdul d'execució és presentat també al treball.

Per solucionar els problemes esmentats en aquest treball presentem un nou paradigma de computació basat en objectius. Aquest paradigma és el de Computació Distribuïda basada en Objectius. Per implementar el paradigma s'ha dissenyat una arquitectura de sistema operatiu orientat a objectius. En aquest SO l'usuari expressa els seus objectius i el SO s'encarrega d'ajudar l'usuari a assolir els seus objectius per mitjà d'una aproximació orientada a serveis. A més es presenta un simulador que funciona amb aquest paradigma, seguint els requisits plantejats al treball. Alguns d'els requisits són paràmetres que comprenen com definir propietats d'un objectiu o els que defineixen la bondat d'un pla que compleix aquests objectius. Alguns dels paràmetres que estan implicats en la creació i selecció de plans són el *temps* i la *confiança*. Per això el sistema operatiu presenta a més un planificador de tasques amb predicció del temps d'execució i raonament basat en beneficis. Finalment s'han realitzat un conjunt d'experiments per analitzar els avantatges de la proposta i s'ha dissenyat un cas d'ús per validar la proposta.

Abstract

For current computing frameworks, the ability to dynamically use the resources that are allocated in the network has become a key success factor. As long as the size of the network increases, it is more difficult to find how to solve the problems that the users are presenting. Users usually do know *what* they want to do, but they don't know *how* to do it. If the user knows its goals it could be easier to help him with a different approach.

The amount of developed software and its complexity has currently grown so huge that it has lead to discover that traditional paradigms of software development are not enough to create complex software. That is why there is a constant work on new paradigms, to improve the level of abstraction needed to develop increasingly complex applications. Among these paradigms, we can highlight the Service-Oriented Computing paradigm and Multi-Agent Systems.

Service-Oriented Computing (SOC) is a paradigm where the fundamental component for developing applications is the service. By using single services or service compositions it is possible to achieve solu-

tions to problems in a decentralized manner with a high degree of adaptability. This paradigm, coupled with the cloud-computing one, is becoming very important today because both paradigms allow to develop applications based on platform-agnostic, distributed and low-cost computational elements. The use of SOC in multi-agent systems is endorsed by the proposal of achieving the agent goals by means of the invocation and composition of a set of services that are available within the multi-agent system.

The analysis of the evolution of current operating systems, with particular attention to the abstractions used, revealed little use of these paradigms in design, maintaining a gap between applications designed using these modern paradigms and designs of the SO. Therefore, our proposal is to focus on major current challenges of computing science that are not solved by existing OS: the presence in the network, service-orientation and, of course, the three major design factors of OS: performance, security and reliability. For all this, our proposal is oriented to increase the level of the abstractions provided by the operating system and their services. This allows us to offer an OS execution layer integrated into the network, and security and reliability mechanisms which are not available in lower abstraction levels of current Operating Systems.

These changes begin by replacing the paradigm that is used. Changing the abstractions that an OS uses is linked to the paradigm used, and by changing these abstractions we need to propose a new execution module for the OS that supports them. The execution module that implements this paradigm is also presented in this work.

To solve this we present a new computing paradigm based on goals. This paradigm is called Distributed Goal-Oriented Computing paradigm. To implement this paradigm an execution framework for a Goal-

oriented Operating System has been designed. In this OS users express their goals and the OS is in charge of helping the achievement of these goals by means of a service-oriented approach. It is also presented a framework which gives support to this paradigm following the requirements defined in this work. Some of these requirements comprise how to define the properties of a goal and the parameters that define the goodness of a plan. Some of the parameters that involve the creation and selection of a plan are *time* and *trust*. To do this, the operating system presents also a scheduler with deadline prediction abilities and benefits-based reasoning.

Agradecimientos

«Fuerte soy por la Fuerza,
pero no tanto.»
–Yoda (Return of the Jedi)

Este trabajo no habría sido posible sin el apoyo y la fuerza de mucha gente a la que quiero agradecer especialmente su respaldo, ánimos y soporte durante tanto tiempo: familia, amigos y compañeros de trabajo. Especialmente a mi familia, gran soporte en mi vida y de la que no se puede estar más orgulloso. Sobre todo a aquellas personas a las que va dedicado este trabajo: a mi padre, que ha sido un gran padre, compañero y ejemplo a seguir, y a mi madre, en la que no he dejado de pensar ni un solo día y de la que me siento orgulloso de haber formado parte y de que ella forme parte de mi aún hoy. Quiero dedicar especialmente también este trabajo a mis sobrinos, Cesar y Julia, que han dado un vuelco a mi vida, cambiando mi escala de valores y prioridades y a los que quiero enormemente.

Los amigos han sido otro gran pilar todo este tiempo. Nombrarlos a todos sería imposible y algunos hasta dudaría en ponerlos en la sección de amigos o de familia. Entre ellos Lolo y toda su familia, Naira

y Diego, al que tengo que agradecerle la estupenda portada que ha diseñado para este trabajo, Alex, con el que puedes contar para lo que sea y sabes que no te fallará, y tantos otros que formáis parte importante de mi vida. Disculpad que no os pueda nombrar a todos.

Finalmente agradecer también a todos mis compañeros de trabajo con los que comparto el día a día. Compañeros de laboratorio presentes y pasados, y todos los grandes profesionales y amigos que he tenido la suerte de conocer aquí. Martí merece un especial agradecimiento dado que ha supuesto un gran apoyo en mi trabajo, siempre a mi lado y siempre dispuesto a que esto saliera adelante sin recibir nada a cambio más que mi más sincera gratitud. Muchos otros compañeros han sido también importantes en este camino: por nombrar a algunos, todos mis compañeros del 208, y también entre otros Stella, Andrés, Carlos, Elena, Sole, Luis... y aquellos que ya marcharon como Gustavo, Juan Ángel, Pepe, etc. Me habéis hecho entre todos el camino más agradable.

Vicent Botti también merece un apartado especial aquí, fue quien me dio la gran oportunidad de trabajar con él hace ya años y ha confiado en mi todo este tiempo, haciéndonos sentir parte de su familia. Con personas así da gusto trabajar, sin ninguna duda. Y por supuesto, agradecer a mis dos directores, Ana, que lleva conmigo desde el primer día, incansable y trabajadora, y Vicente, que además de ayudar a sacar adelante este trabajo de forma desinteresada ha sido un buen jefe y un mejor amigo. Gracias a todos.

Parte I

Motivación y Objetivos

1

Motivación y Objetivos

1.1. Motivación

En este trabajo se presenta una nueva aproximación al diseño de sistemas operativos. Para ello presentaremos un nuevo paradigma, fruto de la integración de paradigmas como el de la Computación Orientada a Servicios (SOC) y los Sistemas Multi-Agente (MAS), que emerge del análisis de los actuales paradigmas de programación. Este análisis ha revelado un espacio no cubierto clasificado dentro de la programación mediante satisfacción de objetivos utilizando estrategias de búsqueda y ejecución de acciones. Este nuevo paradigma será utiliza-

do como modelo de ejecución del sistema operativo. Este paradigma se denominará **Computación Distribuida basada en Objetivos**.

Los Sistemas Operativos (SO) son probablemente uno de los productos software más complejos de diseñar, desarrollar y mantener. Deben proporcionar al mismo tiempo una funcionalidad crítica con un buen rendimiento y dentro de unos parámetros aceptables de seguridad, tolerancia a fallos y eficacia. Su construcción es además de una gran complejidad debido a sus restricciones de fiabilidad [DCCR07] y eficiencia.

El comportamiento de los diseñadores de SO se ha mantenido alejado de las nuevas tendencias y nuevos paradigmas desarrollados en los últimos años, como la *Computación como Interacción* [LM08], la computación *peer-to-peer* [MKLN02], los sistemas autónomos [CL91] o la computación distribuida [OCD+88, SF79, Tan96], donde podemos incluir el *cloud-computing* [Ram08] y los sistemas *grid* [PL05, Coz08, FKNT02].

Las abstracciones utilizadas en el diseño de los SO han ignorado los avances realizados en este tipo de paradigmas, de los cuales podemos destacar las entidades computacionales de agente y de servicio, abstracciones de alto nivel que proporcionan una gran flexibilidad al desarrollo de aplicaciones gracias al paradigma de los sistemas multiagente y de la orientación a servicios. Además, la tecnología de sistemas multi-agente se centra en grupos de agentes que colaboran para conseguir un objetivo específico. En los últimos años el concepto de organización de agentes ha sido ampliamente utilizado, incluyendo un objetivo global, una topología definida y un conjunto de roles y normas que deben ser cumplidos por los miembros de la organización. Estas organizaciones, basadas en la Teoría de Organizaciones Humanas [AJB05], permiten implementar sistemas abiertos, donde diferentes agentes heterogéneos de diferente origen y formación pueden organi-

zarse para conseguir un objetivo global.

Los paradigmas orientados a servicios se integran muy habitualmente en sistemas de proceso de negocio. Estos son sistemas donde un conjunto de servicios se relacionan mediante una lógica de negocio para llevar a cabo una tarea. Estos sistemas son muy comunes en el B2B (Business2Business) y el B2C (Business2Client). Dentro de estos sistemas, donde la competitividad es muy alta, un factor muy valorado es el tiempo de servicio, es decir, cuanto antes y con mejor calidad sirva un proveedor su servicio, mejor reputación adquirirá y por lo tanto más ventas realizará en un futuro. Sin embargo, para poder establecer estos tiempos de servicio es necesario crear compromisos temporales y disponer de la capacidad de cumplirlos, para lo que es necesario la intervención del sistema operativo, que es el que tiene el control sobre la planificación y ejecución de las tareas agendadas.

Para afrontar la problemática expuesta anteriormente este trabajo explorará la hipótesis del uso de una metodología basada en agentes, usando conceptos de teoría de organizaciones de agentes y orientada a servicios distribuidos con restricciones temporales para avanzar en el diseño de un modelo de sistema operativo basado en agentes y orientado a objetivos. De este modo sería posible unificar la gestión del SO y del middleware orientado a agentes para mejorar la funcionalidad y el diseño del propio SO y a su vez eliminar las inconsistencias y replicaciones entre middleware y sistema operativo. Las aplicaciones del sistema pueden ser modeladas como *organizaciones de agentes*, las cuales formen un sistema modular donde cada agente puede participar dinámicamente, proporcionando una parte de la solución o de la funcionalidad requerida en forma de servicios. Del mismo modo, el propio sistema operativo puede ser modelado como una organización de agentes que ofrece sus *servicios* al resto de agentes. Algunos de es-

tos servicios serían: gestión del ciclo de vida de los agentes, gestión del directorio de servicios, acceso a los recursos, etc.

1.2. Objetivos

El objetivo de este trabajo es presentar el paradigma de **Computación Distribuida basada en Objetivos** o **DGOC** (por Distributed Goal-Oriented Computing) e integrarlo en un sistema operativo, abriendo así una nueva linea de investigación dentro de los sistemas operativos. Como veremos en este trabajo, la innovación en sistemas operativos ha sido muy conservadora durante los últimos años, quedando atrás frente a los nuevos retos que han ido apareciendo en las ciencias de la computación como han sido la inteligencia colectiva, los servicios distribuidos, computación en la nube, etc.

Con este objetivo se pretende demostrar que el paradigma de computación expuesto puede mejorar sensiblemente el modelo de ejecución de un sistema operativo con un propósito orientado tanto a los procesos de negocio y de computación en la nube como a dispositivos empotrados y móviles destinados a resolver los objetivos del día a día del usuario.

A continuación se muestran los objetivos específicos de este trabajo partiendo del objetivo general planteado:

- Estudio del estado del arte en diseño de sistemas operativos. Con ello se pretende analizar cómo han evolucionado los sistemas operativos históricamente, clasificándolos en diferentes categorías según su propósito o entorno. Se tratará de identificar sus aportaciones y carencias, con el fin de realizar una nueva propuesta de

SO basada en las técnicas orientadas a objetivos. Para ello se realizará un análisis de los paradigmas y lenguajes orientados a objetivos con el fin de realizar una caracterización de los lenguajes y tecnologías orientados a objetivos, así como las diversas técnicas utilizadas para componer planes que lleven a cabo dichos objetivos (composición de servicios web, planificación basada en casos, ...)

- Modelado y formalización del paradigma de Computación Distribuida basada en Objetivos (DGOC), considerando la especificación de las entidades que intervienen en el mismo, como los agentes, servicios, objetivos, planes, etc, y la relación entre las diferentes entidades.

- Análisis de los requisitos de la nueva propuesta de SO. Realización de un análisis de requisitos sobre aquello que se espera de un sistema operativo siguiendo una metodología específica basada en organizaciones de agentes.

- Diseño de un sistema operativo siguiendo el paradigma DGOC que incluya todos los componentes que permiten llevar a cabo la consecución de objetivos por parte de los agentes. Este diseño se debe centrar en el modelo de ejecución del sistema operativo, que es aquel que va a dar el soporte a este paradigma.

- Desarrollo de un entorno de simulación de sistemas operativos basados en DGOC, donde se desplieguen diferentes sistemas operativos interconectados y que implementen el paradigma DGOC.

- Exploración de cómo un SO puede ayudar a la ejecución de composiciones de servicios donde entren en juego los requisitos temporales y el beneficio conseguido al proveer un servicio dentro

de los parámetros acordados. Para ello se plantea la implementación de un planificador de tareas de un SO con predicción de deadline y basado en beneficios.

- Evaluación y pruebas del trabajo probando el conjunto de prototipos desarrollados en esta tesis y comparándolo con otras aproximaciones, tratando de mostrar las ventajas del mismo.

- Desarrollo de un caso de uso con el fin de validar la utilidad del paradigma propuesto en este trabajo.

1.3. Estructura de la memoria

El resto de capítulos de esta memoria se estructuran de la siguiente forma:

- El Capítulo 2 presenta el estado del arte sobre el estado actual de las técnicas utilizadas en el paradigma orientado a objetivos.

- Este estado del arte se completa en los Capítulos 3 y 4, donde nos centramos en los trabajos relacionados con la representación y razonamiento de objetivos y en la composición de planes.

- El Capítulo 5 presenta un estudio del estado del arte en diseño de sistemas operativos, centrándonos en las abstracciones y modelos de ejecución, así como en los planificadores basados en planes, que son la base del modelo de ejecución planteado en este trabajo.

- Los Capítulos 6, 7 y 8 presentan los pasos llevados a cabo para diseñar el sistema operativo basado en objetivos de este trabajo,

pasando por la definición del paradigma, el análisis y el diseño del Sistema Operativo.

- El Capítulo 9 presenta con más detalle el planificador desarrollado para que el SO tenga soporte de predicción de deadline y razonamiento basado en beneficios.
- En el Capítulo 10 presentamos un caso de uso que nos permita validar la utilidad del paradigma propuesto.
- Finalmente, el Capítulo 11 muestra las conclusiones y trabajo futuro de esta tesis, así como las publicaciones relacionadas del autor.

En el Apéndice A se muestra un middleware desarrollado previamente a este trabajo donde se han implementado y probado las aproximaciones orientadas a objetivos que han podido desarrollarse en una plataforma de agentes. Para ello se utilizó una plataforma de agentes desarrollada anteriormente, SPADE, sobre la que se implementaron algunas de las características del paradigma propuesto en este trabajo. El desarrollo de este middleware no permitió implementar todas las características requeridas, por lo que fue necesario el diseño de un SO que pueda soportar el paradigma presentado en este trabajo en su totalidad. No obstante, debido a la relación directa con la propuesta de tesis, se ha decidido añadir dicho contenido como un apéndice.

Parte II

Estado del Arte

2

El Paradigma de Computación Orientada a Objetivos

En este capítulo se hace un estudio de las diferentes aproximaciones a las técnicas utilizadas para la computación orientada a objetivos. En primer lugar se realizará un repaso por los tipos de paradigma de programación más destacables, situando el foco en el paradigma orientado a objetivos. A continuación se presentan los diferentes modelos

que implementan este paradigma de computación, centrándonos en los modelos de agente orientados a objetivos, y las diferentes tecnologías de soporte asociadas.

2.1. Paradigmas de Programación

En los últimos años se está desarrollando el concepto de computación dirigida por objetivos que presenta una nueva aproximación a los paradigmas clásicos de computación. Estos paradigmas clásicos se pueden clasificar de la siguiente forma:

Programación Funcional

Este paradigma es de tipo declarativo y se basa en la evaluación de una expresión para ser calculada mediante la utilización de funciones. Su origen está en el cálculo lambda y utiliza la reducción como operación fundamental.

Programación Imperativa

Este paradigma está basado en la Máquina de Turing en lugar de en el cálculo lambda, y su operación fundamental es la asignación. Mediante esta operación se cambia el estado de la memoria. Este paradigma es el más cercano al hardware actual dado que los computadores implementan el paradigma de la Máquina de Turing.

Programación Lógica

Este paradigma se basa en la satisfacción de predicados. Utiliza lógica matemática para solucionar el problema expresado. Para la satisfacción de los predicados expresados realiza algún tipo de estrategia de búsqueda (como el backtracking).

Estos tres paradigmas presentados se pueden situar en tres cuadrantes de un sistema de dos ejes, donde un eje representa **Evaluación vs. Ejecución** y otro eje representa **Mecanismo vs. Consecuente**. Por **Mecanismo** nos referimos a aplicar un conjunto de reglas que nos lleven de una expresión a un resultado. Sin embargo, **Consecuente** significa expresar aquello que se quiere conseguir y aplicar algún tipo de búsqueda que nos lleve de la expresión al resultado. Como podemos ver en la Figura 2.1, el paradigma de la Programación Funcional se sitúa en el cuadrante **Evaluación-Consecuente**, mientras que el paradigma de la Programación Lógica se sitúa en el cuadrante **Evaluación-Mecanismo**. La Programación Imperativa se sitúa, según esta clasificación, en el cuadrante **Ejecución-Mecanismo**.

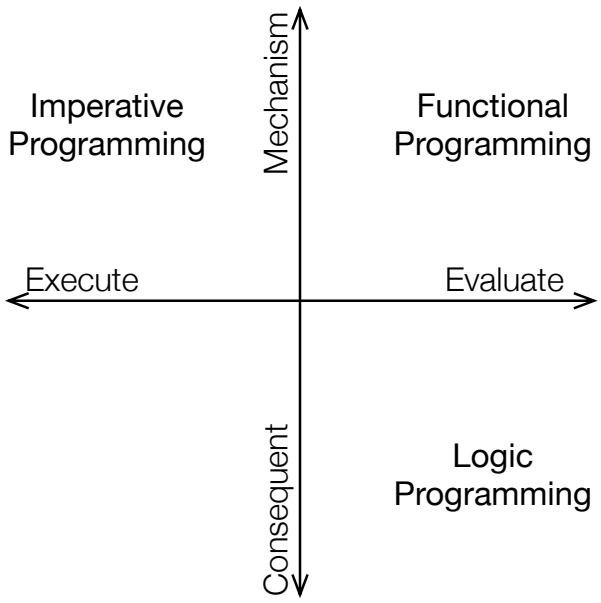


Figura 2.1: Clasificación de los paradigmas de programación

2.2. El Paradigma de Computación Orientada a Objetivos

En la Figura 2.1 hemos podido observar cómo queda un cuadrante donde no se sitúa ningún paradigma, el cuadrante de la **Ejecución-Consecuente**. En este cuadrante encaja un paradigma que trate de conseguir una solución representada por un conjunto de objetivos mediante la ejecución de acciones. Dicho de otro modo, la operación fundamental de este paradigma es la satisfacción de un objetivo (en términos de estado o efecto) mediante estrategias de búsqueda como por ejemplo la planificación iterativa con contingencias y otras técnicas modernas de Inteligencia Artificial. Este tipo de heurísticas en dichas estrategias deben ser técnicas avanzadas de búsqueda dado que en este paradigma basado en la *Ejecución* para la satisfacción de objetivos sería excesivamente costoso (e incluso peligroso) ejecutar cada una de las posibles acciones, puesto que cada acción tomada durante el proceso de satisfacción del objetivo tendría un efecto sobre el entorno. Esta es la principal diferencia con los paradigmas situados en los cuadrantes de *Evaluación*, donde se pueden considerar todas las alternativas sin más efectos colaterales que el coste computacional.

El paradigma de Computación basada en Objetivos es de tipo Ejecutivo-Consecuente, es decir, intenta llegar a la satisfacción de objetivos mediante la ejecución de acciones. A continuación vamos a describir algunos de los elementos y técnicas fundamentales para establecer un paradigma de computación basada en objetivos y el estado del arte en cada una de estas técnicas.

2.3. Agentes BDI

Los agentes BDI (Belief-Desire-Intention) [RG95, BPML04] son entidades capaces de llevar a cabo tareas y que tienen características como la *autonomía*, la *reactividad*, la *proactividad*, habilidades sociales e incluso pueden ser capaces de cooperar, aprender y adaptarse [Nwa96, WJ95, WJK99, Woo00].

El modelo de agente BDI es un modelo basado en el *razonamiento práctico*. En este modelo se espera que sean los agentes los que seleccionen qué acción ejecutar a cada momento de acuerdo a sus deseos. El modelo de razonamiento de un agente BDI se muestra en la Figura 2.2. Las *Creencias*, *Deseos* e *Intenciones* son los componentes básicos de un agente BDI que se ejecute en un entorno desconocido y con incertidumbre. Un agente BDI cambia de un estado a otro cuando lleva a cabo una acción hasta el momento en que alcanza un *Deseo*. Cuando una acción es ejecutada el agente recuerda el efecto de esa acción como una *Creencia* (Belief). Una *Intención* es un conjunto ordenado de creencias que guían el curso de acción de un agente hacia el cumplimiento de sus deseos.

Agente BDI

$$Belief : \{final\_state\} | \{state, action, state\}$$
$$Desire : \{final\_state\}^{+}$$
$$Intention : \{Belief\}^{+}$$

Figura 2.2: Agente BDI

Las creencias son, en términos BDI, expresiones que representan el

estado del entorno, como por ejemplo el estado de una variable, una base de datos relacional o una expresión simbólica en lógica de predicados. Un deseo es un término similar, que representa el estado del entorno al que se quiere llegar, que también puede ser un valor en una variable o estructura, o también una expresión en algún tipo de lógica. Las intenciones son el tercer pilar sobre el que se sustenta la arquitectura BDI. La representación de las mismas es mucho más abstracta y dependiente de la implementación, ya que deben ser planes que lleven al cumplimiento de los deseos del agente. En la arquitectura abstracta del modelo BDI propuesto por Rao y Georgeff en [RG95] se presenta un algoritmo para un intérprete BDI con creencias, deseos, intenciones y una cola de entrada de eventos. El bucle principal del intérprete se muestra en el Algoritmo 1.

Algoritmo 1: Intérprete BDI

```
1  initialize-state();
2  repeat
3  |  options := option-generator(event-queue);
4  |  selected-options := deliberate(options);
5  |  update-intentions(selected-options);
6  |  execute();
7  |  get-new-external-events();
8  |  drop-successful-attitudes();
9  |  drop-impossible-attitudes();
10 until true ;
```

La arquitectura BDI es un modelo de agente muy potente, basado en una lógica bien definida, que propone un modelo de razonamiento basado en objetivos para agentes pero que adolece de ciertas limita-

ciones. La representación de objetivos es un punto clave en este tipo de arquitecturas, puesto que es necesario definir correctamente y con gran precisión cuales son los deseos del agente, evitando estados indeseados con consecuencias potencialmente catastróficas.

Los middleware para BDI más conocidos y utilizados a día de hoy son Jason [BHW08], Jadex [BPL04] y Jack Intelligent Agents [HRHL01]. Todos ellos siguen los principios BDI estándar (deseos, creencias e intenciones) y están implementados sobre Java y en ocasiones utilizando algún lenguaje de programación de agentes específico.

2.4. Servicios Web

Los sistemas multi-agente no son el único campo donde se utilizan técnicas de planificación para lograr objetivos, otro campo donde son utilizadas este tipo de técnicas es en la composición de servicios web dentro del paradigma de la Computación Orientada a Servicios (SOC, por sus siglas en inglés) [PG03, Pap03]. Los servicios web son componentes software que se comunican mediante un conjunto de protocolos e interacciones estándar para intercambiar datos entre aplicaciones. Los servicios web permiten una alta interoperabilidad entre aplicaciones gracias al uso de estándares bien definidos.

Con la aparición de la web semántica ha surgido la necesidad de establecer mecanismos para encontrar los servicios disponibles en las redes y poder invocarlos. La importancia del **descubrimiento de servicios** es tener la capacidad de encontrar aquel servicio que realiza exactamente la función que estamos buscando, partiendo de la base de que es imposible conocer todos los servicios disponibles, debido al tamaño del conjunto y a la dinamicidad del mismo. Para ello es muy importante que los servicios estén bien descritos, como por ejemplo con el

lenguaje WSDL [CCMW01].

La primera aproximación para buscar servicios en la red (a la par que la más obvia y sencilla) es utilizar palabras clave. La sencillez de este método es a la vez su gran debilidad, puesto que depende de encontrar palabras clave en la descripción de los servicios, lo cual no te asegura que diversos servicios con las mismas palabras clave tengan la misma funcionalidad. Del mismo modo, no podemos conocer mediante este método las posibles relaciones entre servicios ni características cualitativas de los mismos.

Para solventar este problema existe una segunda aproximación al descubrimiento de servicios basada en la semántica de los mismos. Para ello podemos utilizar ontologías, que nos proporcionan una especificación formal y explícita de los conceptos compartidos entre los proveedores de servicios y sus clientes. Mediante el uso de ontologías podemos mantener una formalización lógica que nos permite describir con una misma terminología los servicios, expresar relaciones e implicaciones entre ellos y realizar consultas más precisas. En definitiva, al conocer la semántica de lo que realiza el servicio podemos encontrar con mucha más precisión aquello que buscamos, independientemente de la sintaxis y poniendo toda la atención en el significado. Se han desarrollado una serie de lenguajes de descripción de servicios que nos permiten introducir los conceptos semánticos en las descripciones de los servicios. Algunos de los más importantes son WSMO [WdBB$^+$05], SAWSDL [KVBF07, KP08] y OWL-S [MBH$^+$04], siendo este último uno de los más aceptados [LRPF04].

El descubrimiento de servicios puede ser utilizado tanto para localizar un servicio que se adapte a nuestros requisitos como para encontrar una composición de servicios que cumpla con el objetivo que buscamos. Para ello existen muchas y diversas técnicas como IOPE

(Inputs, Outputs, Preconditions and Effects), que son algoritmos que tratan de localizar servicios cuyos valores de entrada y salida (IOPE) coincidan de la mejor forma posible con el patrón buscado, tanto coincidencias exactas como subclases, subsunción, disyunciones, etc. Existen otras muchas técnicas para el descubrimiento de servicios como el uso de hipergrafos, alineamiento de ontologías, model checking, lógica difusa, técnicas de razonamiento, técnicas de descubrimiento eficiente…

En ocasiones las técnicas de descubrimiento de servicios no son capaces de encontrar un único servicio que satisfaga la búsqueda, por lo que se hace necesario utilizar la técnica de composición de servicios, como mostraremos en el Capítulo 4.

2.5. Discusión

En este capítulo se ha presentado el paradigma orientado a objetivos y las principales aproximaciones a este tipo de paradigma. Se ha visto cómo es uno de los paradigmas más nuevos en las ciencias de la computación, alejándose tanto de paradigmas imperativos como de los lógicos y funcionales, aunque hereda características de todos ellos.

El paradigma orientado a objetivos se fundamenta en la expresión del objetivo a cumplir y la definición de los mecanismos que llevan al cumplimiento de dicho objetivo. Entre las aproximaciones a este nuevo tipo de computación hemos destacado la aproximación proactiva de los agentes BDI, donde se definió el formalismo que permite expresar el conjunto de deseos, conocimientos e intenciones que permite a los agentes inteligentes tratar de lograr por sus propios medios sus

objetivos expresados mediante deseos.

Dado que los agentes son entidades inteligentes con habilidades sociales, en este trabajo se adaptarán perfectamente a un entorno distribuido basado en los principios de la computación distribuida orientada a objetivos, donde las intenciones de los agentes BDI serán construidas en función de las interacciones y la colaboración con el resto de agentes. De este modo se podría desarrollar el modelo BDI introduciendo la capacidad de no sólo encontrar, sino también generar nuevas soluciones o planes que les conduzcan a la satisfacción de sus deseos.

Este paradigma orientado a objetivos permitirá que los agentes únicamente necesiten expresar sus deseos y dispongan de los mecanismos necesarios para encontrar un plan de acción que les permita cumplir dicho objetivo con una interacción del usuario mínima.

Este es un campo totalmente abierto en la actualidad. Se encuentran diferentes aproximaciones a la resolución del problema, sin que ninguna cubra todos los aspectos de la misma. En este capítulo se han destacado las dos aproximaciones más importantes, la del modelo de agentes BDI y la de los Servicios Web. Este estudio del estado del arte deja de manifiesto la todavía falta de formalización de este paradigma, así como la falta de arquitecturas que le den soporte completo y autónomo. Estas aproximaciones se presentan siempre sobre un *middleware*, introduciendo una gran sobrecarga sobre el sistema en que se ejecutan e imposibilitando funciones de más bajo nivel y más cercanas al sistema operativo como las políticas de seguridad o características de tiempo real. Es por ello que en este trabajo se va a presentar una propuesta de formalización de este paradigma, así como una arquitectura que de soporte al mismo, poniendo de manifiesto la necesidad de incluir un sistema operativo en esta arquitectura.

3

Representación de Objetivos

El **objetivo** es el elemento fundamental del paradigma orientado a objetivos. Éste es utilizado para expresar un estado que se desea alcanzar mediante la ejecución de un conjunto de acciones admitidas y un proceso de búsqueda que permita llegar del estado actual al estado objetivo. Los objetivos deben ser un conjunto consistente de estados a los cuales se desea llegar y necesitan, por tanto, una representación explícita para poder ser consistentes.

A continuación vamos a mostrar dos ejemplos extraídos de la ficción que ilustran la problemática de la representación correcta de ob-

jetivos y sus consecuencias. El primer ejemplo es del conocido relato de ciencia ficción *Yo Robot* de I. Asimov [Asi50], donde una mala representación de objetivos, en este caso las tres *Leyes de la Robótica*, llevan a una situación catastrófica. Las tres Leyes de la Robótica definidas por Asimov son las siguientes:

Las 3 leyes de la robótica

1. Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la Primera Ley.
3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la Primera o la Segunda Ley.

Estas leyes, definidas para contrarrestar el *efecto Frankenstein* y evitar el temor humano a las máquinas, son continuamente puestas a prueba en la novela que nos ocupa, donde los robots reinterpretan continuamente las tres leyes dado que están llenas de ambigüedades como la propia interpretación de *daño* o la de *ser humano*. Se plantea, por ejemplo, si evitar el daño psicológico de la humanidad está por delante de evitar el daño físico de un único ser humano (e incluso cientos de ellos). En la versión cinematográfica de la novela los robots se plantean que el ser humano es un peligro para el propio ser humano y que lo mejor es que los robots tomen el control para proteger a los humanos de sí mismos.

Otro ejemplo, más cercano a la actualidad y a las técnicas de inteligencia artificial aplicadas hoy en día en la industria, se encuentra en el

videojuego publicado por Bethesda Softworks en 2006: *The Elder Scrolls IV: Oblivion* [UB06]. Este videojuego poseía un potente sistema de inteligencia artificial donde los agentes del sistema disponían de acciones a ejecutar en función de sus objetivos, además de un inventario con los objetos que poseían en cada momento. La situación que se presentó en este juego (y que los diseñadores de IA tuvieron que solventar) fue que cuando se le presentaba a un agente el objetivo *barrer tu casa* y éste no disponía de una escoba en su inventario podía darse el caso en el que ejecutara el siguiente plan: al no encontrar una escoba ejecuta la acción *buscar escoba*, a continuación el agente localiza una escoba en el inventario de su vecino, que está barriendo plácidamente su casa. El agente buscará la mejor forma de conseguir la escoba, y dado que tiene entre sus posibles opciones la acción *combate* decide matar a su vecino y ejecutar a continuación la acción *saquear cadáver* para hacerse con la escoba de su vecino (y de paso con todas sus posesiones). Finalmente puede ejecutar su acción *barrer tu casa*, sin tener en cuenta que para conseguir su objetivo ha tenido que asesinar y saquear durante el proceso.

Como vemos, la problemática de representar correctamente los objetivos de un agente y sus restricciones no sólo tiene problemas técnicos sino incluso filosóficos. La consistencia de los objetivos, la ambigüedad, los conflictos entres diferentes objetivos e incluso los daños colaterales deben ser tenidos en cuenta en este tipo de paradigma orientado a objetivos, donde esperamos que el razonamiento para llegar a cumplir los objetivos emerja de los propios agentes.

3.1. Lenguajes de Representación de Objetivos

La representación correcta de objetivos debe ser útil principalmente para poder razonar sobre los mismos. El sistema debe ser capaz de

decidir si puede ejecutar planes en paralelo, recuperarse de un plan fallido, retrasar o abandonar un objetivo, etc. Asimismo es importante poder especificar las relaciones entre objetivos para poder deliberar correctamente sobre los mismos. Algunos middleware como Jack y Jason no utilizan ningún tipo de representación explícita de objetivos, por lo que no pueden abordar esta problemática. El modelado de objetivos es muy complejo, razón por la que no existe un consenso sobre como representarlos, puesto que puede incluir muchas clasificaciones como por ejemplo la taxonomía (objetivos globales o individuales, estrictos o laxos) o por características temporales (objetivos con fecha de caducidad, persistentes en el tiempo, etc). En [BPML04] se realiza una clasificación según el tipo de objetivo, clasificándolos como *achieve, mantain, cease, avoid, optimise, test, query, perform* y *preserve*. Esta clasificación es bastante completa, puesto que se extrae del análisis realizado por Braubach et al. de diferentes metodologías y plataformas y aglutina un gran número de tipos de objetivo. Los objetivos de tipo *perform* especifican una actividad que debe ser realizada, un objetivo de tipo *achieve* indica una condición que se desea alcanzar, un objetivo de tipo *query* es utilizado para obtener un dato sin necesariamente ejecutar una acción y un objetivo de tipo *mantain* tiene como propósito observar un estado del entorno y hacer lo necesario para restablecerlo cuando éste ha sido cambiado. En función de la metodología o de la plataforma pueden tomar nombres diferentes, pero lo destacable es que ninguna de ellas soporta todos los tipos de objetivo. Cabe remarcar que no todas las metodologías o plataformas implementan este conjunto completo de tipos de objetivo, siendo el único común a todas en el estudio de [BPML04] el objetivo de tipo *achieve*.

En [TPH02] se presenta una propuesta de representación y resolución de conflictos para objetivos. En esta propuesta se tiene en cuenta que puedan aparecer conflictos entre objetivos (un objetivo puede ser

inconsistente con otro objetivo) y entre planes (dos planes necesitan acceso exclusivo a un recurso en un mismo instante de tiempo). Este tipo de conflictos son en general muy complicados de detectar y resolver con anticipación, por lo que suelen ser necesarias técnicas aplicadas en tiempo de ejecución para detectar y resolver este tipo de conflictos.

Los sistemas multi-agente son un campo de aplicación muy fructífero para el desarrollo de paradigmas orientados a objetivos (especialmente BDI), y dentro de estos sistemas existen diversos lenguajes de programación orientados a agentes[Sho93] que contemplan en cierta forma la representación de objetivos. Algunos de los lenguajes más representativos son GOAL [HdBVDHM01], 3APL [HDBVdHM99a], el lenguaje AgentSpeak(L) [VP96] y la propuesta que Hindriks nos presenta en [HBHM98], los cuales guardan mucha relación con Agent-0 [HdBVDHM99b] y con el lenguaje Golog [LRLL97], un lenguaje de programación basado en *Cálculo Situacional*.

Tanto Agent-0 como 3APL y AgentSpeak(L) incluyen en su especificación términos como creencias, objetivos y planes aunque con una diferencia importante con respecto a lenguajes declarativos como es GOAL. En una aproximación lógica un objetivo es una sentencia declarativa, mientras que en estos lenguajes de programación los objetivos se expresan como secuencias de acciones o planes. En 3APL se les llama objetivos, en AgentSpeak se les llama intenciones y en Agent-0 compromisos, pero en todos ellos son secuencias de acciones o, en definitiva, planes.

El lenguaje GOAL (Goal Oriented Agent Language) aparece como alternativa a estos lenguajes que expresan los objetivos de forma procedural. GOAL utiliza una aproximación declarativa que expresa qué estado se debe cumplir, y no qué acciones se deben ejecutar (*goal-to-be* frente a *goal-to-do*). Sin embargo, tal y como comentan Thangarajah y

Padgham en [TPH02], muchos de estos lenguajes (incluido GOAL) tienen una importante contrapartida, no contemplan la consistencia entre objetivos.

3.2. Razonamiento sobre Objetivos

Para solucionar el problema de la consistencia y los conflictos entre objetivos es necesario utilizar técnicas de razonamiento sobre objetivos. Estas técnicas permiten expresar interrelaciones entre objetivos que ayudan a detectar posibles conflictos o inconsistencia en los objetivos declarados.

En [TPH02] se realiza un estudio dedicado a la representación y razonamiento sobre objetivos en agentes BDI. Para ello asume la existencia del predicado $Consistent(\alpha, \beta)$ que implica que los estados objetivo α y β son consistentes. Si se da $\neg Consistent(\alpha, \beta)$ entonces existe un conflicto entre α y β, lo cual representamos mediante $Con(\alpha, \beta)$. Dado que los deseos en un sistema real son impredecibles y pueden aparecer en cualquier momento es imposible evitar la existencia de conflictos entre los mismos. Para ello proponen la existencia de prioridades entre objetivos, representados por la función Pr. Por ejemplo, $Pr(\alpha) > Pr(\beta)$ indica que α es más deseable que β. Además, para establecer la diferencia entre los estados de un objetivo se define $Des(\phi)$ como un objetivo que se desea alcanzar (un deseo), mientras que $Goal(\phi)$ indica que ϕ ha sido adoptado como objetivo (lo cual implica que debe ser consistente con el resto de objetivos adoptados). Con el objetivo de transformar un deseo en un objetivo se han definido una serie de reglas, mediante las cuales un deseo $Des(\phi)$ es adoptado como objetivo $Goal(\phi)$ o como inalcanzable $\neg Goal(\phi)$. La regla R_1 expresa que si existen dos objetivos α y β activados, no existe ningún conflicto entre

ellos.

$$R_1 : Goal(\alpha) \wedge Goal(\beta) \rightarrow \neg Con(\alpha, \beta)$$

Cuando existe conflicto entre objetivos podemos aplicar las reglas R_2 y R_3. R_2 indica que si existen conflictos entre dos deseos, el deseo α no es imposible (no está en conflicto con otros objetivos previamente instanciados) y además la prioridad de α es mayor a la de β, entonces α es instanciado como objetivo y β no.

$$R_2 : Des(\alpha) \wedge Des(\beta) \wedge \neg Imp(\alpha) \wedge Con(\alpha, \beta) \wedge (Pr(\alpha) > Pr(\beta)) \rightarrow Goal(\alpha) \wedge \neg Goal(\beta)$$

R_3 expresa, por el contrario, que dados dos deseos en conflicto y con la misma prioridad donde uno de ellos ya ha sido instanciado como objetivo, no es posible instanciar el segundo deseo.

$$R_3 : Des(\alpha) \wedge Goal(\beta) \wedge Con(\alpha, \beta) \wedge (Pr(\alpha) = Pr(\beta)) \rightarrow \neg Goal(\alpha)$$

En este mismo trabajo se describen además un conjunto de reglas que tienen en cuenta los pasos intermedios en la ejecución de planes que llevan al cumplimiento de los objetivos, tratando de evitar de este modo conflictos también durante la ejecución del plan y no únicamente en el estado final del mismo. Sobre este trabajo han realizado una serie de pruebas de rendimiento, demostrando que el razonamiento sobre objetivos para evitar conflictos introduce una sobrecarga aceptable, siempre dependiente de la profundidad de la jerarquía del plan, el número de objetivos activados concurrentemente, el número de literales en cada objetivo y el número de pasos de un plan.

Otra aproximación para la resolución de conflictos es la propuesta por K. Sycara en [Syc88], donde propone técnicas de negociación para conseguir compromisos que permitan relajar los objetivos cuando

existen conflictos. Sin embargo esta propuesta es únicamente aplicable a entornos no-cooperativos donde no se puede asumir la colaboración entre los agentes en conflicto.

Existen otros trabajos que mejoran las capacidades de razonamiento sobre objetivos extendiendo las relaciones utilizadas en la representación de objetivos. A continuación vamos a presentar algunos lenguajes que aportan técnicas de razonamiento sobre objetivos que pueden resultar útiles en el desarrollo del paradigma de computación basada en objetivos.

3.2.1. Goal Models

En [GMNS03a] y [GMNS03b] se propone un modelo de representación de objetivos basado en Goal Models. Esta propuesta introduce nuevas relaciones entre objetivos además de la relación AND y OR. Algunas de estas relaciones son la relación $++$ y la relación $--$, tales que, si $++(G, G')$, la satisfacción de G implica la satisfacción de G' (y el caso contrario, denegación, para $--$). También etiquetan las relaciones entre objetivos con los operadores $+$ y $-$, que modelan una situación donde un objetivo contribuye positivamente o negativamente a la satisfacción de otro objetivo. Este tipo de etiquetado permite establecer relaciones cualitativas y cuantitativas gracias a la semántica que han definido en el trabajo. Sin embargo, aunque Goal Models permite representar relaciones cuantitativas y cualitativas entre objetivos, limita estas relaciones a un conjunto pequeño y muy orientado a la descomposición de un objetivo en subobjetivos, lo cual no es adecuado para muchos dominios.

3.2.2. 2APL

Un descendiente del lenguaje 3APL llamado 2APL [Das08] propone una aproximación más práctica para la programación de agentes. En 2APL las creencias y objetivos se expresan en un lenguaje declarativo, mientras que los planes siguen un estilo de programación imperativo. Además introduce soporte de eventos y mensajes para gestionar las percepciones del agente, permitiendo así modificar las creencias y objetivos en función del entorno. Los planes en los agentes 2APL consisten en acciones compuestas mediante operaciones de selección condicionales, operadores de iteración, operadores secuenciales y operadores no entrelazados. Mediante un tipo de regla son capaces de generar planes para alcanzar sus objetivos. Un segundo tipo de regla se utiliza para procesar tanto eventos como mensajes recibidos. Finalmente disponen de un tercer tipo de regla destinado a gestionar y reparar planes que han fallado. Además el programador puede implementar planes atómicos que pueden ser ejecutados tanto como parte del comportamiento reactivo del agente (al responder a un evento o un mensaje) como por parte del comportamiento deliberativo del agente (al ser ejecutado para satisfacer un objetivo declarado). En resumen, 2APL es un lenguaje de programación de agentes muy completo que mejora en muchos aspectos a su sucesor 3APL, además está implementado sobre la plataforma JADE, utiliza Prolog para la declaración de creencias y es capaz de generar planes que satisfagan los objetivos de los agentes. Como añadido 2APL puede ejecutar planes en exclusión mutua cuando exista riesgo de crear un problema al ejecutar planes concurrentemente. Sin embargo 2APL carece de la capacidad de expresar, y por tanto evitar, conflictos entre objetivos. Otra desventaja de 2APL es que únicamente compone planes teniendo en cuenta las acciones disponibles por un agente, sin tener en cuenta la posible colaboración entre agentes

del sistema.

3.2.3. Cálculo Situacional

Una aproximación diferente es la que propone el lenguaje Golog, basado en *Cálculo Situacional* [LRLL97] y con un amplio uso en programación de alto nivel en robots, agentes software, simulación discreta con eventos, etc. Éste es un lenguaje de primer orden aunque con algunas características de lenguajes de segundo orden, específicamente diseñado para representar entornos cambiantes dinámicamente. Los cambios en el entorno son el resultado de *acciones*. Una *situación* es un término de primer orden que representa una secuencia de acciones que nos llevan de una situación inicial al estado actual. Puede verse una situación como un histórico de los cambios ocurridos en el entorno. La situación inicial, representada por la constante S_0, indica la situación donde todavía no han ocurrido cambios en el entorno. Las situaciones cambian gracias a los *fluentes*, que son funciones aplicadas a las situaciones y que tienen precondiciones y postcondiciones para minimizar la cantidad de posibles estados sucesores. Existe además una función binaria especial llamada $do(\alpha, s)$ que denota la situación sucesora a s al ejecutar la acción α. Por ejemplo, dada la acción $put(x, y)$, la función $do(put(A, B), s)$ expresa la situación resultante de colocar el objeto A sobre el objeto B cuando el entorno se encuentra en la situación s. Las acciones y las situaciones son términos de primer orden, por lo tanto podemos representar una situación como un histórico de acciones sobre el entorno. Así, la situación $do(putdown(A), do(walk(L), do(pickup(A), S_0)))$ representa el historial de haber ejecutado las acciones $[putdown(A), walk(L), pickup(A)]$ desde la situación inicial S_0.

El intérprete de Golog está implementado sobre Prolog, lo cual lo

convierte en un lenguaje de programación lógico. Esto hace que aunque pueda realizar planes al estilo de las técnicas de planificación clásicas de IA, con una especificación formal de las primitivas y efectos, Golog esté más centrado en la programación a alto nivel que en la planificación en tiempo de ejecución. Golog ha sido extendido para soportar concurrencia [LKM99], tiempo [PR93], razonamiento no-monotónico [Bak91] y eventos [Lak99].

3.3. Discusión

En este capítulo hemos presentado el estado del arte sobre la representación y razonamiento sobre objetivos, que será necesario para la formalización del paradigma de computación orientada a objetivos. Se ha podido observar cómo éste es uno de los campos más complejos y delicados de la Computación Orientada a Objetivos. Existen diversos lenguajes de representación de objetivos que tratan de expresar de la forma más completa posible los objetivos para una arquitectura de este tipo.

Sin embargo hemos visto como es altamente complejo definir un lenguaje que no sólo permita representar objetivos complejos, con propiedades y jerarquía, sino que además es importante definir correctamente las relaciones entre los objetivos, posibles conflictos, implicaciones, e incluso definir el deseo de ejecutar acciones sencillas por medio de este tipo de representación.

En este trabajo necesitamos un lenguaje de objetivos para poder formalizar el paradigma de computación orientada a objetivos, sin embargo ninguno de los presentados se ajusta a los requerimientos que vamos a proponer en esta tesis. Es por ello que será necesario que extendamos un lenguaje de objetivos existente con los requisitos tempo-

rales que presentaremos en este trabajo. Esta extensión será presentada más adelante en la propuesta de esta tesis.

4

Técnicas de planificación para paradigmas orientados a objetivos

Una parte importante del paradigma orientado a objetivos es encontrar una secuencia de acciones que permitan llevar al cumplimiento de los objetivos. Es importante remarcar que no necesariamente se utilizan técnicas de planificación, puesto que en muchos sistemas, como la mayoría de middleware BDI, los planes no son calculados sino que son provistos por el diseñador. Los agentes BDI tienen en su ma-

yoría de implementaciones (por ejemplo Jadex [BPL04]) una biblioteca de planes a la que acudir para buscar un plan que se ajuste al objetivo que desean cumplir.

Sin embargo, con el auge de los sistemas adaptativos han aparecido diversas técnicas de planificación que son capaces de generar un plan nuevo para cumplir un objetivo basándose en las acciones disponibles. En este capítulo presentamos algunas de las más importantes.

4.1. Composición de Servicios Web

La composición de servicios se plantea para solventar el siguiente problema: realizar una búsqueda de servicios considerando únicamente servicios atómicos sería una fuerte limitación a la hora de cumplir los objetivos establecidos. Es por ello que la composición de servicios trata de localizar grupos de servicios que, coordinados de forma particular, lleven a satisfacer los requisitos del cliente.

Existen principalmente tres estilos de composición de servicios: la orquestación de servicios, la coreografía de servicios y la composición de servicios semánticos [PTD+06]. Se diferencian básicamente en el tipo de coordinación utilizada, en la orquestación se utiliza una coordinación centralizada basada en flujos, mientras que en la coreografía de servicios se utiliza una coordinación distribuida basada en protocolos. Los servicios semánticos son compuestos de forma automática mediante el uso de algoritmos que tengan en cuenta la información semántica proporcionada en la descripción de los servicios.

4.1.1. Orquestación de Servicios

La orquestación de servicios es un tipo de composición de servicios proactiva en la que una de las partes envueltas en el proceso es la que se encarga de dirigir a los servicios que está orquestando. Este tipo de composición está basado en lenguajes de control de flujo como WSFL [Ley01], XLANG [Tha01], BPML [TSPB02] o BPEL [JEA+07, CGK+03]. En [Sha01] y [MM03] tenemos diversos estudios que comparan varios de estos lenguajes de control de flujo. Podemos ver que la mayoría de estos lenguajes son muy similares en esencia, ninguno subsume al otro pero todos aportan características similares. En la actualidad el lenguaje con más impacto es BPEL dado que es uno de los más actuales (la versión 2.0 es de 2006) y que además ha recogido la mayor parte de características del resto de lenguajes.

4.1.2. Coreografía de Servicios

La coreografía de servicios es un tipo de composición que requiere de la colaboración de varias partes. Este tipo de composición normalmente requiere del intercambio público de mensajes para establecer reglas y acuerdos que ocurren entre las diversas partes implicadas en el proceso. Mientras que la orquestación de servicios utiliza lenguajes estructurados para su definición, siendo el más representativo el lenguaje BPEL, la coreografía de servicios tiene también sus propios lenguajes de definición como por ejemplo BPNM [Whi04], Let's Dance [ZBDTH06], BPEL4Chor [DKLW07] y uno de los más extendidos, WS-CDL [KBT+05]. WS-CDL especifica los comportamientos comunes observables de todos los participantes envueltos en la coreografía.

La diferencia entre coreografía y orquestación de servicios es pequeña e incluso artificial [BDO05]. Existe en la actualidad el consen-

so implícito en la comunidad científica de ir hacia un único lenguaje y entorno para la composición de servicios para la orquestación y la coreografía de servicios web, siendo los estándares preferidos por la comunidad BPEL y WS-CDL [MH08].

4.1.3. Algoritmos de Composición de Servicios

Los algoritmos de composición de servicios utilizan la información semántica incluida en el modelo del servicio para alcanzar el objetivo establecido. Un tipo de algoritmos de composición están basados en el perfil de los servicios, típicamente utilizando DAML-S [ABH+02] o OWL-S [MBH+04]. Estos algoritmos, como por ejemplo [ACC04], tratan de localizar un servicio cuyo perfil cumpla los requisitos de la búsqueda, y si no lo encuentran realizan un encadenamiento hacia atrás con los servicios hasta encontrar una coincidencia. El problema de este tipo de algoritmos es que tratan los servicios como cajas negras, fijándose únicamente en las entradas y salidas de los servicios, pero sin prestar atención al modelo de proceso del servicio y por tanto a su comportamiento interno. Esto puede traer problemas cuando, por ejemplo, encadenamos servicios cuya salida es condicional en función del flujo interno del propio servicio, lo cual podría llevarnos a una composición incorrecta.

Bansal y Vidal desarrollaron en [BV03] un algoritmo de composición que sí utiliza el modelo de proceso de los servicios, creando árboles que representan los servicios compuestos, siendo las hojas de los árboles servicios atómicos. Este algoritmo fue refinado y mejorado por Brogi, Corfini y Popescu en [BCP05]. Esta extensión del algoritmo de Bansal y Vidal se llama *Service Aggregation Matchmaking (SAM)* y presenta una composición de servicios más flexible, teniendo en cuenta que no siempre la entrada de un servicio será satisfecha por un único

servicio y que puede ser necesario sugerir entradas adicionales de varios servicios. Para ello el algoritmo SAM crea un grafo que represente las dependencias entre todos los procesos atómicos de los servicios disponibles, para después analizar el grafo de dependencia creado y seleccionar una composición que puede satisfacer la totalidad o una parte del objetivo a cumplir. Además SAM es capaz de sugerir al usuario entradas adicionales para ayudar a encontrar una composición válida. Este tipo de algoritmos son muy ventajosos para la composición de servicios web dado que trabajan con un lenguaje estándar de representación (OWL-S) y son incluso capaces de realizar alineamiento de ontologías, sin embargo fallan a la hora de aplicar preferencias del usuario (como por ejemplo la calidad de servicio, tiempo de ejecución etc.) para restringir el conjunto de servicios sobre el que se realiza el proceso de búsqueda. Además, la mayoría de algoritmos no explotan todas las capacidades proporcionadas por los perfiles de los servicios. Por ejemplo, la mayoría de ellos utilizan las entradas y salidas de los servicios pero no tienen en cuenta las precondiciones y postcondiciones.

4.2. Planificación en Inteligencia Artificial

La planificación es un problema complejo que lleva mucho tiempo planteado dentro del campo de la Inteligencia Artificial [Wel99, RN95]. Podemos caracterizar el problema de la planificación como un proceso de búsqueda complejo. Es un tipo de problema donde un agente utiliza su conocimiento sobre las acciones disponibles y sus consecuencias para identificar una solución dentro del conjunto abstracto de posibles planes.

La planificación en IA tiene una correspondencia muy directa con los paradigmas orientados a objetivos, dado que trata de encontrar un camino o plan desde un estado inicial hasta un estado final o estado **objetivo**.

Para los problemas de planificación se utiliza un lenguaje específico llamado PDDL (Planning Domain Description Language) [GHC+98, FD03].

La planificación es un proceso de búsqueda, necesita un estado inicial del que partir, un estado objetivo y un conjunto de posibles acciones modeladas como transformaciones de estados. Dado que este proceso de búsqueda puede ser muy amplio y costoso, las técnicas de planificación tratan de optimizar las técnicas de búsqueda mediante heurísticas y técnicas de resolución de problemas eficientes.

Podemos clasificar las técnicas de planificación en inteligencia artificial según el *tipo de búsqueda* [Hof01, SY03, PW92], el *tipo de nodos* [GS02, Wel94, AB94, PW92] y el *tipo de representación* [Hof01, GS02, SY03, PW92].

Existen otras aproximaciones al margen de esta clasificación que proponen alternativas completamente diferentes. Entre ellas podemos encontrar planificadores como GRAPHPLAN [BM97] y planificadores como SATPLAN [KS96] que construyen estructuras de datos optimizadas sobre las que realizan el proceso de búsqueda. GRAPHPLAN utiliza grafos como estructuras, mientras que SATPLAN utiliza cláusulas.

Otra aproximación muy interesante también es la basada en redes jerárquicas o *Hierarchical Task Planning* (HTN) [EHN94]. En esta técnica de planificación se sustituyen las acciones abstractas por fragmentos de

planes que son capaces de llevar a cabo esas acciones. Un planificador HTN descompone la tarea deseada en subtareas, para descomponer a continuación estas subtareas en más subtareas y así sucesivamente hasta llegar a tareas atómicas. Como en otros paradigmas de planificación, HTN asume un conjunto de operadores que cumplen unos determinados efectos cuando sus precondiciones se cumplen. Además de este tipo de operaciones, HTN soporta un conjunto de *métodos* utilizados para describir cómo descomponer una tarea en subtareas. Este tipo de métodos representan conocimiento del dominio o, si lo vemos desde la perspectiva de un planificador, fragmentos de planes. La planificación HTN también es utilizada para la composición de servicios web, como puede verse en [SPW+04]. Otro trabajo donde se combina la planificación HTN con un planificador FF [Hof01] y capacidades de replanificación para la composición de servicios es el planificador X-Plan [KGS05]. Este trabajo es capaz de transformar servicios OWL-S en el lenguaje estándar de planificación PDDL para así ejecutar un planificador eficiente que encuentre una composición de servicios válida.

Como podemos observar, las técnicas de planificación en inteligencia artificial son muy útiles a la hora de llenar el espacio planteado por los paradigmas orientados a objetivos. Sin embargo todas ellas adolecen de un mismo problema, la eficiencia. La planificación es un proceso de búsqueda muy complejo, razón por la que las heurísticas son de ayuda en la búsqueda de una solución. Sin embargo los planificadores son en general razonablemente eficientes únicamente cuando tienen conocimiento del dominio, dando peores resultados en entornos donde el dominio es desconocido.

El problema de la eficiencia puede ser parcialmente resuelto gracias a los planificadores multi-agente [DWTMW05]. Los planificadores multi-agente distribuyen la tarea de planificar entre varios agen-

tes, encargando a cada uno de los agentes tareas diferentes como la coordinación del plan, refinamiento del objetivo global, planificación individual, etc. El gran problema y a la vez el gran reto de estos planificadores es el de la coordinación.

Otro problema que adolece la planificación en inteligencia artificial es el de la complejidad de los planes. Incluir un flujo de control mínimamente avanzado como condiciones o bucles introduce una enorme complejidad.

4.3. Sistemas de Planificación Basada en Casos

Los sistemas de planificación presentados anteriormente presentan potentes formas de encontrar una secuencia de acciones para llegar desde un estado inicial a un estado objetivo. Sin embargo todos ellos carecen de una potente característica que nos aporta el razonamiento basado en casos: el valor de la experiencia. Las técnicas de razonamiento basado en casos o CBR proponen una aproximación a la resolución de problemas mediante la reutilización o adaptación de soluciones que fueron aplicadas en el pasado a problemas similares.

4.3.1. Razonamiento Basado en Casos

Un sistema CBR utiliza una base de datos llamada **base de casos** donde almacena el conocimiento adquirido en la resolución de problemas anteriores. Este conocimiento es almacenado en forma de **casos**. Cuando un CBR debe tratar un nuevo problema busca en la base de casos un caso similar que comparta suficientes características para ser reutilizado. El caso es adaptado al problema actual y revisado para proponer la solución al problema. Finalmente, el nuevo problema es me-

morizado como un caso nuevo en la base de casos junto a la solución proporcionada, permitiendo así al sistema aprender de casos pasados. Un sistema CBR clásico se compone de una serie de fases que se corresponden con este proceso. Como podemos observar en la Figura 4.1, las fases de un CBR se corresponden con la fase de **RECUPERACIÓN**, en la que se busca un caso similar al actual, la fase de **REUTILIZACIÓN**, donde se aprovecha la información recuperada para proponer una solución, la fase de **REVISIÓN**, donde se adapta la solución propuesta al problema actual y se confirma que sea una solución aceptable, y la fase de **RETENCIÓN**, donde el caso aprendido es almacenado para futuros usos.

Agentes CBR-BDI

Este tipo de razonador permite a los sistemas basados en agentes el aprendizaje de forma autónoma a partir de la experiencia. Esto ha dado como resultado, por ejemplo, el diseño de sistemas BDI basados en el modelo CBR, como podemos ver en el trabajo de Corchado, Laza y otros [CPCC04, LC01]. En estos trabajos se propone una aproximación al modelo BDI de agente mediante la aplicación de las técnicas de razonamiento basado en casos, lo cual permite una aplicación práctica de este modelo de agente añadiendo además capacidades de aprendizaje. En las Tablas 4.1, 4.2 y 4.3 se puede ver una correspondencia entre un caso CBR y un modelo BDI.

Desde la perspectiva del trabajo de Corchado et al. existe una correspondencia clara entre un sistema BDI y un sistema CBR. Mientras los agentes BDI expresan el estado final al que pretenden llegar como un *deseo*, un sistema CBR lo representa como el *resultado* del entorno tras la resolución del problema, el cual puede reconocer como una situación similar de un caso anterior e intente utilizarlo para llegar a un

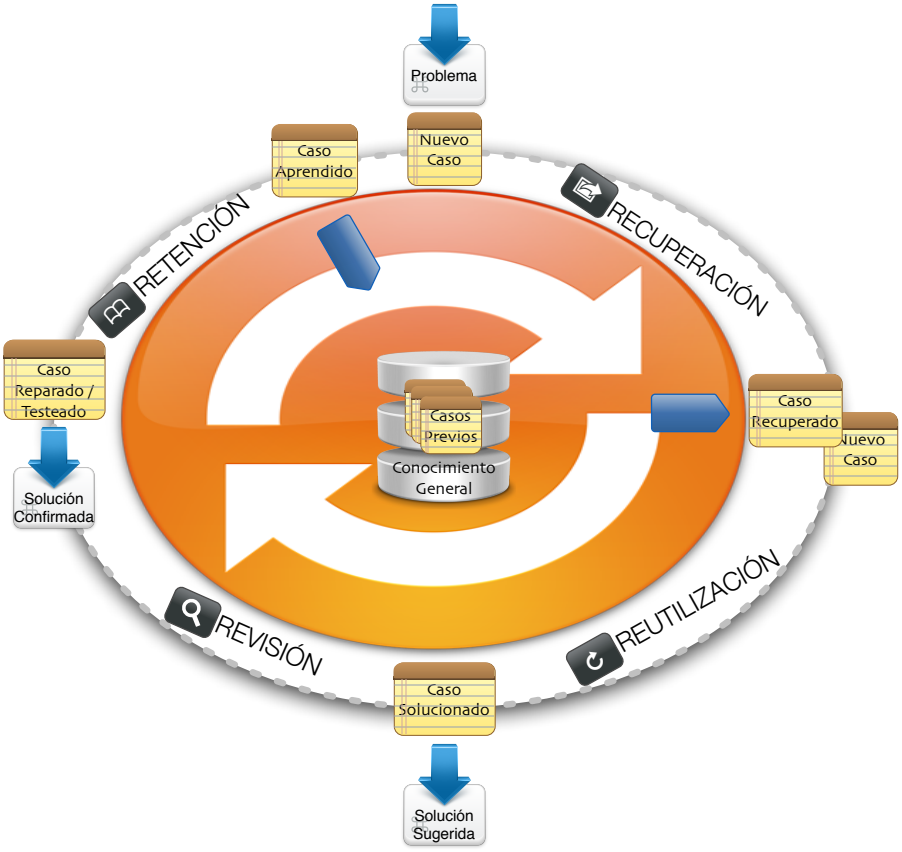


Figura 4.1: Ciclo de un sistema CBR

estado final parecido. Las *creencias* de un agente BDI, que representan el cambio de un estado a otro cuando se ejecuta una acción, tienen una correspondencia en los sistemas CBR con el *problema* que representa la situación en la que se encuentra el entorno tras la resolución del problema. También se puede encontrar una correspondencia entre las *intenciones* de un agente BDI, que representan planes de acciones a ejecutar para llegar a cumplir sus objetivos, y la *solución* de un mode-

Caso: Problema, Solución, Resultado	{}: Secuencia
Problema: estado_inicial	[]: Opcional
Solución: {acción,[estado_intermedio]}*	*: 0 o más repeticiones
Resultado: estado_final	+: 1 o más repeticiones
	|: o

Tabla 4.1: Definición de un caso CBR

Creencia:	{estado_final}|{estado,acción,estado}
Intención:	{Creencias}+
Deseo:	{estado_final}+

Tabla 4.2: Actitudes mentales de un agente BDI

CASO = INTENCIÓN = CONJUNTO DE CREENCIAS

Tabla 4.3: Correspondencia entre CBR y BDI

lo CBR, que representan el conjunto de estados que alcanza el entorno como consecuencia de la ejecución de acciones.

Según estas correspondencias se puede plantear un ciclo de razonamiento para este contexto:

- **Recuperación:** Se trata de recuperar intenciones cuyo estado final sea similar a aquel al que el agente pretende llegar. En [FC01] se plantea realizar la fase de recuperación utilizando el *Método Kernel*.
- **Reutilización:** En esta fase se utiliza la secuencia de acciones recuperada del pasado para obtener una nueva solución. Si es necesario, se adapta dicha secuencia para el caso actual.

- **Revisión:** En esta fase Pavón et al. proponen en [RFRR01] el uso de técnicas de revisión basadas en creencias.
- **Retención:** Como último paso, en esta fase se almacenan los planes en la base de casos del sistema como un nuevo caso.

Cabe destacar que la base de casos de un agente CBR-BDI debe estar inicializada con un conjunto inicial de creencias, deseos e intenciones representados en forma de casos. De este modo el agente puede iniciar su proceso deliberativo siguiendo las fases del ciclo CBR mencionado. La definición de agentes CBR-BDI ha sido formalizada en detalle en los trabajos [GBCSF02] y [GBC03].

4.3.2. Planificación Basada en Casos

La Planificación Basada en Casos puede ser definida como el acto de crear secuencias de acciones que cumplan un objetivo basándose en los recuerdos [Ham90]. Su principio, como el del razonamiento basado en casos, es el de reutilizar la experiencia adquirida en el pasado para poder volver a utilizar planes ejecutados con éxito anteriormente y para reparar planes fallidos.

La Planificación Basada en Casos (CBP) es en esencia una *estrategia de búsqueda local* que comienza desde una solución previamente generada y trata de adaptarla al problema actual.

Es en definitiva diferente de las estrategias de búsqueda local que parten de soluciones aleatorias (en lugar de basarse en la experiencia adquirida) y de las técnicas utilizadas en los problemas de planificación clásica de la inteligencia artificial, que realizan una búsqueda sis-

temática del espacio de soluciones partiendo de una solución vacía. Este tipo de búsqueda sistemática del espacio de soluciones realizado en la planificación clásica ha resultado un problema experimentalmente intratable, razón por la que se han desarrollado métodos de búsqueda optimizados y heurísticas específicas como se comentó en la Sección 4.2. Es por ello que la eficiencia es otro aspecto a tener en cuenta cuenta cuando hablamos de Planificación Basada en Casos, dado que nos permite evitar la repetición de esfuerzos ya realizados en el pasado [GAS95, HW95, IK96, KH92, VCP+95].

No obstante, la eficiencia de la planificación basada en casos no deja de ser un tema controvertido puesto que, aunque que en el caso general el hecho de reutilizar la experiencia pasada ahorra tiempo a la hora de evitar repetir operaciones ya realizadas anteriormente, siempre puede existir un caso peor en el que no sea posible reutilizar ningún caso del histórico de acciones que tenemos almacenado y la generación del plan sea tanto o más costosa que en el caso de la planificación clásica [NK95]. En ese mismo trabajo que muestra experimentalmente que si el planificador está muy optimizado para un dominio particular puede resultar mucho más eficiente, dependiendo de la similitud de los casos recuperados con el problema actual. Por otro lado, los experimentos realizados en [GAS95], [HW95], [IK96], [KH92] y [VCP+95] sugieren que la planificación basada en casos tiende a ser más eficiente.

El ciclo de planificación de un CBP es similar al de un razonador basado en casos, donde en esta ocasión los casos serán planes. Así, tenemos una fase de *RECUPERACIÓN* donde tratamos de encontrar planes que resuelvan problemas similares al problema actual. A continuación tenemos la fase de *REUTILIZACIÓN*, donde tratamos de adaptar el plan recuperado a las características particulares del plan actual. En la fase de *REVISIÓN* ejecutamos el plan, comprobamos que éste sea

correcto, y en caso negativo tratamos de repararlo. Finalmente, en la fase de *RETENCIÓN* almacenamos el nuevo plan generado para que pueda ser utilizado en casos futuros.

4.4. Discusión

En este capítulo hemos presentado las distintas técnicas que pueden ser utilizadas en el paradigma orientado a objetivos para la generación de planes que permitan cumplir los objetivos expresados. Aunque muchos modelos orientados a objetivos no utilizan generación de planes, sino que se basan en colecciones de planes provistas previamente o en interacción con el usuario, en este capítulo nos hemos centrado en las tecnologías que generan planes de forma dinámica en función de las acciones disponibles.

Este proceso de los paradigmas orientados a objetivos está íntimamente ligado al de representación y razonamiento sobre objetivos, puesto que la forma de expresarlos determinará también la técnica y la complejidad de la generación de los planes que los satisfagan.

La Composición de Servicios Web es una de las técnicas más orientadas a entornos distribuidos, donde además es necesario coordinar la interacción de diferentes proveedores de servicios. Los lenguajes de definición de servicios son de los más completos a la hora de expresar las relaciones entre ellos, los requisitos para su invocación como las entradas, salidas, precondiciones y efectos. Por otro lado, la planificación clásica en Inteligencia Artificial es uno de los procesos de planificación más potentes debido a su larga trayectoria en el desarrollo de algoritmos que optimicen el complejo proceso de búsqueda que supone encontrar un camino entre la potencialmente gran cantidad de acciones disponibles. Sin embargo uno de sus mayores problemas, al

margen de la eficiencia, es la complejidad computacional de calcular planes con estructuras de control complejas como ciclos y condiciones.

La Planificación basada en Casos permite basarse en la experiencia y, por lo tanto, aprender de situaciones anteriores para mejorar el proceso de planificación. Además, gracias al aprovechamiento de acciones pasadas, es posible optimizar cada vez más el proceso de planificación, puesto que en entornos donde se repiten continuamente situaciones iguales o similares la CBP resulta óptima.

Por estas razones la técnica de Planificación basada en Casos será la utilizada como estrategia de búsqueda del paradigma de computación orientado a objetivos presentado en este trabajo. Además extenderemos este tipo de planificación con un gestor de compromisos que nos permita realizar el proceso de planificación teniendo en cuenta parámetros del sistema en ejecución como el tiempo de respuesta o la probabilidad de éxito de los servicios que componen un plan.

5

Abstracciones y Planificadores en Sistemas Operativos

En el presente capítulo vamos a realizar un breve repaso por las abstracciones y los planificadores de los modelos de ejecución de varios sistemas operativos representativos tanto por su importancia histórica como por sus aportaciones originales. El objetivo de este trabajo es presentar la hipótesis de cómo un nuevo modelo de ejecución, basado

en objetivos, puede mejorar el diseño en el campo de los sistemas operativos. Para ello vamos a realizar previamente un breve repaso sobre su estado del arte: qué es un sistema operativo y cómo los podemos clasificar, para finalmente hacer un estudio de algunos de sus planificadores más representativos y una breve discusión de las abstracciones que utilizan.

5.1. Qué es un Sistema Operativo

El sistema operativo (SO) es el programa más importante de un computador, pues es el encargado de controlar los recursos del sistema y aportar una base sobre la que los programas de aplicación puedan ejecutarse. Además es el encargado de proteger los accesos al hardware y ofrecer una capa de abstracción que simplifica el uso de la arquitectura.

Existen dos aproximaciones complementarias para definir el sistema operativo, como máquina virtual y como administrador de recursos:

El sistema operativo como máquina virtual: el SO es capaz de abstraer el hardware para hacer más sencillo su uso. Su función es ocultar la complejidad del computador en el uso diario de las aplicaciones, no siendo así necesario preocuparse por complejidades como las interrupciones, los puertos de entrada/salida o la organización de la memoria. Observando al operativo como una máquina virtual que se interpone entre los programas de aplicación y el hardware real, podemos diseñar una gestión mucho más sencilla de los recursos a los que deben acceder dichas aplicaciones puesto que su complejidad queda oculta por el sistema operativo. De este modo se pueden aportar funciones mucho más sencillas como la abstracción de lectura y escritura de dispositivos,

en lugar de tener que acceder directamente a los recursos. Del mismo modo se puede ofrecer, por ejemplo, una interfaz de comunicación entre entidades del sistema sin necesidad de gestionar la memoria para compartir información o enviar señales.

El sistema operativo como administrador de recursos: la otra forma de aproximarse al sistema operativo es entendiéndolo como un gestor de recursos. Existen determinados recursos en un computador que deben ser gestionados de una forma específica para mantener la integridad de los mismos. Esta necesidad de gestión de recursos se acentúa cuando entra en juego la multiprogramación o los sistemas multiusuario, donde diferentes programas pueden estar accediendo de forma simultánea a un mismo recurso. En este escenario se hace necesario disponer de un gestor de los recursos que permita acceder a los mismos sin entrar en conflicto. Dicho papel lo asume el sistema operativo, haciendo uso de los modos de los procesadores actuales, que son el modo *usuario* y el modo *privilegiado*. Estos modos son utilizados para proteger determinadas operaciones sobre el hardware. De esta manera, se hace necesario pasar por el sistema operativo para realizar determinadas tareas, las cuales serán gestionadas por el mismo SO con el fin de no provocar un conflicto con dichos accesos. El sistema operativo se hará cargo además de gestionar los recursos software, como la gestión de procesos, encargándose de decidir cuándo entra cada proceso a ejecución.

5.2. Clasificación de los Sistemas Operativos

Los Sistemas Operativos existentes se pueden clasificar en diferentes categorías en función de diversos parámetros. De forma histórica se clasifican en función de: el modo de trabajo del usuario, el número

de usuarios, el propósito del sistema y el número de procesadores que soporta. La clasificación anteriormente expuesta es una aproximación basada en la evolución de los sistemas operativos desde su creación donde se ejecutaban en máquinas por lotes y eran meros supervisores del sistema. La realidad es que la evolución de la ingeniería de computadores ha hecho que sea muy habitual el trabajar con máquinas multiprocesador que soporten una gran cantidad de usuarios simultáneos con tiempo compartido de forma interactiva. Es por esta razón por la que la anterior clasificación ha quedado un poco obsoleta y es necesario revisarla. A continuación se detalla nuestra propuesta de clasificación revisada.

5.2.1. Clasificación revisada

Hoy en día, *prácticamente cualquier* sistema operativo tiene capacidades multiusuario, es interactivo y multiprocesador debido a que la evolución de los computadores ha hecho necesario que todos los SO lo soporten. Resulta por tanto más interesante la visión por propósito, dado que hay grandes diferencias en el diseño de un sistema operativo en función de su objetivo, ya sea el dispositivo en el que debe funcionar o la funcionalidad específica que deba ofrecer. A continuación se expone una clasificación de sistemas operativos más actualizada donde se distinguen los diferentes tipos de sistema diferenciados por su arquitectura o propósito. Esta clasificación no es excluyente, puesto que representa las diferentes aproximaciones que se pueden tomar a la hora de diseñar un SO, siendo posible elegir varias opciones simultáneamente. Así es posible, por ejemplo, un SO multiprocesador, extensible y de propósito general.

- **Sistemas *mainframe*:** son sistemas orientados a grandes compu-

tadores donde la potencia de cálculo y de entrada/salida (E/S) es importante. Originalmente fueron diseñados para trabajar con grandes volúmenes de datos en empresas con centros de proceso de datos. Estos SO se suelen encontrar en grandes computadores de bancos, centros de datos, etc. Típicamente soportan procesos por lotes, procesamiento por transacciones y tiempo compartido.

- **Sistemas de servidor:** están orientados a ofrecer servicios a través de la red, así como a procesar gran cantidad de peticiones por segundo con gran eficiencia. Estos sistemas normalmente no necesitan de un *mainframe* para funcionar, puesto que no necesitan gran capacidad de proceso, sino mucha eficiencia en sus comunicaciones.

- **Sistemas de propósito general:** son los destinados al gran consumo, donde su único objetivo es presentar las herramientas más comunes para el uso diario de un computador personal, con una interfaz sencilla y amigable. Estos operativos deben estar además preparados para una gran cantidad de aplicaciones no específicas dado su propósito general, como las aplicaciones ofimáticas, de entorno gráfico, juegos, etc.

- **Sistemas extensibles:** la variedad de funciones que debe cumplir un SO hace que sea muy complejo diseñar un SO de propósito general que cubra todos los aspectos con los que éste puede llegar a encontrarse. Del mismo modo, no siempre es posible disponer de SO de propósito específico para cada tarea. Los sistemas extensibles permiten la carga dinámica de nueva funcionalidad como módulos del sistema, permitiendo así *extender* el SO en función de las necesidades particulares de cada momento. De este modo el SO puede adaptar su comportamiento a las tareas que debe resolver en un determinado momento. Por ejemplo, un sistema

extensible será capaz de cambiar el módulo de acceso a disco en función de la tarea que esté ejecutando (por ejemplo, accesos cortos y rápidos a una base de datos). Los sistemas extensibles son además capaces de cambiar dinámicamente cargando o descargando las extensiones sin necesidad de parar el sistema.

- **Sistemas multiprocesador:** en la búsqueda por obtener cada vez más potencia de trabajo se ha abierto una nueva tendencia donde, en lugar de hacer procesadores cada vez más potentes (y más caros) se ha comenzado a poner a trabajar varios procesadores en paralelo como si fueran un único sistema. Estos sistemas, conocidos como multicomputadores, computadores en paralelo o multiprocesadores, necesitan de un sistema operativo adaptado para manejar estos múltiples procesadores, aunque normalmente son variaciones de sistemas operativos ya existentes.

- **Sistemas paralelos:** son una extensión de los sistemas multiprocesador donde la necesidad de ejecutar diferentes aplicaciones en varios procesadores se extiende a una red de computadores o cluster. Estos sistemas trabajan de forma coordinada utilizando envío de mensajes y memoria compartida para la comunicación entre procesos que están realizando una misma tarea y además suelen ser capaces de balancear la carga del sistema, moviendo los procesos entre los computadores del cluster con más recursos disponibles.

- **Sistemas distribuidos:** una tendencia muy extendida hoy en día es la de distribuir los diferentes servicios que ofrece un sistema operativo entre diferentes computadoras, aprovechándose del incremento de velocidad en las redes de computadores. De esta forma podemos diseñar un sistema donde el almacenamiento no se encuentre necesariamente en la misma terminal donde se en-

cuentra el usuario, o destinar los cálculos pesados a un nodo del sistema con gran potencia de procesamiento, o sencillamente redistribuir la carga del sistema de forma dinámica entre todos sus nodos.

- **Sistemas grid:** son una extensión de los sistemas distribuidos donde se tiene acceso a los recursos distribuidos geográficamente por todos los nodos de la red Grid de forma heterogénea. Esto supone una gran innovación en los sistemas distribuidos puesto que permiten el acceso a cualquier recurso sin importar el lugar en que se encuentre o las condiciones del computador donde se ejecute. Los recursos compartidos pueden ser tanto recursos hardware como capacidad computacional. Además un sistema Grid es reconfigurable de forma dinámica y escalable, permitiendo añadir y modificar nodos del sistema de forma dinámica. Esto permite abstraer el acceso a los recursos mediante su virtualización y tener un conjunto de servicios ofrecidos al usuario con independencia de la localización.

- **Sistemas de tiempo real:** existen aplicaciones muy específicas donde no importa únicamente el resultado de una operación sino además el momento preciso en que este resultado es proporcionado. Con este escenario se desarrollan los sistemas de tiempo real, orientados a asegurar que los resultados se proporcionan en un plazo preciso. Podemos encontrar sistemas de tiempo real estricto, donde no es aceptable que un plazo no se cumpla, como por ejemplo las aplicaciones donde se controlan dispositivos físicos como robótica, aeronáutica o automoción; y sistemas de tiempo real no-estricto donde sería aceptable no cumplir algún plazo, como por ejemplo los sistemas multimedia de un televisor moderno.

- **Sistemas empotrados:** con el nacimiento de pequeños sistemas hardware de bolsillo como las PDA o los teléfonos móviles, donde se dispone de poca potencia de proceso y limitaciones de memoria, almacenamiento y energía, pero se requiere de gran rendimiento y disponibilidad, se ha hecho necesario diseñar nuevos sistemas operativos capaces de sacar el máximo rendimiento a estos pequeños dispositivos. Encontramos SO empotrados en los dispositivos móviles PDA o los teléfonos móviles.

- **Sistemas ultra-empotrados:** existen sistemas todavía más pequeños que los sistemas empotrados, como los que aparecen en las tarjetas inteligentes, donde las limitaciones tanto de memoria como de potencia de proceso son todavía más estrictas. Dichos sistemas operativos en muchas ocasiones únicamente realizan una tarea concreta (como un pago electrónico) donde la necesidad específica de un sistema operativo ultra-empotrado no sería demasiado grande. Sin embargo otras propuestas incluyen sistemas más completos como una máquina virtual de Java capaz incluso de ejecutar varias tareas con multiprogramación, con lo que se hace necesario un planificador y un gestor de recursos. Con este fin nacen los sistemas ultra-empotrados.

En este listado de sistemas operativos podemos distinguir varios grupos donde se identifican las tendencias actuales en el desarrollo de sistemas. De este modo, se distinguen los sistemas operativos *orientados a servicios* (como los servidores), los orientados a *mejorar el rendimiento o disponibilidad* de la red (como los sistemas distribuidos, grid, paralelos, etc), los sistemas orientados a un *propósito particular* (como los de tiempo real o los empotrados), donde cada día los sistemas empotrados adquieren más importancia debido a la introducción cada vez más masiva de los dispositivos móviles y, finalmente, los siempre pre-

sentes sistemas operativos de *propósito general*, que a día de hoy siguen teniendo gran importancia dada la gran penetración de los computadores personales en los hogares. Podemos observar cómo las tendencias en la evolución de los computadores afectan muy directamente a la evolución de los sistemas operativos. Por ejemplo, hoy en día existen dos campos que están marcando muy fuertemente esta evolución: los dispositivos móviles y la importancia de la red. Ambos han adquirido gran importancia y han marcado la necesidad de plantear de nuevo la arquitectura de los sistemas operativos que en un principio no fueron diseñados para ese propósito. Aquellos que tratan de mantener su antigua arquitectura y abstracciones son los que más sufren a la hora de adaptarse a estas nuevas tendencias.

5.3. Algoritmos de planificación

Una de las aportaciones de este trabajo de tesis consistirá en un nuevo algoritmo de planificación que permite a un SO crear un plan de ejecución que tenga en cuenta la predicción de cuándo van a finalizar su ejecución las tareas y cuánto beneficio, en términos cuantitativos, aportará dicha ejecución al SO. En esta sección exploraremos un conjunto de planificadores clásicos muy representativos por su uso extensivo o por conformar la base en la teoría de planificadores, que nos serán muy útiles para comparar las diferentes técnicas en las que se puede compartir el tiempo de CPU, como los algoritmos *justos*, algoritmos de tiempo real o algoritmos no expulsivos entre otros. Adicionalmente, en esta sección también exploraremos una categoría específica de planificadores llamados Planificadores basados en Planes[1],

[1] La traducción del inglés de *Planning-based Scheduling* resulta en nuestro idioma un tanto redundante.

los cuales tienen mucha relación con el planificador presentado en este trabajo.

Los planificadores de los sistemas operativos de propósito general utilizan habitualmente algoritmos que comparten el procesador de forma equitativa entre todas las tareas. A esta categoría de planificadores la llamamos algoritmos *justos* (*fair algorithms*, del inglés). El más representativo de esta categoría es el algoritmo Round Robin. Este algoritmo divide el tiempo del procesador en unidades de tiempo llamadas *quantums* y le da a cada tarea la misma cantidad de quantums de forma cíclica. Cuando una tarea que se encuentra en ejecución consume su quantum es expulsada del procesador para dar cabida a la siguiente tarea de la cola. La tarea expulsada se pone al final de la cola. De este modo el algoritmo Round Robin reparte la misma cantidad de tiempo de procesador entre todas las tareas, permitiendo un grado de interactividad bastante bueno. De todas formas, este algoritmo no nos permite realizar predicciones de tiempo dado que no podemos predecir cuantas tareas va a haber en la cola y, por tanto, con cuantas habrá que compartir el tiempo de procesador.

Existe una interesante evolución de este tipo de algoritmos de planificación que tratan de repartir el tiempo de procesador de forma justa. Esta evolución del algoritmo de planificación se denomina *Completely Fair Scheduler* (CFS) [WTKW08]. Este algoritmo se diseñó como sustituto del algoritmo $O(1)$ para el núcleo de Linux. En lugar de utilizar colas, CFS utiliza una estructura más compleja llamada árboles red-black. Los árboles red-black tienen una complejidad temporal de $O(log_n)$ para las operaciones de inserción, búsqueda y borrado. CFS extrae de forma muy eficiente del árbol red-black el proceso que ha utilizado la menor cantidad de CPU (el cual se almacena siempre en el nodo más a la izquierda del árbol). Aunque éste es un algoritmo de

planificación muy justo y que incrementa notablemente el rendimiento de los sistemas multi-tarea, tampoco es capaz de realizar predicciones temporales, lo que es un requisito para nuestro sistema.

Existe un algoritmo muy sencillo, llamado FCFS (First-Come First-Served), que sirve las tareas en orden de llegada de forma no expulsiva. Este algoritmo fue muy popular en los primeros sistemas de procesado por lotes y únicamente necesita de una cola donde va almacenando las tareas por orden de llegada. Al ser un algoritmo no expulsivo las tareas en ejecución nunca serán interrumpidas. Por supuesto, en este tipo de algoritmo no expulsivo sería muy sencillo realizar una predicción del tiempo de ejecución de las tareas, puesto que una vez comienzan su ejecución no van a ser interrumpidas, por lo que conociendo su WCET podemos utilizar este dato para realizar la predicción. Sin embargo, este tipo de algoritmos tiene una gran contrapartida, que es precisamente su característica de ser no expulsivo. Esto conlleva una poca interactividad y nulas capacidades multi-tarea, lo que hoy en día son más requisitos que características deseables, puesto que es importante que el usuario perciba que el sistema no se queda bloqueado haciendo una única tarea mientras el resto de tareas están bloqueadas y esperando. Esta contrapartida no es aceptable para el sistema distribuido que proponemos en nuestro problema.

La predicción del tiempo de finalización de las tareas es un problema diferente al que podemos encontrar en los problemas de tiempo real. Los sistemas de tiempo real deben ser capaces de planificar tareas que obligatoriamente se ejecuten antes de un plazo de tiempo conocido como *deadline*. Este plazo es obligatorio en los sistemas de tiempo real *críticos (hard)*, aunque ocasionalmente puede no ser cumplido en sistemas de tiempo real *no críticos (soft)* (aunque no es deseable en ningún caso). Estos sistemas de tiempo real utilizan planificadores muy es-

pecíficos que aseguran que cada tarea aceptada se ejecutará antes de su plazo establecido. En estos sistemas es también muy importante el disponer de un correcto cálculo del WCET.

La manera más sencilla de planificar tareas de tiempo real es utilizar un *ejecutivo cíclico*. Este ejecutivo sirve para planificar tareas cíclicas. Cada una de estas tareas tiene un periodo definido y un WCET, por lo que el ejecutivo cíclico crea un plan de ejecución fijo, llevado a cabo por el diseñador del sistema. El ejecutivo cíclico puede reemplazar en la práctica a todo el sistema operativo, dado que únicamente necesita un bucle infinito que comprenda el *hiperperiodo* donde caben la ejecución de todas las tareas ordenadas por el diseñador.

Existen planificadores de tiempo real mucho más completos y dinámicos que el ejecutivo cíclico, como los que ordenan por prioridades en función de la proximidad del plazo o los que utilizan dos colas de prioridad para tratar de priorizar las tareas con más riesgo. Sin embargo, en este trabajo vamos a centrarnos en la categoría de planificadores Planning-based Scheduling por su proximidad al problema que nos ocupa.

Planning-based Scheduling es un tipo de planificación dinámica de tiempo real que trata de dar garantías a las tareas que llegan al sistema mediante mecanismos de control de admisión. Estas garantías se centran en la capacidad del sistema de satisfacer las restricciones temporales o plazos de las tareas venideras.

En Planning-based scheduling (PBS) habitualmente se crea un plan de ejecución que comprende todas las tareas que deben ser ejecutadas, lo cual habitualmente implica la asignación de prioridades a las tareas. En el caso de utilizar prioridades dinámicas, las prioridades relativas

de cada tarea pueden cambiar conforme avanza el tiempo, así como al ejecutar las diferentes tareas.

PBS pasa habitualmente por los siguientes tres pasos: Análisis de viabilidad, construcción de la planificación y ejecución. El análisis de viabilidad se realiza mediante la comprobación de la planificabilidad de la tarea, lo cual implica comprobar si las restricciones temporales que la tarea impone podrán ser satisfechas. Este análisis de viabilidad se realiza habitualmente cuando la tarea llega al sistema. Este tipo de análisis es más apropiado para tareas periódicas, dado que conocemos el periodo en el que se van a ejecutar y los recursos que van a necesitar pueden ser fácilmente calculados y reservados. Este análisis de viabilidad también puede ser aplicado a tareas aperiódicas.

La construcción del plan es el proceso mediante el cual se ordenan las tareas que van a ser ejecutadas. Este orden se almacena para su uso en la fase de ejecución. Esta fase de construcción del plan se suele realizar cuando se asignan las prioridades a las tareas. Finalmente, la fase de ejecución se encarga de decidir cual es la tarea que debe ejecutarse a continuación. Este proceso puede consistir sencillamente en seguir el plan establecido, dependiendo de si el sistema es expulsivo o no expulsivo, de la naturaleza de la plataforma de ejecución o incluso de si la construcción del plan forma parte del análisis de viabilidad.

Existen dos algoritmos de referencia para Planning-based Scheduling: RED y Spring. RED (Robust Earliest Deadline) [BS93] es un algoritmo de planificación robusto para tratar con tareas aperiódicas en entornos sobrecargados. Los planificadores robustos separan las restricciones temporales y la importancia mediante la consideración de dos políticas: una para la aceptación de tareas y otra para el rechazo de tareas. Spring es el algoritmo utilizado en el núcleo del SO Spring. Este algoritmo tiene un mecanismos basado en planes para el control

de admisión que es capaz de tratar con los recursos requeridos por las tareas, más allá de el recurso del procesador.

Podemos observar cómo estos algoritmos están muy centrados en sistemas de tiempo real y en sus mecanismos de admisión. Estos algoritmos no sirven para nuestro propósito puesto que en este trabajo no presentamos un sistema de tiempo real ni instalamos mecanismos de admisión en el mismo. Nosotros en este trabajo nos centraremos en estudiar el comportamiento de la ejecución de las tareas para poder realizar predicciones temporales y establecer acuerdos entre los clientes y los proveedores de servicios. Para ello proponemos un nuevo planificador de tareas como veremos en el Capítulo 9.

5.4. Abstracciones de los Sistemas Operativos

Las abstracciones de los sistemas operativos nos permiten simplificar el uso que se hace del hardware sin necesidad de encontrarnos a muy bajo nivel, viendo el computador como una máquina virtual. Estas abstracciones han estado habitualmente muy ligadas a la propia evolución de la arquitectura de computadores, manteniendo un interbloqueo entre ellas que ha resultado en pocas innovaciones en los últimos años.

Además, un obstáculo importante a la hora de introducir innovaciones en los SO actuales ha sido la compatibilidad hacia atrás. A día de hoy existe tal cantidad de aplicaciones dependientes de la arquitectura de un SO que hace muy complicado proponer grandes cambios en el diseño de los sistemas operativos. Existen determinadas abstracciones del sistema, como la orientación a fichero o el modelo de ejecución por procesos e hilos, muy establecidas que dificultan este cambio. Esta es la razón por la que se han producido pocas innovaciones, sobre todo

en la forma en que son ofertados los servicios del SO. Ha habido diversos intentos de integrar nuevas tecnologías en el diseño de los sistemas operativos como la tecnología de la orientación a objetos o las llamadas a procedimientos remotos, pero no todas han tenido una buena aceptación ni una buena introducción en el mercado, que es un indicador bastante fiable.

En la Tabla 5.1 se muestra un extracto de algunos de los sistemas operativos que han marcado una evolución significativa en el diseño de SO, remarcando algunas de sus abstracciones más importantes de forma resumida. Podemos observar como las abstracciones de proceso y de fichero son las dos más comunes en el diseño de SO a lo largo de su historia. Se han mantenido abstracciones muy cercanas al hardware de los computadores y pocos han tratado de introducir abstracciones más cercanas a las metodologías y paradigmas más modernos. La abstracción de objeto ha entrado tímidamente en algunos SO, mientras que otros empiezan a prestar atención a un factor importante como es la seguridad e introducen mecanismos dentro del propio sistema para potenciar esta vertiente. Es el caso de los manifiestos de Singularity. Uno de los sistemas que más rompen con el conjunto clásico de abstracciones es XtreemOS, que apuesta firmemente por el servicio y las organizaciones virtuales. Sin embargo, el gran problema de esta aproximación es que se basa en otro sistema operativo (Linux) por lo que arrastra sus abstracciones (proceso, fichero,...) y tan solo introduce capas entre el usuario y el operativo. Estas capas, si bien otorgan la funcionalidad que se demanda para este tipo de SO grid, introducen una sobrecarga innecesaria que sería fácilmente evitable introduciendo esas nuevas y potentes abstracciones como parte intrínseca del propio XtreemOS.

S.O.	Año	Tipo	Abstracciones	Referencias
UNIX	1973	Prop. general	Proceso Fichero Tubería	[RT73, Rit78]
Mach	1986	Microkernel	Tarea Hilo Puerto Mensaje	[RJO+89] [ABB+86]
ExOS	1995	Exokernel	Ninguna	[Eng98, EKO95]
NanOS	1998	Microkernel	Objetos Agentes Tareas	[MiEBA98, Esc01]
Amoeba	1981	Distribuido	Proceso Mensaje Puerto	[TM81, MvRTa90]
Clouds	1991	Distribuido	Objetos Threads Mensajes	[DLAR91]
Scout	1995	Empotrado	Path	[MMO+95]
Plan 9	1995	Distribuido Híbrido	Fichero	[PPTT95]
Paramecium	2001	Extensible	Objetos	[vDHT95, Doo01]
MINIX 3	2006	Microkernel	Proceso Fichero Mensaje	[HBG+06]
Linux	1991	Prop. general	Proceso Fichero Usuario	[BC05, Rus98]

Windows NT	1993	Prop. general	objetos (procesos, hilos y sync)	[SSS98]
Singularity	2003	Experimental	SIP[2] Manifiestos	[HLTW05] [HLA+05]
XtreemOS	2006	Grid	VO[3] Fichero Servicio	[CFJK08, JMM07]

Tabla 5.1: Resumen de abstracciones principales de los SO

5.5. Discusión

La mayor innovación en el campo de los sistemas operativos en los últimos tiempos ha sido probablemente la introducción y expansión de la red. El salto de la computación centralizada a una computación distribuida a lo largo de los computadores en la red, comúnmente llamada *nube*, ha hecho emerger la necesidad de hacer un completo rediseño de los sistemas operativos tal cual los conocemos para adaptarse a esta tecnología.

La funcionalidad demandada hoy en día a un sistema operativo ha cambiado por completo. Factores como la capacidad multi-plataforma, soporte multi-procesador, o la concurrencia no suponen retos tecnológicos hoy en día, como hemos comentado previamente, y se encuentran en la vasta mayoría de sistemas desarrollados en los últimos años. Sin embargo aquello que se exige hoy en día a un sistema operativo es que esté plenamente integrado en la red (acceso a servicios, transparencia,

[2]Software Isolated Processes
[3]Virtual Organizations

distribuido,...) y además ofrezca garantías para mover con seguridad e integridad la información en dicha red.

Además, las tendencias en el diseño de SO se han dirigido hacia la reducción progresiva de los núcleos, encapsulando la funcionalidad crítica del sistema en ellos y dejando el resto en componentes externos. La expresión última de esta tendencia son los exokernels, aunque la opción más extendida e implementada es la del microkernel. En definitiva, contestando a la pregunta *¿Han alcanzado los sistemas operativos su mínima expresión de funcionalidad?*, podemos concluir que, dado el hardware con el que actualmente se trabaja, *probablemente* han alcanzado su mínima expresión. Luego, *¿En qué dirección es interesante continuar en el diseño de sistemas operativos?*. La propuesta de este trabajo es centrarse en uno de los retos actuales de la computación más extensos e influyentes y que no está plenamente resuelto: la presencia en la red, aprovechando los nuevos paradigmas de computación que ésta nos abre, como la orientación a servicios. Para ello, la propuesta de este trabajo está orientada a aumentar el nivel de abstracción proporcionado por el sistema operativo y sus servicios, integrándolos más en la red (algo para lo que no estuvo preparado cuando esto apareció) y sacando provecho de los paradigmas asociados.

Otro aspecto influyente en todo este proceso es la sofisticación del software creado hoy en día. Como consecuencia, la complejidad del software creado se ha visto incrementado de forma exponencial. Esto ha hecho que los paradigmas clásicos de computación no sean suficientes para crear este tipo de software tan complejo en un tiempo razonable y libre de errores. Nuevos paradigmas han sido desarrollados para facilitar la creación de software y su uso. Cabe destacar el paradigma de los *sistemas multi-agente*, que proporciona abstracciones inteligentes para llevar a cabo las tareas exigidas tanto de forma colaborativa como

individual y con una fuerte presencia en la red y una sintonización alta con paradigmas orientados a la red como los *servicios*. En este trabajo presentaremos el resultado de diseñar un nuevo sistema operativo utilizando un paradigma definido a partir de estas tendencias, el paradigma de computación basada en objetivos.

Parte III

Propuesta

6

Análisis de un Sistema Operativo orientado a Agentes

Introducir los conceptos de la tecnología de Sistemas Multi-Agente como nuevas abstracciones del Sistema Operativo nos introduce una serie de interesantes avances en el campo de los Sistemas Operativos. Los agentes son entidades reactivas, proactivas, autónomas y sociales. Todas estas características dotarían a las entidades computacionales de

nuestro sistema de una alta flexibilidad y dinamicidad. Teniendo la tecnología de agentes como parte de las entidades computacionales de nuestro sistema podemos dotarles de esa autonomía, sociabilidad y proactividad, características completamente ausentes en la abstracción actual de proceso.

Al introducir entidades computacionales autónomas podemos tener en el sistema aplicaciones que llevan a cabo ellas solas sus objetivos sin necesidad de interacción externa. Este tipo de entidad computacional, a diferencia del proceso, no es un contador de programa que ejecuta instrucciones de forma secuencial, sino que busca la forma de cumplir sus objetivos basándose en las tareas que es capaz de ejecutar.

Dado que un agente es una entidad social, es capaz de comunicarse con su entorno con el fin de buscar la mejor solución a su problema. Tiene la capacidad de colaborar con los agentes de su entorno tanto para demandar un servicio como para ofrecer sus servicios a otros agentes.

Finalmente, una ventaja de introducir el concepto de agente como entidad computacional del Sistema Operativo es la posibilidad de tener entidades reactivas y proactivas. De este modo, las entidades computacionales del sistema no sólo ejecutan código secuencial sin mirar su entorno, sino que son capaces de reaccionar a su entorno e incluso de tomar la iniciativa y realizar determinadas acciones sin necesidad de interacción externa.

Con todas estas características podemos diseñar un Sistema Operativo donde la entidad computacional sea el agente, una abstracción de alto nivel que dota de gran potencia a la construcción de aplicaciones mediante el uso del paradigma de los agentes. Sin embargo, en el paradigma de los agentes se trabaja con Sistemas Multi-Agente donde un conjunto de agentes colaboran para conseguir un fin. Paralelamente, en

los últimos años se ha estado desarrollando el concepto de *organización de agentes*. Una organización de agentes es una agrupación de agentes con una misión común, una topología definida, un conjunto de roles y un conjunto de normas y reglas a cumplir por parte de los miembros de la organización. Estas organizaciones, basadas en la Teoría de Organizaciones Humanas [AJB05], permiten la implementación de sistemas abiertos y heterogéneos donde agentes de diferente origen y formación pueden *organizarse* para cumplir un objetivo común.

La inclusión de una metodología de agentes orientada a organizaciones [Arg05, Arg08, AJB06] permite la modelización de entidades computacionales mucho más completas, basadas en componentes formados por agentes abstractos que interactúan mediante el intercambio de mensajes y el uso de servicios y con una fuerte arquitectura de seguridad basada en roles, normas y reglas, como veremos en próximas secciones.

Con este nuevo planteamiento de sistema operativo presentamos en este trabajo una evolución en la construcción de sistemas operativos donde modelamos las aplicaciones del sistema como *organizaciones de agentes*. Dichas organizaciones de agentes forman un sistema modular donde cada agente puede entrar a formar parte dinámicamente de cualquier organización del sistema, colaborando con la consecución de los objetivos de las organizaciones a las que pertenece y con sus propios objetivos. Del mismo modo, el propio Sistema Operativo es modelado como una organización de agentes que ofrece servicios al resto de organizaciones que se ejecutan en el sistema. Algunos de estos servicios son: la gestión de su ciclo de vida, acceso a servicios del sistema, registro de nuevos servicios en el sistema, acceso a los recursos, etc.

Un valor añadido que obtenemos al modelar nuestro sistema operativo utilizando Organizaciones de Agentes es la construcción de un

sistema distribuido abierto. Al ser nuestro sistema operativo un gestor de organizaciones virtuales y las aplicaciones de nuestro sistema organizaciones de agentes nos abrimos a nuevas posibilidades como el poder ejecutar nuestras aplicaciones en maquinas distribuidas eligiendo de forma dinámica los componentes que mejor se adaptan a los objetivos que intenta cumplir nuestra organización.

Con este modelo de ejecución basado en organizaciones de agentes se hace sencillo encontrar aplicaciones que subcontratan sus operaciones de cálculo a una organización externa de entre un conjunto de opciones que se ofrecen a realizar un determinado servicio. La organización es capaz de elegir qué servicio utilizar de entre las opciones ofrecidas en función de la precisión elegida, el coste del servicio, u otros parámetros que considere oportunos. Con este modelo de ejecución basado en organizaciones de agentes extendemos el modelo de computación un poco mas allá del *cloud-computing*, puesto que el sistema es capaz de forma autónoma de elegir qué componentes incluir en su organización para llevar a cabo sus objetivos. Este trabajo puede verse publicado en [PNA+12].

6.1. Misión de la Organización

Para realizar el modelado de nuestro Sistema Operativo vamos a utilizar como apoyo una metodología de modelado de organizaciones de agentes llamada *GORMAS* (Guidelines for Organizational-based Multi-Agent Systems) [Arg08]. Dicha metodología proporciona diversos meta-modelos y vistas que pueden ayudar a modelar nuestro sistema basándonos en la tecnología de agentes orientados a organizaciones.

El principal obstáculo a la hora de diseñar un sistema operativo

de estas características que a su vez de soporte para ejecutar organizaciones de agentes y donde se puedan registrar servicios abiertos es que tenemos a la vez una organización de agentes que es en si misma la plataforma donde se ejecutarán otras organizaciones de agentes. En definitiva lo que estamos haciendo es elevar la plataforma de agentes al entorno del sistema operativo, dado que al integrar el middleware de agentes en el sistema podemos ofrecer de forma más óptima todas las características comentadas en anteriores secciones.

6.2. Objetivos

Para modelar nuestro sistema operativo vamos a definir los objetivos globales del sistema. Los objetivos que deberá perseguir la organización que compone el sistema operativo (recordemos que es a su vez la plataforma de agentes que da soporte a las organizaciones) son los correspondientes a un Sistema Operativo clásico junto a los de una plataforma de agentes basada en organizaciones. En definitiva, podemos definir la *Misión* de nuestro sistema como un conjunto de objetivos:

1. Ofrecer una capa de abstracción sencilla de los recursos hardware
2. Proteger el acceso a los recursos hardware
3. Maximizar el uso de los recursos gestionados por el sistema (hardware y software)
4. Ofrecer un soporte de ejecución y de gestión del ciclo de vida de las organizaciones y agentes que se ejecutan en el sistema
5. Dar soporte al registro, búsqueda e invocación de servicios abiertos en el sistema

6.3. Servicios

Para cumplir esta serie de objetivos abstractos (que finalmente se descomponen en Objetivos Funcionales y Objetivos Operativos) es necesario que el sistema ofrezca una serie de servicios que permitan llevar a cabo la misión del sistema. En este trabajo vamos a agrupar los servicios que ofrece el sistema en tres bloques de meta-servicios diferenciados:

Gestión de Organizaciones: Este grupo de servicios ofrecerán la funcionalidad básica para la creación de organizaciones en el sistema, gestión de su ciclo de vida, entrada a una organización, salida de una organización, topología de la organización, gestión de normas y reglas, etc.

Gestión de Servicios Abiertos: Bajo el paradigma que estamos trabajando hemos modelado un sistema con orientación a servicios [MM07], donde cada organización consume servicios para cumplir sus objetivos correctamente u ofrece servicios al resto de organizaciones del sistema (o de otros sistemas, dado que es abierto y distribuido). Hemos llamado a estos servicios abiertos ***operaciones***. Para llevar a cabo este objetivo es necesario que el sistema ofrezca los servicios básicos de registro, desregistro y búsqueda de estas operaciones. Dichos servicios podrán ser registrados y utilizados por cualquier entidad computacional que interactúe con el sistema.

Gestión de Recursos y Operaciones: A parte de ofrecer un soporte para gestionar las entidades computacionales y los servicios básicos para registrar las operaciones en el sistema es necesario ofrecer los servicios básicos para invocar a dichas operaciones. Si ade-

mas estas operaciones necesitan acceder a los recursos hardware del sistema será necesario ofrecer una capa de acceso mínima que proteja dicho acceso. Este conjunto de servicios queda englobado en el meta-servicio de Gestión de Recursos y Operaciones, puesto que de el se descomponen los servicios que ofrecerán las organizaciones del sistema que tengan como objetivo ofrecer operaciones o recursos al resto de organizaciones del sistema.

6.4. Actores y Roles

Dado que nuestro sistema operativo se fundamenta en sistemas orientados a servicios (donde estos servicios ofrecidos en el sistema se componen de operaciones y recursos) únicamente modelamos dos tipos de actores que interaccionan con el sistema: los clientes y los servidores. Podemos observar las interacciones de los actores con el sistema en la Figura 6.1.

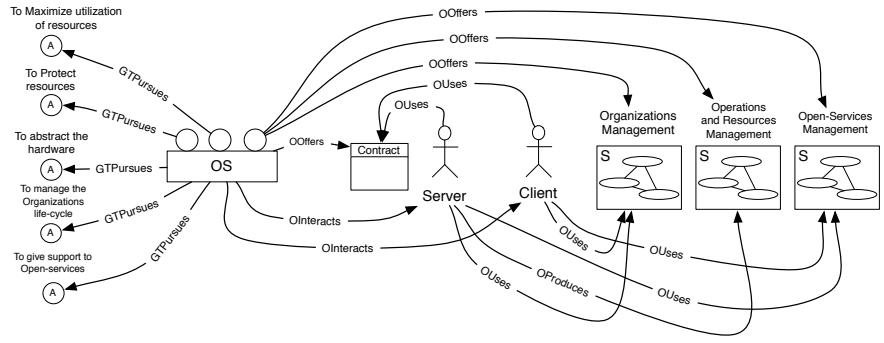


Figura 6.1: Modelo de Organización. Vista Funcional. Misión.

Los actores *Cliente* interactúan con el Sistema Operativo para consumir servicios del sistema, tanto recursos como operaciones. Dichos actores son agentes abstractos (de momento no los modelaremos como agentes u organizaciones) y su única finalidad en el sistema es cumplir sus objetivos haciendo uso para ello de los recursos y operaciones ofrecidos en el sistema.

Nuestro sistema es un sistema abierto. Esto quiere decir que no únicamente se ofrecen y consumen los servicios proporcionados por la propia plataforma o Sistema Operativo, sino que cualquier entidad computacional puede entrar en el sistema (siguiendo siempre una serie de políticas de seguridad) y ofrecer sus servicios al resto de entidades del sistema. Este tipo de actor será conocido como *Servidor*.

La finalidad de los actores *Servidor* es la de ofrecer los diferentes servicios que el sistema puede albergar: recursos y operaciones. Los recursos son servicios de acceso al hardware o relacionados con el mismo, por lo que harán falta una serie de privilegios especiales. Las operaciones son servicios software que cualquier agente puede ofrecer, por lo que no serán necesarios tantos privilegios.

Para gestionar las políticas de seguridad y los privilegios que hacen falta para realizar determinadas acciones en este trabajo se propone la creación de un *sistema de roles* dentro del Sistema Operativo. Mediante un sistema de roles correctamente definido los agentes del sistema pueden adoptar diferentes roles de forma dinámica, quedando así definido qué acciones tiene permiso para realizar y cuales no. Un agente puede asumir diferentes roles dentro del sistema a un mismo tiempo con lo que, por ejemplo, puede ejercer de servidor de recursos y cliente de operaciones a un mismo tiempo, mientras es ademas miembro de una organización.

Los roles definidos en nuestro sistema, conforme puede verse en la

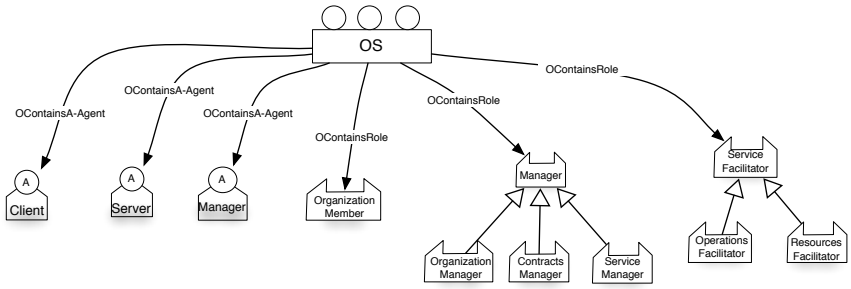


Figura 6.2: Modelo de Organización. Vista Estructural.

Figura 6.2, son los siguientes:

Miembro de una organización: Este es el rol mas básico que asume todo agente que entra en el sistema. Cada agente del sistema pertenece al menos a una organización (aunque sea una organización con estructura plana de la que el es el único miembro), con lo que adquiere al asumir este rol los diferentes privilegios o obligaciones que tiene todo agente en el sistema. Entre otros obtiene permisos para invocar determinados servicios de gestión de organizaciones (como solicitar la entrada a una organización, obtener los objetivos de la misma o salir de la organización cuando ha terminado su tarea). Este rol se especializará en otros más específicos como *propietario de la organización, gestor de la organización o invitado de la organización*, entre otros. Este rol también habilita la posibilidad de invocar servicios en el sistema.

Facilitador de servicios: Este rol es asumido por todo agente que desee ofrecer un servicio en el sistema. Como hemos visto anteriormente los servicios ofrecidos en el sistema pueden ser *recursos* y *operaciones*. Dado que cada tipo de servicio implica unos privilegios

muy diferentes (no es lo mismo ofrecer un servicio software que un acceso a un dispositivo hardware) este rol se especializa en dos diferentes: **Facilitador de recursos** y **Facilitador de operaciones**.

Gestores: En el sistema existirán una serie de agentes internos encargados de gestionar las diferentes tareas críticas del sistema. Estos agentes serán los *Gestores* del sistema y para ellos existen una serie de roles específicos (que no puede asumir un agente común del sistema) que les habilitan en determinadas tareas. El rol de *Gestor* se especializa en los siguientes:

Gestor de Organizaciones: El gestor de organizaciones es el encargado de velar por el ciclo de vida de las organizaciones del sistema. Este rol permite ofrecer los servicios relacionados con la gestión de organizaciones en el sistema (creación, acceso, mantenimiento, etc) y es por tanto una pieza fundamental del Sistema Operativo.

Gestor de Servicios: Este rol es el encargado de ofrecer el acceso a registro, búsqueda y desregistro de servicios en el sistema. El agente que ofrece este rol asume la responsabilidad de ofrecer la gestión básica de servicios para que el resto de agentes del sistema sea capaz de ofrecer sus propios servicios e invocar los de otros agentes.

Gestor de Contratos: Un producto adicional que ofrece el sistema para incrementar la confiabilidad y estabilidad del sistema son los *contratos* o *compromisos*. Mediante el uso de estos contratos, generados y gestionados por el rol de gestor de contratos, es posible determinar de antemano mediante negociación previa los parámetros bajo los cuales se va a llevar a cabo determinada interacción en el sistema. Por ejemplo se

generara un contrato antes de entrar a una organización, o para ofrecer un servicio en el sistema o incluso se puede exigir un contrato para consumir un recurso del sistema. De este modo se puede controlar con mucha mas precisión las acciones de los agentes dentro del sistema. Estos contratos pueden definir qué interacciones están permitidas, qué recursos puede acceder el agente o incluso un tiempo límite para cumplir sus objetivos dentro de una organización, entre otros ejemplos.

En la Figura 6.3 observamos la Vista Funcional Interna del modelo de organización donde podemos observar cómo se relacionan los roles del sistema con los diferentes agentes abstractos que participan en el sistema: clientes, servidores y gestores. Cada agente puede ser a su vez cliente, servidor o gestor, por lo que puede asumir cualquiera de los roles presentados en función de la tarea o tareas que vaya a llevar a cabo.

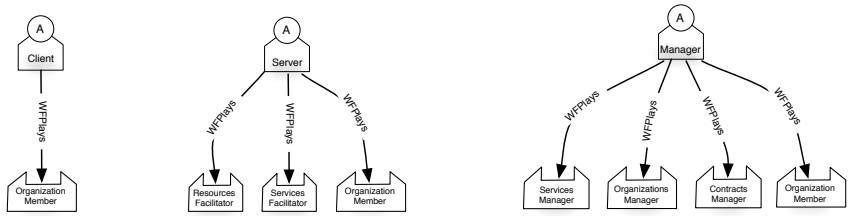


Figura 6.3: Modelo de Organización. Vista Funcional Interna.

Para tener una visión global del sistema modelado hemos utilizado el diagrama de Funcionalidad Externa del Modelo de Organización

de la metodología GORMAS. En dicho diagrama (ver Figura 6.4) podemos ver las relaciones entre los servicios que el Sistema Operativo proporciona (mediante la relación `OProvides`)y los roles que asumen los agentes para proporcionar o utilizar dichos servicios. Como podemos ver en la figura el rol de *Organization Member* es el rol mas básico que obtiene un agente y cuya funcionalidad provee la capacidad de invocar los servicios propios del sistema: Gestión de Organizaciones, Gestión de Servicios y Gestión de Recursos. Nótese que en este diagrama no aparece el servicio de operaciones puesto que las operaciones son funcionalidades software ofrecidas por los agentes del sistema. No corresponde al sistema operativo ofrecer las operaciones propiamente dichas, sino el soporte para que se puedan registrar y utilizar dichas operaciones (y que es ofrecido por un agente con el rol de Gestor de Servicios). La capacidad de ofrecer recursos y operaciones corresponde al rol de *Service Facilitator*. Dicho rol se especializa en los roles *Resources Facilitator* y *Operations Facilitator*. Como podemos observar, los agentes que asumen el rol de Resources Facilitator pueden proporcionar el servicio de acceso a recursos, que es parte del SO por la necesidad de ejecutar dichos servicios bajo unos parámetros mínimos de protección. Estos servicios mínimos de acceso a recursos hardware pueden a su vez ser especializados por agentes con el rol Resources Facilitator. Se puede observar que el rol de Operations Facilitator no proporciona inicialmente ningún servicio puesto que estos son los servicios abiertos (no pertenecientes al sistema operativo) que serán registrados de forma dinámica en el sistema.

Finalmente, las aplicaciones que se ejecutan en el sistema tienen la abstracción de organizaciones virtuales de agentes en un sistema orientado a servicios y dirigido por objetivos. Cada aplicación es una organización de agentes con una topología determinada (estructura plana, jerarquía, burocracia, etc) y con una serie de objetivos o misión global

a cumplir. Los agentes que pertenecen a una organización asumen la obligación de ayudar a conseguir la misión de la organización y pueden ser castigados o recompensados mediante algún sistema de penalizaciones y recompensas por los agentes manager del sistema operativo. Estos agentes internos con el rol manager son los encargados de velar por el cumplimiento de las normas de las organizaciones y de ofrecer recompensas o penalizaciones (como un privilegio en la planificación del sistema o la negación a acceder a un recurso por un tiempo determinado).

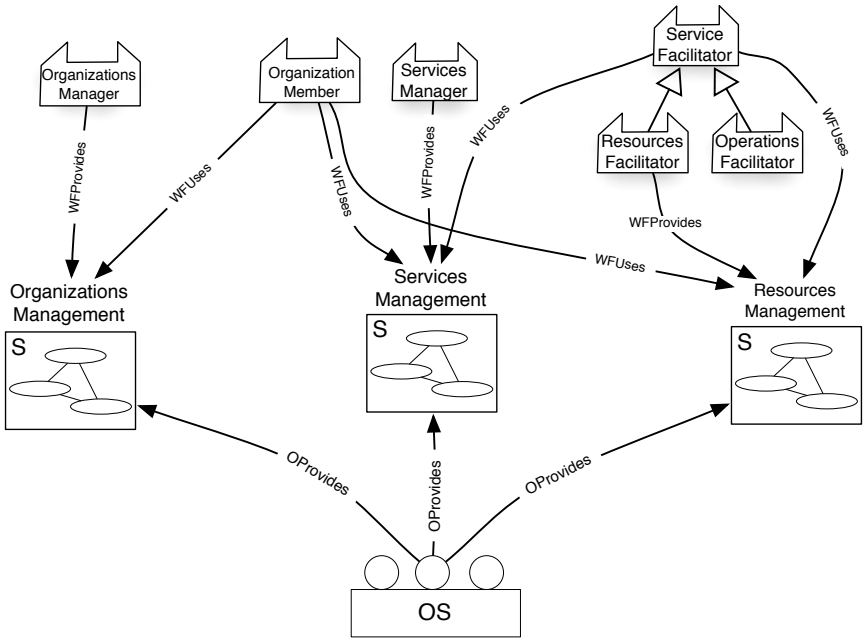


Figura 6.4: Modelo de Organización. Vista Funcional Externa.

Cada organización puede comunicarse con cualquier otra organi-

zación del sistema (o incluso externa) que desee para ofrecer un servicio (operación o recurso) o negociar el uso de un servicio externo a ella. Nótese que en este sistema no es necesario diferenciar entre aplicaciones y bibliotecas dado que es un sistema orientado a servicios y cualquier aplicación puede registrar un nuevo servicio que ofrecer al sistema (y como es un sistema eminentemente distribuido debido a sus características intrínsecas podemos ofrecer servicios abiertos al resto del mundo externo a nuestro sistema) o consumir un servicio que desee previa negociación con el proveedor del servicio y el gestor de contratos. Unicamente es necesario haber adquirido previamente el rol de facilitador del servicio o de miembro de una organización para consumirlo.

Como ya hemos comentado previamente, la seguridad del sistema es controlada por los agentes internos del sistema con el rol de Gestores. Estos agentes se encargan de asignar roles a los agentes que lo demandan, ejercer de notarios en la firma de contratos, vigilar que los contratos se cumplan (así como las normas y reglas de las organizaciones) y tomar las medidas adecuadas cuando no se cumplan, penalizando o incluso expulsando del sistema al implicado.

6.5. Consideraciones finales

El objetivo de este capítulo ha sido estudiar cómo la tecnología que ha sido desarrollada para crear sistemas multi-agente durante los últimos años puede ser utilizada como una solución factible para el diseño de sistemas operativos. Esta tecnología ha demostrado ser suficientemente robusta y flexible para crear sistemas complejos, donde se encuentran capacidades como la comunicación entre entidades del sistema y la deliberación sobre su entorno.

Al proporcionar al SO de abstracciones más avanzadas abrimos la posibilidad de crear aplicaciones más complejas utilizando el soporte proporcionado por el mismo sistema operativo, sin la necesidad de *middleware* intermedio que introduzca una sobrecarga innecesaria en el sistema, tal y como presentamos en las conclusiones del Capítulo 2.

Como resultado de este trabajo se han identificado algunas abstracciones de alto nivel para el SO, como son las organizaciones de agentes, los roles, los contratos o los objetivos, de los cuales hicimos un análisis en el Capítulo 3. Estas abstracciones nos han proporcionado un estudio inicial sobre los elementos que pueden proporcionar nuevas tecnologías en el diseño de sistemas operativos que mejoren aspectos de los mismos como la seguridad, la eficiencia o los modelos de ejecución. Por ejemplo, al introducir tecnologías del acuerdo en las comunicaciones entre entidades del SO podemos mejorar la seguridad proporcionada por el propio sistema operativo. Esto se debe a que es posible utilizar abstracciones como el contrato y la negociación que permitan el control de los derechos y obligaciones que una entidad computacional del sistema debe observar. Esta mejora en el nivel de las abstracciones propuestas por un sistema operativo nos permitirá afrontar los nuevos retos en el diseño de sistemas operativos presentados en el estudio del Capítulo 5.

Mediante éste análisis hemos identificado al SO como un sistema orientado a servicios con entidades computacionales complejas mucho más potentes que la abstracción clásica de proceso, y donde la abstracción de objetivo y el entorno distribuido son necesarias para la interacción entre entidades y que éstas puedan producir y consumir dichos servicios.

A partir de este análisis hemos propuesto un paradigma de computación distribuido y orientado a objetivos que será utilizado para la

propuesta de sistema operativo orientado a objetivos. En este paradigma nos hemos centrado principalmente en el modelo de ejecución del sistema, abarcando cómo los agentes utilizan sus objetivos para producir y consumir servicios que satisfagan sus deseos. Aspectos complementarios como la seguridad o la organización de los agentes en organizaciones virtuales quedarán propuestos para un trabajo futuro.

7

Computación Distribuida basada en Objetivos

En los capítulos previos se ha mostrado cómo la computación basada en objetivos cubre un espacio dentro del cuadrante de paradigmas de computación Ejecutivo-Consecuente, donde el eje Consecuente indica la expresión de aquello que se desea conseguir y el eje Ejecutivo indica la ejecución de acciones para la satisfacción del objetivo, tal y como vimos en el Capítulo 2. Además, al realizar el modelado de un sistema operativo en el capítulo anterior utilizando tecnología de agentes, ha emergido la necesidad de realizar una definición más formal, donde este paradigma situado en el cuadrante Ejecutivo-Consecuente

esté basado en servicios y se soporte sobre un sistema operativo basado en agentes, como presentamos en nuestra publicación [PBGF09]. A esta propuesta la llamamos Computación Distribuida basada en Objetivos.

7.1. Definición

En este trabajo proponemos una definición de Computación Distribuida basada en Objetivos (DGOC) como el paradigma donde una serie de agentes heterogéneos expresan sus deseos mediante el uso de objetivos. Dichos agentes son además capaces de cumplir sus objetivos mediante el uso de la composición automática de servicios. Estos servicios son proporcionados a su vez por agentes que se encuentran en el mismo sistema o distribuidos en la *nube*.

El paradigma propuesto en este trabajo aporta un nuevo método de presentar y solucionar problemas en computación, donde la integración con la red es absoluta y la autonomía del sistema permite al usuario no preocuparse de cómo solventar un problema, sino de qué problema desea solventar.

El paradigma aquí presentado incorpora las abstracciones de *agente*, *base de conocimiento*, *servicios*, *objetivos* y *planes* (composiciones de servicios) [dSP05], algunos de los cuales son tomados del modelo de agentes BDI [RG95]. El modelo que implementa este paradigma tiene como propósito definir un nuevo soporte de ejecución sobre un núcleo de sistema operativo cuya entidad de ejecución principal es el servicio. Los agentes son gestionados por el SO, que ejecuta los servicios en función de los objetivos expresados por los agentes. En este trabajo nos

centramos en el modelo de ejecución del sistema operativo.

En primer lugar definimos un agente a_i mediante la siguiente tupla:

$$a_i = \langle KB_i, S_i, P_i, G_i \rangle \tag{7.1}$$

Donde:

- KB_i representa la Base de Conocimiento del agente a_i. La Base de Conocimiento almacena la información sobre los estados del agente. Estos estados son aquellos hechos que el agente estima que son ciertos, por ejemplo su propia representación del entorno. Un ejemplo de un posible ítem de conocimiento almacenado en la KB sería, después de haber guardado una canción en un reproductor de música, el ítem «*Song in iPod*».

- S_i representa el conjunto de servicios ofrecidos por el agente. Estos servicios son utilizados por el agente para conseguir sus propios objetivos, pero también pueden ser ofertados a otros agentes para ayudar a la consecución de sus propios objetivos. Los servicios se describen mediante el modelo OWL-S como se verá en la sección 7.2. Parte importante de un servicio serán por tanto sus precondiciones y sus postcondiciones. Siguiendo con el ejemplo de guardar una canción en un reproductor de música conectado a un puerto USB, algunos de los servicios proporcionados por los agentes del entorno podrían ser «*Entropy Encoding*» (para realizar la codificación de una canción PCM en formato MP3), o «*Validate Song in iPod*» (para verificar que la canción ha sido correctamente copiada al reproductor de música). El SO puede proporcionar además algunos servicios propios dado que en determinadas circunstancias se puede comportar como un agente con el fin de interactuar con otros agentes. Un ejemplo de servicio ofrecido por

el SO sería «*USB Write*». En el ejemplo mencionado este servicio se encargaría de enviar el fichero en formato MP3 como un flujo de datos al reproductor móvil de música. De manera más general, este sería un servicio que actuaría como parte del controlador USB del SO.

- P_i representa un conjunto de planes pre-compilados proporcionados por el agente para la consecución de sus objetivos. Un plan es una secuencia ordenada de servicios donde la postcondición del servicio j ($Post(s_j)$) es igual a la precondición del servicio $j+1$ ($Pre(s_{j+1})$), donde $s_j, s_{j+1} \in S$, siendo $S = \bigcup_{a_i} S_i$ el dominio del conjunto de servicios ofrecidos por todos los agentes del sistema. De este modo, podemos definir el conjunto de planes pre-compilados como $P_i \subseteq P$, donde P representa al dominio del conjunto total de posibles planes, esto es, el conjunto de todas las secuencias de servicios posibles:

$$P = \{(s_1..s_n / \forall i \in 1..n, s_i \in S \wedge \\ \forall j \in 1..n-1, Pre(s_{j+1}) = Post(s_j))\}$$

Los planes pre-compilados existen con el objetivo de optimizar el proceso de componer nuevos planes en tiempo de ejecución. Para ello, cada uno de los planes candidatos a ser pre-compilados son creados con anterioridad en la fase de diseño del agente. Esta tarea es especialmente útil para aquellos planes más críticos del sistema, dado que son un tipo de planes que no suelen cambiar mucho con el paso del tiempo. También es útil para aquellos planes ejecutados con cierta frecuencia o donde la eficiencia es vital. Un ejemplo de este tipo de planes que deben permanecer inalterados sería el plan proporcionado por el SO para dar de alta un

nuevo usuario en el sistema. Ésta es una tarea crítica con un impacto importante en el protocolo de seguridad del sistema, por lo que debe ser controlada firmemente en la fase de diseño del SO para evitar problemas de seguridad.

- G_i representa el conjunto de objetivos que el agente desea conseguir. Cuando un objetivo es alcanzado, el agente lo marca como una entrada en su Base de Conocimiento, lo que significa que el agente cree que ese hecho es cierto. En el ejemplo del reproductor de música, el objetivo que lanzó el plan «*Save Song to iPod*» sería «*Song in iPod*». Por lo tanto, cuando el plan finaliza con éxito el agente escribirá en su Base de Conocimiento el hecho «*Song in iPod*», lo que representa que el objetivo ha sido conseguido. De este modo podemos representar los objetivos de la forma habitual en sistemas orientados a objetivos, como estados que el agente desea alcanzar en su KB, y por lo tanto cuando el agente cumple el objetivo significa que él cree que la representación del objetivo tiene valor verdadero.

Una vez presentada la definición de agente podemos presentar la definición del paradigma de Computación Distribuida basada en Objetivos:

$$DGOC = \langle A, \gamma_g, \kappa_p, \delta_p \rangle \tag{7.2}$$

Donde:

- A representa al conjunto de agentes presentes en el sistema: $A = \{a_i\}$.
- $\gamma_g : 2^A \rightarrow G$ es la función de selección de objetivos, donde $G = \bigcup_{a_i} G_i$ representa el dominio del conjunto de objetivos de todos

los agentes del sistema y $G_i = \{g_{ij}\}$ representa el conjunto de objetivos del agente a_i, por lo que g_{ij} hace referencia al objetivo j-ésimo del agente a_i.

- $\kappa_p : A \times 2^S \rightarrow 2^P$ es la función de composición de nuevos planes. Esta función es utilizada cuando no se ha encontrado ningún plan en el conjunto de planes pre-compilados del agente. La función de composición κ_p crea las posibles composiciones de servicios que pueden ser alcanzadas desde la base de conocimiento KB_i del agente a_i hasta el objetivo g_{ij}:

$$\begin{aligned}\kappa_p(g_{ij}, a_i) = &\{(s_1..s_n / \forall i \in 1..n, s_i \in S \wedge \\ &\forall j \in 1..n-1, Pre(s_{j+1}) = Post(s_j)) \\ &\wedge Pre(s_1) \in KB_i \wedge (Post(s_n) = g_{ij})\}\end{aligned}$$

- $\delta_p : G \times 2^P \rightarrow P$ es la función de selección de planes. Esta función selecciona un plan para ser ejecutado y tratar de cumplir el objetivo activo. Para ello, la función δ_p tiene en cuenta el conjunto de planes pre-compilados P_i y el conjunto de planes compuestos mediante la función $\kappa_p(g_{ij}, a_i)$ y selecciona un plan válido, esto es, un plan cuyas precondiciones se cumplan en la base de conocimiento del agente que activa el objetivo y cuya postcondición coincida con el objetivo a cumplir. La invocación de esta función será: $\delta_p(g_i, \{p_1..p_n\} \cup \kappa_p(g_i, a_j))$

En el Algoritmo 2 se muestra el intérprete de DGOC en cada periodo de activación. Inicialmente se extrae un nuevo objetivo del conjunto de posibles objetivos mediante la función de selección γ_g. A continuación se comprueba si dicho objetivo es alcanzable y es consistente con el resto de objetivos activados, es decir, que no entre en conflicto con alguno de ellos. Una vez seleccionado el objetivo se utiliza la función de

selección δ_p para encontrar un plan que cumpla el objetivo. A continuación se manda a ejecución el plan. En caso de fallo se seleccionará un nuevo plan. Finalmente se comprueba si el plan ha finalizado con éxito comprobando su postcondición. En caso afirmativo se marca el objetivo como alcanzado. Para cualquier caso no controlado se marcará el objetivo como no-alcanzable.

Algoritmo 2: Intérprete DGOC

```
1  repeat
2      g_i ← γ_g({a_1..a_n}) ;
3      if IsPossible(g_i) ∧ IsConsistent(g_i) then
4          p_i ← δ_p(g_i, {p_1..p_n}, κ_p(g_i, a_j));
5          while ¬ IsFinished(p_i) do
6              Execute(p_i);
7              if HasFailed(p_i) then
8                  p_i ← δ_p(g_i, {p_1..p_n} ∪ κ_p(g_i, a_j));
9              end
10         end
11         if CheckPostCondition(p_i) == True then
12             GoalPursued(g_i);
13         else
14             GoalNotPursued(g_i);
15         end
16     else
17         GoalNotPursued(g_i);
18     end
19 until True ;
```

Este intérprete presenta una visión generalizada de cómo funcio-

na el paradigma DGOC. Se ha utilizado para esta definición elementos extraídos de los agentes BDI, como la función de selección de planes, o de las técnicas de planificación en Inteligencia Artificial y Planificación Basada en Casos para la composición de nuevos planes. Además se ha definido el elemento fundamental de ejecución, el servicio, utilizando las definiciones del ámbito de los servicios web e introduciendo de este modo los planes compuestos por servicios distribuidos. En el Capítulo 8 se muestra cómo se ha implementado este modelo en un sistema operativo. A continuación mostraremos la definición de servicio que utiliza DGOC. Finalmente mostraremos el lenguaje de objetivos utilizado para definir el conjunto G.

7.2. Servicios

Para describir los **Servicios** en este modelo de ejecución se ha utilizado la definición de servicio OWL-S [MBH$^+$04]. Un servicio OWL-S $s_i \in S_i$ se define con la tupla:

$$s_i = \langle SP, GR, PM \rangle \tag{7.3}$$

donde SP es el Perfil del Servicio, GR es el *Grounding* y PM es el Modelo de Proceso del servicio. El perfil del servicio define qué hace el servicio. El *grounding* define cómo interactuar con el servicio y el modelo de proceso define cómo se usa el servicio.

El modelo de proceso de un servicio OWL-S está formado por *procesos compuestos* y *procesos atómicos*. Un proceso compuesto es un conjunto de procesos atómicos (los cuales no tienen una estructura interna y se ejecutan en un único paso) con una estructura interna construida a base de procesos atómicos y compuestos y unas pocas estructuras de control de flujo (`sequence`, `if-then-else`, `choice`, etc).

Este tipo de servicio OWL-S es un estándar bien definido que proporciona a este paradigma suficiente potencia para construir toda la funcionalidad proporcionada por los agentes.

En nuestra propuesta los servicios que componen un plan son el verdadero elemento ejecutable de un plan. Un servicio s_i está además compuesto por una *precondición* P, una *postcondición* Q, y un conjunto de *entradas* y *salidas*. La postcondición es la entidad **Objetivo** que el agente pretende alcanzar. La precondición P es el **pre-requisito** necesario para la ejecución del servicio. La postcondición expresa además el **impacto** que tendrá la ejecución del servicio s_i. Ambos, P y Q, se definen en el *aspecto funcional* del perfil del servicio.

7.2.1. Descubrimiento de Servicios

El Descubrimiento de Servicios es utilizado para localizar servicios que resulten útiles para la composición de un nuevo plan, lo cual permite a los agentes la consecución de sus objetivos. De todas formas, el descubrimiento de servicios es una tarea muy complicada dado que el espacio de búsqueda puede llegar a ser considerablemente grande. El número potencial de servicios disponibles puede ser inmanejable, lo cual hace necesario el uso de protocolos de descubrimiento que permitan al sistema encontrar y catalogar de forma eficiente toda esta cantidad de servicios. Una tecnología que puede ser utilizada para ello son los servicios de directorio, donde los agentes pueden registrar sus servicios y buscar otros servicios de otros agentes. Estos directorios pueden estar federados o interconectados, creando redes de directorios de servicios donde es sencillo encontrar nuevos servicios útiles. Otra posible tecnología que se puede utilizar para el descubrimiento de servicios es la tecnología de publicación-subscripción, como por ejemplo el protocolo XMPP PubSub [MSAM99], que permite a los agentes suscribirse

únicamente a los perfiles de servicios en los que está interesado. Tecnologías como ZeroConf [Gut01] son muy útiles para descubrir servicios que se encuentran en la misma subred, aunque esto tiende a realizarse a costa de una gran sobrecarga de emisión de paquetes en la red (broadcast).

En nuestra propuesta, los planes pre-compilados se crean en la fase de diseño previa, lo cual implica que los servicios de estos planes no tienen que ser necesariamente descubiertos antes de ser invocados. Esto se debe a que este tipo de planes habitualmente no permiten el uso de servicios desconocidos (y por lo tanto poco fiables a priori) y además dan importancia a la optimización del rendimiento. Esto se hace para evitar un proceso de descubrimiento excesivamente largo.

El Descubrimiento de Servicios es un problema abierto [BCP05, PTD+06] en este trabajo dado que resulta una tarea extremadamente compleja que debe ser llevada a cabo en una de las fases de este paradigma. Expertos en servicios web han abierto interesantes líneas de investigación tratando de encontrar el mejor método para descubrir los servicios que mejor se adaptan a las necesidades de cada aplicación, pero sin introducir una gran sobrecarga en el proceso. En este trabajo se ha utilizado la aproximación de la tecnología ZeroConf [Gut01] dado que es lo bastante eficiente para descubrir servicios en redes locales. De todos modos, el paradigma soporta la introducción de modelos de descubrimiento de servicios más complejos y avanzados como XMPP PubSub [MSAM99].

7.3. Lenguaje de Objetivos

En este trabajo vamos a utilizar un lenguaje de objetivos que nos permita expresar los objetivos del usuario de una forma lo más expre-

siva posible. Necesitamos expresar las condiciones que se deben cumplir para considerar un objetivo cumplido, qué conflictos pueden existir con otros objetivos del sistema y qué requisitos temporales se deben cumplir para la ejecución del objetivo (como que no comience antes de un determinado momento o que finalice antes de una hora dada).

Para ello, hemos extendido el lenguaje de objetivos utilizado en la plataforma JadeX [BPL04] con las características temporales requeridas por nuestro sistema operativo.

A continuación presentamos la serie de características comunes que poseen los objetivos, como las condiciones de activación o de borrado, así como determinadas propiedades:

- Si un objetivo debe tener un invariante que se cumpla durante toda la ejecución del plan se puede expresar mediante una `<contextcondition>`, en caso de no cumplirse esta condición se parará la ejecución del plan.

- Si la condición determina la cancelación inmediata del objetivo será una `<dropcondition>`.

- Si la condición expresa una situación inicial que debe cumplirse para activar el objetivo será una `<creationcondition>`.

- Si el objetivo es persistente (debe mantenerse activo siempre) podrá tener una `<recurcondition>` que indique en qué casos se puede considerar la reactivación del objetivo.

- Si la condición expresada es *débil*, es decir, es deseable cumplirla aunque no obligatoria, se utilizará la etiqueta `<softcondition>`.

- Por último, <targetcondition> es la condición utilizada para expresar el estado que se desea alcanzar. En esta etiqueta se expresa en definitiva el núcleo semántico del objetivo.

Para añadir restricciones temporales a una condición de un objetivo hemos introducido los operadores begin y end que nos permiten indicar que dicha condición debe cumplirse antes o después de un determinado instante de tiempo indicado en un parámetro del operador. Por ejemplo la condición (end (?condition) ?time) indicaría que la condición ?condition debe cumplirse antes del instante ?time.

En una representación de objetivo se puede encontrar además el nodo <deliberation> donde se aporta información adicional que pueda ser útil en la selección y activación de objetivos. Esta información es utilizada para expresar interacciones entre objetivos como conflictos o prioridades. Para ello se utiliza el elemento <inhibits/>. Un objetivo puede tener además la etiqueta <unique/> para expresar que no se pueden adoptar dos instancias iguales de un objetivo al mismo tiempo.

Los objetivos pueden tener además una serie de propiedades. La propiedad retry indica que, en caso de fallar el plan seleccionado para cumplir un objetivo, se seguirá intentando encontrar un plan hasta alcanzar el objetivo. La propiedad retrydelay indica los milisegundos que se esperará hasta volver a tratar de alcanzar el objetivo. La etiqueta exclude se utiliza en conjunción con retry e indica en qué casos se podrá seleccionar un determinado plan cuando se está buscando uno nuevo para un objetivo fallido. Si la etiqueta exclude tiene el valor when_tried (por defecto) no se seleccionará ningún plan que ya haya sido ejecutado. Si el valor es when_succeeded no se elegirá ningún plan ejecutado con éxito previamente. El valor when_failed indica justo lo contrario (ningún plan fallido previamente). Por

último se puede configurar con el valor `never`, que evita que se excluya ningún plan en ningún caso. Estas propiedades son tenidas en cuenta cuando se está componiendo un nuevo plan. Finalmente, la propiedad `recur` indica si un objetivo es persistente, esto es, que debe mantenerse activo en el tiempo. Cuando un objetivo con la propiedad `recur` ha sido ejecutado, el sistema esperará `recurdelay` milisegundos y volverá a activarlo, reactivando todo el proceso deliberativo de dicho objetivo.

Los objetivos pueden ser de cuatro tipos diferentes: `achieve`, `perform`, `query` y `mantain`. Cada tipo de objetivo tiene diferente finalidad. El objetivo de tipo `achieve` es el modelo clásico de objetivo donde se expresa un estado a alcanzar utilizando el elemento `<target-condition>`. Un objetivo de tipo `perform` es utilizado para expresar una actividad que debe ser llevada a cabo, es decir, la ejecución de un servicio. Aunque en estos objetivos no es necesaria una `<target-condition>`, sí que pueden utilizarse otras condiciones como `<contextcondition>`, como `<creationcondition>`, etc. Un objetivo de tipo `query` trata de consultar el valor de un parámetro. Este objetivo tiene éxito cuando dicho parámetro contiene un valor en la base de conocimiento del agente. Por último, un objetivo de tipo `mantain` monitoriza un estado y trata de mantenerlo en un valor predeterminado. Para ello utiliza la etiqueta `<mantaincondition>`.

7.4. Consideraciones finales

El paradigma de la **Computación Distribuida basada en Objetivos** emerge de la necesidad de facilitar y automatizar la creación de software para responder a las necesidades del usuario haciendo uso de los nuevos retos tecnológicos como son la presencia en una nube

de servicios distribuidos en la red y los sistemas multi-agente. En este paradigma los agentes expresan sus deseos por medio de objetivos, los cuales son alcanzados gracias a la composición basada en casos de servicios distribuidos. Esta composición tiene como resultado un conjunto de planes que el agente, con ayuda del sistema operativo, puede ejecutar para cumplir su objetivo. Al ocurrir en un entorno distribuido será habitual que cada uno de los servicios requeridos se encuentre en localizaciones completamente diferentes, por lo que la comunicación entre entidades será un factor vital para este paradigma. Del mismo modo una característica deseable será la recuperación frente a errores, por lo que los agentes tendrán la capacidad de reparar los planes en ejecución si estos fallan. Para todo esto será necesario disponer de una arquitectura de sistema operativo que de soporte a esta funcionalidad, como será presentado en el Capítulo 8.

Esta arquitectura presta especial atención a la calidad de las respuestas obtenidas, utilizando como factores de calidad la confianza y el tiempo de respuesta de los servicios implicados. Para medir la confianza utiliza sistemas de razonamiento basados en casos, aprendiendo de casos anteriores para valorar que proveedor de servicio le inspira más confianza. Para medir el tiempo de respuesta es necesario que el sistema operativo tenga capacidades de tiempo real, puesto que el proveedor del servicio adquirirá un compromiso temporal con el cliente, dentro del cual se entregará el resultado. Para ello debe ser capaz de evaluar la carga del sistema y realizar una predicción temporal para la ejecución del servicio. Dicha predicción afectará a la calidad y fiabilidad del proveedor del servicio.

El paradigma tiene en cuenta además otros aspectos como:

- *reparación de planes*: cuando fallan durante su ejecución (un servicio del plan no está disponible o no cumple su postcondición)

- *compartición de planes*: los agentes pueden compartir el conocimiento que tienen sobre planes compuestos para mejorar la inteligencia colectiva, basándose en casos anteriores y en la confianza que se deposite sobre el agente
- *compromisos temporales*: los agentes (con ayuda del sistema operativo) establecen un compromiso temporal para el tiempo de respuesta del servicio. Este compromiso tiene la finalidad de poder elegir el servicio de mayor calidad (donde el tiempo de respuesta es uno de los factores de calidad) entre todas las opciones disponibles en la nube de servicios

Con este paradigma se ha realizado el diseño de una arquitectura de sistema operativo que da soporte a la computación distribuida basada en objetivos. Además, se ha desarrollado un entorno de simulación que implementa la arquitectura de sistema operativo donde se han desplegado las características del paradigma que requieren de la modificación del modelo de ejecución del sistema operativo. Estas características son principalmente las capacidades de tiempo real que permiten al sistema controlar la ejecución de los servicios dentro de unos parámetros temporales establecidos. Igualmente el sistema operativo se comportará como un agente más del sistema, pudiendo activar sus propios objetivos y ofreciendo sus servicios en la nube para que sean invocados (por ejemplo un sistema de ficheros distribuido).

8

Diseño de un Sistema Operativo orientado a Objetivos

En este capítulo se presenta el diseño de un sistema operativo basado en el paradigma de Computación Distribuida basada en Objetivos. Se mostrará la arquitectura general propuesta del modelo de ejecución del SO, desgranando los componentes elementales del mismo y su funcionamiento e interacción con el resto de componentes. Esta

arquitectura ha sido presentada y publicada en [PJGF11], [PNGFJ12] y [PNJGF12],

8.1. Una arquitectura orientada a la ejecución de objetivos

Dado que una composición de servicios OWL-S es un conjunto de servicios que incluyen tanto procesos atómicos y compuestos como estructuras de control, se define un *Plan* como un modelo de proceso compuesto por uno o más modelos de procesos compuestos (de nuevo, incluyendo servicios compuestos, servicios atómicos y estructuras de control). El Plan define la forma de lograr algunos resultados o post-condiciones uniendo los diferentes servicios OWL-S que se pueden conectar. Servicios compuestos o incluso servicios atómicos pueden ser vistos como planes muy simples, pero también definimos un plan como el resultado de unir los distintos servicios compuestos, con el fin de lograr un objetivo.

Para dar soporte al paradigma presentado, en este capítulo se desarrolla el diseño de la arquitectura de ejecución orientada a objetivos a partir del análisis realizado previamente. La arquitectura se compone de los siguientes componentes (Figura 8.1):

- **Runtime Engine**: El Runtime Engine toma los planes proporcionados por los planificadores on-line o por las librerías de planes pre-compilados y gestiona su ejecución transfiriendo la ejecución de los servicios que componen el plan al scheduler. Este componente es también capaz de invocar remotamente los servicios ofrecidos por otros agentes que se encuentran en otros nodos y

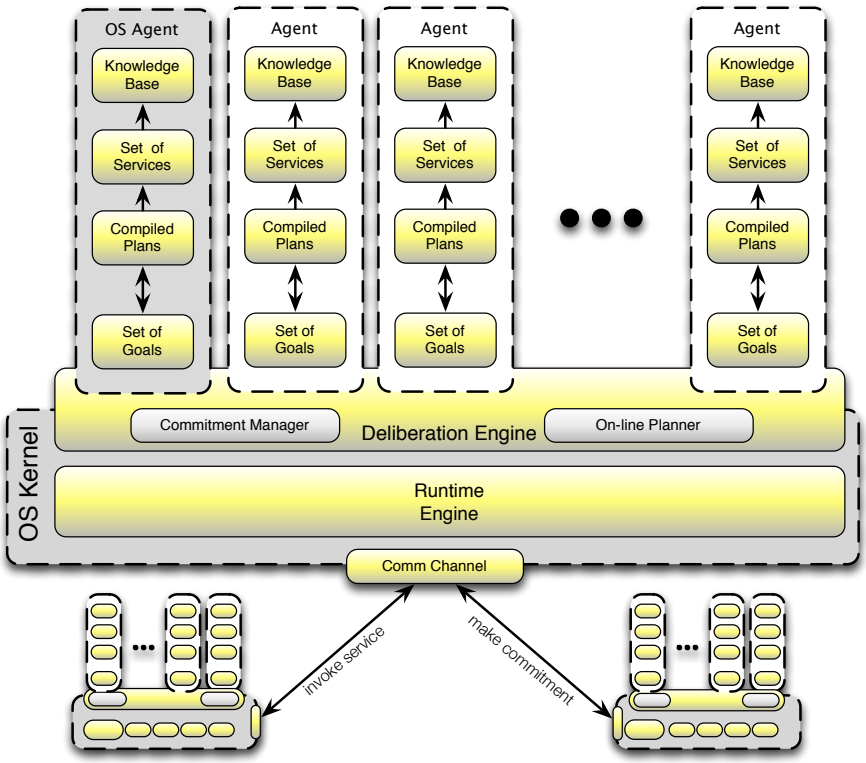


Figura 8.1: Componentes del módulo de ejecución y de los agentes

que fueron localizados mediante el protocolo de descubrimiento de servicios e incluidos en el plan que se encuentra en ejecución.

- **Deliberation Engine**: Es responsable de decidir cómo y en qué orden se ejecutan los planes. Este motor negocia con los agentes que prestan un servicio de un plan en ejecución. Este motor está permanentemente en ejecución en segundo plano evaluando los objetivos que los agentes quieren lograr y seleccionándolos para su ejecución. Este componente interactúa simultáneamente con el

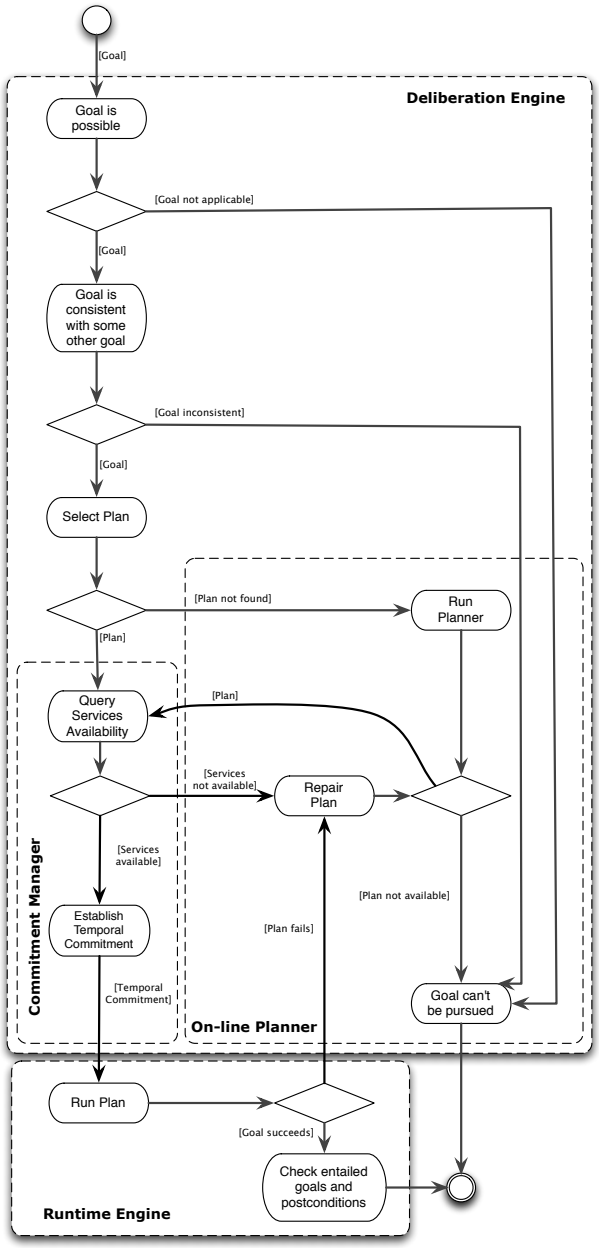


Figura 8.2: Diagrama de flujo del modelo de ejecución basado en objetivos

Runtime Engine, el Gestor de Compromisos (Commitment Manager) y el Planificador On-line.

- **Commitment Manager**: Los agentes proveedores de servicios negocian con el Gestor de Compromisos su disponibilidad y, si es así, su ejecución dentro de un marco temporal. El Commitment Manager no sólo negocia que el servicio se encuentre disponible para ser invocado, también negocia cuándo va a ser capaz el servicio de proporcionar su respuesta. De este modo el Commitment Manager es capaz de escoger aquellos servicios que ofrecen las mejores condiciones temporales y de confianza en el caso de que existan diversos agentes proporcionando la misma interfaz de servicio. Para calcular esta predicción de tiempo de respuesta el agente debe tener en cuenta algunos puntos como: (i) la carga de trabajo actual, (ii) la latencia de la red para invocar servicios remotos, (iii) la disponibilidad del servicio en el momento de la solicitud y (iv) la disponibilidad de los recursos hardware y software necesarios para ejecutar el servicio. Para esta tarea el agente necesita de la ayuda del sistema operativo. El sistema operativo puede ayudar al agente a predecir si va a ser capaz de satisfacer la petición dentro de los límites temporales definidos, y si es así establecer un compromiso con el Deliberation Engine. Esta funcionalidad es ofrecida por el Commitment Manager.
- **On-line Planner**: El Planificador On-line es capaz de reparar o mejorar los planes en ejecución. Este planificador se ejecuta simultáneamente en el interior del motor de deliberación y su tarea es ayudar a los agentes a alcanzar un objetivo cuando el agente no tiene planes pre-compilados que le guíen a la realización del objetivo mediante la composi-

ción o la reparación de planes. El planificador utiliza un TB-CBP (Temporal Bounded Case Based Planner) para generar los planes en tiempo de ejecución. El tiempo requerido para ejecutar el TB-CBP es conocido, por lo que el proceso deliberativo está acotado temporalmente. Además, el TB-CBP utiliza casos que fueron ejecutados en el pasado para tomar la decisión acerca de si el plan calculado podrá ser ejecutado dentro del límite temporal comprometido.

- **OS Agent**: El Sistema Operativo es un participante activo del paradigma de Computación Distribuida basada en Objetivos. Esto significa que el SO, independientemente de las tareas más críticas llevadas a cabo por el núcleo, es capaz de expresar sus propios objetivos, ofrecer servicios a otros agentes y almacenar sus propios ítems de conocimiento en la KB. Este comportamiento se representa mediante la abstracción que llamamos *OS Agent*, la cual es homóloga de la abstracción de agente mostrada en la Figura 8.1.

  - **Objetivos del SO**: El SO tiene sus propios objetivos para llevar a cabo las tareas que corresponden a un sistema operativo. Este conjunto de objetivos incluye las tareas de mantenimiento del sistema y otra funcionalidad no crítica que el SO debe llevar a cabo para asegurar el correcto funcionamiento del nodo en que se ejecuta el SO. Un ejemplo de un objetivo del SO sería «*Keep the HD defragmented*».
  - **Base de Conocimiento del SO**: La KB del SO representa las creencias del SO. El sistema operativo utiliza esta base de conocimiento para poder llevar a cabo sus objetivos mediante el uso de servicios que éste es capaz de invocar. Este conocimiento únicamente está relacionado con la información que

el OS Agent maneja, lo cual no incluye información interna del núcleo, información del proceso de ejecución o información del proceso deliberativo. Dos ejemplos de ítems almacenados en la KB del OS Agent son «*The HD is fragmented*» o «*The HD was defragmented at 2011-11-31 23:50:45*».

- **Servicios del SO**: Esta estructura almacena el conjunto de servicios básicos proporcionados por el sistema operativo. Este conjunto de servicios es utilizado por el sistema operativo para proporcionar la funcionalidad básica de bajo nivel a los agentes del sistema. Esto incluye todo lo necesario para gestionar el sistema y para acceder a funciones restringidas solamente a través del sistema operativo tanto por razones de seguridad como de estabilidad. Algunas de estas características son la comunicación de los controladores del sistema con el hardware, así como otras características que permiten la correcta interacción entre los agentes, proveedores de servicios y el sistema operativo. Algunos ejemplos de servicios del SO son: «*USB Write*», «*Analyze HD*» and «*Defragment HD*».

- **Librería de Planes del SO**: Ofrece planes de pre-compilados para su ejecución asociados a objetivos comunes. Este componente se crea en la fase de diseño del sistema operativo y su motivación es ofrecer planes pre-compilados para objetivos fundamentales que no pueden esperar para una composición diferente o no pueden variar en su flujo de ejecución por razones de seguridad y eficiencia. Un ejemplo de plan pre-compilados del SO es «*Defragment HD*», que incluiría diversos servicios, entre otros «*Analyze HD*», «*Check last defragmentation date*» y «*Perform Defragmentation*».

Bajo el paradigma de Computación Distribuida basada en Objetivos, los objetivos activados por cada agente son seleccionados por el motor deliberativo para su consecución, ayudando éste a elegir el plan adecuado para cumplir cada objetivo. Posteriormente es el Runtime Engine el encargado de planificar su ejecución en función de la disponibilidad de los servicios y de la carga del sistema. Es importante destacar que este modelo de agente preserva las características deseables que definen a un sistema multi-agente como son la autonomía o la proactividad, dado que es el agente el que activa aquellos objetivos que desea cumplir y en el momento que él decide llevarlos a cabo. El Deliberation Engine y el Runtime Engine son herramientas que asisten a la consecución de dichos objetivos como parte del sistema operativo donde se ejecutan los agentes.

Los planes pueden ser proporcionados por el propio agente o pueden ser compuestos *on-line*. Estos planes son una secuencia de **servicios** ofrecidos por los propios agentes tanto de forma local como remota. Como puede verse en la Figura 8.2, los elementos de ejecución básica son los servicios que componen los planes. Los planes son proporcionados al sistema operativo desde dos vías diferentes: la generación *off-line* del plan o la generación *on-line* por parte del planificador del Deliberation Engine. El sistema operativo recibe los objetivos activados para cumplir y los posibles planes que pueden ayudar a la consecución del objetivo. Una vez seleccionado el plan que cumple el objetivo activo, el Runtime Engine ejecuta los servicios que componen el plan seleccionado.

Una vez presentada la arquitectura del SO orientado a objetivos, en la próxima sección se muestra el proceso deliberativo utilizado para llevar a cabo la ejecución de los objetivos de los agentes.

8.2. Deliberation Engine

Este motor deliberativo es el *cerebro* del módulo de ejecución, responsable de decidir qué acciones deben ser llevadas a cabo para cumplir un objetivo. Este componente se encarga de analizar los objetivos que se encuentran actualmente activados por los agentes y de proporcionar los recursos necesarios para su consecución. El Deliberation Engine es el componente principal que gestiona todo el flujo del proceso de ejecución. Se comunica con el Runtime Engine para ejecutar los servicios, con el Gestor de Compromisos para administrar los compromisos temporales de los agentes y con el Planificador On-line para componer nuevos planes.

Desde que un objetivo es activado por un agente hasta que se logra completar, el motor deliberativo pasa a través de diferentes etapas que implican los diferentes componentes del módulo de ejecución. Estos pasos son:

1. Comprobar si es posible activar el objetivo.
2. Comprobar si el objetivo es consistente y no existen conflictos.
3. Pedir al planificador los planes que lleven al objetivo deseado.
4. Pedir al Commitment Manager un compromiso temporal con cada servicio del plan seleccionado.
5. Si no hay compromisos posibles pedir al planificador un nuevo plan o marcar el objetivo como *no-alcanzable*.
6. Seleccionar el plan que nos ofrezca un factor mayor de calidad teniendo en cuenta el compromiso temporal y la fiabilidad basada en casos anteriores almacenados en la Base de Casos.

7. Enviar el plan al Runtime Engine para ser ejecutado.
8. Si el plan falla, pedir al Planificador On-line un nuevo plan, en caso negativo marcar el objetivo como *no-alcanzable*.
9. Cuando el plan finaliza, actualizar la Base de Casos con los resultados de los compromisos.
10. Comprobar los objetivos activos relacionados y las postcondiciones y, en caso de que se cumplan, marcar el objetivo como *alcanzado*.

Existen dos componentes que facilitan al Deliberation Engine la toma de decisiones a la hora de determinar una composición de servicios. Estos componentes son el ***On-line Planner*** y el ***Commitment Manager***. A continuación se explican con más detalle estos componentes del Deliberation Engine.

8.2.1. On-line Planner

Dentro del módulo de ejecución, el encargado de proporcionar los planes que resuelven los objetivos de los agentes del sistema es el Online Planner. Este planificador se basa en el uso de un CBP (Case-Based Planner)[Spa01] que ha sido modificado para dar respuestas en un tiempo acotado, de este modo el tiempo de ejecución del planificador es predecible. Este nuevo modelo de razonador basado en casos llamado TB-CBP (Temporal Bounded CBP) consta de las mismas fases que el CBP clásico, pero modificadas para determinar cuánto tiempo va a costar ejecutar cada una de ellas. De esta forma el tiempo de ejecución del proceso de composición de servicios es conocido y tenido en cuenta cuando se debe dar una respuesta en un tiempo máximo. Una descripción detallada de esta aproximación se encuentra en [NHJB11, NBJ10].

No obstante, a continuación se verá una descripción general del funcionamiento del *TB-CBP on-line planner*.

La estructura de los casos utilizada en el TB-CBP se define como:

$$\langle Postcondition, Precondition, \{Service\}, Quality, ExecutionTime \rangle$$

donde:

- $Postcondition$ es el objetivo a conseguir una vez ejecutados los servicios.
- $Precondition$ es el conjunto de condiciones que se deben dar para iniciar la ejecución de los servicios que lograrán cumplir el objetivo.
- $Service$ es la lista de servicios que deben ser ejecutados desde el estado $Precondition$ para llegar al estado $Postcondition$.
- $Quality$ indica la confianza que tiene el sistema en cada uno de los agentes para que ejecuten los servicios correctamente. Esta confianza se basa en casos pasados ejecutados con éxito.
- $ExecutionTime$ es el tiempo que se ha estimado en ejecuciones pasadas para la ejecución de los servicios.

Un ejemplo de la Base de Casos puede verse en la Tabla 8.1.

Para realizar la búsqueda de una composición, el agente informará del objetivo ($Postcondition$) que desea satisfacer y de su conocimiento previo ($Precondition$). Con esta información el *On-line Planner* puede construir una composición de servicios extrayendo casos de la

Tabla 8.1: Ejemplo de Base de Casos del TB-CBP

Postcondition	Precondition	Services	Quality	Time
B	A	{S1}	1	4t
C	A	{S1,S3}	0.85	10t
C	B	{S3}	0.85	6t
D	C	{S6}	0.9	7t
E	B	{S7,S10,S11}	0.76	11t
E	D	{S8}	0.99	3t
E	D	{S4,S12}	0.98	7t
F	C	{S5,S9}	0.81	7t
F	E	{S13,S14}	0.98	10t
...	...	...	...	...

base de casos que dispone y componiendo un camino entre el objetivo que se desea alcanzar hasta llegar a alguna de las creencias que el agente posee, lo cual significaría que el agente almacenará la creencia de que el objetivo ha sido cumplido.

Imaginemos la siguiente situación teniendo en cuenta la información de ejemplo de la Tabla 8.1. Un agente desea satisfacer el objetivo F y su creencias son {A,B}. El *On-line Planner* extraerá de la Base de Casos aquellos casos que tengan como $Postcondition$ el objetivo F. Por cada caso extraído el algoritmo volverá a realizar una búsqueda en la Base de Casos pero considerando como $Postcondition$ las precondiciones de los casos extraídos (el parámetro $Precondition$). Se seguirá con este proceso hasta que se llegue a extraer un caso con el que su $Precondition$ se cumpla para las creencias del agente, en este caso $Precondition = A \vee B$. En la Figura 8.3 podemos apreciar el avance de la búsqueda desde F hasta A o B. En el caso presentado podemos apreciar que se obtienen varios planes posibles. Atendiendo a las necesidades del agente se escogerá uno u otro. Si el agente desea obtener

un resultado con la mayor calidad posible entonces escogerá cualquiera de los planes marcados como (3). Puede darse el caso que se desee obtener un plan que obtenga el objetivo lo antes posible, escogiéndose entonces el plan marcado como (1). El agente puede incluso desear un plan que se cumpla dentro de un periodo temporal marcado, por ejemplo antes de 22 unidades de tiempo, con la mayor calidad, en este caso se escogerá la opción (2). Como se aprecia, el agente tiene la libertad de escoger que clase de plan quiere obtener haciendo al sistema adaptable a las necesidades del mismo. Para ello utilizará los servicios del Commitment Manager, como veremos más adelante.

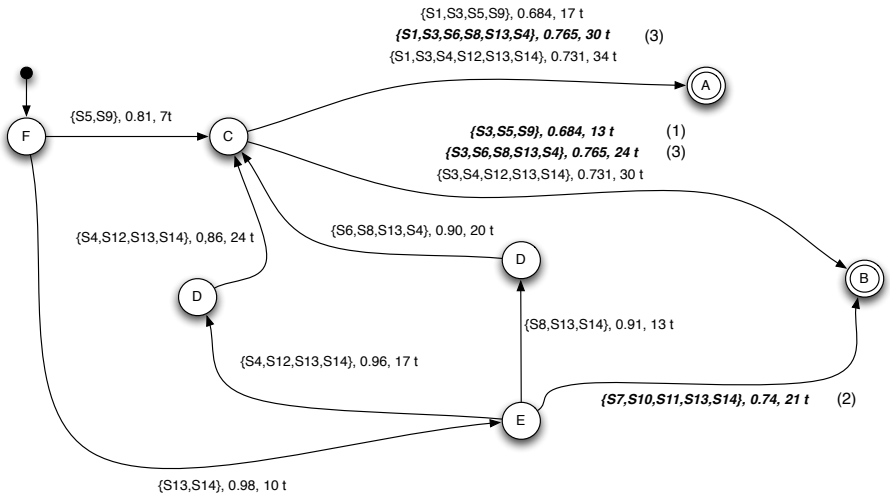


Figura 8.3: Secuencia de búsqueda en la Base de Casos

En la Figura 8.4 podemos observar las distintas fases del TB-CBP implementado por el On-line Planner. Dado que este planificador utiliza el modelo expuesto en los trabajos [NHJB11, NBJ10] podemos ver cómo el orden de las fases habituales de un CBP ha sido alterado para

poder acotar temporalmente su ejecución. En primer lugar se ejecutan las fases de una etapa de aprendizaje, que engloba la fase de *revise* y la de *retain* durante un tiempo acotado (T_L), y a continuación se ejecutan las fases de la etapa deliberativa del CBP, que engloba la fase de *retrieve* y la de *reuse*. Vamos a ver brevemente cada una de las fases.

Revise

En esta fase el TB-CBP se encarga de tomar los casos previamente ejecutados y almacenados en una *cache* de casos (que han sido realimentados por el Commitment Manager como veremos más adelante) y formatearlos para su introducción en la base de casos. La fase de revisión del ciclo CBP se divide en las subtareas de evaluación y reparación del plan. En este caso realizamos una *evaluación en ejecución* del plan, valorando su utilidad tras la ejecución, y una *reparación automática*, que actualiza los valores del plan, realizando una adaptación similar a la de los planificadores CHEF [Ham90, Ham86] y DERSNLP [IK96]. En esta fase se comprueba que la ejecución del caso ha sido correcta, esto es, que ha finalizado exitosamente su ejecución y además se ha ejecutado dentro del plazo establecido. En caso afirmativo premia el caso mejorando su valor de calidad, que representa la confianza que tiene el TB-CBP en dicho caso, para posteriormente ser actualizado en la base de casos en la próxima fase. En caso de que la ejecución resultara fallida el caso será penalizado, decrementando su valor de calidad. Si además, la razón del fallo ha sido superar el tiempo de ejecución establecido, se revisará el valor de tiempo de ejecución actualizándolo al nuevo peor caso.

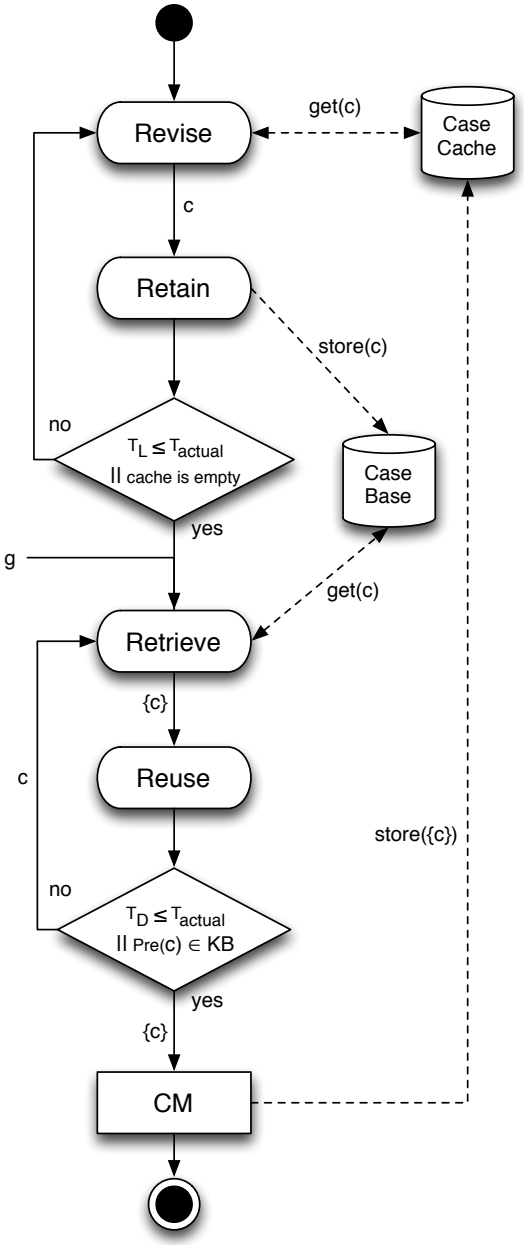


Figura 8.4: Ciclo del Planificador Basado en Casos Acotado Temporalmente

Retain

La fase de retención es de las menos costosas en este planificador. Dado que este tipo de planificación es *hacia atrás* (partiendo del objetivo buscamos un camino que parta del conocimiento almacenado por el agente) la estructura de la base de casos está indexada por el valor de postcondición de cada caso. Utiliza una tabla de dispersión donde la clave es la postcondición y el valor es el caso. Es un tipo de almacenamiento basado en *clasificación*.

Estas dos fases, pertenecientes a la etapa de aprendizaje del TB-CBP, son ejecutadas en segundo plano por el sistema, con un tiempo acotado para mantener lo más actualizada posibles la base de casos. Además, el On-line Planner se encarga de mantener la base de casos optimizada y actualizada. Para ello se realiza una limpieza periódica de aquellos casos que son muy raramente utilizados y que fueron generados hace mucho tiempo. Se mantienen preferiblemente en la base de casos aquellos casos que han sido más costosos de generar o que se utilizan más a menudo, pudiendo ser estos factores parametrizados en el SO. Dado que el tamaño de la base de casos debe mantenerse acotado y razonablemente bajo, el algoritmo utilizado para mantener el tamaño controlado y los casos actualizados ha sido adaptado de los algoritmos de reemplazo de páginas utilizados por los sistemas operativos para la gestión de la memoria virtual. Estos algoritmos combinan técnicas como [ADU71] o [YSP99].

Retrieve

En esta fase es cuando se comienza la composición de un nuevo plan. Para ello se introduce en el sistema el objetivo que se desea cumplir, g, y se comienza la búsqueda hacia atrás extrayendo casos de la

base de casos cuya postcondición coincida con el objetivo buscado. Este tipo de recuperación se clasifica como *recuperación jerárquica*, dado que se basa en haber almacenado los casos en la base de casos con una estructura particular (una tabla de dispersión en este caso en particular).

Reuse

La fase de reutilización se encarga de adaptar los casos recuperados y construir la solución que será entregada como salida del TB-CBP. Dado que la adaptación automática puede resultar poco fiable y muy costosa sin supervisión [MSH96], esta fase utiliza la técnica de *copia sin adaptación automática* dado que el caso recuperado puede ser introducido directamente en la solución. Como la planificación es una búsqueda hacia atrás, la precondición del caso reutilizado se pasa a la fase de *retrieve* como nuevo objetivo para la recuperación de nuevos casos. Esto se repite hasta que se encuentra un caso cuya precondición esté contenida en la base de conocimiento del agente que invocó la planificación. La solución que construye el planificador es un grafo cuyo nodo final es el objetivo, al cual se llega por los diferentes caminos de dicho grafo. Para ello se utilizó la función $\kappa(g_{ij}, a_i)$ vista anteriormente. Una vez el grafo ha sido calculado, se debe comprobar que todos los servicios que forman parte del plan están disponibles y cuál es el nivel de carga de trabajo de los agentes que los deben de ejecutar. Esta función es realizada por el Commitment Manager, el cual veremos a continuación.

Otra consideración de rendimiento de esta etapa de deliberación ocurre gracias a que el algoritmo del planificador está también acotado temporalmente (T_D). Dado que el TB-CBP utiliza un algoritmo *anytime* [Zil96] para la composición de planes, cuando el tiempo asignado para la composición se agota el TB-CBP devuelve la mejor solución que ha

sido capaz de componer en el tiempo delimitado.

8.2.2. Commitment Manager

El Commitment Manager es una mejora de un framework llamado *SAES* [VNJR09], el cual permite componer servicios y garantizar su ejecución dentro de un límite temporal. La gran diferencia con la aproximación SAES es que, al introducir el framework dentro del sistema operativo con capacidades de tiempo real, se dispone de más información para hacer más precisa la ejecución en tiempo real y, en consecuencia, los compromisos temporales.

El Commitment Manager tiene dos funciones principales: (i) comprobar que el conjunto de servicios propuestos por el On-line Planner para la ejecución de un plan se encuentra disponible para su ejecución y (ii) establecer un compromiso con los agentes que proporcionan cada uno de los servicios seleccionados sobre la ejecución de los mismos (ver Figura 8.5).

Para cumplir su primera función, el Commitment Manager envía una petición de propuestas (call for proposals) a todos los agentes que oferten los servicios implicados en la composición. Cada agente pregunta a su SO cuándo puede completar el servicio y devuelve una propuesta de tiempo de respuesta al Commitment Manager. La propuesta consiste en la siguiente tupla:

$$< T_{start}, T_{duration}, PS >$$

Donde:

- T_{start} indica el momento de activación del servicio. Esto es, cuan-

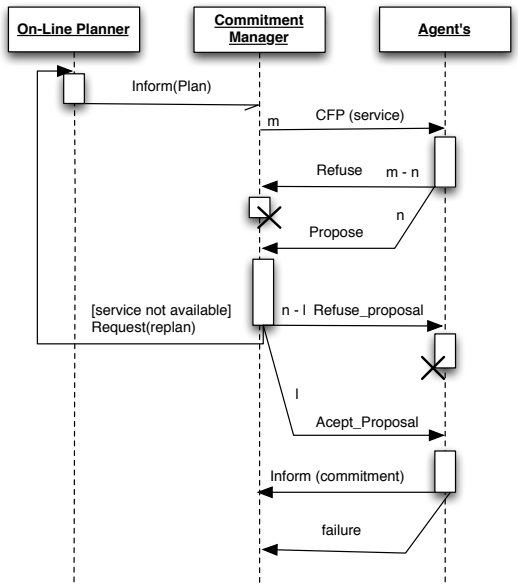


Figura 8.5: Protocolo de interacción de consulta de la disponibilidad de servicios

do puede empezar su ejecución.

- $T_{duration}$ indica el tiempo necesario para completar la ejecución del servicio.
- PS indica la probabilidad de éxito de la ejecución del servicio en el tiempo indicado.

En la ejecución del On-line Planner el sistema ha obtenido una medida de calidad utilizada para estimar el mejor plan. En este caso, el Commitment Manager calcula la probabilidad que indica si el agente puede completar el servicio en el tiempo indicado teniendo en cuenta la carga de trabajo y la fiabilidad del algoritmo de predicción de tiempo de respuesta. Esta información es más precisa que la calidad obtenida

del plan, puesto que tiene en cuenta la situación actual del sistema y no únicamente casos pasados. Con toda esta información se realiza un pre-acuerdo entre el agente y el Commitment Manager para la ejecución del servicio.

Cuando todos los agentes han contestado al Commitment Manager, éste debe calcular la probabilidad de éxito asociada al conjunto de la composición de servicios. Para realizar esta tarea, el Commitment Manager utiliza la probabilidad de éxito enviada por cada uno de los agentes junto con la información de ejecuciones previas similares extraída de la base de casos. La probabilidad de éxito de la composición de servicios se calcula de la siguiente forma:

$$PS_{composition} = \prod_{i=0}^{N} PS_i * \omega_i$$

donde $\omega_i \in [0, 1]$ es el peso asociado al servicio i. Este peso está relacionado con la experiencia de compromisos previos; un agente con muchos compromisos fallidos tendrá un peso más bajo.

Una vez el Commitment Manager ha calculado la probabilidad de éxito de cada uno de los caminos de la composición de servicios, enviará al Deliberation Engine la composición elegida y su probabilidad $PS_{composition}$ para que éste analice si la composición es viable. Para seleccionar la mejor composición utiliza tanto el valor de $PS_{composition}$ como el valor normalizado del tiempo de ejecución de la composición ($\widetilde{T}_{composition}$). Entendemos que la composición es viable si, por ejemplo, la probabilidad de éxito es admisible para los estándares de calidad exigidos por el Deliberation Engine; estos son valores parametrizables del SO. Si el Deliberation Engine está de acuerdo con la composición se lo comunicará al Runtime Engine para que comience la ejecución del plan. En este caso, los pre-acuerdos establecidos por el Commitment Manager son confirmados a los agentes. En caso contrario el Commit-

ment Manager informará de que los pre-acuerdos deben ser cancelados, liberando así la reserva de recursos hecha por los agentes.

El Commitment Manager está también a cargo de asegurar que los compromisos establecidos se cumplan. En caso de que no se cumplan el Commitment Manager penalizará al agente proveedor del servicio. Esta penalización se ve reflejada en los pesos aplicados por el Commitment Manager en los cálculos de probabilidad de éxito de las composiciones de servicios.

8.3. Runtime Engine

El Runtime Engine es el componente a cargo de controlar el ciclo de vida de las entidades en ejecución. Esto incluye guiar la ejecución del modelo de proceso de los planes que se encuentran activos y planificar los servicios que han sido invocados por un plan, tanto invocaciones locales como remotas.

Un servicio *atómico* posee un ciclo de vida y proceso de ejecución similar al de la abstracción clásica de proceso de un sistema operativo [RT73]. Los servicios son entidades independientes que el Runtime Engine planifica y ejecuta con un contexto propio para cada servicio. El ciclo de vida de los servicios se compone de los siguientes estados: (i) `preparado`, (ii) `en ejecución` y (iii) `durmiendo`.

Como hemos visto anteriormente, el Runtime Engine se encarga también de la gestión de la ejecución de los *planes*. Mientras que el Commitment Manager se encarga de asegurar que los compromisos temporales son cuidadosamente cumplidos, el Runtime Engine se encarga de verificar a cada paso que el plan está siendo ejecutado correctamente. Esto implica asegurarse de que, antes de ejecutar un servicio,

Algoritmo 3: Algoritmo de ejecución del Runtime Engine

```
1  foreach Plan in selectedPlans() do
2      if checkPreCondition(Plan) == True then
3          ServiceQueue = emptyQueue()
4          n = selectFirstNode(Plan)
5          append(ServiceQueue, n)
6          while hasNodes(ServiceQueue) do
7              n = getNode(ServiceQueue)
8              if checkPreCondition(n) == True then
9                  invoke(n)
10                 if checkPostCondition(n)== True then
11                     foreach Node in neighbors(n) do
12                         append(ServiceQueue,Node)
13                     end
14                 end
15             end
16             remove(ServiceQueue, n)
17         end
18         if checkPostCondition(Plan) == True then
19             return True
20         end
21         else
22             replanning()
23         end
24     end
25 end
```

todas las precondiciones son verdaderas y que, después de ejecutar el mencionado servicio, todas las postcondiciones han sido cumplidas. Esta supervisión se consigue mediante el seguimiento del modelo de proceso OWL, siguiendo el flujo lógico descrito y comprobando que se cumplan las precondiciones y postcondiciones. La tarea de visitar el modelo de proceso corresponde al Runtime Engine, comprobando que se cumplen a cada paso las postcondiciones y que existen las precondiciones necesarias para avanzar al siguiente nodo.

El algoritmo 3 detalla los pasos seguidos por el Runtime Engine y que son detallados a continuación:

1. The Runtime Engine (RE) extrae un plan de la lista de planes programados para su ejecución por el Deliberation Engine.

2. Comprobar si la precondición del plan es válida y el plan puede ser ejecutado.

3. En este momento el plan es seleccionado como plan en ejecución. El Runtime Engine selecciona el primer nodo del plan de su grafo de servicios e invoca este servicio, añadiéndolo a la cola de preparados del planificador de tareas.

4. Antes de ejecutar un servicio el Runtime Engine siempre comprobará previamente que se cumplen sus precondiciones y, tras la ejecución del servicio, comprobará que se han cumplido sus postcondiciones. Si la postcondición es válida la ejecución del plan puede continuar.

5. Una vez el servicio finaliza su ejecución, el Runtime Engine extrae del modelo de proceso todos los vecinos del nodo que representa al servicio y comprueba sus precondiciones. Estos vecinos

son los nodos directamente accesibles desde el nodo del servicio recién ejecutado.

6. Este proceso continua hasta que el modelo de proceso alcanza un nodo final o los servicios fallan y el plan debe ser reparado (utilizando para ello el On-line Planner).

7. Cuando el plan termina, el Runtime Engine comprueba la post-condición del plan. Si ésta es válida, el objetivo que motivó la ejecución del plan es marcado como *objetivo alcanzado*. En otro caso, será necesario buscar un nuevo plan mediante el On-line Planner.

El *agente* es la entidad que motiva este modelo de ejecución. Los agentes pueden tener cuatro posibles estados, dependiendo de su rol actual:

- **Applicant**: El agente tiene objetivos que desea cumplir y no ofrece ningún servicio.
- **Provider**: El agente ofrece servicios a otros agentes pero no tiene objetivos activados.
- **Provider-Applicant**: El agente tiene tanto objetivos que desea cumplir como servicios disponibles para ser invocados por sí mismo o por otros agentes con roles *Applicant* o *Provider-Applicant*.
- **Inert**: El agente no dispone ni de objetivos ni de servicios. Este es habitualmente el caso de un agente que está disponible para salir del sistema.

Una vez el Runtime Engine ha ejecutado un plan éste notificará al On-line Planner, encargado de ejecutar la fase de *retain* sobre la Base

de Casos. Esta fase consiste en el almacenamiento de un nuevo caso (tanto si la ejecución ha sido exitosa como si no) para mantener la Base de Casos actualizada.

8.3.1. Traza de ejecución

En esta sección se va a exponer una traza de ejemplo donde se puedan ver los diferentes pasos que sigue este modelo de ejecución para la consecución de un objetivo. Por razones de sencillez se ha preparado un entorno sencillo con pocos elementos y un único objetivo a obtener. Para mostrar la flexibilidad del sistema simularemos un error en la traza, mostrando la tolerancia a fallos del modelo de ejecución. En el Capítulo 10 se mostrará un ejemplo de forma más extensa.

El escenario que hemos preparado ha sido diseñado para llevar a cabo una tarea muy común: *almacenar una canción en un dispositivo iPod*. En este escenario el agente cliente únicamente se limita a expresar su objetivo (**`Song in Ipod`**), además de poseer cierto conocimiento previo en su base de conocimiento: el audio que desea guardar (**`PCM Audio`**) y ciertos metadatos adicionales (título, autor, género,...) sobre la canción (**`Song Metadata`**). Estos ítems de conocimiento actuarán como las precondiciones del plan que va a ser ejecutado. En este ejemplo existe un agente que actúa como interfaz con el usuario (llamaremos a este agente el *client agent*), también existe un conjunto de servicios distribuidos por los diferentes nodos de la red. Cada uno de estos servicios es proporcionado por un agente que se encuentra alojado en un nodo de la red, conectado a su vez al nodo donde se encuentra el agente cliente.

Cuando el *client agent* activa el objetivo, el Deliberation Engine tratará de buscar un plan que cumpla dicho objetivo. Dado que el *client*

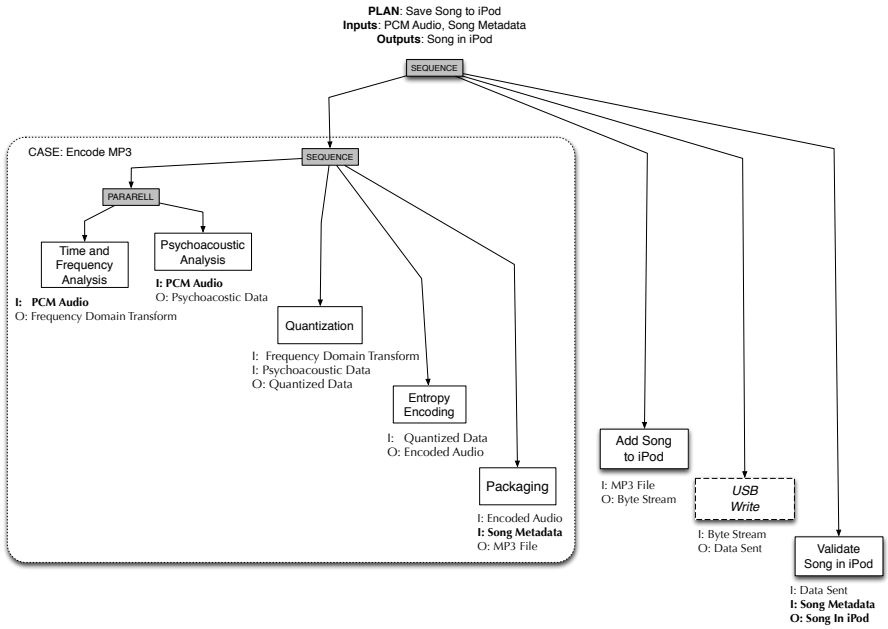


Figura 8.6: Modelo de proceso del plan `Save Song to iPod`

agent no posee ningún plan que tenga como función este objetivo, el On-line Planner será el encargado de componer un plan que lo lleve a cabo. Para ello partirá del conocimiento del agente expresado en su KB para ser utilizado como precondiciones. El plan generado se muestra en la Figura 8.6. El iPod únicamente trabaja con formatos de codificación de audio MP3 (que es diferente de la precondición expresada por el agente, que utiliza el formato PCM), por lo que el plan generado incluye automáticamente los servicios necesarios para codificar la canción desde el formato PCM al formato MP3. Supongamos que este tipo de codificación de audio había sido previamente ejecutado en el sistema, razón por la cual existe un Caso encargado de la codificación de audio en el plan generado. Las cajas con lineas de puntos repre-

sentan servicios de bajo nivel proporcionados por el sistema operativo del *client agent*, por lo que se encuentran en el mismo nodo que dicho cliente.

A modo de ejemplo vamos a seguir una traza de ejecución utilizando el plan de la Figura 8.6:

1. Inicialmente, el Deliberation Engine seleccionaría un objetivo de un agente. Por simplicidad en este ejemplo tan sólo tendremos un objetivo: *Song in iPod*. Dado que no hay más objetivos en el sistema, el Deliberation Engine selecciona este objetivo.

2. El On-line Planner genera un plan que lleve a cabo el objetivo (Figura 8.6).

3. Dado que la precondición del plan se cumple (el agente *conoce* `PCM Audio` y `Song Metadata`), el motor deliberativo selecciona el plan para su ejecución. Se comprueba que su postcondición es compatible con el objetivo deseado (genera el estado `Song in iPod`).

4. Los primeros servicios a ser ejecutados son `Psychoacoustic Analysis` y `Time and Frequency Analysis`. Antes de ejecutarlos, el Commitment Manager establece compromisos temporales con los nodos donde se alojan.

5. El Runtime Engine ejecuta los servicios `Time and Frequency Analysis` y `Psychoacoustic Analysis`, alcanzado como efectos los ítems de conocimiento `Frequency Domain Transform` and `Psychoacoustic Data`. El Commitment Manager comprueba que los compromisos temporales han sido respetados, por lo que premia a los servicios y ejecuta la fase de *retain* en la base de casos del On-line Planner.

6. A continuación sigue el servicio `Quantization`. Tras establecer un compromiso temporal, el Runtime Engine ejecuta este servicio, consiguiendo como efecto el valor `Quantized Data`. Una vez más el Commitment Manager premia al servicio y retiene el caso.

7. Para mostrar todas las ventajas de este modelo de ejecución, vamos a introducir un error en este punto. Vamos a asumir que el servicio `Entropy Encoding` no se encuentra disponible (esto puede significar que el agente que proporciona el servicio no está conectado, el servicio se encuentra saturado en este momento o quizás la salida que devuelve es incorrecta y no se encuentra en formato MP3). Esta situación provoca que el Commitment Manager penalice al caso que representa a dicho servicio.

8. En este momento, el Runtime Engine necesita pedir al On-line Planner que repare el plan en ejecución para poder continuar con la ejecución demandada por el agente.

9. El planificador nos devuelve el plan mostrado en la Figura 8.7. Este plan reparado continua donde el plan antiguo falló, y reemplaza el servicio fallido por otra estructura gracias a los servicios encontrados en la red de nodos distribuidos. El nuevo plan tiene una estructura muy similar, pero reemplaza el servicio de codificación por una estructura de selección de tres tipos de codificación alternativos (`PCM Encoding`, `Differential PCM Encoding` y `Adaptive PCM Encoding`).

10. En estos momentos el Commitment Manager necesita establecer un compromiso temporal con aquel servicio que le asegure un menor tiempo de ejecución y le proporcione un mayor valor de confianza. Para ello, el Commitment Manager pide a la base de

casos valores de confianza de estos servicios y le pregunta por un compromiso temporal de ejecución a cada uno de los servicios. Con esta información el Runtime Engine selecciona para su ejecución el servicio `Adaptive PCM Encoding`, cancelando los pre-acuerdos con los agentes proveedores de los otros dos servicios.

11. Finalmente la ejecución del plan pasa por los servicios `Packaging`, `Add Song to iPod`, `USB Write` y `Validate Song in iPod`. A cada paso se ha establecido un compromiso temporal con cada proveedor y el servicio ha sido premiado o castigado en función de cada caso.

12. Cuando la ejecución del servicio `Validate Song in iPod` ha finalizado, el *client agent* tiene en su base de conocimiento el hecho `Song in iPod`, por lo que el objetivo ha sido cumplido y puede ser eliminado del conjunto de objetivos activos.

Un aspecto interesante del *client agent* es que, independientemente de que el plan original haya fallado, el agente ha sido capaz de alcanzar su objetivo de manera completamente transparente gracias a la habilidad de replanificación del módulo de ejecución del sistema operativo. Con este módulo, el grado de éxito en la ejecución de objetivos es mucho mayor que en sistemas BDI clásicos. Este módulo de ejecución tiene además la capacidad de proporcionar servicios del SO para la composición de planes, permitiendo al sistema operativo interactuar con este paradigma, como en el caso del servicio `USB Write`.

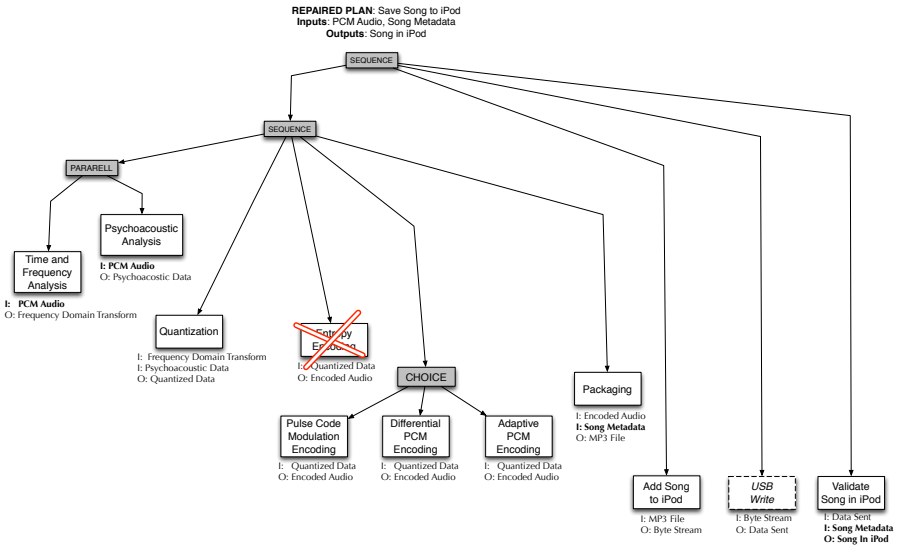


Figura 8.7: Plan `Save Song to iPod` reparado

8.4. Experimentos y Resultados

Con el fin de evaluar la arquitectura aquí presentada para el desarrollo de sistemas operativos orientados a objetivos, en esta sección vamos a presentar un conjunto de experimentos y resultados que validen la propuesta. Se ha desarrollado un simulador discreto para probar todas las funcionalidades y ventajas proporcionadas por el Sistema Operativo que implementa el paradigma de Computación Distribuida orientada a Objetivos. En esta sección presentaremos cómo funciona el simulador y los diferentes experimentos que se han desarrollado para analizar cómo los componentes de la propuesta (el Runtime Engine y los componentes del Deliberation Engine: Commitment Manager y On-line Planner) contribuyen al funcionamiento del sistema.

8.4.1. El simulador

El simulador del sistema operativo nos permite experimentar con la funcionalidad propuesta en este trabajo, pero evitando la complejidad de desarrollar un sistema operativo completo con su problemática de bajo nivel (como el acceso y gestión del hardware). El simulador implementa los componentes principales del módulo de ejecución orientado a objetivos, abstrayendo el resto de componentes del SO. El módulo de ejecución incluye el Runtime Engine y el Deliberation Engine (con sus componentes: el On-line Planner y el Commitment Manager).

El simulador también soporta la representación de un entorno distribuido, donde diversos sistemas operativos orientados a objetivos ofrecen sus servicios en una red interconectada utilizando protocolos de publicación-subscripción de servicios (como ZeroConf [Gut01] o XMPP [MSAM99]). Utilizando el protocolo XMPP PubSub, un agente es capaz de suscribirse a un servicio de directorio para recibir notificaciones cuando se registra o desregistra un nuevo servicio que encaje con las preferencias descritas por el agente. Estas notificaciones fuerzan una actualización de la base de casos. De todas formas, no es obligatorio estar suscrito a un servicio de directorio, puesto que el Deliberation Engine es capaz de buscar nuevos servicios en la fase de composición de planes o cuando un servicio falla en la fase de ejecución. En estos casos la base de casos también es actualizada, buscando así reducir la sobrecarga de la red.

Cada sistema operativo del entorno de simulación tiene además un módulo de comunicación que se encarga de mantener las comunicaciones entre los nodos de la red. Cada nodo es una representación de un sistema operativo orientado a objetivos. Este sistema de paso de mensajes simula un entorno acotado en el tiempo donde es posible predecir el tiempo de latencia de las comunicaciones. Existen varios

trabajos donde se profundiza en protocolos para el establecimiento de paso de mensajes en entornos distribuidos que proporcionan tiempos de latencia garantizados [TBW95, SK00].

El simulador asume que se dispone de un entorno acotado temporalmente, lo cual requiere de un mecanismo de sincronización de reloj y de un sistema de comunicaciones predecible en el tiempo. La sincronización de reloj no es necesariamente una restricción obligatoria, aunque sí deseable. Existen diversas propuestas que utilizan *servicios de tiempo global* para la sincronización de reloj entre los nodos de una red [KO87]. Con estos mecanismos, si un nuevo SO es añadido al entorno de red, su reloj se sincroniza automáticamente con el resto del entorno. Este tipo de servicio de reloj global es muy útil para poder establecer compromisos temporales, para ello utiliza soluciones conocidas de sincronización de relojes de los sistemas distribuidos de tiempo real.

A pesar de todo, un entorno distribuido con comunicaciones acotadas temporalmente no siempre es posible. En determinados escenarios reales, donde las comunicaciones por red no pueden ser acotadas temporalmente (como por ejemplo en Internet), las predicciones de tiempo serán menos precisas. No obstante, la propuesta de SO tiene la capacidad de aprender del entorno gracias a los valores de confianza gestionados por el Commitment Manager, los fallos en las predicciones temporales no serán críticos, puesto que los premios y penalizaciones en la confianza otorgados por el Commitment Manager dirigirán este proceso de aprendizaje.

Este simulador permite cambiar algunos parámetros de su funcionamiento para poder explorar diferentes comportamientos. Podemos parametrizar aspectos de la arquitectura, como el número de nodos en la red, el número de agentes por nodo o el número de servicios que

un agente proporciona. Además, el simulador posee un lenguaje de *scripting* que permite cargar lotes de configuraciones para diferentes entornos de simulación. En estos scripts se puede configurar la situación inicial de la simulación (número de nodos, agentes, distribución de los servicios por agente, objetivos, precondiciones, etc). Además se pueden programar diversos eventos que irán ocurriendo en tiempo de ejecución durante la simulación, como el fallo programado de un nodo, la adición de nuevos agentes o servicios o cualquier otro parámetro configurable del simulador.

La probabilidad de que un servicio falle durante su ejecución es un parámetro configurable. De este modo podemos comprobar cómo se comporta el SO en un entorno tolerante a fallos. Podemos además modificar la precisión con la que se calculan los compromisos temporales en el simulador. Cambiar la precisión de este valor nos permite modificar la calidad de la respuesta del algoritmo, con lo que podemos comparar nodos que ofrezcan diferentes respuestas en las mismas situaciones y cual sería el comportamiento del sistema en estos casos. Este tipo de experimentos será expuesto en las próximas secciones.

Todos los experimentos han sido desarrollados utilizando la misma metodología. Cada experimento ha sido realizado un mínimo de 20 repeticiones. Hemos encontrado que, tras 20 repeticiones, la desviación estándar converge. Se ha aplicado el *test de significancia* de t-student a los resultados de los experimentos. Este test trata de comprobar que las diferencias entre las aproximaciones aplicadas en los experimentos son estadísticamente significativas. La probabilidad de obtener el mismo resultado en aproximaciones diferentes (lo cual será nuestra *hipótesis nula*) resulta siempre muy baja (por debajo de $0{,}05$).

A continuación presentamos el conjunto de experimentos desarrollados. Se han dividido en dos bancos de pruebas: Experimentos del

Deliberation Engine y experimentos de rendimiento, donde se presentan algunas ventajas del sistema distribuido aquí propuesto.

8.4.2. Experimentos del Deliberation Engine

En este trabajo hemos desarrollado una serie de experimentos que validan la funcionalidad esperada del Deliberation Engine. Esto incluye el proceso de razonamiento del sistema operativo para la selección de los mejores servicios disponibles para cumplir los objetivos de los agentes en las mejores condiciones. Para ello vamos a desarrollar experimentos específicos para el Commitment Manager y para el On-line Planner.

Commitment Manager

Como hemos visto anteriormente, el Commitment Manager tiene como función principal establecer compromisos temporales entre los proveedores de servicios y los clientes. Estos compromisos implican que, cuando el cliente invoca un servicio, debe comunicarse con el CM para obtener una predicción temporal de **cuándo** va a obtener la respuesta del servicio. Esta predicción no es una estimación fácil (como veremos en el próximo capítulo), puesto que implica a una gran cantidad de factores que pueden afectar en el resultado (principalmente la carga futura del sistema). El Commitment Manager debe trabajar mano a mano con el Runtime Engine, que es quien planifica la ejecución de los servicios en el procesador. El algoritmo de planificación es muy importante para la tarea de la predicción, puesto que debe ser capaz de cumplir los compromisos temporales establecidos. Al mismo tiempo, este algoritmo debe proporcionar buenos resultados de rendimiento y de interactividad, dado que es un servicio crítico del sistema.

En el próximo capítulo veremos en detalle cómo funciona este algoritmo de predicción de tiempo para el planificador del SO que se ha implementado en este simulador.

A continuación se exponen los diferentes experimentos que se han desarrollado para comprobar la funcionalidad del Commitment Manager. Estos experimentos muestran cómo el SO trata de seleccionar siempre aquellos servicios disponibles que son más deseables para llevar a cabo los objetivos de los agentes.

Experimento 1: Evolución de la confianza con diferentes predicciones temporales

En este experimento vamos a mostrar cómo la confianza que un nodo cliente tiene en sus diferentes proveedores evoluciona conforme pasa el tiempo, centrándose en aquellos nodos que son más fiables. La confianza irá cambiando debido a que no todos los nodos del sistema distribuido tienen la misma precisión a la hora de calcular los compromisos temporales.

El primer experimento ha sido diseñado utilizando como estado inicial el siguiente escenario:

- Existen 3 agentes registrados en el sistema: 1 agente cliente y 2 agentes proveedores, los cuales ofrecen los *mismos* servicios, con la misma precondición P y la misma postcondición Q.
- La red se compone de 3 nodos: cada agente se aloja en un nodo diferente.
- El agente cliente posee el conocimiento necesario para ejecutar el servicio (P) y activa el objetivo G que coincide con las postcondiciones de los servicios (esto es, G=Q).

- Los nodos que hospedan a los servicios tienen diferentes precisiones para calcular los compromisos temporales:
  - El primer nodo tiene una precisión del 90 % calculando el instante de finalización del compromiso temporal. Llamaremos a este nodo *GoodProviderHost*).
  - El segundo nodo tiene una precisión del 20 % calculando el instante de finalización del compromiso temporal. Llamaremos a este nodo *BadProviderHost*).

Durante el experimento se realiza una petición de la ejecución del servicio en cada iteración. Esto significa que el cliente activa el objetivo y selecciona el plan a ejecutar en cada iteración. Tras cada ejecución el agente restaura su conocimiento al estado inicial y vuelve a activar el objetivo una vez más. Conforme pasa el tiempo, la base de casos adquiere más experiencia sobre la confianza de los nodos. La Figura 8.8 muestra los resultados de este experimento. El eje X representa el tiempo (en pasos del simulador) y el eje Y representa la suma acumulada de los servicios proporcionados por cada nodo, lo que es una buena representación de la confianza que el cliente tiene en cada nodo (conforme pasa el tiempo, a más número de peticiones, más confianza en el nodo). En el instante inicial la confianza de ambos nodos es prácticamente la misma. Esto se debe a que no existe experiencia previa en la que basarse, por lo que la base de casos del cliente está vacía y el cliente confía por igual en ambos proveedores. Conforme pasa el tiempo, el número de invocaciones a cada nodo varía debido a que la precisión del nodo *BadProviderHost* no es demasiado buena y falla continuamente en el cálculo del tiempo de finalización de la ejecución del servicio. Esto provoca que la confianza del cliente en ese nodo disminuya y, en consecuencia, la mayoría de invocaciones se realicen al nodo *GoodProviderHost*, como puede verse en la figura.

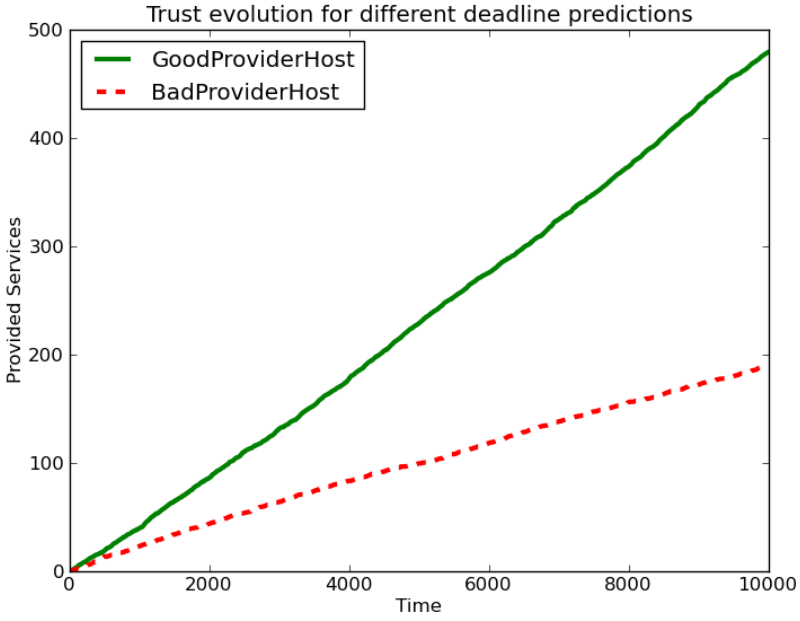


Figura 8.8: Experimento 1: Evolución de la confianza con diferentes predicciones temporales

Cabe destacar que el Deliberation Engine no utiliza *únicamente* el valor de confianza (extraído de la base de casos) para determinar qué proveedor escoger. Con el fin de dar la oportunidad de redimirse a proveedores con mala reputación y descubrir nuevos proveedores con mejores resultados, el Deliberation Engine utiliza un algoritmo de aprendizaje on-line [MCM85] que decide cuándo explorar o cuándo explotar sus soluciones. Esto se ha realizado mediante el ajuste de un umbral que va variando durante la ejecución del sistema operativo. Esa es la razón por la que el *BadProviderHost* no se bloquea y recibe ocasionalmente peticiones de servicio. Pese a tener muchas menos peticio-

nes que el nodo *GoodProviderHost*, de vez en cuando recibe una nueva oportunidad de dar un buen resultado.

Experimento 2: Evolución de la confianza en un escenario grande

Este test muestra cómo la confianza en los nodos con buena precisión en el cálculo de los compromisos temporales crece a medida que pasa el tiempo y los clientes aprenden qué nodos son más fiables. El escenario para este experimento ha sido diseñado con los siguientes elementos:

- Existen 51 agentes registrados: 1 agente con el rol de cliente y 50 agentes con el rol de proveedores que ofrecen el mismo servicio.
- Los agentes están repartidos en 51 nodos diferentes, únicamente un agente por nodo.
- El cliente activa el objetivo que invoca a alguno de los servicios ofrecidos por los proveedores.
- Los nodos se han clasificado en cuatro grupos, dependiendo de la precisión que tienen en el cálculo de su compromiso temporal para sus servicios. Estas precisiones se dividen en: 5 %, 33 %, 67 % y 100 %.

La ejecución de este experimento ha seguido el mismo patrón que en el experimento 1. El agente cliente restaura su base de conocimiento y reactiva su objetivo cada vez que este es conseguido. La Figura 8.9 muestra los resultados del experimento. El eje X representa el tiempo. El eje Y representa si el servicio ha sido demandado por el cliente para ese nodo. De este modo, un punto en esta figura significa que el

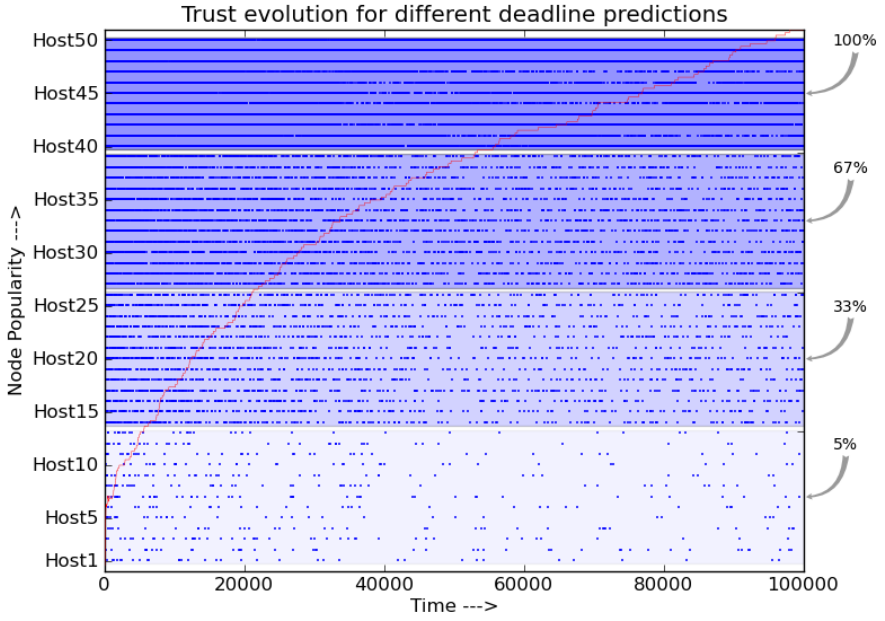


Figura 8.9: Experimento 2: Evolución de la confianza en un escenario grande

servicio ha sido demandado para un nodo en un instante de tiempo. Así pues, la densidad de la nube de puntos nos muestra cuan *popular* es un grupo de nodos. Cuando la densidad es muy alta y los puntos están muy cerca la representación pasa a ser una linea.

La curva que divide la gráfica tiene como finalidad separar la nube de puntos entre la zona densa (parte superior izquierda) y la zona dispersa (parte inferior derecha) para mayor claridad. Ambas áreas muestran cómo la mayor parte de peticiones de servicio se encuentran en la zona densa. Esto se debe a que la base de casos del cliente aprende conforme pasa el tiempo qué nodos son más fiables. Estos resultados muestran el comportamiento esperado del sistema para este

experimento. Del mismo modo se ve que, conforme pasa el tiempo, el número de peticiones a los nodos menos confiables va decreciendo.

Experimento 3: Sistema Operativo Adaptativo

Este experimento nos muestra cómo el Sistema Operativo es capaz de auto-adaptarse a los cambios que percibe en el entorno. La adaptación del SO es un factor muy importante, dado que permite al sistema tener un comportamiento dinámico que permita reconfigurarse para tener la mayor ventaja posible frente a cada una de las circunstancias que ocurran. El escenario que se ha diseñado para este experimento se conforma de los siguientes elementos:

- Se han registrado 5 agentes: un agente cliente y cuatro agentes proveedores, que ofrecen el mismo servicio.
- En la red hay 5 nodos: cada uno de los agentes se encuentra distribuido en un nodo diferente.
- El cliente activa el objetivo que invoca el servicio ofrecido por los agentes proveedores.
- La precisión de los nodos en el instante inicial para el cálculo de los compromisos temporales se distribuye como sigue:
  - Host1: 100 %
  - Host2: 75 %
  - Host3: 50 %
  - Host4: 25 %

En este experimento vamos a cambiar la precisión de alguno de los nodos de forma artificial para mostrar cómo el Sistema Operativo

del cliente es capaz de adaptarse a los cambios de su entorno. Vamos a activar 3 eventos diferentes que tengan influencia sobre el entorno. Los eventos que han sido programados son los siguientes:

- Instante 50000: La precisión de Host 1 se decrementa al 20 %
- Instante 300000: La precisión de Host 3 aumenta al 80 %
- Instante 600000: La precisión de Host 4 aumenta al 90 % y la de Host 2 se decrementa al 20 %

La Figura 8.10 muestra cómo la confianza de los nodos (eje Y) cambia conforme el entorno experimenta los cambios programados (envueltos en la figura con rectángulos verticales). Este valor de confianza representa cuánto se fía el nodo cliente de cada uno de los otros nodos. La adaptación lleva algo de tiempo en verse reflejada, dado que el proceso de aprendizaje del algoritmo del Deliberation Engine debe darse cuenta de que la fiabilidad ha bajado. En el instante 50000 podemos ver cómo la confianza en *Host 1* deja de incrementarse debido al primer evento programado. Podemos notar como esta adaptación tarda un tiempo en consolidarse. Cuando ocurre el segundo evento (instante 300000), la confianza en el *Host 3* comienza a incrementar (su precisión aumenta al 80 %). Mientras tanto, la confianza en *Host 1* sigue decrementándose y el resto de nodos mantienen su valor. El tercer evento cambia de nuevo el comportamiento del sistema, dándole más fiabilidad a *Host 4*, el cual ve incrementado su precisión al 90 %. La confianza en este nodo crece rápidamente debido a que su nueva precisión es muy alta. Paralelamente a esto, *Host 2* comienza a perder valores de confianza.

Estos resultados muestran cómo el Sistema Operativo es capaz de adaptarse cuando ocurren eventos inesperados que cambian el entorno

conocido. En este experimento el agente cliente cambia sus valores de confianza en los diferentes nodos de la red con los que interactúa, cambiando en consecuencia el número de peticiones que realiza a cada uno de los nodos.

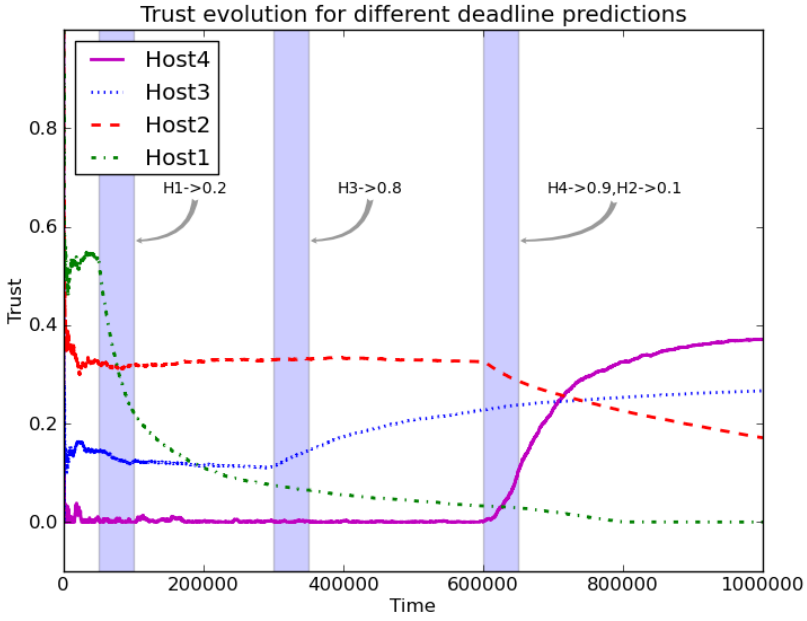


Figura 8.10: Experimento 3: Sistema Operativo Adaptativo

On-line Planner

Como hemos visto anteriormente el On-line Planner es el componente que permite crear planes que lleven a la consecución de los objetivos activos. Este planificador utiliza un TB-CBP (planificador basado

en casos acotado temporalmente) para componer los planes demandados mediante el razonamiento sobre casos pasados similares. Utilizar este planificador para conseguir los objetivos que los agentes han activado aporta una muy interesante característica al sistema: la capacidad de reparar planes on-line, lo que permite a las aplicaciones ejecutadas en el Sistema Operativo ser más tolerantes a fallos.

Esta es la funcionalidad que vamos a probar en los próximos experimentos; esto es, cómo el sistema incrementa su tolerancia a fallos en entornos con poca fiabilidad y cómo esto afecta al ratio de objetivos completados.

Experimento 4: Sistema Operativo tolerante a fallos

Este experimento tiene como fin comprobar cómo el Sistema Operativo es capaz de completar los objetivos que se encuentran activos, incluso si la ejecución de un servicio falla y el plan deja de ser utilizable. Esto significa que parte de un plan que se está ejecutando deja de ser válido y debe ser reemplazado por otro servicio o conjunto de servicios para poder continuar con la ejecución. Para poder llevar a cabo este experimento el simulador permite parametrizar la **probabilidad de error** de un servicio, que indica la posibilidad de que falle la ejecución de un servicio.

El experimento se ha diseñado con los siguientes elementos:

- Se ha creado únicamente un nodo, no hay necesidad de distribuir el experimento en este caso.
- Se han registrado 50 agentes, cada uno con 50 objetivos para activar.

- Finalmente se han registrado 300 servicios, distribuidos de forma equitativa por todos los agentes.

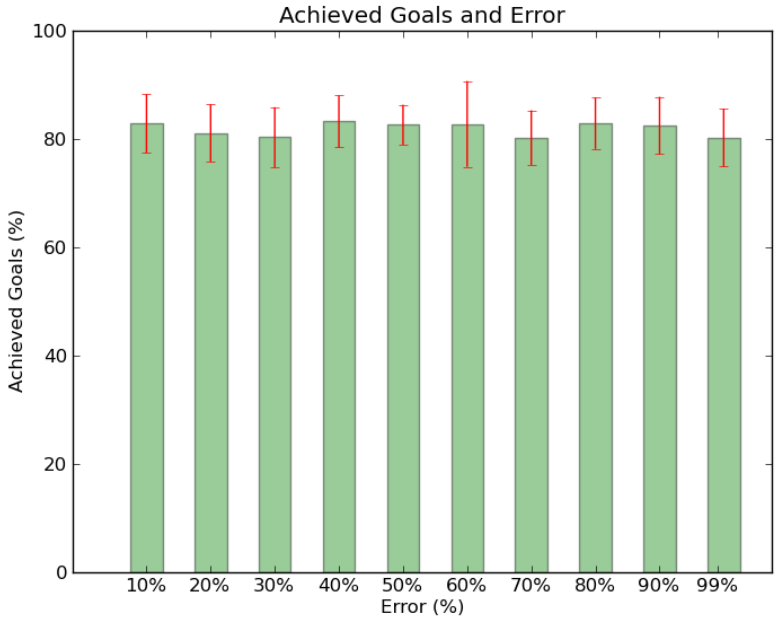


Figura 8.11: Experimento 4: Sistema Operativo tolerante a fallos

En este experimento todos los objetivos, servicios e ítems de conocimiento de los agentes han sido generados aleatoriamente. Únicamente un parámetro es modificado durante las pruebas, la probabilidad de error de los servicios. Este parámetro ha sido variado entre el 10 % y el 99 % en pasos de 10.

La Figura 8.11 muestra los resultados de este experimento. El eje X muestra la probabilidad de error asignada a los servicios. El eje Y

muestra el porcentaje de éxito de los objetivos activados. Hay que destacar que el porcentaje de éxito no es del 100 % dado que la carga ha sido generada aleatoriamente y es estadísticamente improbable que exista siempre un camino para la totalidad de los objetivos creados. Lo que nos muestra la Figura 8.11 es que el porcentaje de éxito de los objetivos es constante, independientemente del error que los servicios tengan. Estos resultados son muy relevantes puesto que muestran que el Sistema Operativo propuesto es capaz de ser muy tolerante a fallos, sin importar en exceso la fiabilidad de los servicios de un plan.

Experimento 5: Evolución de la confianza y errores múltiples

Este experimento nos muestra cómo la combinación de experimentos previos puede afectar a la confianza de los nodos del sistema. Este experimento combina la probabilidad de error de los servicios en ejecución y la precisión de los compromisos temporales emitidos por el Commitment Manager.

El experimento ha sido diseñado con el siguiente escenario:

- Hay 5 agentes registrados en el sistema: 1 agente cliente y 4 agentes proveedores de servicios.
- La red se compone de 5 nodos: cada agente se encuentra alojado en uno de los cinco nodos.
- El agente cliente posee el conocimiento inicial necesario para ejecutar el servicio (P) y activa el objetivo G que coincide con las postcondiciones de los servicios (G=Q).
- Todos los servicios tienen el mismo comportamiento, únicamente varían los grados de precisión de cada uno de los nodos para

calcular los compromisos temporales y la probabilidad de error de los servicios siguiendo la siguiente distribución:

- *Host1* tiene una precisión del 90 % al calcular los compromisos temporales y una probabilidad de error del 10 %.
- *Host2* tiene una precisión del 90 % al calcular los compromisos temporales y una probabilidad de error del 90 %.
- *Host3* tiene una precisión del 10 % al calcular los compromisos temporales y una probabilidad de error del 10 %.
- *Host4* tiene una precisión del 10 % al calcular los compromisos temporales y una probabilidad de error del 90 %.

En la Figura 8.12 podemos ver los resultados de este experimento. Esta prueba nos muestra que no existen demasiadas diferencias entre nodos con diferentes configuraciones. El cliente no discrimina en base a la situación que generó un error (un mal cálculo del compromiso temporal o un fallo en la ejecución de un servicio y su posterior replanificación). Aquello que el cliente ve es que no se le ha proporcionado correctamente el servicio, por lo que el proveedor es penalizado. En la figura podemos ver cómo el nodo *Host 1*, que es el más estable y fiable de todos, tiene el mayor número de peticiones. Por otro lado, el nodo *Host 4* es probabilísticamente el nodo menos fiable, lo que se refleja en un menor número de peticiones con respecto al resto a medida que pasa el tiempo.

8.4.3. Experimento 6: Pruebas de rendimiento distribuidas

En este experimento hemos realizado una serie de pruebas de rendimiento que nos muestran cómo este paradigma puede mejorar la consecución de objetivos cuando aumenta el número de nodos de la

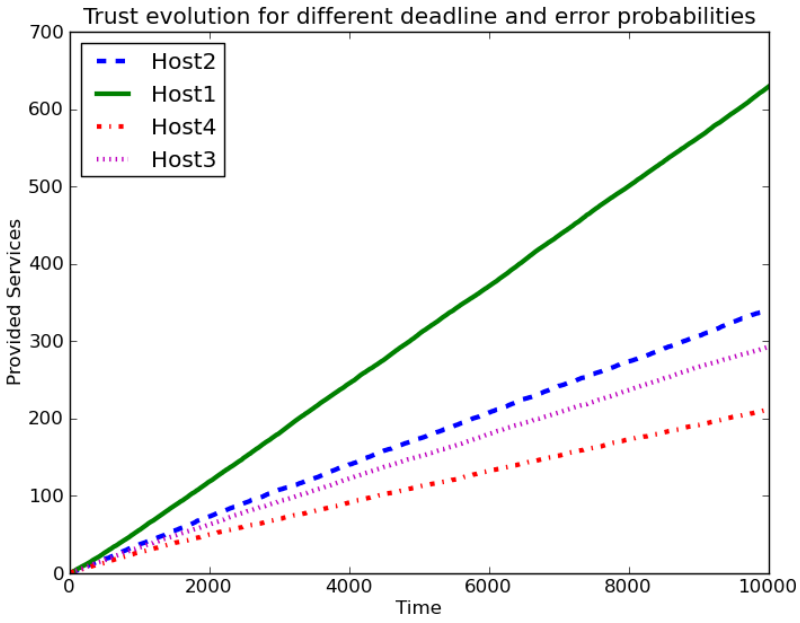


Figura 8.12: Experimento 5: Evolución de la confianza y errores múltiples

red. El sistema operativo que implementa el paradigma de Computación Distribuida orientada a Objetivos puede tener un buen impacto en los resultados de rendimiento del sistema distribuido.

Este tipo de sistema operativo no sólo ayuda a los agentes a cumplir sus objetivos, sino que también es capaz de componer planes utilizando servicios de otros nodos, lo que beneficia ampliamente en la concurrencia del sistema distribuido, mejorando su rendimiento global.

Este experimento (Figura 8.13) muestra cómo incrementar el número de nodos que están ofertando servicios (eje X) decrementa el tiempo

medio necesario para cumplir los objetivos de los agentes (eje Y). Este comportamiento es muy significativo conforme se añaden nodos a la red.

Para realizar este experimento se ha creado un conjunto de objetivos y servicios muy grande en cada una de las pruebas del experimento. Cada una de ellas ha tenido un número diferente de nodos (entre 1 y 50 nodos) donde los agentes han sido distribuidos de forma equitativa. De este modo hemos conseguido que los objetivos sean perseguidos con un alto nivel de concurrencia, obteniendo un menor tiempo en los resultados.

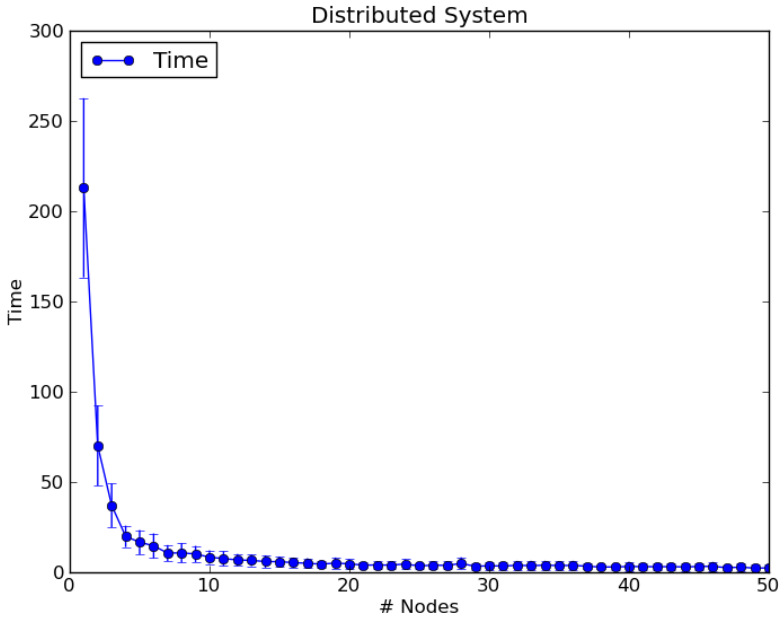


Figura 8.13: Experimento 6: Pruebas de rendimiento distribuidas

Se puede observar en este experimento lo importante que puede resultar el incremento de nodos en la red. En el caso de este experimento se puede ver como los diez primeros nodos contribuyen de manera muy significativa en el decrecimiento de los tiempos medios necesarios para cumplir los objetivos de los agentes. A partir de 10 nodos se puede observar en los experimentos que el impacto es menor, aunque el decrecimiento en los tiempos no cesa. En conclusión, la habilidad de distribuir la ejecución de los planes en la red distribuida de una forma transparente y automática incrementa notablemente el rendimiento medio del sistema.

8.4.4. Experimento 7: Ratio de aceptación de planes

En estos experimentos se ha tratado de mostrar la adaptabilidad del sistema en situaciones diferentes de carga del sistema y de las preferencias del usuario. Para simular esto, se han parametrizado dos variables durante los experimentos. Estas variables han sido establecidas por el Deliberation Engine para gestionar la calidad y la cantidad de planes que son aceptados para su ejecución.

Las dos variables seleccionadas han sido: (i) el máximo tiempo disponible que el Deliberation Engine otorga para la ejecución de un plan y (ii) la mínima calidad requerida a un servicio para ser seleccionado para su ejecución. Además, en este experimento se ha ido incrementado en cada iteración el número de agentes que interactúan, aumentando así la cantidad de objetivos activados. Con esto se ha conseguido aumentar la carga del sistema, dado que cada objetivo nuevo lanza la ejecución de un nuevo plan.

El tiempo máximo para la ejecución de un plan (**MaxTime**) representa la cantidad de tiempo que el Deliberation Engine está dispues-

to a otorgar a un plan para su ejecución. Cualquier plan cuyo cálculo del tiempo de ejecución supere MaxTime será descartado en favor de planes más rápidos. Del mismo modo, la calidad mínima (**MinQ**) representa el valor mínimo de confianza que el Deliberation Engine aceptará de entre las propuestas realizadas por el On-line Planner. Recordemos que este valor de calidad se obtiene del Commitment Manager y representa la probabilidad de éxito de la ejecución de un plan en función de la experiencia pasada.

Ambos parámetros, MaxTime y MinQ, son ajustados dinámicamente por el Deliberation Engine para adaptarse a los requisitos actuales del sistema. Éste siempre tratará de minimizar los tiempos de ejecución y de maximizar la calidad obtenida. En este experimento vamos a estudiar cómo estos parámetros afectan al ratio de *planes aceptados*, dado que no queremos un sistema que sea tan estricto que no acepte nuevos trabajos, pues la finalidad del mismo sigue siendo ejecutar planes y ayudar a cumplir los objetivos de sus agentes. Esta es la razón por la que los parámetros MaxTime y MinQ son dinámicos, para adaptarse a la demanda actual de carga de trabajo del sistema.

El experimento se ha diseñado siguiendo el escenario:

- Se han realizado 100 pruebas con diferente número de agentes en ejecución. En esta sección vamos a mostrar, por brevedad y representatividad las pruebas con 20, 50, 70 y 100 agentes.
- El parámetro MinQ se ha modificado aumentándolo desde el 10 % (0.1) hasta el 100 % (1.0).
- El parámetro MaxTime se ha modificado incrementándolo desde 1.0 unidades de tiempo hasta 1.9 unidades de tiempo. Dado que este simulador es discreto no mediremos el tiempo en segundos, sino en unidades de tiempo teóricas.

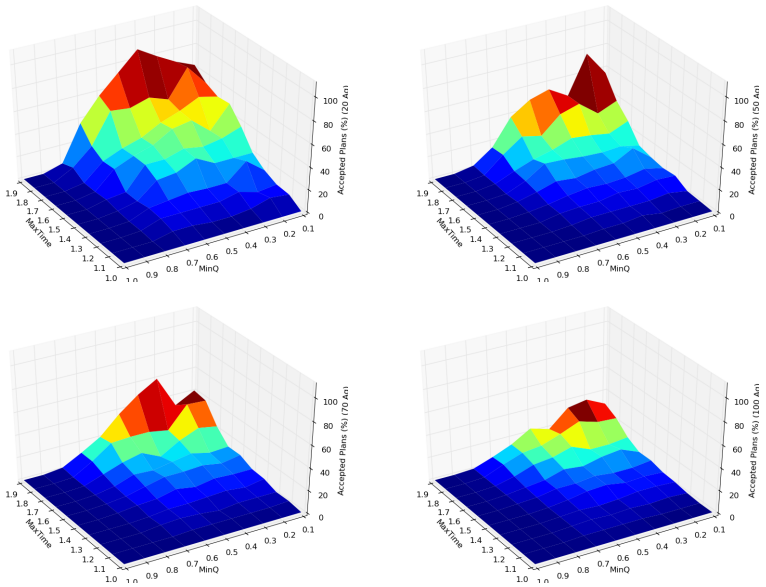


Figura 8.14: Ratio de planes aceptados (20, 50, 70 y 100 agentes)

Como se puede observar en los resultados, el ratio de planes aceptados siempre crece cuando relajamos las condiciones de tiempo y calidad. Además, cuantos más agentes tenemos en el sistema, menor ratio de planes aceptados tenemos. Esto se debe a que la carga del sistema es cada vez más alta, por lo que no hay suficientes recursos para llevar a cabo todos los planes. En esta situación el SO se vuelve más exigente y comienza a rechazar aquellos planes cuyos valores de calidad o tiempo son menos aceptables. De este modo se modifican dinámicamente estos parámetros para conseguir un SO más adaptativo.

8.5. Consideraciones finales

En este capítulo hemos mostrado la arquitectura general del módulo de ejecución del sistema operativo. Se han desarrollado los componentes principales del módulo deliberativo (Commitment Manager y On-line Planner) y se ha presentado el estado actual del Runtime Engine. Finalmente se han diseñado y desarrollado diversos experimentos que explotan las características descritas en este capítulo del SO basado en el paradigma de Computación Distribuida basada en Objetivos. Mediante la arquitectura aquí presentada y las pruebas realizadas se ha podido validar la viabilidad del paradigma de computación propuesto en este trabajo, desgranando muchas de sus características como la tolerancia a fallos, la adaptabilidad o el razonamiento basado en casos pasados.

Sin embargo no ha sido posible introducir las características de tiempo real en el Runtime Engine que permitan al SO realizar los compromisos temporales para la ejecución de servicios debido a que es necesario que el planificador de tareas sea capaz de predecir los tiempos de ejecución que van a tener los servicios. Estas predicciones de tiempo permitirán realizar reservas de recursos que aseguren que un servicio se ejecutará en un tiempo estimado.

Para ello se ha propuesto un algoritmo de planificación con predicción temporal basado en técnicas de *planning-based scheduling*, que permite predecir qué utilización tendrá el SO en el futuro en función de las reservas realizadas y una asignación de prioridades. Este planificador es expuesto en el siguiente capítulo.

9

Planificación con Predicción del Deadline basada en Beneficios

El Sistema Operativo propuesto en este trabajo necesita un planificador de tareas que le ayude a establecer compromisos temporales con garantías de éxito, esto es, que sea capaz de predecir cuándo acabarán los servicios para no perder la confianza de sus clientes. Este planificador es una parte vital del SO puesto que es el responsable de distribuir el tiempo de la CPU entre todos los servicios que se encuentran en ejecución. Para ello, el algoritmo de planificación necesita crear un plan donde se detalle el orden en que los servicios serán ejecutados.

Este tipo de planificadores es muy útil en escenarios donde se desarrolla la Computación Orientada a Servicios o, como es nuestro caso, la Computación Distribuida orientada a Objetivos (DGOC), donde el tiempo de ejecución es un factor crítico utilizado para medir la calidad de un servicio (QoS).

Debido a la dinamicidad y la impredecibilidad de la naturaleza de la web, proporcionar una QoS aceptable es una tarea compleja y desafiante. Los consumidores de servicios necesitan garantías de que sus servicios serán ejecutados con unos mínimos de calidad, por lo que los proveedores de servicios deben de proporcionar los mecanismos necesarios para cumplir con estos criterios de calidad. Existen algunos sistemas, como RT-MOVICAB-IDS system [HNCJ11], donde la ejecución de un servicio en un tiempo predeterminado se considera un factor crítico y, por lo tanto, es necesario proporcionar una respuesta válida antes de determinado instante de tiempo. De otro modo el sistema podría ser inestable o ineficiente.

Existen diversos algoritmos de planificación en los SO de propósito general. Estas aproximaciones, como hemos visto previamente en la sección 5.3, no satisfacen los requisitos del paradigma de Computación Distribuida basada en Objetivos, por lo que en este capítulo mostraremos una solución a este problema donde el algoritmo de planificación no sólo será capaz de planificar las tareas que le llegan desde un entorno distribuido [HHW11], sino que también será capaz de analizar cuándo va a finalizar la ejecución de los servicios para poder realizar una planificación con la intención de cumplir los compromisos temporales y maximizar el beneficio obtenido por la ejecución de los mismos. Además, el planificador será también capaz de priorizar aquellas tareas que le aportan mayores beneficios a nivel cuantitativo (sean beneficios económicos o no), lo cual será un factor de calidad más a añadir

al servicio.

Existen diversas propuestas que introducen el parámetro del tiempo en los servicios web como forma de expresar las restricciones temporales. En [Pan05], [HZj06] y [MDCDM05] se muestran algunas de estas propuestas. Además, existen otras propuestas como [NT07], como [SCZ04] y como [FOGC+07] que utilizan el parámetro del tiempo para guiar la composición de servicios. En todas estas propuestas el tiempo se considera desde un punto de vista descriptivo, pero no se proporcionan garantías en el nivel de ejecución de que estas restricciones se cumplan.

En este capítulo se presenta un algoritmo de planificación de tareas que es capaz de componer un plan de ejecución (*scheduling plan*) con la característica de poder predecir el instante de tiempo en que terminará la ejecución de las tareas que llegan al sistema de forma dinámica. Esto se va a conseguir gracias al mecanismo de reserva de CPU. Este trabajo ha sido publicado además en [PNGFJ13].

9.1. Deadline Prediction Scheduler

En el paradigma de Computación Distribuida basada en Objetivos uno de los valores más importantes que definen la calidad de un proveedor de servicios es la precisión del compromiso temporal que es capaz de establecer con sus clientes. Los clientes que demandan la ejecución de un servicio valoran diferentes parámetros a la hora de establecer la calidad de un servicio. Uno de los más importantes es el *tiempo*. Los clientes utilizan la predicción del deadline de un servicio para establecer lo bueno que es el servicio. Aunque cada agente es una entidad autónoma y puede elegir su propio método para valorar la calidad de un deadline, lo más habitual es establecer que un menor

tiempo de ejecución es siempre mejor. Es por ello que los proveedores de servicios se esfuerzan siempre en proporcionar las predicciones de deadline más bajas posibles, pero siempre siendo cuidadoso para no incumplir el compromiso temporal.

En esta sección vamos a presentar el algoritmo de planificación con predicción del deadline, llamado Deadline Prediction Scheduler (DPS). Para hacer más clara la presentación del algoritmo DPS vamos a mostrar iterativamente diversos refinamientos del algoritmo que aportan nuevas mejoras al mismo.

9.1.1. El problema de la planificación con predicción del deadline

El WCET es un valor que proporciona el tiempo máximo que una tarea va a necesitar para ejecutarse completamente utilizando en exclusividad los recursos del sistema (principalmente la CPU). Sin embargo, en los sistemas operativos actuales la interactividad y la multi-tarea son requisitos imprescindibles, por lo que debemos permitir que las tareas utilicen los recursos del sistema de la manera más justa posible, compartiéndolos, para poder mostrar al usuario una interactividad alta y pocos tiempos de espera en las aplicaciones.

El problema presentado en este trabajo no tiene requisitos de tiempo real. Los servicios no imponen un plazo antes del cual deban ser ejecutados. Esta es la principal razón por la que los planificadores clásicos de tiempo real no son adecuados para este problema. Los proveedores de servicios establecen un compromiso temporal con el cliente en el momento de la ejecución del servicio, asumiendo la obligación de completar el servicio antes de ese instante de tiempo negociado. Este compromiso ha sido otorgado al mejor postor, quien probablemente ha

sido el que ofreció un menor deadline. Para el proveedor es importante satisfacer sus compromisos temporales, puesto que desea mantener la confianza de sus clientes para que en el futuro sigan realizándole peticiones de servicios. Es por ello que el proveedor debe establecer un equilibrio entre una predicción lo más ajustada posible, pero siempre sin incumplir sus compromisos, lo que tendría un efecto negativo sobre su reputación. Este tipo de predicción requiere no únicamente conocer el WCET de sus tareas correctamente calculado, sino también la carga del sistema en el presente y mientras dure la ejecución del servicio.

Realizar este tipo de predicciones es una tarea muy compleja, dado que no podemos conocer cual va a ser la carga del sistema en un instante futuro debido a que la tasa de llegada de nuevas tareas es desconocida en un sistema abierto como es este. Así pues, surge la problemática de cómo vamos a poder asegurar el instante de finalización de un servicio si no somos capaces de saber cuántos servicios habrá ejecutándose al mismo tiempo.

Deadline

En este trabajo definimos el **Deadline** de una tarea τ (D_τ) al instante de tiempo en el cual la tarea finaliza su ejecución en el peor de los casos. Esta medida es diferente del WCET, que hace referencia a la cantidad de tiempo que necesitará la tarea para ejecutarse. Asimismo, difiere del concepto de deadline de los sistemas de tiempo real, donde es la tarea es la que impone el plazo en el que debe ejecutarse. Llamaremos también tiempo de activación t_a al instante de tiempo en el que la tarea llega al sistema y está preparada para ser ejecutada.

A continuación presentamos la primera iteración del planificador con predicción del deadline utilizando reserva de CPU.

9.1.2. Deadline Prediction Scheduler con Reserva de CPU (DPS)

El principio básico de este algoritmo surge del espacio que existe entre los algoritmos con reparto justo de procesador y los algoritmos no expulsivos. En un sistema donde cualquier tarea puede ser aceptada para su ejecución en cualquier momento y donde queremos mantener la interactividad y multi-tarea sin romper los compromisos temporales, necesitamos asegurar que los recursos mínimos para cumplir los compromisos son preservados. Para llevar a cabo esta función utilizaremos la *reserva de CPU*.

Para que un compromiso temporal se cumpla, el planificador debe asegurarse de que la tarea tendrá suficiente tiempo de procesador para ejecutarse. El tiempo de procesador es un recurso muy valioso que está directamente relacionado con el instante de tiempo en que una tarea será capaz de finalizar su ejecución. Por tanto, si realizamos una reserva de CPU que nunca pueda ser disminuida después de haber establecido el compromiso temporal, podremos asegurar que el compromiso podrá ser cumplido. Esta técnica se conoce como *Procesador Garantizado* [JRR97, Lei80]. Sin embargo, las técnicas de Procesador Garantizado se utilizan habitualmente en entornos de tiempo real *duro*, donde son necesarios tests de aceptabilidad o grafos de análisis de planificabilidad pre-calculados, lo cual no es útil para la aproximación de este trabajo.

Hay que hacer notar que, en el caso más sencillo, si reserváramos el 100 % del tiempo de procesador a una tarea obtendríamos de forma implícita un planificador FCFS, lo cual no es deseable en términos de interactividad y multi-tarea como ya hemos comentado. La solución a este problema es reservar un porcentaje del tiempo de procesador para cada tarea y asegurar este tiempo durante todo el ciclo de vida de

la misma. Este porcentaje es asignado dinámicamente a cada tarea en orden de llegada. Por ejemplo, la primera tarea puede recibir el 50 % de CPU, la segunda tarea el 25 %, etc. Podríamos elegir cualquier tipo de distribución para repartir el porcentaje de procesador: en este ejemplo hemos utilizado la función $\frac{100}{2^n}$, donde n es el orden de llegada que hemos utilizado para asignar el tiempo de procesador en *slots*. Otra posible distribución podría ser asignar siempre el mismo porcentaje de procesador (por ejemplo el 10 %), lo cual nos acercaría más a los planificadores *justos* del tipo Round Robin o CFS. La gran diferencia con estos planificadores es que, si hemos establecido un compromiso de reservar un porcentaje de procesador para una tarea, este porcentaje nunca será menor, independientemente del número de tareas que lleguen después.

La granularidad del tiempo de procesador es limitada, por lo que el planificador deberá aplicar alguna política cuando no exista más espacio para nuevas tareas. Por ejemplo puede retrasar el momento de activación de la tarea o incluso podría rechazar la entrada de la tarea, siguiendo los pasos de los tests de aceptación.

Utilizando este algoritmo el planificador será capaz de establecer una predicción del deadline ($\mathcal{D}$) basándose en la ecuación (9.1), donde P_i es la posición de la tarea i basándose en el orden de llegada y $\mathcal{WCET}_i$ es el tiempo de ejecución en el peor caso de la tarea i. Este es claramente un cálculo muy pesimista puesto que no tiene en cuenta el *gain time*, que es el tiempo no utilizado debido a que no hay suficientes tareas para sobrecargar el procesador.

$$\mathcal{D} = 2^{P_i} * \mathcal{WCET}_i \tag{9.1}$$

Gain Time

En este trabajo definimos **gain time** como la holgura de tiempo que se obtiene cuando no existen más tareas con tiempo reservado para ejecutar y es posible reutilizar este tiempo para adelantar la ejecución de tareas pendientes. También se incluye en la definición de *gain time* el tiempo que se obtiene cuando una tarea finaliza su ejecución antes del cálculo previsto por su WCET.

El *gain time* es utilizado para anticipar la ejecución de aquellas tareas que ya han consumido todo su tiempo de ejecución reservado. Por ejemplo, si únicamente existe una tarea en ejecución en el procesador, ésta tendría reservado el 50 % del tiempo de CPU y, por lo tanto, tendría un deadline estimado de $2^1 * \mathcal{WCET}_i$. Sin embargo, una vez consumido el tiempo de CPU reservado y dado que no existen más tareas preparadas, el planificador entraría en gain time. Este tiempo sería utilizado para la única tarea que existe en ejecución, por lo que la utilización real de la CPU sería del 100 %.

El planificador puede utilizar la política que considere mejor para repartir el tiempo de CPU cuando se encuentra en modo gain time. Esta política de gain time podría ser un reparto aleatorio, un algoritmo round robin, un algoritmo EDF (Earliest Deadline First) o cualquier otra política elegida por el diseñador. Además, nada impide cambiar la política de forma dinámica de acuerdo a la carga o las características del SO, dado que nunca va a perjudicar a los compromisos temporales establecidos, puesto que se ejecuta en el tiempo sobrante y únicamente puede adelantar el tiempo de finalización de una tarea pero nunca retrasarlo.

El algoritmo DPS realiza una predicción de deadline con la garantía total de ser correcta, siempre sujeto al cumplimiento de las reservas

establecidas en orden de llegada y al correcto cálculo del WCET. Sin embargo, esta predicción es un cálculo extremadamente pesimista, dado que no tiene en cuenta la relación entre las tareas en ejecución y el gain time conseguido de la holgura. De todas formas, la complejidad temporal de este algoritmo es $O(1)$ para las funciones típicas de un planificador `append` y `retrieve`. Estas funciones son utilizadas para insertar una nueva tarea en la cola del planificador y para eliminar una tarea de la misma cola. Esta complejidad temporal se debe a que el algoritmo no necesita recorrer la cola cada vez, sino únicamente apilar o extraer un elemento de la misma. La predicción del deadline también es constante debido a la sencillez de la ecuación (9.1). Finalmente, la función *sched*, encargada de seleccionar la próxima tarea a entrar en la CPU, también tiene un coste constante. Esto se debe a que el trabajo más costoso se realiza en la función `append`, y durante la selección de la próxima tarea a ejecutar únicamente hay que seguir la reserva de CPU establecida.

En la Figura 9.1 se puede observar un ejemplo de traza del algoritmo DPS. Por sencillez se ha eliminado la tarea que representa al planificador. En la figura se representa cómo se distribuye el tiempo de CPU (separado en ventanas temporales) entre las diversas tareas en ejecución. La reserva se ha realizado siguiendo la política de entregar la mitad del tiempo de CPU restante, hasta llegar al grano de granularidad mínimo. Sin embargo, no se representan las predicciones de deadline dado que quedan muy alejadas en el tiempo. Para solventar la mala precisión de las predicciones de deadline proponemos un refinamiento a este algoritmo en la próxima sección.

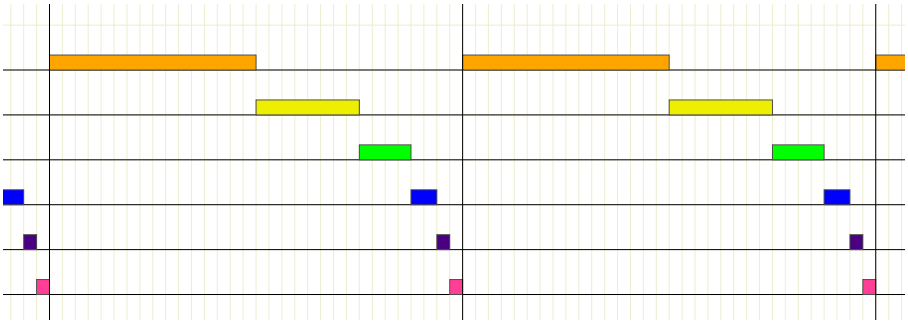


Figura 9.1: Ejemplo de traza del algoritmo DPS

9.1.3. DPS con Reserva de CPU utilizando Promoción de Prioridades Dinámica (DPS-Dy)

La reserva de CPU es una buena aproximación para solventar el problema de la predicción de deadline. Sin embargo, tiene algunas contrapartidas que deben ser resueltas. No es posible mejorar la predicción asumiendo que dispondremos de gain time para utilizar, dado que no sabemos la tasa de llegada de nuevas tareas y el gain time debe quedar libre para cubrir esta posibilidad. Este algoritmo se basa en asegurar que el tiempo de CPU reservado siempre se encontrará disponible mientras dure la ejecución de la tarea. Al mismo tiempo trata de asegurar que habrá recursos disponibles para nuevas tareas entrantes siempre que sea posible, es decir, cuando la utilización de la CPU no sea del 100 %. En ese caso las tareas deberán esperar hasta que se libere algún slot de prioridad.

Slot de prioridad

Llamamos **slot de prioridad** a un porcentaje de tiempo de CPU que puede ser reservado. La prioridad indica la relación en la cantidad de tiempo de CPU reservado. Así, un slot del 50 % tiene una prioridad mayor que uno del 25 %.

Aun así, existe una mejora que puede ser llevada a cabo con el fin de mejorar el cálculo del deadline y hacerlo menos pesimista. Esta mejora se basa en calcular no sólo cuándo una tarea en un slot de prioridad mayor se va a ejecutar, sino también anticipar cuándo va esa tarea a terminar. De ese modo, cuando una tarea finaliza su ejecución y no hay otra tarea planificada para ocupar su slot de prioridad en ese mismo momento, podemos tratar de promocionar una tarea de menor prioridad a ese slot recién liberado.

Con esta técnica, el planificador asigna slots con prioridad dinámica a cada tarea en cada instante de tiempo, dependiendo de qué slots se encuentren disponibles, asegurándose de que nunca queda un slot de alta prioridad libre mientras algún slot de menor prioridad siga ocupado. Esta promoción de prioridades ayuda al planificador a realizar predicciones de deadline mucho más realistas y precisas, puesto que tiene en cuenta la posibilidad de que una tarea pase a utilizar de forma dinámica el tiempo de CPU liberado por otra tarea que ha finalizado.

La reserva de CPU de este algoritmo se representa mediante el uso de una linea temporal. Esta linea temporal se divide a su vez en ventanas temporales por razones de eficiencia. En cada ventana temporal se mantiene una cola con las tareas reservadas, donde el orden en la cola representa el slot de prioridad. Como puede verse en el Algoritmo 4, añadir una nueva tarea al planificador supone recorrer la linea temporal desde el instante de activación t_a hasta consumir el WCET

de la tarea. El algoritmo realiza esta función añadiendo la nueva tarea al final de la cola de cada ventana temporal y decrementando el WCET de acuerdo a la prioridad asignada en cada ventana temporal y al tamaño de la propia ventana. Por ejemplo, dada una tarea τ con un $WCET = 20$ y un tamaño de ventana $\omega = 10$, si añadimos τ en una ventana en la posición 1 (esto es, con la máxima prioridad) su reserva de CPU será del 50 %. Esto significa que τ podrá reservar la mitad del tiempo de la ventana, por lo que el algoritmo decrementará su cómputo de WCET en $\frac{\omega}{2}$. Este proceso se repetirá a lo largo de la linea temporal hasta que se consuma el WCET de la tarea. La ventana temporal en la que el WCET llega a 0 nos permite calcular el deadline de la tarea. Para ello multiplicaremos el índice de la ventana por el tamaño de ventana, que es constante.

Para entender mejor el algoritmo propuesto vamos a mostrar una traza de ejemplo donde añadiremos una nueva tarea para ser planificada en el sistema. La función `append` se utiliza cuando llega una nueva tarea y debe ser preparada para su ejecución (colocándola en una cola de prioridad o aquello que necesite realizar el planificador). En nuestro caso, la función `append` es especialmente relevante, puesto que es el momento en que se realiza la predicción del deadline. También es importante dado que es la función más compleja del planificador en términos de coste temporal.

Supongamos que existen tres tareas ya planificadas en el SO llamadas τ_1, τ_2 y τ_3. Han sido planificadas en la linea temporal tal y como se muestra en la Figura 9.2, con un tamaño de ventana $\omega = 8$, y el escenario inicial definido como vemos a continuación:

Algoritmo 4: Deadline Prediction Scheduler: Función Append

Input : Tarea τ con tiempo de ejecución en el peor caso $WCET_\tau$
Input : Ventana temporal inicial t_a para ejecutar la tarea τ
Input : Tamaño de ventana ω
Output: Deadline D_τ de la tarea τ

```
1  Window ← t_a;
2  if Window ≤ CurrentWindow then
3  |   Window ← CurrentWindow +1
4  end
5  while WCET_τ > 0 do
6  |   D_τ ← Window;
7  |   P ← EnqueueTaskAtWindow(τ, Window);
8  |   WCET_τ ← WCET_τ − ω/2^P;
9  |   Window ← Window +1;
10 end
```

Tarea	t_a	$\mathcal{WCET}$	$\mathcal{D}$
τ_1	1	16	4
τ_2	2	14	6
τ_3	6	10	8

En este escenario con tres tareas ya planificadas vamos a tratar de añadir una nueva tarea, reservando el tiempo de CPU necesario y calculando su deadline. La nueva tarea (τ_4) tiene un $WCET_{\tau_4} = 16$ y su t_a es 4. La traza de la función `append` para esta tarea será la siguiente (Figura 9.3):

1. Primero τ_4 es encolada en t_4. Dado que ya existen dos tareas en

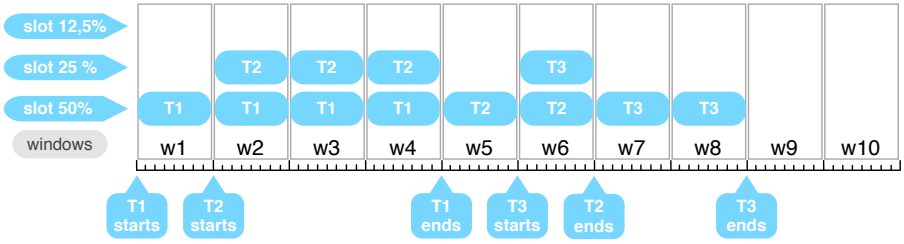


Figura 9.2: Ejemplo de función append para DPS-Dy: Situación Inicial

esa ventana, τ_4 es encolada con un slot de prioridad 3. Con este slot y un tamaño de ventana $\omega = 8$, el WCET es decrementado a $WCET_{\tau_4} = WCET_{\tau_4} - \frac{\omega}{2^P} = 16 - \frac{8}{2^3} = 15$.

2. En t_5 la tarea τ_4 es promocionada al slot de prioridad 2, por lo que su WCET restante pasa a ser $15 - \frac{8}{2^2} = 13$.

3. En t_6 la tarea τ_4 es encolada otra vez en un slot de prioridad 3, por lo que su WCET restante será ahora $13 - \frac{8}{2^3} = 12$.

4. En t_7 y t_8 se encola con un slot de prioridad 2. Tras estos dos pasos, el WCET restante será $12 - \frac{8}{2^2} - \frac{8}{2^2} = 8$.

5. En t_9 ha obtenido el mayor slot de prioridad. Su WCET se reduce a $8 - \frac{8}{2^1} = 4$.

6. Finalmente, en t_{10} la tarea obtiene de nuevo el slot de prioridad 1, el WCET es $4 - \frac{8}{2^1} = 0$ y, dado que el WCET restante ha alcanzado el valor 0, la función `append` ha finalizado y el deadline es $D_{\tau_4} = 10$.

La mejora alcanzada por el planificador DPS con Promoción Dinámica de Prioridades es notable con respecto al algoritmo sin promoción

de prioridades donde el deadline era calculado mediante la ecuación (9.1). Hay que remarcar que con la ecuación mencionada la predicción del deadline hubiera sido realizada teniendo en cuenta únicamente el slot de prioridad adquirido en el momento t_a (que era el slot de prioridad 3), por lo que la predicción de deadline hubiera sido $t_a + 2^P * WCET_{\tau_4} = 4 + 2^3 * 16 = 132$, lo cual es claramente mucho más pesimista que el valor calculado mediante la técnica de promoción de prioridades dinámica.

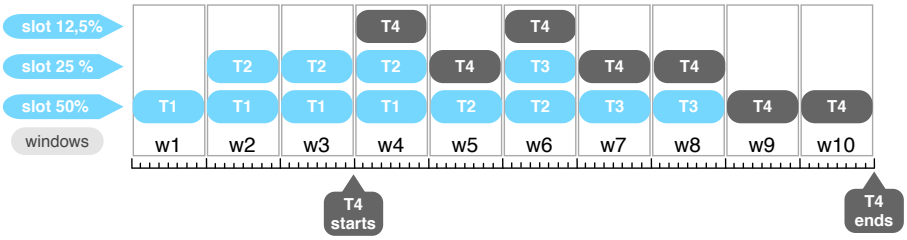


Figura 9.3: Ejemplo de traza de la función append de DPS-Dy

Con respecto a la complejidad computacional, la función `append` tiene un coste lineal, dado que no utiliza ningún bucle anidado ni operaciones complejas en cada iteración, sencillamente añade la tarea al final de cada cola. Es por ello que la función `append` del planificador DPS-Dy tiene un comportamiento muy veloz.

Finalmente, la función *sched* se ejecuta cada vez que el planificador es invocado para seleccionar y poner en ejecución la siguiente tarea que debe entrar a la CPU. Esta función debe seleccionar una de las tareas que tienen una reserva pendiente para la ventana temporal activa actualmente. El planificador debe recorrer la cola de la ventana activa y seleccionar una tarea que todavía tenga tiempo reservado pendien-

te de ser consumido. Si todas las tareas han consumido su reserva el planificador entra en *gain time* y podrá seleccionar una tarea utilizando la política que considere oportuna (aleatoria, Round Robin, EDF, Shortest Job First, etc). El coste computacional de la función `sched` es $O(n)$, donde n es el número de tareas programadas en la ventana temporal activa.

Con este algoritmo hemos presentado un método más preciso para calcular la predicción del deadline, aunque todavía no somos capaces de predecir cómo va a ser aprovechado el tiempo obtenido en gain time, al no conocer la tasa de llegada de las tareas. Aún así, todavía podemos realizar un refinamiento más al planificador, asignando el porcentaje de CPU de una manera más inteligente. La siguiente iteración del planificador utilizará una asignación de recursos más informada utilizando los parámetros extraídos de las tareas.

9.1.4. DPS con Razonamiento basado en Beneficios (BDPS)

Compartir el recurso del procesador utilizando el orden de llegada como criterio no es siempre la solución más justa ni necesariamente la más racional. Dado que existe información que puede ser extraída del cliente que está pidiendo la ejecución de una tarea, el planificador puede utilizar esta información para realizar un mejor plan de ejecución. Podemos definir un mejor plan de ejecución como el reparto del recurso de CPU (slots de CPU) que maximiza el beneficio que el sistema operativo puede obtener para los proveedores de servicios de la ejecución de las tareas de los clientes. Este beneficio se mide en términos cuantitativos como el precio que los clientes pagan (en valor económico u otro valor) por la ejecución de los servicios que están invocando, parametrizado en función del tiempo que ha costado ejecutarlos.

Una razón por la que el orden de llegada no es necesariamente un buen criterio para repartir el recurso de CPU es que existen ciertas situaciones donde otros privilegios deben prevalecer. Un ejemplo de estos privilegios es la importancia del cliente, que llamaremos *prioridad* (Pr). Cuando un cliente solicita la ejecución de una tarea no es lo mismo que el cliente sea el administrador del sistema que sea un cliente remoto al cual no conocemos demasiado y en el cual no confiamos mucho. Esta prioridad se refiere no sólo a la jerarquía de los clientes, sino también a la importancia o la urgencia de la tarea. Los planificadores expulsivos basados en prioridades son ampliamente conocidos, estos expulsan una tarea que se encuentra en ejecución siempre que una tarea de mayor prioridad accede al sistema. Sin embargo, en esta iteración del planificador, la prioridad es utilizada para decidir cuánto porcentaje de procesador será capaz de reservar el cliente. Por ejemplo, el planificador puede decidir que una tarea de prioridad baja puede reservar como mucho el 25 % del tiempo de procesador. Las tareas críticas, como por ejemplo aquellas que pertenecen al propio funcionamiento del SO o que pertenecen al administrador del sistema, serán capaces de realizar reservas con slots más prioritarios. Aún así, las tareas podrán utilizar el gain time cuando se consuma todo el tiempo reservado de una ventana temporal.

Este método de razonamiento puede ser mejorado todavía más introduciendo un modelo basado en beneficios. Este planificador está diseñado para un SO basado en servicios, donde diversos agentes proporcionan servicios y otros agentes consumen esos servicios. Es muy razonable suponer que un proveedor no ofrecerá sus servicios de forma gratuita. Lo más lógico es pedir una cierta compensación (económica o de otro tipo) por la ejecución del servicio. En este trabajo llamaremos a estas compensaciones **beneficios**.

En el escenario que estamos presentando, el planificador presumiblemente tratará de maximizar el beneficio mediante la priorización de aquellas tareas que ofrezcan mayores gratificaciones. Es incluso posible tener en cuenta que el beneficio se degrade conforme pase el tiempo por lo que el planificador utilizará un proceso de razonamiento para decidir no sólo que tareas priorizar, sino también a partir de qué momento el beneficio deja de resultarle interesante.

El beneficio que la ejecución de un servicio puede aportar al SO se puede modelar como una función decreciente. Por ejemplo, una función decreciente linear $f(x) = b - ax$ o una función exponencial $f(x) = b * e^{-ax}$, donde x es la variable que indica el instante de tiempo y a y b son los parámetros constantes proporcionados por el cliente para representar el gradiente y el punto de intersección entre la gráfica de la función y el eje y. Dado que el beneficio debe ser siempre mayor de 0, la constante b deberá ser positiva. El cliente es además capaz de definir su propia función de beneficio (por ejemplo: una constante, función escalón, basada en el WCET, etc).

Así, el planificador puede calcular el porcentaje de tiempo de procesador (que llamaremos Ψ) que una tarea obtiene utilizando la ecuación (9.2), donde Pr es la prioridad estática asignada (un valor de Pr bajo indica una alta prioridad), Pr_{max} indica la máxima prioridad estática posible y b_{max} indica el máximo beneficio asignable establecido por el propio SO.

$$\Psi = \frac{Pr_{max} - Pr}{Pr_{max}} * \frac{b}{b_{max}} * (1 - a) \tag{9.2}$$

El valor Ψ no tiene en cuenta la cantidad de tiempo de procesador libre que queda disponible en la ventana temporal (llamaremos a este tiempo restante φ), así que Ψ se ajusta siguiendo la ecuación (9.3). En

definitiva, la función `append` para el Deadline Prediction Scheduler con Razonamiento basado en Beneficios se muestra en el Algoritmo 5.

$$\Psi = min(\varphi, \frac{Pr_{max} - Pr}{Pr_{max}} * \frac{b}{b_{max}} * (1 - a)) \tag{9.3}$$

A continuación vamos a realizar una traza de ejemplo mostrando cómo diversas tareas son añadidas a este planificador y cómo su porcentaje de procesador y si predicción de deadline son calculados. Para ello utilizaremos la ecuación (9.3) para calcular el porcentaje de procesador asignado y utilizaremos una función exponencial para calcular los beneficios ($B(x) = b * e^{-ax}$). En este ejemplo vamos a asumir que $Pr_{max} = 20$ y $b_{max} = 1000$. El escenario inicial queda definido de la siguiente forma:

Tarea	t_a	$\mathcal{WCET}$	Pr	a	b
τ_1	1	16	8	0.001	900
τ_2	2	14	1	0.0	300
τ_3	4	10	3	0.005	800

1. En primer lugar entra τ_1 al sistema. Aplicando la ecuación (9.3) calculamos $\Psi = \frac{20-8}{20} * \frac{900}{1000} * (1 - 0{,}001) = 0{,}53946$. Dado que el tamaño de ventana es $\omega = 8$, el slot asignado a esta tarea es $\lfloor \omega * \Psi \rfloor = 4$.

2. Esto significa que la tarea τ_1 tiene 4 quantums garantizados en cada ventana, por lo que su deadline será el final de la cuarta ventana temporal $D_{\tau_1} = 5$. Si la tarea finaliza en ese instante de tiempo ($x = (D_{\tau_1} - 1) * \omega = 32$) su beneficio será $B(32) = 900e^{-0{,}001*32} = 871{,}65$.

Algoritmo 5: DPS con Razonamiento basado en Beneficios: Función Append

Input : Tarea τ con tiempo de ejecución en el peor caso $WCET_\tau$
Input : Ventana temporal inicial t_a para ejecutar la tarea τ
Input : Tamaño de ventana ω
Input : Constantes de beneficio a y b
Input : Beneficio máximo b_{max}
Input : Prioridad máxima Pr_{max}
Output: Deadline D_τ para la tarea τ

```
1  Window ← t_a;
2  if Window ≤ CurrentWindow then
3  |  Window ← CurrentWindow +1
4  end
5  while WCET_τ > 0 do
6  |  D_τ ← Window;
7  |  Pr ← getPriority(τ);
8  |  φ ← getFreeSpace(Window);
9  |  Ψ ← min(φ, (Pr_max − Pr)/Pr_max * b/b_max * (1 − a));
10 |  if Ψ > 0 then
11 |  |  EnqueueTaskAtWindow(τ, Ψ, Window);
12 |  |  WCET_τ ← WCET_τ − ⌊Ψω⌋;
13 |  end
14 |  Window ← Window +1;
15 end
```

3. En ω_2 la tarea τ_2 se activa. Ψ es $min(\varphi, \frac{20-1}{20} * \frac{300}{1000} * (1 - 0{,}0)) = min(1 - 0{,}53946, 0{,}285) = 0{,}285$. Esto proporciona un slot de 2 a la tarea τ_2, mientras el nuevo φ es $1 - 0{,}53946 - 0{,}285 = 0{,}17554$.

4. Con 2 quantums garantizados en cada ventana y un $\mathcal{WCET} = 14$,

τ_2 necesitará al menos 7 ventanas para finalizar su ejecución. Por lo tanto, su predicción de deadline será $D_{\tau_2} = 9$, como puede verse en la Figura 9.4.

5. Finalmente, τ_3 se activa en ω_4. El porcentaje de procesador asignado es $\Psi = \frac{20-3}{20} * \frac{800}{1000} * (1 - 0{,}005) = 0{,}6766$. Pero, dado que en ω_4 el valor $\varphi > \Psi$, el porcentaje de procesador será $\Psi = \varphi = 0{,}17554$ en la primera ventana. En ω_5 el valor $\varphi = 1 - 0{,}285 = 0{,}715$, por lo que $\Psi = min(0{,}715, 0{,}6766) = 0{,}6766$.

6. La ventana ω_4 se encuentra ahora completamente reservada, por lo que los quantums asignados a τ_3 serán todos los quantums restantes, que son 2. Desde ω_5 hasta consumir su $\mathcal{WCET}$, el procesador garantizado será $0{,}6766$, lo que se traduce en 5 quantums. Con esto, el deadline de τ_3 es $D_{\tau_3} = 7$.

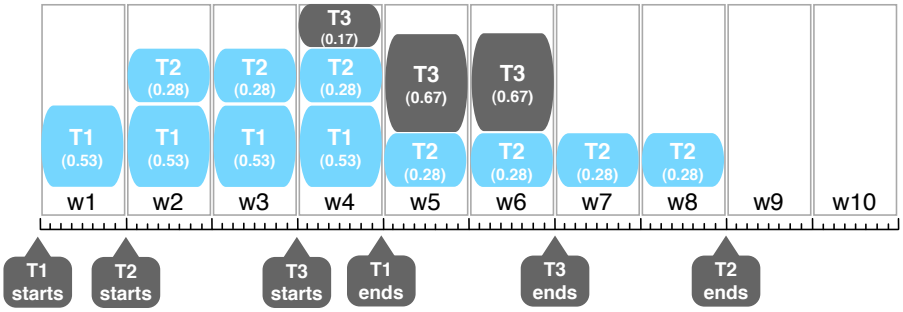


Figura 9.4: Ejemplo de traza de la función append del algoritmo BDPS

Como hemos podido ver en esta sección, el planificador BDPS es un algoritmo que trata de maximizar el beneficio que puede extraer de la ejecución de las tareas, pero siempre tratando de mantener las restricciones deseadas como el uso repartido y justo de la CPU y el cálculo

preciso de una predicción de deadline que nos permita establecer compromisos temporales con los clientes.

9.2. Experimentos y Resultados

Con el fin de poder evaluar el nuevo planificador presentado en este trabajo, se ha modificado el simulador del Capítulo 8 para poder probar todas las características y ventajas proporcionadas por un sistema operativo que implemente el planificador basado en beneficios con predicción del deadline.

Esta evolución del simulador nos permitirá evaluar cómo se comporta el planificador en un entorno distribuido basado en servicios donde los parámetros de tiempo y calidad (medida como beneficio) son factores clave.

En esta sección presentaremos cómo funciona este simulador y cómo se han desarrollado los diferentes experimentos mediante el análisis de diversas aproximaciones a la propuesta y cómo dichas aproximaciones han contribuido al funcionamiento del sistema operativo. Además compararemos estos resultados con un conjunto representativo de algunas de las técnicas de planificación más conocidas.

9.2.1. El simulador

Este simulador es una evolución del software desarrollado en el Capítulo 8, donde se desarrolló un entorno distribuido de acuerdo al paradigma de Computación Distribuida orientada a Objetivos, presentado también en [PNJGF12]. En este entorno distribuido cada uno de los nodos representa a un sistema operativo orientado a objetivos que

ofrece servicios al resto de nodos de la red. Cada nodo de SO implementa además las diferentes versiones del planificador presentado en este capítulo.

En este simulador de SO podemos modificar determinados parámetros que cambian las condiciones del entorno donde se ejecuta el SO, así como la configuración del mismo sistema operativo. Así, podemos cambiar la política de planificación, el número de tareas concurrentes lanzadas en el experimento, el tamaño de las ventanas temporales y de las ráfagas de quantums, etc.

Utilizando el sistema de scripting que se desarrolló para este simulador podemos describir escenarios y programar eventos a ocurrir durante la prueba. En esta configuración podemos cambiar, entre otras cosas, el número de nodos de la red, el número de agentes, la distribución de los servicios por los diferentes agentes, etc. También es posible definir cargas aleatorias para el planificador, lo cual es útil cuando deseamos evaluar un mismo escenario con tantas cargas diferentes como sea posible.

Estos experimentos han sido desarrollados utilizando la misma metodología, con un mínimo de 1000 repeticiones. Se ha comprobado que, tras 1000 repeticiones, la desviación estándar converge. Se ha aplicado a los resultados el *test de significancia* t-student. Con este test se ha podido probar que las diferencias entre las diversas aproximaciones comparadas en el experimento son estadísticamente significativas. Es decir, la probabilidad de obtener el mismo resultado para diferentes aproximaciones (lo que sería nuestra *hipótesis nula*) es muy baja (por debajo de $0{,}05$).

Todos los experimentos se han desarrollado sobre un mismo escenario. El escenario ha sido configurado de la siguiente manera:

- Dos nodos de SO: uno llamado *ClientHost*, que contiene un agente llamado *ClientAgent*, y otro nodo llamado *ProviderHost*, que contiene un agente llamado *ProviderAgent*.
- ProviderAgent es un agente que ofrece hasta 500 servicios atómicos diferentes. Dado que son servicios atómicos, cada uno de ellos será representado por una tarea en el planificador.
- ClientAgent es el agente que invoca cada uno de los servicios registrados. La carga del sistema es activada al inicio de la simulación, lo cual significa que el conjunto de servicios es activado al comienzo del experimento.

Dependiendo del experimento, algunos parámetros serán cambiados en la simulación. De todas formas, existe un valor por defecto para todos estos parámetros cuando no se especifica de otra forma:

- El WCET de cada servicio se asigna siguiendo una distribución gaussiana entre 1 y 200 unidades de tiempo.
- El tamaño por defecto de una ventana temporal es de 256.
- El tamaño por defecto de una ráfaga de quantum es de 64.
- La función de beneficio por defecto es una función exponencial ($f(x) = b * e^{-ax}$), donde los parámetros a y b son asignados siguiendo una distribución normal. Es obligatorio que el parámetro a se encuentre en el rango $[0 - 1]$. El parámetro b debe encontrarse en el rango $[1 - 500000]$, lo cual será suficiente para el propósito de estos experimentos.

A continuación presentamos los experimentos llevados a cabo en este trabajo. Se han dividido en cuatro baterías de pruebas. El primer

experimento analiza los tres refinamientos del planificador con predicción del deadline para comprobar la precisión en el cálculo de las predicciones. El segundo experimento se centra en los beneficios, comparando los tres planificadores de este trabajo con algunos de los planificadores más conocidos. El tercer experimento analizará algunas de las métricas más relevantes para planificadores, como la utilización, el rendimiento y algunas medidas de tiempo. Se compararán estos resultados con el conjunto representativo de planificadores seleccionado para esta batería de pruebas. Finalmente, en el último experimento nos centraremos en el algoritmo BDPS, dado que es el más evolucionado de los tres, tratando de encontrar los parámetros de configuración que mejoren el comportamiento del planificador.

9.2.2. Experimento 1: Precisión de la predicción del deadline

Este primer experimento se ha diseñado para comparar los tres refinamientos del Planificador con Predicción de Deadline. La principal diferencia entre ellos reside en la precisión que son capaces de alcanzar al predecir el instante de tiempo en que una tarea va a terminar (*deadline*). Cada uno de los refinamientos incluye de manera incremental todas las características de su antecesor. Así, el planificador DPS-Dy incluye las técnicas de reserva de CPU y añade la Promoción Dinámica de Prioridades. Del mismo modo, BDPS soporta reserva de CPU y Promoción Dinámica de Prioridades y añade a su vez el Razonamiento basado en Beneficios.

Las tres aproximaciones del planificador (DPS, DPS-Dy y BDPS) han sido comparadas en este experimento para poder comprobar que la precisión de la predicción de deadline mejora con la técnica de Promoción Dinámica de Prioridades.

Este experimento ejecuta la simulación con diferentes conjuntos de servicios (de 10 a 500 servicios en pasos de 10) y compara la diferencia entre el tiempo real de ejecución y el deadline establecido en la predicción. Los resultados se muestran en la Figura 9.5. En esta gráfica se puede observar lo mala que es la precisión del algoritmo DPS, dado que siempre sitúa el deadline en el peor de los casos, sin tener en cuenta las posibles interacciones entre tareas. Sin embargo, DPS-Dy y BDPS son capaces de realizar predicciones mucho más precisas. Esto se debe a que han sido capaces de simular cuándo va a terminar una tarea teniendo en cuenta la Promoción Dinámica de Prioridades. DPS-Dy realiza la promoción teniendo en cuenta el orden de llegada de las tareas, mientras que BDPS utiliza la garantía de porcentaje de CPU basándose en el beneficio estimado de la tarea. Las pequeñas diferencias que pueden observarse entre la predicción del deadline y la ejecución real se deben a que estos algoritmos siguen sin ser capaces de calcular cómo se aprovechará el *gain time*, por lo que la predicción siempre se situará en un límite superior. Cuando todas las tareas que tienen garantizado un porcentaje de CPU han consumido su reserva y la ventana temporal no ha terminado todavía, el planificador entra en modo *gain time*, donde se ejecutan tareas que no han terminado su ejecución utilizando una política diferente.

Highest Benefit First Scheduler

En los experimentos de este trabajo hemos utilizado como política de *gain time* un algoritmo que hemos llamado **Highest Benefit First** o HBF (primero el de más beneficio). Esta política selecciona como siguiente tarea a ejecutar durante la siguiente ráfaga de ejecución a aquella que proporcione el mayor beneficio.

La situación límite para el algoritmo DPS ocurre al calcular el deadline cuando la CPU está sobrecargada y nunca entra en modo *gain time*.

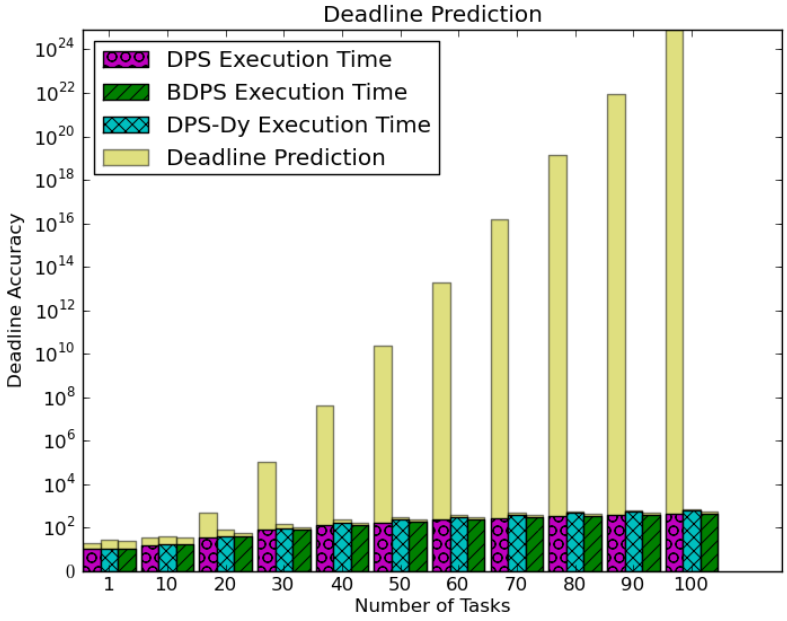


Figura 9.5: Precisión de la predicción del deadline

En este experimento se ha forzado esta situación activando todas las tareas al mismo tiempo y con un mismo WCET. Aún así este es un caso peor poco probable, lo más habitual es que una tarea que finaliza su ejecución deje libre la CPU para que entren otras tareas a utilizarla. Es por ello que en esta situación destacan los algoritmos DPS-Dy y BDPS en el cálculo del deadline, dado que utilizan la Promoción Dinámica de Prioridades presentada en este trabajo.

Los resultados del experimento nos muestran que la aproximación del algoritmo DPS es realmente pesimista, dado que proporciona predicciones de deadline muy altas e irreales, especialmente para un número alto de tareas concurrentes. Por otro lado, DPS-Dy y BDPS muestran

un comportamiento muy similar entre ellas y muy aceptable, muy cercano en definitiva al tiempo real de ejecución. Este experimento nos muestra que la Promoción Dinámica de Prioridades mejora en efecto la precisión del cálculo del deadline.

Se debe tener en cuenta que en un escenario real no conocemos la tasa de llegada de las tareas, lo que dificulta en extremo crear un plan de ejecución off-line sin correr peligro de romper nuestros compromisos.

Sin embargo, la técnica de Reserva de CPU nos permite solventar este problema. Es más, la ventaja que este algoritmo de planificación nos proporciona permite que, incluso cuando tenemos grandes cantidades de tiempo de holgura debido a una baja tasa de llegada de tareas, el planificador todavía será capaz de conseguir mejores tiempos de ejecución y, por lo tanto, mejorar la diferencia entre el tiempo de ejecución real y la predicción de deadline realizada.

9.2.3. Experimento 2: Beneficios

Este experimento ha sido diseñado para comprobar cómo un algoritmo orientado a beneficios ayuda a los proveedores de servicios a alcanzar altas cotas de beneficios. En este experimento compararemos los tres planificadores desarrollados en este trabajo (DPS, DPS-Dy y BDPS) con un conjunto representativo de planificadores bien conocidos.

A continuación vamos a presentar brevemente estos algoritmos de planificación:

FCFS El algoritmo *First-Come First Served* [SGGS98] es un planificador no expulsivo que otorga la CPU a las tareas por estricto orden de llegada. Cuando una tarea toma la CPU mantiene el procesador ocupado hasta que finaliza su ejecución.

RR *Round Robin* [SGGS98] es un algoritmo desarrollado para compartir las CPU entre tareas con la misma cantidad de tiempo. Round Robin divide el tiempo en ráfagas de quantums y otorga la CPU de manera cíclica a cada tarea por la duración de un a ráfaga de quantums.

CFS El *Completely Fair Scheduler* [WTKW08] es un algoritmo que trata de hacer el uso más justo de la CPU. Para ello otorga la CPU a la tarea que menos uso ha hecho del procesador. De este modo trata de maximizar el rendimiento de la interactividad, mientras mantiene una alta utilización de la CPU.

SRTF *Shortest Remaining Time First* [SGGS98] es una evolución del bien conocido algoritmo SJF (Shortest Job First). Este planificador selecciona para ejecutar a continuación a la tarea con el menor tiempo de ejecución. SJF es un algoritmo no expulsivo, mientras que SRTF es la variante expulsiva del algoritmo. Esta variante tiene en cuenta el tiempo de ejecución restante, en lugar de todo el tiempo de ejecución. SRTF toma la decisión de asignar la CPU cuando una tarea llega al sistema o cuando una tarea termina. SJF y SRTF tienen un rendimiento mejor que FCFS. Sin embargo, al contrario que FCFS, estos algoritmos tienen el potencial peligro de *inanición*. La inanición ocurre cuando una tarea muy grande nunca llega a conseguir entrar en la CPU debido a que tareas más pequeñas entran continuamente en la cola de preparados.

Además vamos a comparar estos planificadores con un algoritmo

desarrollado con el único propósito de optimizar el beneficio: *Highest Benefit First* (HBF). HBF es un algoritmo no expulsivo, originalmente diseñado para la fase de *gain time* de BDPS en lugar de políticas menos optimizadas como FCFS, selección aleatoria, etc. Sin embargo hemos desarrollado con propósitos experimentales la política HBF como un planificador completo e independiente que puede ser comparado con el resto de planificadores.

El experimento en cuestión consiste en un conjunto de tareas ejecutadas por todos estos algoritmos. La carga del sistema será la misma para cada uno de los algoritmos, por lo que el único cambio será el orden en que se ejecutan las tareas, dependiendo del algoritmo de planificación. Al final del experimento hemos calculado cuánto beneficio ha sido obtenido globalmente por el agente proveedor del SO servidor.

En este experimento cada tarea utiliza una función exponencial decreciente ($f(x) = b * e^{-ax}$), donde los parámetros a y b han sido asignados siguiendo una distribución gaussiana. El parámetro a se encuentra en el rango $[0 - 1]$. Para el parámetro b hemos seleccionado un rango menor ($[1 - 500]$) para hacer los resultados más manejables. Cada vez que una tarea deja la CPU se calcula el beneficio parcial siguiendo la función $f(x)$.

La Figura 9.6 muestra los resultados del experimento. Se puede observar en la gráfica que el algoritmo que alcanza los mayores valores de beneficio es el del planificador HBF. Este es un resultado esperado dado que el algoritmo está optimizado para ejecutar siempre la tarea que más beneficio le va a dar a corto plazo. Cada tarea aporta una parte del beneficio siempre que sale de la CPU, razón por la cual el algoritmo Round Robin también alcanza altas cotas de beneficio, puesto que comparte de forma igualitaria la CPU con todas las tareas.

El algoritmo BDPS cumple todas las condiciones deseadas presen-

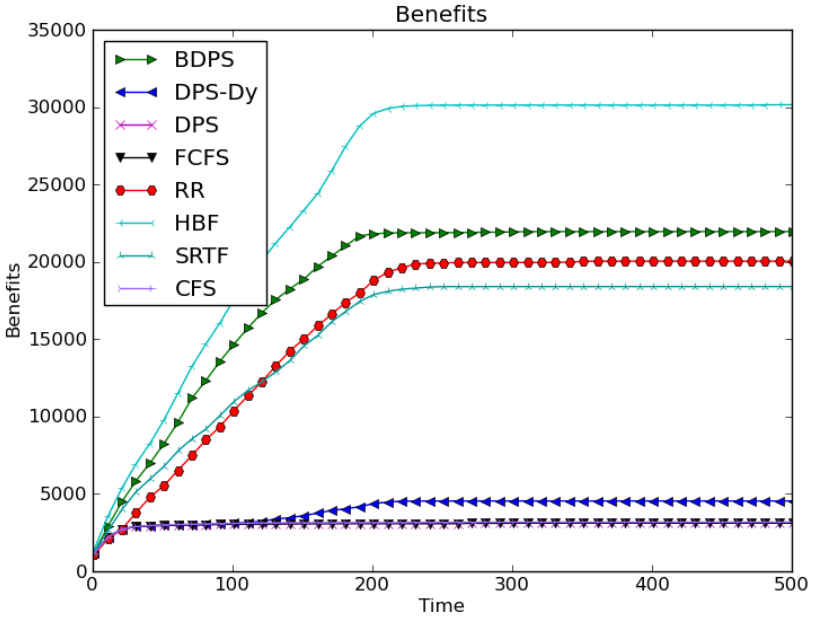


Figura 9.6: Experimento de resultados de beneficios

tadas en este trabajo: BDPS comparte el tiempo de CPU entre todas las tareas utilizando la función de beneficio y utilizando ventanas temporales donde el procesador reservado ha sido asignado en ráfagas de quantums. Por esta razón, el planificador alcanza altas cotas de beneficio sin peligro de inanición para ninguna tarea ni problemas en el rendimiento de la interactividad. Además, es capaz de realizar predicciones del deadline de manera precisa, lo que permite a los agentes establecer compromisos temporales correctos con sus clientes.

El planificador SRTF tiene una de las cotas más altas de tareas ejecutadas por unidad de tiempo. Esto hace que el algoritmo alcance valores muy altos de beneficio, cercanos a los del algoritmo Round Robin.

Los algoritmos FCFS, DPS y DPS-Dy se encuentran en la misma banda de beneficios. Ninguno de ellos tiene en cuenta los beneficios a la hora de planificar y tampoco reparten la CPU de manera justa entre las tareas. Todos estos algoritmos reparten la CPU por orden de llegada, lo cual tiene un efecto muy negativo en los valores de beneficio obtenidos.

En este experimento hemos comprobado cómo el algoritmo BDPS alcanza muy buenos valores de beneficio. Sin embargo existe otro planificador, el HBF, que consigue valores de beneficio todavía mejores. Aún así, como veremos en próximos experimentos, existen otras métricas que deben ser tomadas en cuenta, como el tiempo de respuesta, la utilización o la predicción del deadline. En futuros experimentos veremos que la métrica de predicción del deadline invalidará la opción del algoritmo HBF, dado que ésta será un requisito imprescindible para poder aceptar tareas en el sistema y respetar al mismo tiempo los compromisos temporales.

9.2.4. Experimento 3: Métricas de planificación

En este experimento vamos a presentar algunas métricas comunes de planificadores [Tan08] que nos permitirán comparar los algoritmos desarrollados en este trabajo con algunos de los algoritmos de referencia en términos de rendimiento, utilización, etc. Hemos seleccionado cinco métricas para comparar los mismos algoritmos del experimento anterior.

Las métricas seleccionadas son las siguientes:

Utilización (Utilization) En este trabajo definimos esta métrica como la ocupación de la CPU con respecto al peso del planificador. La

definición de esta métrica puede verse en la ecuación (9.4), donde nQ es la cantidad total de quantums utilizados, nCS es el número de cambios de contexto realizados y cCS es el coste de un cambio de contexto.

$$U = \frac{nQ}{nQ + nCS * cCS} \tag{9.4}$$

Rendimiento (Throughput) Mide el número de tareas (nT) que han completado su ejecución por unidad de tiempo (ecuación 9.5).

$$T = \frac{nT}{nQ + nCS * cCS} \tag{9.5}$$

Tiempo de ejecución (Turnaround Time) El tiempo de ejecución o tiempo de *turnaround* es la cantidad de tiempo que ha sido necesaria para ejecutar una tarea. Esta medida se realiza desde el instante en que una tarea es activada hasta su instante de finalización. Habitualmente este valor se calcula como el tiempo medio de todos los tiempos de ejecución de las tareas.

Tiempo de espera (Waiting Time) Esta métrica mide la cantidad de tiempo que una tarea ha estado esperando en la cola de preparados. Este valor será siempre inferior al tiempo de ejecución, puesto que éste incluye el tiempo de espera y el tiempo que ha estado dentro de la CPU.

Tiempo de respuesta (Response Time) Es la cantidad de tiempo que una tarea ha necesitado desde su instante de activación hasta que se ejecuta por primera vez. En este trabajo mediremos el tiempo de respuesta como el instante de tiempo en que una tarea consigue entrar por primera vez en la CPU.

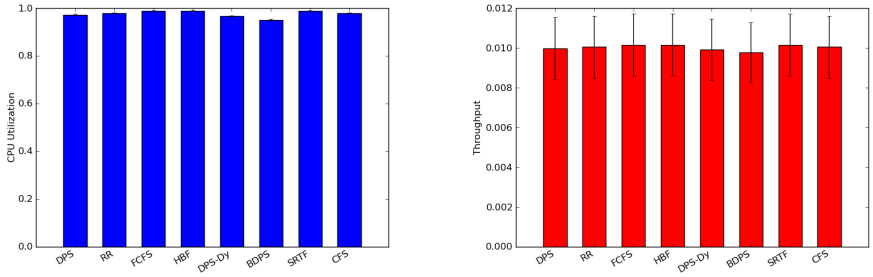


Figura 9.7: Utilización y Rendimiento

La Figura 9.7 nos presenta los resultados obtenidos para las métricas de utilización y rendimiento de cada uno de los algoritmos implementados. Podemos observar cómo la gran mayoría de algoritmos tienen una buena proporción de utilización, dado que este valor es completamente dependiente del número de cambios de contexto que realiza el planificador. El algoritmo BDPS es uno de los que más cambios de contexto ha realizado en estos experimentos. Esto es debido a que el tamaño de ventana adoptado ha sido cuatro veces mayor que el tamaño de ráfaga de los quantums, lo que provoca un número potencialmente alto de cambios de contexto en cada ventana temporal dado que BDPS no siempre utiliza una ráfaga de quantum completa antes de cambiar de tarea en ejecución. La CPU reservada para una tarea en una ventana temporal puede ser menor que el tamaño de ráfaga establecido (dependiendo del beneficio que la tarea proporciona y de su prioridad estática), por lo tanto en término medio se realizan más cambios de contexto que en otros algoritmos justos como RR o CFS. Aún así, la utilización del algoritmo BDPS está por encima del 90 %, lo cual es un ratio muy aceptable. Los algoritmos con mayor valor de utilización son los no expulsivos, puesto que al no intercambiar la CPU entre las tareas

minimizan el número de cambios de contexto. El valor de la métrica de rendimiento está muy relacionado con la métrica de la utilización. De hecho, los resultados son muy similares a los de utilización de la CPU.

La Figura 9.8 nos muestra el Tiempo Medio de Ejecución (Turnaround) y el Tiempo Medio de Espera. Podemos confirmar que el algoritmo SRTF es el que mejores valores de turnaround (y por tanto también de espera) proporciona, dado que siempre selecciona la tarea que está más próxima a terminar su ejecución. De todas formas, todos los algoritmos *justos* (RR, CFS, DPS, DPS-Dy y BDPS) que comparten la CPU de forma equitativa tienen, como contrapartida, mayores tiempos de turnaround y mayores tiempos de espera en la cola de preparados. En el término medio encontramos a otro algoritmo no expulsivo (FCFS) y al algoritmo diseñado a medida HBF.

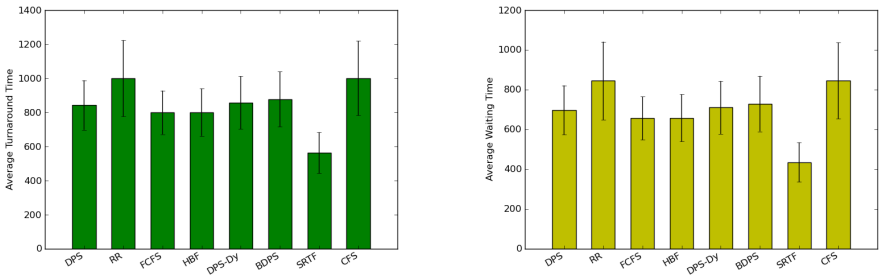


Figura 9.8: Tiempo medio de espera y de turnaround

La siguiente prueba, representada en la Figura 9.9, muestra el Tiempo Medio de Respuesta para cada uno de los algoritmos analizados. El tiempo de respuesta es una métrica muy interesante que nos marca la velocidad con la que el usuario recibe una interacción de la tarea

en ejecución. Esto se debe a que esta métrica mide cuánto tiempo tarda una tarea en entrar en la CPU desde su instante de activación. El grupo de algoritmos *justos* tiene tradicionalmente un buen tiempo de respuesta medio, dado que da las mismas oportunidades a todas las tareas. Ocurre lo contrario con los algoritmos con alto riesgo de inanición (como FCFS y HBF), dado que una tarea puede verse bloqueada durante largos periodos de tiempo si tareas más prioritarias están entrando persistentemente en el sistema.

El algoritmo BDPS tiene una posición especial en esta métrica, dado que obtiene un muy buen tiempo de respuesta. Esto es debido al uso de ventanas temporales puesto que dentro de cada ventana temporal el algoritmo divide el tiempo en ráfagas de un tamaño dependiente de los parámetros de beneficio. Por esta razón todas las tareas, incluso las menos importantes, tienen una oportunidad de entrar en la CPU en un corto periodo de tiempo. Ni el algoritmo de DPS ni el de DPS-Dy presentan resultados de tiempo de respuesta tan buenos como los de BDPS. Esto es debido a que tanto DPS como DPS-Dy utilizan la misma técnica para reservar la CPU (reservar la mitad del tiempo restante de la ventana). Esta técnica tiene una peor granularidad, soportando menos tareas por ventana temporal, lo cual retrasa la entrada de las últimas tareas en llegar.

Para terminar, en la Tabla 9.1 mostramos una comparación cualitativa de todos los algoritmos utilizando una distribución normal sobre la diferencia de los errores. La tabla utiliza la siguiente leyenda: ● indica que el algoritmo tiene un buen resultado en la métrica considerada. ○ indica que el algoritmo tiene un mal resultado en la métrica considerada. ◑ indica que el algoritmo tiene un valor medio en la métrica considerada, lo cual significa que el planificador no se distingue por esta métrica. Finalmente, ⊠ indica que la métrica no es aplicable para

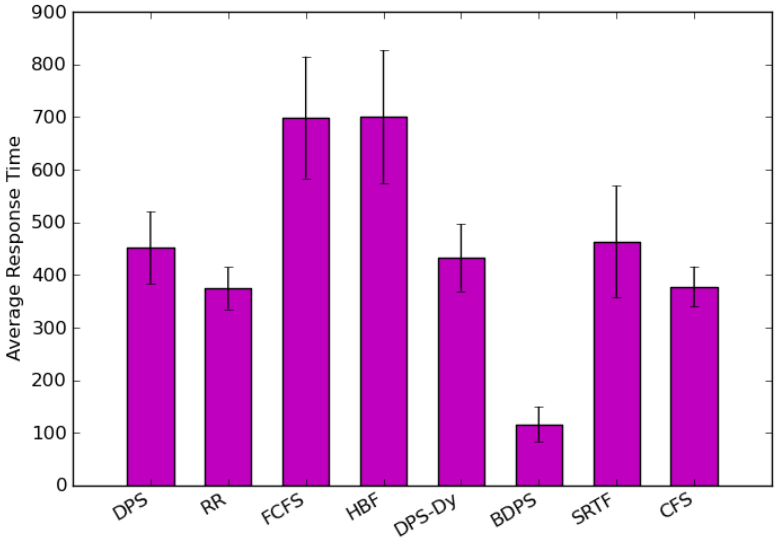


Figura 9.9: Tiempo medio de respuesta

este algoritmo, lo que invalida al planificador para los propósitos del caso de estudio de este trabajo.

En la tabla resumen se puede apreciar que no existe ningún algoritmo que obtenga la máxima puntuación en todas las métricas. El planificador SRTF tiene una de las mejores puntuaciones en la medida en que presenta muy buenos valores de ejecución y de *turnaround* y al mismo tiempo tiene valores aceptables de beneficio y tiempo de respuesta. De todas formas, la imposibilidad de calcular una predicción de deadline segura lo invalida para el propósito de este trabajo. La aproximación no expulsiva (SJF) podría simplificar el proceso de calcular el deadline, pero una aproximación no expulsiva siempre tendrá el peligro de la

	Benefit	Utilization	Turnaround	Waiting	Deadline
DPS	○	◑	◑	◑	○
DPS-Dy	○	○	◑	◑	●
BDPS	◑	○	◑	●	●
HBF	●	●	◑	○	⊠
FCFS	○	●	◑	○	●
RR	◑	◑	○	◑	⊠
CFS	○	◑	○	◑	⊠
SRTF	◑	●	●	◑	⊠

Tabla 9.1: Tabla comparativa

inanición y problemas de interactividad [Tan08].

Además de los nuevos algoritmos desarrollados en este trabajo, el único algoritmo que es capaz de realizar una predicción de deadline segura es el del planificador FCFS. Dado que es un algoritmo no expulsivo que ordena la ejecución según el orden de llegada, y siempre dando por hecho que disponemos de un WCET correctamente calculado, sería sencillo conocer cuándo una tarea va a finalizar su ejecución. Sin embargo, el algoritmo de FCFS presenta muy malos resultados para las métricas de beneficio y de tiempo de respuesta.

Los resultados presentados en este experimento nos muestran que el algoritmo BDPS obtiene buenos resultados para la mayoría de las métricas, dando resultados aceptables para el beneficio, así como buenos tiempos de turnaround, y presentando resultados especialmente buenos en los tiempos de respuesta y predicciones de deadline muy precisas. En definitiva esto hace de BDPS la mejor elección para la arquitectura de sistema operativo presentada en este trabajo y en [PJGF11, PNJGF12].

9.2.5. Experimento 4: Puesta a punto del planificador BDPS

Una vez hemos seleccionado a BDPS como el algoritmo de planificación más apropiado para la arquitectura de SO basada en el paradigma de Computación Distribuida orientada a Objetivos, vamos a centrarnos en la puesta a punto del algoritmo. El propósito de este experimento es encontrar la mejor configuración del planificador, poniendo especial atención en los parámetros que mejor afinan el rendimiento del mismo. Hemos seleccionado dos parámetros que pueden ser modificados con el fin de observar cómo cambia el comportamiento del planificador y, por tanto, cómo cambian los resultados de las métricas que nos indican cuan bueno es el algoritmo. Los dos parámetros que vamos a modificar son el tamaño de la ventana temporal y el tamaño de la ráfaga de quantums.

Hemos diseñado un experimento donde se ejecutan 500 tareas en diferentes escenarios, variando el tamaño de la ventana temporal desde 64 a 2048. El tamaño máximo de la ráfaga de quantums se ha establecido a un cuarto de la ventana temporal. El escenario para cada prueba ha sido replicado, cambiando únicamente el tamaño de la ventana. En la Figura 9.10 podemos apreciar cómo las métricas de utilización y de rendimiento crecen conforme incrementamos el tamaño de la ventana temporal. Esto se debe a que mayores valores de ventana generan menos cambios de contexto, lo que deja más tiempo para ejecutar tareas.

La Figura 9.11 presenta los resultados para el *Tiempo Medio de Turnaround* y el *Tiempo Medio de Espera* de los mismos experimentos. En estas gráficas podemos observar como los tiempos medios tienen sus valores más bajos para valores intermedios del tamaño de ventana.

Podemos comprobar como, conforme crece el tamaño de ventana, se obtienen mejores tiempos de *turnaround*. Esto se debe a que ventanas

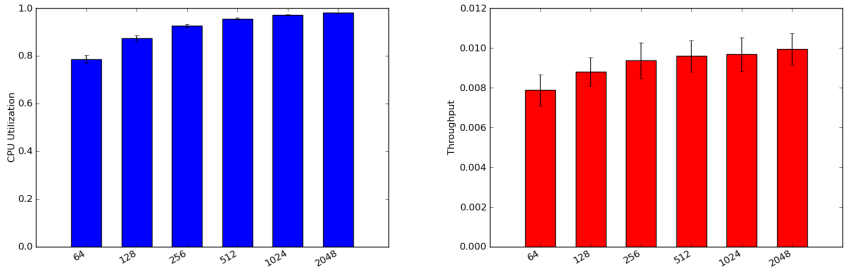


Figura 9.10: Tamaño de ventana: Utilización y Rendimiento

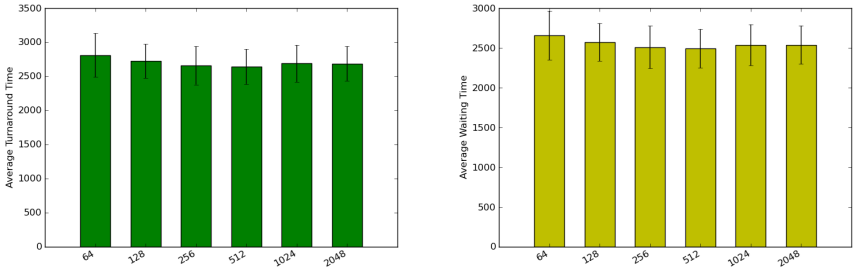
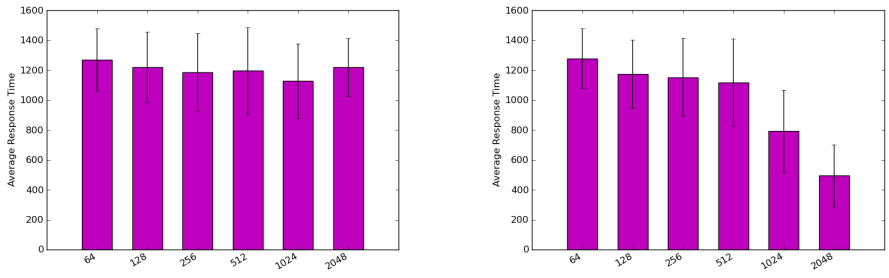


Figura 9.11: Tamaño de ventana: Tiempos de Turnaround y de Espera

más grandes permiten al planificador reservar mayores slots de CPU, lo que hace que el planificador se comporte de manera similar a un planificador no expulsivo. Este comportamiento se compensa usando las ráfagas de quantum tal y como hacen los algoritmos justos, lo cual permite al planificador compartir el tiempo de CPU de forma justa y evitando el riesgo de inanición. De todas formas, si incrementamos demasiado el tamaño de las ráfagas de quantum nos encontramos con la misma situación que teníamos al incrementar desmesuradamente el

tamaño de las ventanas temporales.

Es importante notar que los valores del tamaño de ventana son directamente proporcionales con los del tamaño de la ráfaga de quantums. Esto significa que con tamaños grandes de ventana temporal y pequeños de ráfagas de quantum obtendremos un número elevado de cambios de contexto, lo que afectará a la métrica de la utilización. Sin embargo por otro lado obtendremos buenos valores de tiempo de respuesta. Para comparar estas interacciones hemos desarrollado una nueva prueba donde mantenemos el tamaño de la ráfaga de quantum constante en un valor pequeño e incrementamos el tamaño de ventana temporal. La Figura 9.12 compara los valores de tiempo de respuesta para diferentes tamaños de ventana con un tamaño fijo de ráfaga. En (a) el tamaño de ráfaga se fija a un cuarto del tamaño de ventana, mientras que en (b) se fija a un valor constante de 32.



(a) (b)

Figura 9.12: Tamaño de ventana: Tiempos de Respuesta variando el tamaño de ráfaga de quantums

Estos resultados nos muestran que, tal y como esperábamos, valo-

res altos de tamaño de ventana y bajos de tamaño de la ráfaga de quantums nos proporcionan los mejores valores de tiempo de respuesta para el algoritmo de BDPS. De todas formas, existen ciertas contrapartidas que deben ser tenidas en cuenta con esta configuración. Cuanto más pequeño sea el tamaño de ráfaga, menor utilización tendrá el planificador. Y lo que es más importante aún, dado que la predicción de deadline se ajusta siempre al final de una ventana (dado que podemos predecir en qué ventana terminará una tarea, pero no en qué instante de la ventana), valores muy altos del tamaño de la ventana temporal perjudicarán la precisión del cálculo de la predicción del deadline, como puede verse en la Figura 9.13, donde los valores de ventana más altos (como 1024 y 2048) dan predicciones tremendamente pesimistas con respecto al tiempo de ejecución real.

En resumen, en estos experimentos hemos comprobado cómo el tamaño de la ventana temporal y el tamaño de las ráfagas de quantum afectan directamente al comportamiento del algoritmo de BDPS. Una buena proporción entre ambos parámetros puede decrementar el tiempo de respuesta significativamente. De todas formas, dependiendo de la métrica que nos interese potenciar, el SO puede seleccionar diferentes valores para estos parámetros. Altos valores del tamaño de ventana proporcionarán una buena utilización y rendimiento, pero deteriorarán la predicción del deadline y los tiempos de espera y de turnaround. Este experimento nos ha mostrado que, con un ratio de 1 a 4 entre el tamaño de ventana y el de ráfaga, valores intermedios de tamaño de ventana (alrededor de 512) devuelven resultados medios aceptables para estas métricas.

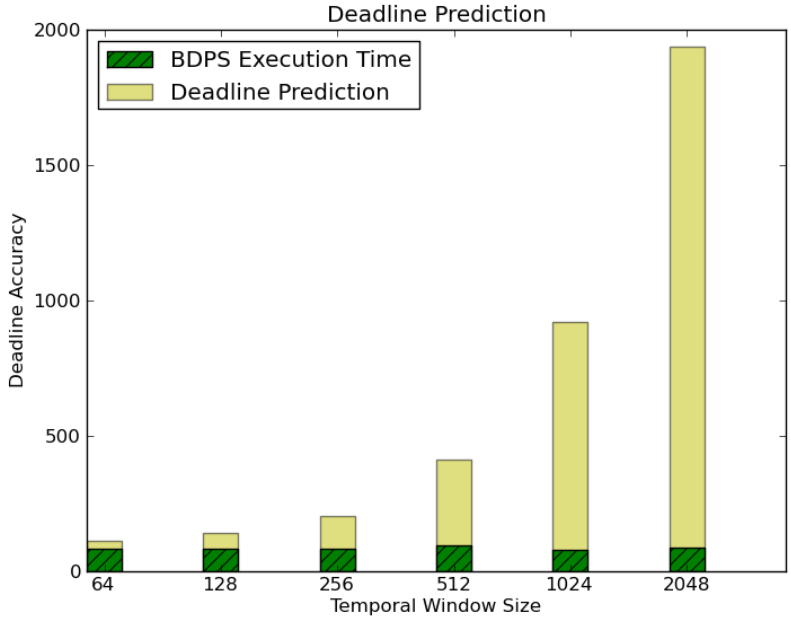


Figura 9.13: Precisión de la predicción de deadline para diferentes tamaños de ventana

9.3. Consideraciones finales

En este capítulo hemos presentado un algoritmo de planificación que puede ser utilizado para predecir el instante de finalización de una tarea en un Sistema Operativo que implemente el paradigma de Computación Distribuida orientada a Objetivos. Este planificador es útil para priorizar aquellas tareas con funciones de beneficio ventajosas y para establecer compromisos temporales con los clientes que han invocado las tareas.

Garantizar el tiempo de finalización de una tarea en ejecución es

un proceso complicado si deseamos establecer estas garantías en tiempo de ejecución sin conocer la frecuencia de llegada de las tareas y, además, tener en cuenta parámetros proporcionados por los clientes como el beneficio obtenido de la ejecución de la tarea. El algoritmo BDPS es capaz de crear un plan de ejecución que predice cuándo van a completarse las tareas en ejecución en el peor de los casos. Esto se consigue mediante técnicas de garantía de procesador. BDPS reserva el recurso de CPU de acuerdo a la prioridad del cliente y a su función de beneficio.

En resumen, los experimentos de este capítulo han demostrado que el planificador BDPS encaja perfectamente en un entorno de Computación Distribuida orientada a Objetivos. Es capaz además de establecer buenas predicciones de deadline, permitiendo ser competitivo a la hora de conseguir compromisos temporales con sus clientes. BDPS ha demostrado además un muy buen tiempo medio de respuesta, lo cual es un indicador de que el reparto de procesador entre tareas se hace de forma bastante equitativa. Una de las mayores ventajas de BDPS es que crea un plan de ejecución utilizando el parámetro del beneficio, lo que permite al SO ordenar la ejecución de las tareas maximizando el potencial beneficio que puede conseguir. Además, el resto de métricas analizadas en este capítulo han resultado muy similares a las de otros planificadores conocidos y comparados en este trabajo.

10

Caso de Uso

Un Sistema Operativo que implementa el paradigma de Computación Distribuida basada en Objetivos está diseñado para funcionar idealmente en entornos orientados a servicios, en entornos Grid o en entornos de procesos de negocios (BPEL). En estos entornos existen multitud de nodos interconectados donde se ofrecen multitud de servicios proporcionados por diferentes proveedores. Estos proveedores pueden ser empresas de servicios, particulares, proyectos comunitarios, etc. Por ejemplo, la empresa Google posee una gran red de nodos que ofrecen servicios de computación en la nube (buscador, traductor, mapas, etc). Estos servicios pueden utilizarse mediante interfaces de

programación proporcionadas por la propia empresa o mediante la interfaz web proporcionada a través de un navegador. También pueden seguir algún estándar de servicios web. En el caso de este trabajo seguimos el estándar OWL-S.

Es en estos entornos donde el SO propuesto en este trabajo adquiere su máximo interés. Un ejemplo claro son las redes de procesos de negocio donde el B2B (Business 2 Business) o B2C (Business 2 Client) marcan el flujo de trabajo. Diversas empresas ofrecen sus servicios para ser utilizados por otras empresas o por clientes. Estos servicios habitualmente tienen un coste, por lo que el cliente les pagará un beneficio al finalizar el servicio. Pero igualmente los clientes exigen unos requisitos mínimos de calidad, como puede ser la velocidad a la que se sirve el servicio, la seguridad del mismo o la fiabilidad.

El Caso de Uso que presentamos en este capítulo se centra en un entorno domótico, donde un usuario expone un deseo con ciertas restricciones y el sistema domótico del hogar tratará de llevar a cabo el deseo de su usuario expresado como un objetivo. En este ejemplo entrarán en juego tanto servicios ofrecidos por los diferentes nodos computacionales del hogar domótico de forma pervasiva o ubicua [Red06, MCB08], como servicios externos con los que se puede interactuar como serán en este caso servicios de pago electrónico o de alquiler de películas.

10.1. Descripción del Caso de Uso

El caso de uso que se presenta a continuación se centra en el visionado de una película por parte de un usuario de un hogar inteligente. En este escenario el usuario expresa a su hogar inteligente su objetivo de ver determinada película a la hora a la que llegue a casa. La forma de introducir el objetivo en el sistema es variada, el usuario puede hacerlo

mediante un SMS, mediante una aplicación de su iPad o mediante una conversación por mensajería instantánea. En cualquier caso, la aplicación que recibe la intención del usuario la transforma en un objetivo que es introducido en el Sistema Operativo orientado a Objetivos del hogar inteligente.

Dado que existen diversas formas de ver una película hoy en día (como el alquiler en una tienda on-line, la descarga mediante protocolos P2P, o los servicios de streaming de vídeo) el usuario únicamente expresa que desea ver la película, la forma de conseguirla será responsabilidad del motor deliberativo. Sin embargo podrá poner determinadas restricciones y determinados parámetros de entrada. Un parámetro de entrada será, por ejemplo, el título de la película que desea ver. Otros parámetros pueden ser que no desea pagar más de 5 euros por ver la película y que no desea utilizar como medio de pago el proveedor *Paypal*. Existen incluso determinados parámetros que pueden ser importantes para llevar a cabo este objetivo, como la calidad a la que desea ver la película o el dispositivo en el que desea verla (su televisor inteligente, su iPad, su PC,...). En este ejemplo veremos cómo determinados parámetros necesarios para el cumplimiento de un objetivo no tienen porqué ser especificados en la propia declaración del objetivo.

En las próximas secciones vamos a presentar en detalle este Caso de Uso, mostrando cómo se define el objetivo que lleva al cumplimiento del deseo del usuario de ver la película, cómo se construye el plan y finalmente cómo se seleccionan los servicios que van a ser ejecutados para llevar a cabo el objetivo.

10.2. Objetivo del Caso de Uso

En el caso de uso que nos ocupa hemos determinado que el usuario desea ver una película al llegar a su casa, con una determinada calidad (por ejemplo 720p) y sin pagar más de 5 euros, siempre evitando como método de pago *Paypal*. El medio mediante el cual se introduce el objetivo en el sistema puede variar en función de la interfaz utilizada. El usuario utiliza una aplicación externa que hace la función de agente interfaz y que traslada los deseos del usuario al sistema DGOC como un objetivo activo. En [LE06] se presenta una interfaz para electrónica de consumo que utiliza la base de conocimiento OpenMind Commonsense [Sin02] y ConceptNet [LS04] para transformar los deseos y motivaciones del usuario en objetivos de su sistema. Otro tipo de interfaces que se ven hoy en día son aquellas dirigidas por reconocimiento del lenguaje natural (como Siri en los dispositivos de Apple) o mediante el reconocimiento de gestos (como Kinect en la XBOX de Microsoft).

En el caso de nuestro ejemplo, una vez capturados los requisitos del usuario y transformados en un objetivo obtenemos una representación como la del Listado 10.1.

Listado 10.1: Objetivo del Caso de Uso

```
<goal type='achieve' retry='true' retrydelay='0'
        recur='false' exclude='when_failed'
        name='ViewFilm'>
    <parameter name='film:title'>Casablanca</parameter>
    <parameter name='film:quality'>720p</parameter>
    <targetcondition>
        (video:viewed ?film:title)
    </targetcondition>
    <softcondition>
        (≥ (film:current-quality ?film:title) ?film:quality)
    </softcondition>
    <contextcondition>
        (> (begin (video:prepared ?film:title)) "21:15:00 GMT")
```

```
14      </contextcondition>
15      <dropcondition>
16        (and
17          (> bank:price 5)
18          (= bank:paymentMethod 'PayPal')
19
20          (≤ (end (video:viewed ?film:title)) "23:59:59 GMT" )
21        )
22      </dropcondition>
23      <deliberation>
24              <inhibits rel='ScreenSaver'/>
25      </deliberation>
26  </goal>
```

La `targetcondition` del objetivo representa aquello que el usuario busca conseguir (ver una película). Además el usuario ha representado otras condiciones donde desea expresar determinadas restricciones a su objetivo. En la `softcondition` ha expresado que desearía (nótese el condicional, puesto que es una restricción débil) ver la película con una calidad igual o superior a 720p.

En la `contextcondition` expresa una restricción temporal utilizando el operador `begin`. Mediante este operador se indica el instante en que comienza la ejecución del servicio que genera el estado indicado en el primer operando. De este modo se está indicando que la película no comience a reproducirse antes de las 21:15 horas. Si no se cumpliera esta restricción el Deliberation Engine suspendería la ejecución del plan hasta que todas sus restricciones de contexto se cumplan. En este caso, esperaría a que fuera la hora deseada por el usuario para comenzar la reproducción de la película. Del mismo modo se puede utilizar el operador `end` para indicar una restricción temporal en la finalización de un servicio. Si deseamos aplicar la restricción temporal a todo el plan utilizaremos como operando `current-plan` en lugar del estado generado por un servicio.

Finalmente la `dropcondition` del objetivo expresa diversas restricciones. Exige que el precio pagado por la película sea menor a 5 euros y que el medio de pago no sea *Paypal*. Si se cumple cualquiera de estas condiciones el plan será automáticamente abortado y se iniciará la recuperación o *rollback* de los servicios ejecutados. Existe además otra `dropcondition` con restricciones temporales que indica que quiere terminar de ver la película antes de la medianoche. Cuando la ejecución del plan vaya a comenzar la reproducción de la película el Deliberation Engine preguntará al servicio de reproducción por su tiempo de ejecución en el peor caso, este responderá (como mínimo) con la duración de la película a reproducir (el análisis del WCET puede ser paramétrico [VHMW01]) y de este modo el Deliberation Engine podrá comprobar las restricciones y, en caso de no cumplir alguna tomar las acciones correspondientes. En este caso abortaría el plan y efectuaría un rollback del mismo.

En este objetivo se especifica además una interacción con otro objetivo con identificador *ScreenSaver*. Así, durante la reproducción de la película se inhabilita el salvapantallas.

Con el fin de introducir variables que hagan más completo el ejemplo vamos a definir un objetivo de mantenimiento en el sistema que pueda llevarnos a un posterior conflicto. Pudiendo mostrar así cómo se comportará el sistema cuando detecta conflictos entre objetivos. El objetivo de mantenimiento que vamos a introducir tiene como finalidad mantener la batería de nuestro dispositivo cargada. Este objetivo pertenece al conjunto de objetivos del *OS Agent*, que es el agente que representa al sistema operativo y trata de mantener el sistema en las mejores condiciones para el cumplimiento de los objetivos del resto de agentes del sistema. Para ello hemos definido un objetivo de tipo `mantain` que se muestra en el Listado 10.2.

Como podemos observar en este objetivo de mantenimiento, la intención del mismo es mantener la carga de la batería por encima de 200 miliamperios/hora. En el momento que se incumpla esta condición se tratará de encontrar un plan que lleve la carga de la batería a la condición objetivo, que son 1500 mAh.

Listado 10.2: Objetivo de mantenimiento

```
<goal type='mantain' recur='true' recurdelay='0' exclude='never'
    name='KeepBattery'>
  <mantaincondition>
    (> ?battery 200mAh)
  </mantaincondition>
  <targetcondition>
    (> ?battery 1500mAh)
  </targetcondition>
</goal>
```

10.3. Planificación del Caso de Uso

Siguiendo el modelo de ejecución del paradigma DGOC, una vez expresado el objetivo a alcanzar el sistema tratará de encontrar, o en su defecto componer, un plan que permita llevar al cumplimiento del objetivo. Esto ocurre cuando el Deliberation Engine selecciona el objetivo como próximo objetivo a cumplir. En nuestro ejemplo, dado que no tenemos más objetivos que cumplir, seleccionaremos éste como próximo objetivo.

La situación inicial de la que parte el On-line Planner puede ser muy compleja, con una gran cantidad de servicios disponibles y una base de casos donde se ha ido almacenando el razonamiento sobre ejecuciones pasadas. Con el fin de no complicar en exceso este ejemplo vamos a presentar un escenario reducido, con una base de casos pe-

queña, que nos permita realizar una traza del proceso de planificación de forma asequible para el lector.

En la Figura 10.3 podemos ver cómo sería la situación inicial de este caso de uso, donde mediante un interfaz móvil se introduce el objetivo que el usuario desea cumplir y éste es enviado a un computador que utiliza un sistema operativo DGOC. Este sistema forma parte del sistema domótico del hogar, que es capaz de localizar servicios adicionales en una nube de servicios accesible a través de su red. Podemos ver los proveedores de servicios localizados en la nube que van a intervenir en este ejemplo, entre ellos tenemos proveedores de servicios de pago como Visa (Tabla 10.4), MasterCard (Tabla 10.5) o PayPal (Tabla 10.6). Encontramos también proveedores de alquiler y venta de películas como Netflix (Tabla 10.1) e iTunes (Tabla 10.2). Proveedores de servicios de compartición de ficheros como P2P (Tabla 10.3) y proveedores de servicios de reproducción y reconversión de películas como VideoPlayer (Tabla 10.8). Además, el computador situado en el hogar domótico ofrece sus propios servicios asociados al sistema operativo (Tabla 10.7).

Una vez el objetivo es seleccionado por el Deliberation Engine el On-line Planner trata de localizar aquellos planes que permiten la consecución del objetivo. El planificador devolverá un conjunto de planes que contiene todos los posibles caminos que se pueden tomar, dados los servicios localizados, y se lo entregará al Commitment Manager para establecer pre-acuerdos con dichos servicios y poder así seleccionar qué camino es el más interesante para los fines del usuario. El grafo de planes generado por el On-line Planner es el de la Figura 10.2.

Podemos observar en la figura del grafo de planes cómo existen diversas alternativas en función del proveedor de películas, del medio de pago o de la reproducción de la película. Por ejemplo, el planificador ha encontrado tres posibles servicios de búsqueda de películas en

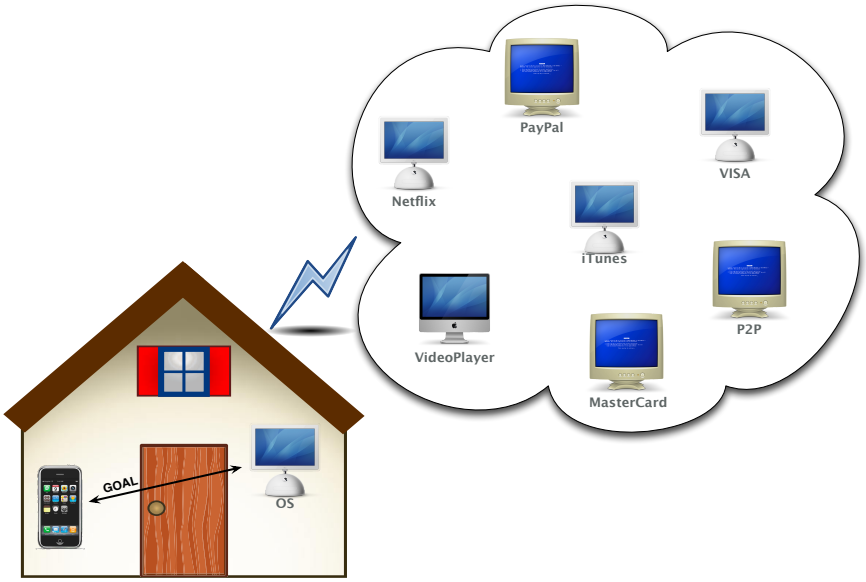


Figura 10.1: Situación inicial del Caso de Uso

ID	Name	Conditions	PS	T
A	film:search_film	P: ?film:title P: ?film:quality Q: ?film:ID Q: ?bank:price	0,9	8
B	film:buy_film	P: ?film:ID P: ?bank:price Q: ?bank:objectToPurchase	0,9	3
C	film:prepare_download	P: ?film:ID P: ?bank:objectPurchased Q: ?os:url	0,93	5

Tabla 10.1: Servicios ofertados por el proveedor Netflix

ID	Name	Conditions	PS	T
D	film:search_film	P: ?film:title P: ?film:quality Q: ?film:ID Q: ?bank:price	0,7	7
E	film:buy_film	P: ?film:ID P: ?bank:price Q: ?bank:objectToPurchase	0,8	3
F	film:prepare_download	P: ?film:ID P: ?bank:objectPurchased Q: ?os:url	0,81	4

Tabla 10.2: Servicios ofertados por el proveedor iTunes

ID	Name	Conditions	PS	T
G	film:search_film	P: ?film:title P: ?film:quality Q: ?film:ID	0,4	7
H	film:prepare_donwload	P: ?film:ID Q: ?os:url	0,8	23

Tabla 10.3: Servicios ofertados por el proveedor P2P

ID	Name	Conditions	PS	T
I	bank:pay	P: ?bank:price P: ?bank:objectToPurchase Q: ?bank:objectPurchased Q: ?bank:confirmationMethod Q: ?bank:paymentMethod(VISA)	0,95	4

Tabla 10.4: Servicios ofertados por el proveedor VISA

ID	Name	Conditions	PS	T
J	bank:pay	P: ?bank:price P: ?bank:objectToPurchase Q: ?bank:objectPurchased Q: ?bank:confirmationMethod Q: ?bank:paymentMethod(MC)	0,63	6

Tabla 10.5: Servicios ofertados por el proveedor MasterCard

ID	Name	Conditions	PS	T
K	bank:pay	P: ?bank:price P: ?bank:objectToPurchase Q: ?bank:objectPurchased Q: ?bank:confirmationMethod Q: ?bank:paymentMethod(PayPal)	0,61	7

Tabla 10.6: Servicios ofertados por el proveedor PayPal

ID	Name	Conditions	PS	T
L	os:download_url	P: ?os:url Q: ?os:file	0,76	345
M	os:move_file	P: ?os:file P: ?os:where Q: ?os:file_available	0,99	112

Tabla 10.7: Servicios ofertados por el Sistema Operativo

ID	Name	Conditions	PS	T
N	video:prepare_film	P: ?os:file_available P: ?os:file Q: (or (?video:prepared) (?video:need_encode))	0,89	3
O	video:encode	P: ?os:file_available P: ?os:file P: ?video:need_encode Q: ?video:prepared	0,78	156
P	video:play	P: ?video:prepared Q: ?video:viewed(?film:title)	0,97	412
Q	video:play_stream	P: ?os:url Q: ?video:viewed(?film:title)	0,71	412

Tabla 10.8: Servicios ofertados por el proveedor VideoPlayer

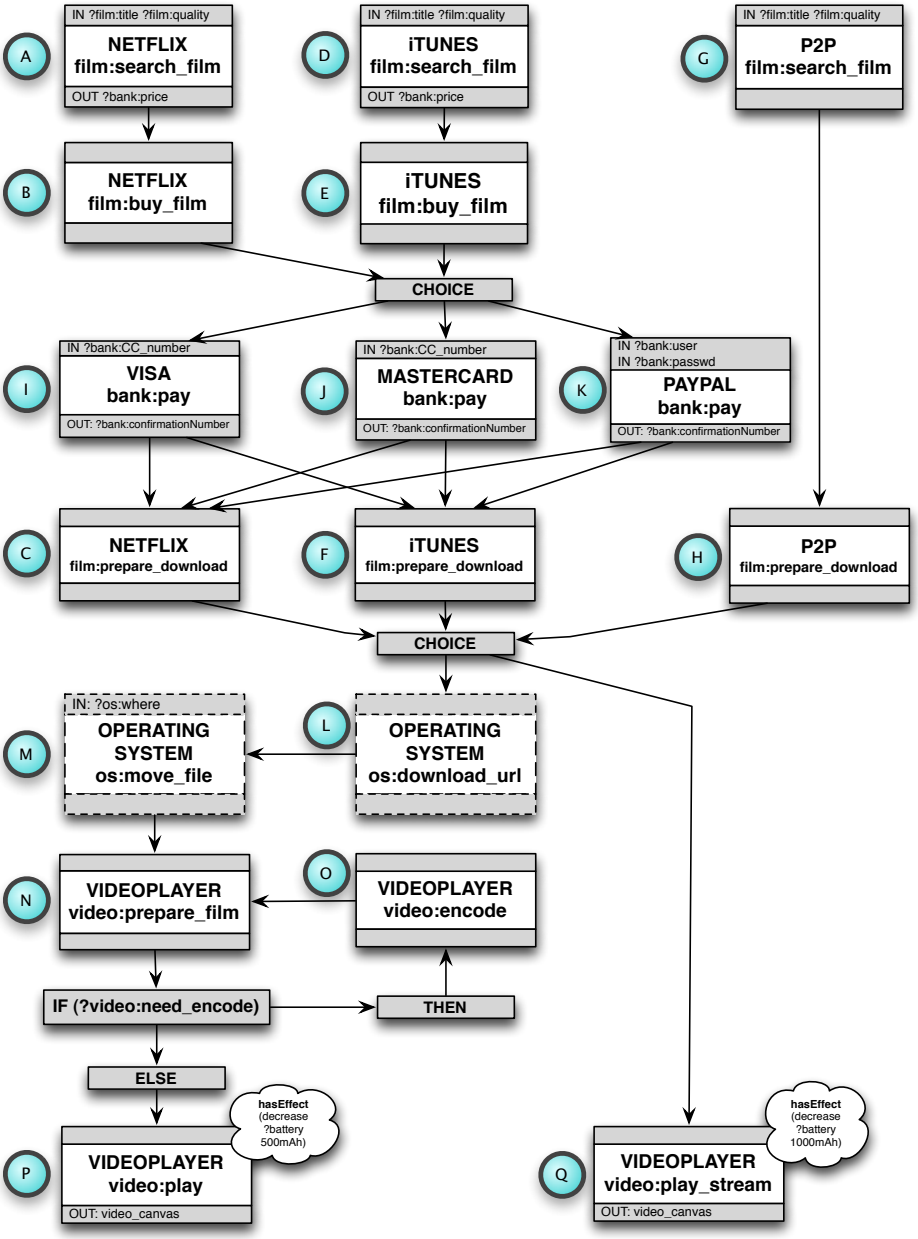


Figura 10.2: Grafo de planes del Caso de Uso

diversos proveedores: Netflix y iTunes, que son proveedores de pago, y descarga P2P, que permite la descarga punto a punto sin coste. Podemos observar como en las precondiciones y postcondiciones se utilizan predicados de cuatro ontologías diferentes que intervienen en el plan: `film` para la gestión de búsqueda y alquiler de películas, `bank` para la gestión de pagos bancarios, `os` para servicios relacionados con el sistema operativo (estos servicios se muestran con línea de puntos) y `video` para los servicios de reproducción de video.

En las tablas de los proveedores se muestran las precondiciones y postcondiciones más relevantes de los servicios que intervienen en el plan, así como el tiempo de cómputo estimado y el valor de calidad, que indica la confianza que tiene el On-line Planner en la correcta ejecución del servicio. A modo de ejemplo presentamos además en el Listado 10.3 el *profile* OWL-S de uno de los servicios del plan: `film:search_film`. En el Listado puede verse, además de las precondiciones y postcondiciones del servicio, las entradas y salidas, efectos y resultados, y demás datos que son utilizados por el Deliberation Engine durante la ejecución de los planes.

Listado 10.3: Servicio film:search_film del Caso de Uso

```
<process:AtomicProcess rdf:ID='film:search_film'>
   <process:hasInput>
      <process:Input rdf:ID='film:title'/>
     <UI:input/>
   </process:hasInput>
   <process:hasInput>
      <process:Input rdf:ID='film:quality'/>
      <UI:input/>
   </process:hasInput>
   <process:hasOutput>
      <process:Output rdf:ID='film:ID'/>
   </process:hasOutput>
  <process:hasOutput>
     <process:Output rdf:ID='bank:price'/>
    <UI:output/>
```

```
16    </process:hasOutput>
17    <process:hasResult>
18      <process:Result>
19        <process:hasResultVar>
20            <process:ResultVar rdf:ID='film:QualityHD'>
21               <process:parameterType rdf:resource='integer'/>
22            </process:ResultVar>
23         </process:hasResultVar>
24         <process:inCondition>
25           <expr:KIF-Condition>
26             <expr:expressionBody>
27          ( and
28            (current-value (film-quality ?film:ID) ?film:QualityHD)
29            (≥ ?film:QualityHD ?film:quality)
30          )
31             </expr:expressionBody>
32           </expr:KIF-Condition>
33         </process:inCondition>
34        <process:withOutput>
35            <process:OutputBinding>
36               <process:toParam rdf:resource='#film:ID'/>
37               <process:valueFunction rdf:parseType='Literal'>
38                  <film:FilmID xsd:datatype='&xsd;#string'/>
39               </process:valueFunction>
40            </process:OutputBinding>
41            <process:OutputBinding>
42               <process:toParam rdf:resource='#bank:price'/>
43               <process:valueFunction rdf:parseType='Literal'>
44                  <bank:Price xsd:datatype='&xsd;#integer'/>
45               </process:valueFunction>
46            </process:OutputBinding>
47         </process:withOutput>
48        <process:hasEffect>
49           <expr:KIF-Condition>
50             <expr:expressionBody>
51             (and
52                (> ?FilmID 0)
53                (current-value (film-title ?film:ID) ?title)
54                (= ?title ?film:title)
55             )
56             </expr:expressionBody>
57           </expr:KIF-Condition>
58         </process:hasEffect>
```

```
59
60      </process:Result>
61
62      <process:Result>
63         <process:hasResultVar>
64            <process:ResultVar rdf:ID='film:QualitySD'>
65               <process:parameterType rdf:resource='integer'/>
66            </process:ResultVar>
67         </process:hasResultVar>
68         <process:inCondition>
69           <expr:KIF-Condition>
70            <expr:expressionBody>
71          (and
72            (current-value (film-quality ?film:ID) ?film:QualitySD)
73            (> ?film:quality ?film:QualitySD)
74          )
75             </expr:expressionBody>
76           </expr:KIF-Condition>
77         </process:inCondition>
78         <process:withOutput rdf:resource='failureNotice'/>
79            <process:OutputBinding>
80               <process:toParam rdf:resource='#FilmID'/>
81               <process:valueData rdf:parseType='Literal'>
82                  <drs:Literal>
83                     <drs:litdefn xsd:datatype='&xsd;#integer'>
84                       0 </drs:litdefn>
85                  </drs:Literal>
86               </process:valueData>
87            </process:OutputBinding>
88         </process:withOutput>
89      </process:Result>
90   </process:hasResult>
91 </process:AtomicProcess>
```

En este perfil de servicio, cuya finalidad es buscar una película, se pueden ver parámetros de entrada (título y calidad de la película) y dos parámetros de salida, uno un identificador que será utilizado por posteriores servicios para gestionar la comprar y visualización de la misma y otro parámetro que indicará el precio de la película. El servicio tiene, además de los parámetros de salida, unos efectos. Estos efectos per-

miten al Deliberation Engine comprobar si se seguirán cumpliendo las condiciones establecidas por el objetivo una vez ejecutado el servicio. Por ejemplo, de los dos resultados posibles (uno con la salida correcta y un resultado en caso de error) el primero se asegura de que la calidad establecida se cumple y de que el título corresponde con el valor introducido como parámetro de entrada.

Interfaz de usuario

En la descripción OWL-S de este ejemplo hemos introducido dos elementos extra con el fin de gestionar la posible interacción con el usuario. Estos elementos son `<UI:input/>` y `<UI:output/>`. El paradigma presentado en este trabajo utiliza la base de conocimiento del agente para crear y llevar a cabo los planes que cumplen sus objetivos. Sin embargo, en entornos reales como los de este ejemplo es muy habitual que determinados datos no existan inicialmente en la base de conocimiento o sencillamente necesiten de determinada confirmación o actualización por parte del usuario. Del mismo modo es muy habitual que exista la necesidad de mostrar información al usuario tras la ejecución de determinados servicios, tanto con fines informativos como para adquirir determinada información que permita al plan continuar su ejecución. Para ello hemos extendido los servicios propuestos por OWL-S en nuestro modelo de Sistema Operativo, de modo que puedan tener una interfaz que se ejecute antes del servicio y otra que se ejecute después del servicio (mostrados como bandas grises antes y después de cada servicio en la Figura 10.2), donde se puedan requerir datos extra del usuario, mostrar información o incluso pequeñas lógicas de negocio para tomar decisiones durante la ejecución del plan.

Gestionar la interacción con el usuario en entornos distribuidos donde cada servicio de un plan puede estar ejecutándose en un compu-

tador diferente y a mucha distancia de donde se encuentra el usuario es realmente complicado. Para solventar este problema hemos utilizado un lenguaje muy extendido y estandarizado que además puede integrarse fácilmente en la definición basada en etiquetas de un servicio OWL-S. Las interfaces de usuario de nuestros servicios se definen en HTML y Javascript. De este modo se puede delegar en el dispositivo donde se encuentre el usuario toda la carga de representación, adquisición de datos intermedios y determinada lógica intermedia. Con ello el usuario únicamente necesita de un navegador web con soporte de Javascript para poder interactuar con el sistema operativo presentado en este trabajo, utilizando un dispositivo desde el que comunicarse con el mismo, lo cual permite una gran movilidad a los usuarios, dado que es posible introducir nuevos objetivos al sistema desde dispositivos remotos.

La etiqueta `<UI:input/>` es utilizada para indicar que un parámetro de entrada puede ser requerido antes de la ejecución de un servicio, de modo que si en el momento de construcción del plan el planificador no dispone de ese parámetro, no descartará la utilización del servicio porque sabe que será requerido previamente a su ejecución. Del mismo modo, la etiqueta `<UI:output/>` indica que un parámetro será requerido tras la ejecución del servicio. Habitualmente para mostrar por pantalla un dato que sea necesario transmitir al usuario. En nuestro ejemplo, mostraremos al usuario el precio de la película que hemos encontrado para él.

Resolución de conflictos

Si observamos el plan que hemos generado para nuestro caso de uso donde deseamos visualizar una película (Figura 10.2) veremos como hemos representado dentro de una nube parte de los efectos que

se producen al ejecutar los dos servicios de reproducción de vídeo. En ambos casos nos informa de que la carga de la batería se decrementará considerablemente, 1000 mAh en el caso de la reproducción en streaming y 500 mAh en el caso de la reproducción de una película descargada. Esta información puede ser muy útil al Deliberation Engine puesto que, en el momento de tomar la decisión de descargar la película o verla a través de streaming (sin descarga previa) la interacción con el objetivo de mantenimiento de la batería mostrado previamente puede resultar decisiva. En el caso de que la carga de la batería se encuentre en ese momento por debajo de 1000 mAh el Deliberation Engine no seleccionará el servicio de streaming, puesto que la carga de batería no será suficiente para completar la ejecución del servicio.

10.4. Traza de ejecución

A continuación vamos a mostrar una traza de cómo el Deliberation Engine elige el plan a ejecutar partiendo del grafo de planes proporcionado por el On-line Planner y de la negociación que realizará el Commitment Manager con los proveedores de servicio. la traza comienza una vez el objetivo ha sido activado y el On-line Planner ha seleccionado un conjunto de planes posibles para cumplir el objetivo (Figura 10.2). En este punto el Commitment Manager debe establecer pre-acuerdos con los proveedores de servicio con el fin de aportar información necesaria para la toma de decisión por parte del Deliberation Engine.

El On-line Planner realiza la composición de planes *hacia atrás*, partiendo del objetivo busca servicios cuyas postcondiciones coincidan con la `targetcondition` del objetivo y va conectando servicios hasta llegar a un estado conocido. Para ello consulta tanto la base de co-

nocimiento como los parámetros de entrada del objetivo. Si durante la composición surgen diferentes alternativas se utiliza un nodo de tipo `CHOICE`, que podrá ser resuelto por el Deliberation Engine durante la ejecución del plan. Del mismo modo, si encuentra operadores lógicos (como un OR) podrá introducir nodos condicionales como el `IF-THEN-ELSE` para seguir un camino u otro durante la ejecución del plan.

El planificador no devuelve un plan único e inamovible, sino que devuelve un grafo con diversas opciones y gran cantidad de información para resolver en tiempo de ejecución. Esto se debe a que determinadas variables no pueden ser resueltas hasta el momento de la ejecución y a que es más óptimo mantener alternativas por si falla una condición durante la ejecución (en lugar de volver a iniciar el proceso de composición de un nuevo plan). Por ejemplo, en la Figura 10.3 se muestra el camino más corto del plan y, en principio, más ventajoso: tiene el menor número de servicios, y además no necesita pasar por el proveedor de pagos, por lo que la condición de no pagar más de un determinado precio por la película se cumplirá con toda seguridad. Además evitamos la descarga previa de la película al utilizar el reproductor en streaming, por lo que presumiblemente ahorraremos tiempo. Durante la ejecución, al llegar al segundo nodo `CHOICE`, donde se elige si se descarga la película o se visualiza por streaming, el Deliberation Engine puede comprobar que el servicio `video:play_stream` tiene como efecto: `(decrease ?battery 1000mAh)`. Como ya hemos comentado en la sección sobre conflictos entre objetivos, el Deliberation Engine no seleccionará un servicio si va a tener como consecuencia el no disponer de suficiente carga en la batería para finalizar la ejecución del mismo.

Otra situación que puede ocurrir ejecutando este mismo camino es

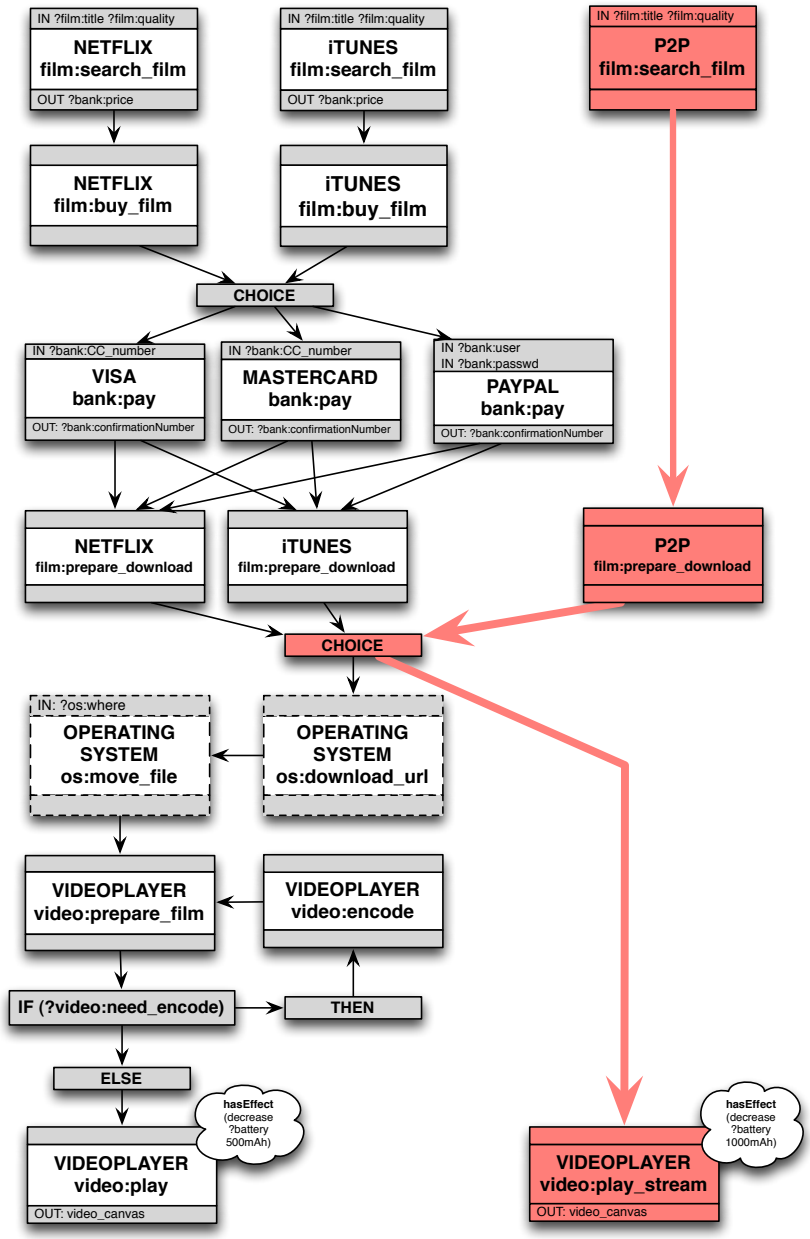


Figura 10.3: Camino más corto del plan

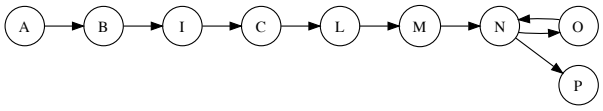
la siguiente: El proveedor de películas P2P es, como ya hemos visto, el más barato, dado que se basa en la compartición de contenido entre los usuarios de la red y no en el alquiler y venta. Sin embargo, es muy habitual que en las redes de descarga P2P la calidad no sea óptima. Esta calidad, que es un dato de entrada del objetivo, únicamente puede ser resuelta tras la ejecución del servicio, por lo que será en ese momento cuando el Deliberation Engine tome la decisión de qué ejecutar a continuación. Si la condición de ver la película a una determinada calidad fuera una `dropcondition`, al no cumplirse el requisito debería buscar otro camino en el plan, usando los proveedores Netflix o iTunes. Sin embargo, dado que es una `softcondition` la ejecución del plan podría continuar. Sin embargo, en el caso de que la búsqueda fuera vacía, la ejecución del plan fallaría y se debería reiniciar su ejecución (realizando en el peor de los casos una replanificación).

La función del Deliberation Engine será evitar que se incumplan estas condiciones. Por ejemplo, dado que el objetivo expresa como `dropcondition` que no desea utilizar como método de pago el del proveedor de PayPal, el servicio de pago `bank:pay` del proveedor PayPal no será seleccionado para su ejecución.

Una vez el On-line Planner ha calculado el conjunto de caminos posibles que pueden ejecutarse, el Commitment Manager calcula la probabilidad de éxito de cada uno de los caminos para tomar la decisión final de qué camino del plan ejecutar. Todo camino que pasa por el servicio `K` (`bank:pay`) del proveedor *PayPal* ha sido podado, puesto que al escribir la sentencia `bank:paymentMethod(PayPal)` se cumplía una de las dropconditions del objetivo. Cuando aparecen sentencias IF-THEN-ELSE en el camino el Deliberation Engine toma el peor caso y, para evitar bucles, limita los ciclos a dos pasos como máximo por cada servicio. Para cada camino x calcularemos su probabilidad de éxito con

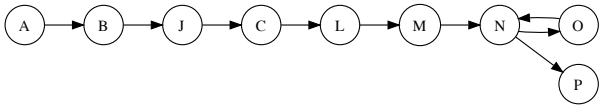
la ecuación $PS_x = \prod_{i=0}^{N} PS_i * \omega_i$ y su tiempo de ejecución propuesto mediante la ecuación $T_x = \sum_{i=0}^{N} T_i$. Para simplificar esta traza hemos dejado el valor de $\omega_i = 1$, simulando que la base de casos todavía no posee experiencia previa. Finalmente normalizaremos el valor de T_x. Para ello utilizaremos el valor medio de todos los T_x, al que llamaremos $\overline{T}$, y la desviación típica σ. El valor normalizado se calcula mediante la ecuación: $\widetilde{T_x} = \frac{T_x - \overline{T}}{\sigma}$. Con esta información mostramos a continuación los caminos extraídos del plan de la Figura 10.2 y sus valores PS_x, T_x y $\widetilde{T_x}$. Podemos ver como en el camino número 1 la probabilidad de éxito de la composición es de $0,32$ y su tiempo normalizado es $0,028$.

1



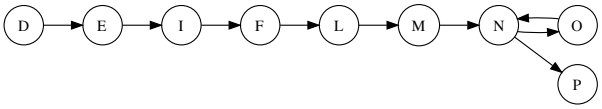
$PS_1 = 0{,}32$ $T_1 = 1051$ $\widetilde{T_1} = 0{,}028$

2



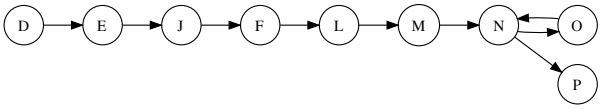
$PS_2 = 0{,}21$ $T_2 = 1053$ $\widetilde{T_2} = 0{,}025$

3



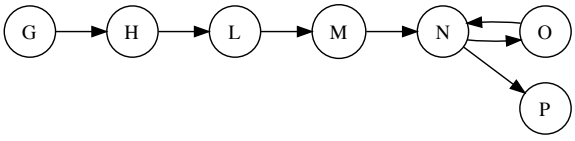
$PS_3 = 0{,}19$ $T_3 = 1049$ $\widetilde{T_3} = 0{,}031$

4



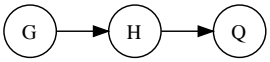
$PS_4 = 0{,}12$ $T_4 = 1051$ $\widetilde{T_4} = 0{,}028$

5



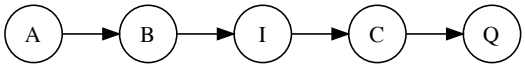
$PS_5 = 0{,}14$ $T_5 = 1061$ $\widetilde{T_5} = 0{,}013$

6



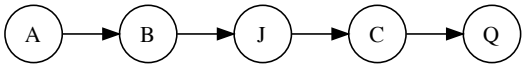
$PS_6 = 0{,}23$ $T_6 = 442$ $\widetilde{T_6} = 0{,}96$

7



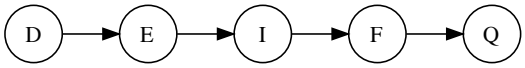
$PS_7 = 0{,}50$ $T_7 = 432$ $\widetilde{T_7} = 0{,}97$

8



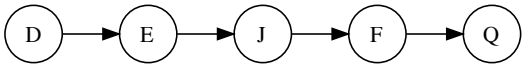
$PS_8 = 0{,}33$ $T_8 = 434$ $\widetilde{T_8} = 0{,}97$

9



$PS_9 = 0{,}30$ $T_9 = 430$ $\widetilde{T_9} = 0{,}98$

10



$PS_{10} = 0{,}20$ $T_{10} = 432$ $\widetilde{T_{10}} = 0{,}97$

El siguiente paso, una vez calculados los valores PS_x y $\widetilde{T_x}$ de cada camino del plan, es comparar todos los caminos y tomar una decisión. Para ello podemos simplemente sumar ambos valores (dado que ya los

tenemos normalizados) y con esta información veremos que el camino con mejor puntuación es el número 7 ($PS_7 + \widetilde{T_7} = 1{,}47$), por lo que los compromisos temporales establecidos serán confirmados y se pasará al Runtime Engine para su ejecución. Del mismo modo, los compromisos preestablecidos para el resto de caminos son cancelados, liberando así la reserva realizada en los proveedores de dichos servicios.

Si durante la ejecución del plan seleccionado fallara la ejecución de algún servicio entraría en funcionamiento la recuperación del plan. La información previamente recopilada puede ser reutilizada para establecer nuevos compromisos temporales y, en caso de no poder utilizar ninguna de las opciones previamente calculadas (quizás por la caída de un proveedor), sería necesario que el On-line Planner tratara de realizar un nuevo plan. Finalmente, en el peor de los casos en el que no se encontrara un nuevo plan, el objetivo sería marcado como no alcanzable.

10.5. Consideraciones finales

En este capítulo hemos presentado un caso de uso donde se puede ver cómo el modelo de ejecución de sistema operativo presentado en este trabajo funcionaría en un entorno real. Se ha mostrado paso a paso, desde la instanciación del objetivo hasta la puesta en funcionamiento de un plan que cumpla dicho objetivo.

En este caso de uso hemos podido demostrar cómo el sistema operativo basado en la computación distribuida orientada a objetivos presenta su utilidad para este tipo de entornos. Aprovechando la distribución de los servicios el sistema operativo ha podido asistir al usuario para cumplir su objetivo sin necesidad de gran interacción por parte del mismo.

El sistema operativo no sólo ha sido capaz de localizar diferentes composiciones de planes que llevan al cumplimiento del objetivo, sino que además tiene la habilidad de razonar sobre esas composiciones y realizar una toma de decisión, seleccionando la más apropiada para el usuario, teniendo en cuenta condiciones de rechazo, condiciones débiles e interacciones y conflictos con otros objetivos.

En esta deliberación se ha incluido también una fase de negociación entre los proveedores y el Commitment Manager para establecer los términos de la ejecución. Entre estos términos se ha tenido en cuenta la reputación de los proveedores de servicios y la predicción del tiempo de ejecución previsto. Para la predicción del tiempo de ejecución de los servicios se ha utilizado el algoritmo BDPS. Este algoritmo otorga más prioridad a aquellos servicios que proporcionen más beneficio al proveedor, lo que se ve reflejado en un mejor tiempo de ejecución. Dado que ya se ha mostrado el funcionamiento de BDPS en el Capítulo 9, hemos omitido su traza en este ejemplo, que se ha centrado en la parte del cliente.

En definitiva hemos podido comprobar cómo el sistema operativo basado en objetivos nos presenta una nueva dimensión de cómo enfocar aplicaciones de Inteligencia Ambiental de una forma más sencilla y directa, dado que el usuario no necesita conocer ni qué servicios debe invocar, ni preocuparse de buscar alternativas cuando algo falla, ni tampoco preocuparse de buscar aquellos servicios más eficientes o que más le beneficien. Tan solo expresando su objetivo (mediante una interfaz sencilla y adecuada) podrá alcanzarlo de forma transparente gracias a este sistema operativo y a los agentes que lo pueblan.

Parte IV

Conclusiones

11

Conclusiones y Trabajo Futuro

Este capítulo presenta las conclusiones obtenidas durante esta investigación. Las contribuciones principales se presentan en la sección 11.1. En la sección 11.2 se presenta el trabajo futuro planteado. Finalmente, en la sección 11.3 se presentan los trabajos publicados con relación a este trabajo de tesis.

11.1. Contribución

Teniendo en cuenta los objetivos planteados en esta Tesis así como los resultados del trabajo desarrollado, la principal contribución de este trabajo ha sido la propuesta de Sistema Operativo basado en el paradigma de *Computación Distribuida Orientada a Objetivos*.

Esta propuesta abre una nueva línea en el desarrollo de sistemas

operativos, elevando las abstracciones utilizadas por los SO a aquellas propuestas por metodologías más modernas como la de sistemas multi-agente y la de computación orientada a servicios.

Con este paradigma se ha conseguido el desarrollo de un sistema distribuido abierto, donde los agentes son la entidad de ejecución y buscan la consecución de sus objetivos mediante la invocación de servicios externos de manera dinámica, esto es, realizan una composición de servicios cuya ejecución les lleva al cumplimiento de su objetivo. Dado que los servicios invocados no tienen por qué localizarse en el mismo nodo en que se encuentra el agente que demanda el servicio, consideramos que este sistema operativo tiene una orientación a sistemas distribuidos y *computación en la nube*.

Este enfoque a la computación *en la nube* y orientada a objetivos y el hecho de ser un SO *abierto* son características que derivan de la existencia de un ecosistema potencialmente amplio donde los agentes que residen en dicho sistema tienen multitud de opciones para elegir cuando buscan la forma de cumplir sus objetivos. Dichos agentes tratarán siempre de buscar la solución de mayor *calidad* que sean capaces de encontrar. Los parámetros que definen la calidad de un servicio o de una composición de servicios pueden ser muchos y muy diversos (precisión, confianza, tiempo, fiabilidad, seguridad,...). En este trabajo se han utilizado algunos de ellos (como la fiabilidad y el tiempo) para ver cómo el SO puede ayudar a los agentes en esta tarea.

El principal parámetro utilizado ha sido el del tiempo de ejecución de un servicio. El periodo de tiempo que tarda un servicio en devolvernos un resultado puede ser tomado en cuenta como uno de los parámetros de calidad. Para ello el SO adquiere características heredadas de los sistemas operativos de tiempo real no-críticos que le permiten tener determinado control sobre los tiempos de ejecución, la planificación de

los servicios para poder cumplir sus compromisos temporales y sistemas de recompensa y castigo que modifiquen el valor de fiabilidad de cada servicio.

Para llevar a cabo el paradigma propuesto se ha realizado el diseño de una arquitectura general del modelo de ejecución de un sistema operativo orientado a objetivos. Esta arquitectura presenta los componentes necesarios para desplegar el paradigma utilizando agentes y objetivos. Se ha desarrollado un módulo denominado Deliberation Engine que permite seleccionar objetivos de los agentes a cumplir y localizar o componer planes que permitan la consecución del objetivo. Para ello se ha presentado un componente llamado On-line Planner que utiliza un planificador basado en casos y acotado temporalmente. Para establecer aquellos planes más ventajosos para el usuario se ha presentado un componente denominado Commitment Manager que se encarga de establecer compromisos entre los usuarios y los proveedores de servicios.

Con esta propuesta se han realizado una serie de prototipos y experimentos que han permitido verificar la aproximación orientada a objetivos aplicada al diseño de un Sistema Operativo. Se ha podido ver en los experimentos cómo mejorar la tolerancia a fallos gracias a la capacidad de replanificación y a las técnicas de razonamiento aplicadas en la planificación basada en casos, que nos permite elegir aquellos servicios y proveedores de servicios que más confianza nos aportan. Igualmente se ha verificado cómo mejora la adaptabilidad del sistema, manteniendo altos niveles de calidad de servicio a pesar de los posibles fallos en nodos de la red.

Finalmente, otra aportación de este trabajo ha sido el diseño y desarrollo de un planificador de tareas para el sistema operativo con la capacidad de realizar predicciones del tiempo total de ejecución de una

tarea (lo que hemos llamado deadline) dando garantías de este tiempo de finalización, lo que permite establecer compromisos temporales firmes, y maximizando a su vez el beneficio obtenido por la ejecución de tareas. Se ha demostrado cómo este algoritmo de planificación obtiene buenos resultados en las métricas realizadas entre diferentes algoritmos de planificación, obteniendo muy buenos resultados en el tiempo de respuesta, al mismo tiempo que realiza predicciones temporales fiables que permiten establecer compromisos temporales entre proveedores y clientes para mejorar la competitividad del sistema con estos parámetros de tiempo de ejecución y beneficio obtenido.

11.2. Trabajo Futuro

En esta tesis existen ciertas ideas que no han sido abordadas y que al mismo tiempo resultan muy interesantes para ser tratadas en un trabajo futuro:

- En este trabajo se ha desarrollado un prototipo de sistema operativo que cubre toda la funcionalidad propuesta en esta tesis. Dicho simulador ha permitido realizar diversos experimentos en entornos cercanos a los reales. Sin embargo, con el fin de estudiar en más profundidad este paradigma, sería necesario avanzar hacia la implementación del propio sistema operativo basado en objetivos. Se ha comenzado un estudio para analizar la viabilidad de modificar un sistema operativo existente o comenzar la implementación de un SO por completo. Modificar un sistema operativo existente tiene la ventaja de no repetir trabajo ya largamente realizado en otros operativos. Sin embargo, el nuevo enfoque realizado en las abstracciones propuestas por esta tesis complica la

adaptación de un SO existente. Las aproximaciones pueden pasar por utilizar un SO implementado en lenguajes de alto nivel (como Singularity), un SO grid basado en organizaciones virtuales (como XtreemOS) o un exokernel sobre el que implementar las nuevas abstracciones (como ExOS).

- En este trabajo se han realizado experimentos para analizar la configuración ideal de los parámetros utilizados por el algoritmo BDPS. Se plantea como trabajo futuro el soportar la modificación dinámica de estos parámetros (tamaño de ventana, tamaño de quantum, etc) para adaptarse automáticamente a la carga del sistema. Se puede estudiar cómo variar estos parámetros en función del tiempo de ejecución medio de las tareas, su tasa de llegada, etc.

- La base de casos del On-line Planner utiliza una estructura de rápido acceso por índice como es la tabla de dispersión. Sin embargo, aunque esta estructura permite acceder muy rápidamente a los casos almacenados conociendo únicamente su postcondición, todavía existe un proceso donde el tiempo puede resultar un factor crítico: la composición de planes. Para ello se plantea como trabajo futuro realizar parte del trabajo de la composición en la fase de *retain* del planificador. Almacenando los casos nuevos en un grafo conectado donde los arcos conectan la postcondición de un servicio con la precondición de otro tendríamos gran parte del trabajo realizado en la fase de *retain* del TB-CBP, que es menos crítica que la fase de recuperación de casos. Al tener almacenado en un grafo todos los casos conocidos e interconectados el proceso de composición de planes se basaría en una búsqueda de caminos en el grafo, para lo que existen diversos algoritmos probados y muy eficientes.

- El proceso de negociación entre los proveedores de servicios y el Commitment Manager es también un punto considerado para ampliar en un trabajo futuro. Se plantea introducir estrategias de negociación multi-atributo con el objetivo de conseguir acuerdos más flexibles para la ejecución de servicios. Se podría utilizar negociación bilateral y manejo de *outside options* para establecer las estrategias de negociación. Otro aspecto que también ha sido lanzado durante la fase de análisis es la posibilidad de establecer contratos entre los agentes. Actualmente se utilizan métodos de bonificación y penalización en la base de casos para los proveedores de servicios. En un futuro se plantea extender esta funcionalidad con contratos que determinen determinados derechos y obligaciones, estudiando cómo afectará el incumplimiento de los mismos al proveedor de servicios.

- La seguridad es un aspecto planteado para trabajar en un futuro. Esta tesis se ha centrado en el modelo de ejecución del sistema operativo. Sin embargo existen muchos más aspectos que es interesante tener en cuenta durante el diseño del mismo, como la seguridad. Se plantea estudiar cómo un usuario interactúa con el sistema sin provocar brechas de seguridad, cómo compartir información entre agentes de forma eficiente y segura (modelos de compartición de memoria, de paso de mensajes,etc), cómo se autentifica un usuario para poder activar objetivos, qué objetivos puede activar, qué conjunto de servicios tiene restringidos en función de su nivel de seguridad, etc.

11.3. Publicaciones relacionadas

A continuación se presentan por orden de relevancia las publicaciones del autor relacionadas con el trabajo de tesis:

Deadline Prediction Scheduling based on Benefits

- Autores: J. Palanca, M. Navarro, A. García-Fornes y V. Julian
- Revista: Future Generation Computer Systems (FGCS) Volume 29, Issue 1, pp 61–73. (2013)
- **Índice de Impacto: 1.978**

En este trabajo se presenta la planificación del sistema operativo propuesto utilizando predicción del deadline y con razonamiento basado en beneficios. Se presentan además los experimentos realizados sobre la implementación de este planificador y su comparación con otros planificadores usualmente empleados por SO comerciales.

Distributed Goal-Oriented Computing

- Autores: J. Palanca, M. Navarro, V. Julian y A. García-Fornes
- Revista: Journal of Systems and Software (JSS) Volume 85, Issue 7, pp 1540-1557. (2012)
- **Índice de Impacto: 0.836**

En este trabajo se presenta una visión global y completa del paradigma de Computación Distribuida orientada a Objetivos. Se repasan todos los componentes que forman el sistema operativo que implementa

dicho paradigma y se realiza una evaluación exhaustiva de la funcionalidad del paradigma. También se presenta el simulador discreto implementado en esta propuesta.

A new deliberation mechanism for Service-Oriented Operating Systems

- Autores: J. Palanca, M. Navarro, A. García-Fornes y V. Julian
- Conferencia: 45 Hawaii International Conference on System Sciences (HICSS-45) pp. 5463–5472 (2012)
- **Core Ranking: A**

En este artículo se profundiza en el diseño del motor deliberativo del sistema operativo propuesto. Se presenta el planificador on-line basado en un TB-CBP (Planificador basado en casos acotado temporalmente) y en el gestor de compromisos que establece compromisos temporales y de seguridad con los clientes.

A Goal-oriented Execution Module based on Agents

- Autores: J. Palanca, V. Julian y A. García-Fornes.
- Conferencia: 44 Hawaii International Conference on System Sciences (HICSS-44) pp. 1-10. (2011)
- **Core Ranking: A**

En este trabajo se presenta el modelo general de ejecución para un sistema operativo diseñado bajo el paradigma de Computación Distribuida basada en Objetivos. Se presenta el funcionamiento del Deliberation

Engine del sistema operativo incluyendo el algoritmo que lleva a la activación de planes y su ejecución. Finalmente se presenta un ejemplo y unas pruebas basadas en una primera versión del simulador.

Modeling an Operating System based on Agents

- Autores: J. Palanca, M. Navarro, E. Argente, A. García-Fornes y V. Julian
- Conferencia: International Conference on Hybrid Artificial Intelligence Systems (HAIS 2012) pp.588-599 (2012)
- **Core Ranking: C**

En este artículo se presenta el modelado de un sistema operativo basado en agentes utilizando la metodología GORMAS. Mediante esta metodología se identifican los elementos principales (agentes, organizaciones, roles,...) que han servido para el diseño del sistema operativo orientado a objetivos.

A Jabber-based Multi-Agent System Platform

- Autores: M. Escriva, J. Palanca, G. Aranda, A. García-Fornes, V. Julian and V. Botti
- Conferencia: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS06) pp. 1282-1284. (2006)
- **Core Ranking: A+**

En este trabajo se presenta la plataforma SPADE. Se define la plataforma, su modelo de agente y el núcleo de comunicación basado en el protocolo de Mensajería Instantánea XMPP. Este es el trabajo fundacional de la plataforma, sirvió de base para detectar las carencias de los sistemas multiagente y de los frameworks que les dan soporte. Sobre él se han introducido posteriormente las características del paradigma de Computación Distribuida basada en Objetivos.

Towards Organizational Agent-Oriented Operating Systems

- Autores: J. Palanca, V. Botti and A. García-Fornes.
- Conferencia: 24th ACM Symposium on Applied Computing (SAC 2009) pp. 752-756. (2009)
- **Core Ranking: B**

Este trabajo presenta los resultados fruto del análisis y modelado de un sistema operativo utilizando una metodología basada en organizaciones virtuales de agentes. En este artículo se presentan las primeras conclusiones que llevarán posteriormente a la especificación del paradigma de Computación Distribuida basada en Objetivos.

Supporting Agent Organizations

- Autores: E. Argente, J. Palanca, G. Aranda, V. Julian, V. Botti, A. García-Fornes and A. Espinosa.
- Conferencia: 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'07) Vol. 4696 pp. 236-245. (2007)

- **Core Ranking: B**

Este trabajo presenta el uso de la extensión de multiconferencia del protocolo XMPP para la creación de organizaciones virtuales de agentes. Se presenta el API desarrollado para la plataforma SPADE y los elementos fundacionales como las Unidades Organizativas y la topología de organizaciones en SPADE.

Adding New Communication Services to the FIPA Message Transport System

- Autores: J. Palanca, M. Escriva, G. Aranda, A. García-Fornes, V. Julian and V. Botti
- Conferencia: Proceedings of the 4th German Conference on Multiagent System Technologies (MATES06) Vol. 4196 pp. 1-11. (2006)
- **Core Ranking: B**

En este trabajo se presenta el modelo de comunicación de la plataforma SPADE. Se extiende el protocolo XMPP para ser compatible con FIPA.

Towards Organizational Agent-Oriented Operating Systems

- Autores: J. Palanca, V. Botti and A. García-Fornes.
- Conferencia: EUMAS 2008 pp. 1-10. (2008)
- **Core Ranking: C**

Este trabajo presenta también los resultados fruto del análisis y modelado de un sistema operativo utilizando una metodología basada en organizaciones virtuales de agentes. Este es el mismo trabajo que el presentado en la conferencia SAC 2009 dado que esta conferencia admite trabajos ya publicados para su discusión.

Otras publicaciones relacionadas

Building Service-Based Applications for the iPhone Using RDF: a Tourism Application

- Autores: J. Palanca, G. Aranda and A. García-Fornes
- Conferencia: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009) N. 55 pp. 421-429. (2009)
- **Core Ranking: C**

En este trabajo se exploran los sistemas multi-agente orientados a servicios e integrados en dispositivos móviles.

Hybrid MAS to solve problems with different temporal constraints

- Autores: J. Palanca, G. Aranda, C. Carrascosa and L. Hernández.
- Conferencia: New Trends in Real-Time Artificial Intelligence (NTeRTAIn 2006 / ECAI 2006) pp. 19-28. (2006)

En este trabajo se explora la interacción de agentes con plataformas de tiempo real. En este caso se construye un sistema multi-agente híbrido

donde se pone en funcionamiento un conjunto de agentes sin características de tiempo real (SPADE) interactuando con una plataforma de agentes en tiempo real (ARTIS). De este trabajo se extrajo la experiencia con sistemas multi-agente en tiempo real y su interacción con paradigmas orientados a servicios.

Towards Developing Multi-agent Systems in Ada

- Autores: G. Aranda, J. Palanca, A. Espinosa, A. Terrasa and A. García-Fornes
- Conferencia: Proceedings of the 11th Ada-Europe International Conference on Reliable Software Technologies Vol. 4006 pp. 131-142. (2006)
- **Core Ranking: A**

En este trabajo se expone cómo ampliar el campo de acción de la plataforma de sistemas multi-agente SPADE añadiendo soporte para el lenguaje de programación ADA. Se presenta además el API de programación en ADA.

Uso de Técnicas Híbridas en el Cálculo del WCET

- Autores: J. Palanca and A. García-Fornes
- Conferencia: VIII Jornadas de Tiempo Real pp. 243-249. (2005)

En este trabajo se realiza un estudio de uno de los parámetros más importantes de los sistemas de tiempo real: el tiempo de ejecución en el peor caso. Se propone una nueva aproximación para realizar el cálculo de dicho tiempo.

Desarrollo de un SMA de Tiempo Real para la gestión automatizada del correo interno de un edificio de oficinas

- Autores: C. Carrascosa, L. Hernández, A. García-Fornes, J. Palanca and V. Botti
- Conferencia: VI Jornadas de Transferencia Tecnológica de Inteligencia Artificial, TTIA'2005 (AEPIA)/I Congreso Español de Informática CEDI'2005 pp. 183-190. (2005)

En este trabajo se explora cómo construir un ejemplo complejo de sistemas multi-agente con restricciones temporales.

A

SPADE: Un sistema multiagente basado en XMPP

SPADE (Smart Python multi-Agent Development Environment) es una plataforma de sistemas multi-agente construida alrededor de un nuevo modelo de comunicación (Jabber [Fou04a]) el cual provee de nuevas y potentes características a la capa de comunicación. El núcleo de la plataforma está construido en torno a esta capa de comunicación, que comunica los elementos de la plataforma entre ellos.

A continuación se introducirán los diferentes modelos que componen SPADE. En primer lugar presentaremos el nuevo **modelo de comunicación** basado en el protocolo Jabber. A continuación presentaremos todos los elementos que componen el **modelo de la plataforma** (compatible con FIPA). Una vez presentado el modelo de plataforma presentaremos el **modelo de agente** de SPADE, describiendo la arquitectura interna de los agentes en la plataforma. Finalmente presentaremos las principales características de la plataforma.

A.1. Modelo de Comunicación

Las comunicaciones en SPADE son gestionadas por medio del protocolo Jabber. Jabber es el protocolo estándar de Internet para la comunicación de mensajería instantánea y presencia [Fou04a, Fou04b]. Su tecnología base es el estándar *Extensible Messaging and Presence Protocol* (XMPP), que es un subconjunto del protocolo XML para difusión de contenidos y tecnologías, que permite interconectar dos entidades dentro de una red para el intercambio de mensajes, presencia y otra información estructurada en tiempo real.

La aplicación más popular del protocolo Jabber es una red de mensajería instantánea con funcionalidades similares a otras redes propietarias existentes como AIM[1], ICQ[2], MSN[3] o Yahoo Messenger. Estas redes permiten la interconexión de usuarios conocidos por medio de redes confiables, usadas y probadas por miles de usuarios simultáneamente.

[1] AIM: AOL Instant Messaging
[2] ICQ: I seek you
[3] MSN: Microsoft Messenger

A.1.1. El protocolo Jabber

La topología básica de Jabber está basada en la arquitectura cliente-servidor; la mayor diferencia, dado que se compone de una red descentralizada y distribuida, es una arquitectura también distribuida, similar a los servidores de correo electrónico convencionales. Desde la perspectiva de una red de computadores un servidor Jabber se comporta como un enrutador de mensajes XML, es decir, un distribuidor y conector de flujos XML entre conexiones de clientes. Además, un servidor Jabber ofrece servicios a sus clientes, como un directorio de usuarios, salas de chat públicas o puentes a otros servicios de mensajería.

Un cliente Jabber puede conectarse a diferentes servidores al mismo tiempo y contactar con cualquier cliente conectado a cualquiera de esos servidores. De este modo cuando un servidor recibe un mensaje para un cliente en otro servidor, éste lo enruta al servidor del cliente destino de forma transparente para ambos clientes.

La tecnología Jabber/XMPP ofrece diferentes ventajas descritas a continuación:

- **Abierto, público y libre:** La tecnología XMPP es libre para su uso y está bien documentada en diferentes estándares RFC[4] y en la Fundación XMPP. El hecho de que el protocolo sea libre significa que puede ser utilizado, ampliado o modificado por cualquier entidad, individuo u organización sin pagar ningún tipo de derechos o royalties por ello.

- **Estándar:** La tecnología XMPP está estandarizada en diferentes documentos RFC y aprobada por el IETF (Internet Engineering

[4]Request For Comments

Task Force)[Fou04a, Fou04b], lo que asegura su fiabilidad y continuidad.

- **Fiable:** La tecnología Jabber fue desarrollada inicialmente en 1998 y hoy en día se encuentra en un estado muy estable. Lleva años de uso y se encuentra muy probada, siendo integrada en productos de grandes compañías como Google (Gtalk), Apple (iChat), IBM (XMPP-to-SIP), etc.

- **Descentralizado:** La arquitectura de la red Jabber es similar a la del correo electrónico. Esto implica que cualquiera puede alojar su propio servidor Jabber creando redes que se unan a la red principal de Jabber o subredes independientes de Mensajería Instantánea que no dependan de una red centralizada. Aunque Jabber parezca una arquitectura cliente-servidor puesto que los usuarios deben conectarse a un servidor que enrute sus mensajes, en realidad esto servidores son pequeños componentes que se conectan con el resto de servidores de la red, creando la red distribuida de Jabber. Por tanto, únicamente cada nodo tiene una arquitectura cliente-servidor, la red es, sin embargo, completamente distribuida.

- **Seguro:** Jabber utiliza una seguridad robusta basada en SASL (Simple Authentication and Security Layer) y TLS (Transport Layer Security), que han sido incluidas en las especificaciones del núcleo de XMPP.

- **Extensible:** Usando la potencia de los espacios de nombres de XML, cualquiera puede construir una extensión propia sobre el núcleo del protocolo, sencillamente creando su propio espacio de nombres.

- **Flexible:** Las aplicaciones Jabber, a parte de la Mensajería Ins-

tantánea, han permitido otros muchos usos como la gestión de redes, sindicación de contenidos, herramientas de colaboración, compartición de ficheros, juegos, monitorización de sistemas remotos,..., y ahora, comunicación de agentes.

- **Diverso:** Una gran cantidad de empresas y proyectos de software libre utilizan el protocolo Jabber. Compañías como IBM, HP, Google, Apple y muchas otras se basan en Jabber para las comunicaciones de sus productos o sistemas.

Notificación de Presencia

La Notificación de Presencia permite llevar un control de lo que llamamos *latido del corazón* o *heartbeat*, por ejemplo, la habilidad de saber, por parte de una entidad de la red, cuando otra entidad está conectada o no. Esta funcionalidad elimina la necesidad del típico servicio de *Ping* para agentes.

Los contactos pueden ser usados además para crear *círculos sociales*, esto es, grupos de agentes que son conscientes de la presencia y estado del resto de agentes, eliminando así la necesidad de realizar excesivas consultas al AMS[5] y al DF[6], con la consecuente mejora de rendimiento.

La Notificación de Presencia es una de las características más potentes y exclusivas que aporta la plataforma SPADE. Es un sistema mediante el cual una entidad participante en una comunicación Jabber puede saber en cualquier momento el estado del resto de entidades involucradas en la conversación. Las entidades Jabber pueden declarar su estado actual, de modo que cualquier otra entidad asociada a ella es notificada inmediatamente. Normalmente una entidad publica su

[5] Agent Management System
[6] Directory Facilitator

estado y su disponibilidad para participar en una conversación (*Disponible, Ocupado, No molestar,...*), aunque esta lista no está cerrada; las entidades Jabber pueden definir su propio estado y significado.

Dos o más entidades Jabber pueden suscribir su presencia la una a la otra. Para suscribir un contacto ambas entidades deben autorizar al otro contacto mediante un simple proceso de negociación, proceso que puede ser automatizado.

Publicación y subscripción de eventos

SPADE permite a las entidades publicar y suscribirse a eventos personales. Ésta es extensión del protocolo XMPP, la `XEP-60: Publish--Subscribe (PubSub)`, y puede ser utilizada por todas las aplicaciones que requieran de notificación de eventos como sindicación de contenidos, geolocalización, carga de recursos, etc.

Para utilizar esta extensión una entidad debe publicar información sobre un evento a un nodo del servicio de publicación-subscripción (PubSub). Acto seguido el servicio notificará a todas las entidades que estén subscritas al evento y posean los permisos adecuados. El elemento de tipo `evento` puede contener cualquier tipo de información correspondiente a la semántica de su espacio de nombres gracias al lenguaje XML del que XMPP deriva.

Esta tecnología es especialmente útil dentro de nuestro sistema multiagente tanto para la creación de eventos personales de los agentes del sistema como para la creación de un completo sistema de trazas que permita conocer en tiempo real el estado de la plataforma, sencillamente suscribiendo agentes de `log` al servicio de trazas.

Organizaciones

La plataforma SPADE da además soporte a organizaciones virtuales de agentes [APA+07]. Una organización se compone de uno o más nodos donde se reúnen y comunican agentes. Es por esto que realizamos la analogía nodo-sala con la multi-conferencia Jabber. Cuando se desee diseñar una organización el agente propietario crea la sala de conferencia donde se reúnen los agentes que colaboran en la organización y configura los permisos de la sala para que se adapten a los requisitos de la organización.

Definimos **nodo** de una organización como *unidad mínima de una organización donde cooperan agentes subordinados a un mismo nivel y, opcionalmente, con un supervisor directo*.

Las organizaciones más sencillas están compuestas por un único nodo como la *jerarquía simple* donde hay un único supervisor y n subordinados que reciben órdenes. Otras organizaciones más complejas pueden formarse por más nodos como la burocracia, donde participan varios nodos que a su vez representan cada uno una jerarquía simple.

En esta arquitectura cada nodo queda modelado como una sala de conferencia y cada agente como un participante de la sala. La variedad de responsabilidades entre agentes se modela con los roles y afiliaciones. Finalmente, cuando la organización es compleja y se compone de varios nodos, estos se anidan como participantes de su nodo superior.

Para la comunicación de un agente con el miembro representativo de una organización SPADE utiliza otro estándar definido por la Jabber Software Foundation basado en grupos de trabajo. El documento es el «**XEP-0142: Workgroups**»[Tuc05] y define una extensión al «**XEP-0042: Multiuser Conference**»[SA06] sobre multiconferencia. Gracias a este protocolo se puede permitir el que un agente se dirija a una or-

ganización sin conocer necesariamente a su representante, puesto que se dirige al *servicio* que representa a la organización y gestiona sus comunicaciones con el exterior. Del mismo modo, este servicio de representante (que será siempre el supervisor del nodo) es utilizado como enlace para anidar nodos entre sí cuando la arquitectura de la organización es compleja.

Una de las grandes ventajas de esta arquitectura para modelar organizaciones es su flexibilidad. Cuando nos encontramos en el proceso de análisis y diseño de nuestra organización es muy probable que las estructuras que ofrece la metodología no se adapten a nuestras necesidades específicas. Es por eso que el diseño de organizaciones basado en Mensajería Instantánea provee una gran flexibilidad puesto que ofrece una serie de organizaciones básicas para las necesidades más comunes, pero todas ellas son extremadamente configurables tanto en la fase de diseño como dinámicamente por el agente administrador para ajustarse a las necesidades específicas de cada organización.

Invocación Remota de Servicios

Dado que esta plataforma nace con el objetivo de dar soporte a un paradigma donde los servicios son un elemento muy importante es necesario dar un soporte adecuado a los mismos. Como se verá más adelante la plataforma dispone de un servicio de directorio donde los agentes pueden registrar nuevos servicios, buscar servicios que se adecuen a sus objetivos o eliminar servicios que ya no se encuentran disponibles.

Sin embargo también es necesario dar soporte a la invocación remota de servicios. Aunque un agente es capaz de enviar un mensaje (siguiendo el protocolo FIPA o no) a otro agente demandando que le

provea de un servicio, ofrecer un estándar de invocación de servicios supone una gran ayuda para el diseñador y desarrollador del sistema. La plataforma SPADE implementa la extensión «**XEP-009: Jabber-RPC**». Esta extensión es principalmente una encapsulación del protocolo XML-RPC que codifica peticiones RPC y las envía mediante el protocolo HTTP. Jabber-RPC encapsula las peticiones RPC en el mismo formato que XML-RPC pero a diferencia de éste envía el paquete mediante el protocolo XMPP empotrando el mensaje RPC dentro de un mensaje Jabber (que ya está escrito en XML). El estándar introduce además protocolos de descubrimiento de servicios para comprobar si un determinado agente soporta este tipo de invocación.

A.2. El Modelo de Plataforma

La plataforma SPADE está modelada de acuerdo a los estándares propuestos por FIPA para plataformas multi-agente [FIP02a]. Implementa servicios estándares básicos (como el sistema de gestión de agentes, AMS, o el servicio de directorio, DF) que han sido diseñados como componentes internos de la plataforma (o add-ins).

Aunque el núcleo de la plataforma está desarrollado sobre la tecnología Jabber, la plataforma soporta más protocolos de transporte de mensajes, como *HTTP* [FIP02b].

Elementos de la Plataforma

En esta sección vamos a describir los elementos que componen SPADE.

- El principal componente de la plataforma es el **Enrutador XML**,

al cual se conectan el resto de componentes de la plataforma. Es un servidor XMPP estándar [Fou04a] que enruta todos los mensajes desde un origen a un destinatario especificado de forma transparente y sin necesidad de interacción humana. Este enrutador actúa como Sistema de Transporte de Mensajes (MTS). Es un componente muy flexible (gracias a su estandarización) de modo que puede ser sustituido por cualquier implementación de servidor Jabber en función de las necesidades (velocidad, capacidades, tolerancia a fallos,...) sin necesidad de reescribir nada.

- Uno de los componentes conectados al enrutador es el **Canal de Comunicación de Agentes de SPADE** (ACC), el cual gestiona todas las comunicaciones dentro de la plataforma. Este componente recibe mensajes en formato FIPA-ACL (FIPA Agent Communication Language [FIP01]) que llegan a la plataforma y los redirige a su destino correcto. Este destino debe ser o un agente o un componente de la plataforma; si el destino es un agente el MTS entrega el mensaje al agente para su proceso.

- El **Protocolo de Transporte de Mensajes** (MTP) por defecto de la plataforma es el protocolo Jabber, inherente a la misma. Además de conectar internamente los diferentes componente de la plataforma, el enrutador XML es quien conecta la plataforma con el mundo exterior (como otras plataformas o incluso comunicaciones agente-humano). Además se pueden conectar MTPs adicionales como plug-ins a la plataforma. El MTS es capaz de enrutar mensajes a través del MTP adecuado cuando recibe un mensaje de un protocolo diferente.

- El **Sistema de Gestión de Agentes** (AMS) es el componente que implementa los servicios de *registro*, *des-registro*, *modificación*, *obtención de información de la plataforma* y *búsqueda* para los agentes.

Son servicios para registrar la presencia de los agentes en la plataforma y poder realizar consultas sobre el mismo. Este componente es interno a la plataforma y se conecta al MTS como si fuera un agente más para gestionar el resto de agentes. El AMS implementa por completo la ontología `fipa-agent-management`, aunque también utiliza la tecnología de notificación de presencia para mantener actualizado el estado de todos los agentes conectados a la plataforma.

- El **Servicio de Directorio** (DF) es un componente que provee una base de datos donde registrar y consultar los servicios ofrecidos por otros agentes registrados en la plataforma. Este componente también se conecta a la plataforma como un agente más. Esto es, tiene el comportamiento de un agente de la plataforma (recibe mensajes, se gestiona por el AMS, etc) pero implementado internamente. Además el DF implementa por completo la ontología `fipa-agent-management` para Directorio de Servicios, con registro, desregistro, modificación y búsqueda.

La Figura A.1 muestra un esquema básico de la plataforma SPADE.

El bloque que representa el MTS (utilizando el protocolo XMPP) toma todos los mensajes recibidos y los redirige a su destinatario. El destinatario puede ser un agente, un componente de la plataforma, o incluso un usuario humano. Además el MTS gestiona la información de control (mediante consultas de tipo `iq`) y la información de enrutado. El ACC, AMS y DF se conectan a la plataforma como componentes internos. Estos componentes han sido construidos para comportarse como agentes de la plataforma, por lo tanto disponen de toda la funcionalidad de que dispone un agente. Como podemos ver en la figura, estos componentes están conectados también al MTS, de modo que están en contacto con el resto de la plataforma SPADE.

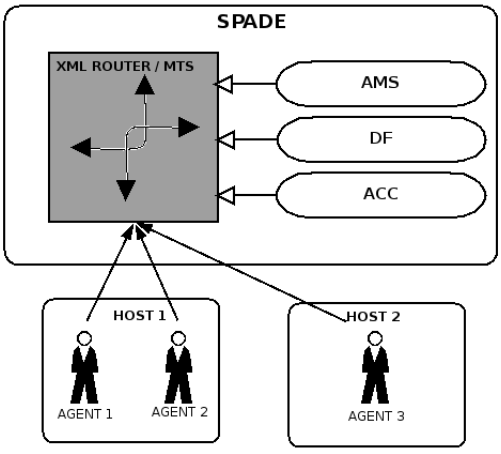


Figura A.1: El Modelo de Plataforma

Aspectos de Seguridad

La plataforma SPADE aporta algunos mecanismos internos de seguridad que ayudan a mantener la integridad del sistema. Estos mecanismos han sido desarrollados en diferentes niveles de seguridad:

- Para conectarse al servidor Jabber es necesario un **usuario y contraseña**. Este mecanismo previene de conexiones no autorizadas a la plataforma si no han sido autorizadas por un administrador. Por descontado, este mecanismo de seguridad es configurable (en el caso, por ejemplo, de que queramos un acceso abierto a todo el mundo). El administrador de la plataforma puede configurar el servidor Jabber para aceptar cualquier petición de registro, para denegar todas las peticiones de registro, para aceptar únicamente determinadas conexiones (por ejemplo, en función

del dominio), etc.

- La conexión al servidor Jabber puede ser codificada con un algoritmo de cifrado simétrico utilizando SSL (Secure Socket Layer). SSL aporta **cifrado** de datos, **autenticación** en el servidor, **integridad** de los mensajes y, opcionalmente, autenticación del cliente para las conexiones TCP-IP. Este mecanismo asegura que cada mensaje que viaja por la plataforma está fuertemente cifrado y no puede ser leído por alguna entidad maliciosa. Este proceso es necesario para asegurar la **confidencialidad** del sistema. SSL opera entre los agentes y la plataforma SPADE y también entre los componentes y la plataforma.
- Para asegurar la autenticidad de un mensaje, el servidor Jabber aporta otro mecanismo que evita la suplantación de identidad (o *spoofing*). Este método sencillamente previene el robo de identidad bloqueando el campo `from` de un mensaje y dejándolo como el ID del emisor.

Interoperabilidad de la Plataforma

El MTS puede manejar tantos MTPs como sea necesario. Si el protocolo es XMPP (protocolo Jabber), el MTS entrega directamente el mensaje al agente receptor. Es nuestro escenario (–ver Figura A.2), existen dos plataformas que soportan el protocolo Jabber. Si el Agente *X* de la Plataforma *A* desea enviar un mensaje al Agente *Y* de la Plataforma *B*, este entrega el mensaje a su propio MTS. Cuando un mensaje llega a un MTS y su destinatario es otro MTS, se redirige el mensaje a su MTS. Cuando un mensaje cuyo receptor es otro MTS llega a un MTS, este es redirigido a través de un mecanismo llamado *'s2s'*[7].

[7]Server To Server

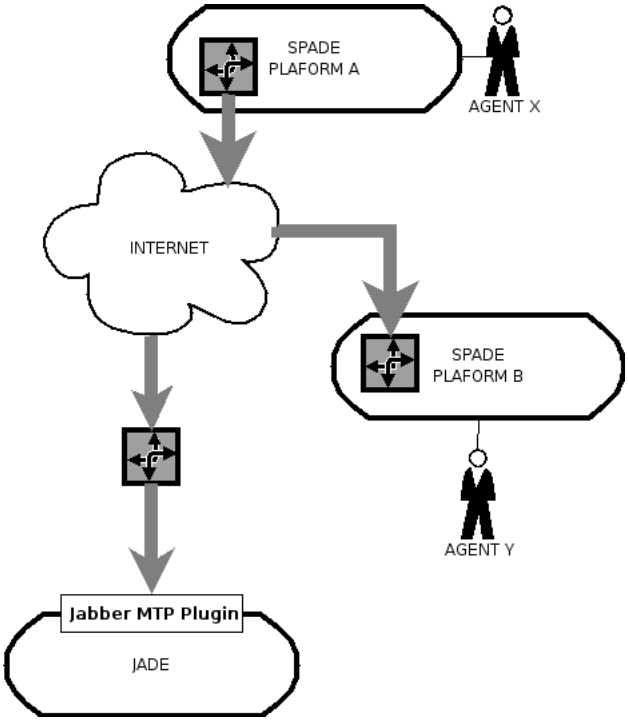


Figura A.2: Arquitectura Distribuida

Esta característica permite a SPADE comunicarse con cualquier plataforma de agentes que siga el estándar FIPA y soporte el Protocolo de Transporte de Mensajes XMPP.

Uno de los hitos de este trabajo ha sido el desarrollo de un plug-in para una de las plataformas de agente más conocidas, JADE, que permite utilizar el protocolo XMPP. Gracias a este plug-in podemos comunicar una plataforma JADE y una plataforma SPADE usando el protocolo Jabber (–ver Figura A.2). Del mismo modo, dos plataformas JADE pueden comunicarse utilizando este plug-in XMPP, aprovechando de este modo todas las ventajas del Protocolo de Transporte propuesto.

A.3. El Modelo de Agente

Todos los componentes desarrollados dentro de la plataforma están construidos siguiendo la arquitectura de agente. Los agentes SPADE son elementos conectados al MTS con la capacidad de enviar mensajes, tanto entre agentes, como entre plataformas o incluso entre usuarios humanos. De este modo se ha utilizado la tecnología de agentes para construir el propio soporte de agentes.

El Modelo de Agente de SPADE está compuesto básicamente de un mecanismo de conexión a la plataforma, un distribuidor de mensajes, y un conjunto de diferentes *comportamientos* a los que el distribuidor entrega los mensajes (–ver Figura A.3). Cada agente necesita un identificador que llamamos Jabber ID (JID) y una contraseña válida para establecer una conexión con la plataforma. Si el proceso de registro en la plataforma está deshabilitado, el administrador de la plataforma es el encargado de definir las políticas de registro.

El *JID* (compuesto de un **nombre de usuario**, una '@'y un **dominio del servidor**) será el nombre que identifique al agente en la plataforma durante toda su vida. Para permitir el uso de diferentes protocolos de transporte dentro de la plataforma se ha definido el prefijo `xmpp://` para las direcciones XMPP.

Un agente puede ejecutar varios comportamientos de forma simultánea. Un **comportamiento** es una tarea que el agente puede ejecutar utilizando patrones de repetición.

SPADE aporta varios tipos de comportamiento predefinidos: Cíclico, Periódico, One-Shot, Time-Out, y Máquina de Estados Finita. Estos tipos de comportamiento ayudan a implementar las diferentes tareas que un agente debe llevar a cabo. Los comportamientos más típicos

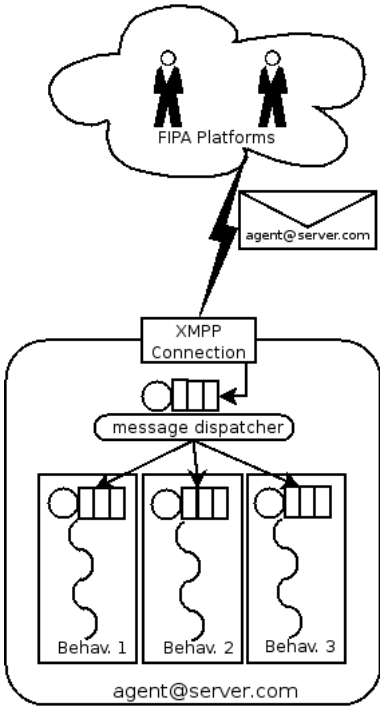


Figura A.3: Modelo de Agente de SPADE

son los siguientes:

- Los comportamientos Cíclicos y Periódicos son útiles para realizar tareas cada cierto tiempo.
- Los comportamientos One-Shot y Time-Out para tareas concretas o aperiódicas.
- La Máquina de Estados Finita permite construir comportamientos mucho más complejos.

Cada agente puede tener tantos comportamientos simultáneamen-

te como sea necesario. Cuando un mensaje llega a un agente, el distribuidor de mensajes lo redirige a la cola del comportamiento adecuado (–ver Figura A.3).

Un comportamiento tiene una plantilla de mensaje asociada a él. De este modo, el distribuidor de mensajes puede determinar a qué comportamiento corresponde un mensaje comparándolo con la plantilla adecuada. Este es el mecanismo utilizado por los comportamientos para seleccionar que tipo de mensajes o conversaciones le interesan.

La plataforma soporta además dos tipos más de comportamientos con un funcionamiento especial. Estos comportamientos han sido creados pensando específicamente en el soporte del paradigma de la Computación Distribuida basada en Objetivos. Los comportamientos son de tipo **Evento** y de tipo **Objetivo**.

Comportamientos Evento

Estos comportamiento están diseñados para dar respuesta a eventos aperiódicos que ocurran y a los cuales queramos responder de una forma particular. Su principal característica es no ocupar espacio físico en memoria, dado que es posible que ocurran con una frecuencia muy baja, para despertar en el momento en que ocurre un determinado evento. En el instante en que ocurre el evento que dispara este comportamiento se instancia en memoria el código necesario para gestionar el evento y una vez terminado se vuelve a eliminar.

Un agente puede registrar cualquier tipo de evento, teniendo en cuenta que en esta plataforma todo evento es convertido en un mensaje (gracias al protocolo PubSub), por lo que un evento puede ser un nodo al que estemos subscritos o un mensaje de entrada proveniente de un agente en particular o con una ontología determinada, etc.

Comportamientos Objetivo

Este tipo de comportamiento es el encargado de cumplir los objetivos expresados por el agente. Es un comportamiento *singleton* (es decir, existe una única ocurrencia de este comportamiento por agente) y es el encargado de buscar de forma completamente automática y proactiva los servicios disponibles que puedan llevarle a la composición de un plan que cumpla sus objetivos.

Este tipo de comportamiento hace uso de extensiones antes comentadas como el protocolo Jabber-RPC para invocar servicios y el agente de directorio DF para buscar los servicios necesarios. Para realizar la composición de un plan puede hacer uso de su propio planificador así como invocar algún servicio de planificación ofertado por algún otro agente.

A.4. Características Principales

En esta sección vamos a resumir las características principales de SPADE:

1. SPADE ha sido desarrollado utilizando el lenguaje de programación Python [Lut96]. Python es un lenguaje de programación de alto nivel con una enorme librería de clases que incluye numerosos y útiles componentes. Tiene una sintaxis muy sencilla, lo cual permite al programador centrarse en la funcionalidad y olvidarse de detalles de más bajo nivel.

2. La plataforma SPADE cubre la arquitectura FIPA. Los servicios AMS y DF soportan la ontología *fipa-agent-management* por com-

pleto. De todos modos, la plataforma también soporta mensajería entre agentes utilizando exclusivamente el protocolo XMPP.

3. SPADE soporta dos Protocolos de Transporte de Mensajes: HTTP y XMPP. Esto permite a diferentes plataformas el comunicarse utilizando este nuevo modelo de comunicación o pueden incluso utilizar el ampliamente extendido HTTP si no disponen del protocolo XMPP. Los MTPs han sido diseñados como plug-ins, de modo que se puedan añadir nuevos protocolos de forma sencilla y transparente al usuario.

4. SPADE implementa dos lenguajes de contenido diferentes: **FIPA-SL** y **RDF**, puesto que son los dos lenguajes de contenido más utilizados actualmente en la comunidad de agentes. La interfaz semántica ofrecida al agente es *independiente del lenguaje*, puesto que la librería del agente es capaz de traducir cualquiera de estos lenguajes a un lenguaje intermedio entendido por los agentes de la plataforma. Se realiza del mismo modo para el envío de mensajes, traduciendo de forma automática a cualquiera de los lenguajes de contenido soportados desde este interfaz.

5. Otra característica definida por FIPA y que SPADE soporta es la **movilidad**. Un agente SPADE puede moverse de una plataforma a otra plataforma autorizada sencillamente desconectándose de una plataforma y solicitando la conexión a otra, independientemente de donde se encuentre físicamente el agente.

6. Como comentamos en la Sección A.3, los agentes SPADE persiguen sus objetivos mediante la ejecución de uno o más **comportamientos** de forma concurrente.

7. Los agentes en SPADE tienen, a parte de los comportamientos básicos comunes a otras plataformas, dos tipos de comportamien-

to de funcionalidad avanzada: los comportamientos de tipo **Evento**, para responder a eventos ocurridos en el sistema de forma automática, y los comportamientos de tipo **Objetivo**, encargados de cumplir los objetivos del agente siguiendo el paradigma *DGOC*.

8. Para cumplir con el paradigma *DGOC* los agentes implementan además un completo soporte de invocación de servicios utilizando el estándar XML-RPC integrado en el protocolo XMPP, conocido como **Jabber-RPC**.

9. SPADE proporciona soporte para subscribirse y recibir notificaciones de eventos personalizadas utilizando el protocolo **Pub-Sub**. De este modo los agentes en SPADE pueden decidir qué tipo de eventos desean recibir y automatizar su comportamiento con respecto a estos eventos.

10. SPADE ofrece una interfaz gráfica multiplataforma basada en el lenguaje HTML. De este modo se hace posible interactuar con la plataforma desde cualquier dispositivo con un navegador web sencillo. Tanto el núcleo de la plataforma como los propios agentes ofrecen una interfaz web. Esta interfaz permite buscar agentes y servicios en la plataforma, así como enviar mensajes a cualquier agente, seguir los registros de mensajes enviados, realizar introspección de los agentes, etc.

11. Como mencionamos anteriormente, con el objetivo de ser muy *portable* SPADE ha sido desarrollado en Python. Esto permite la ejecución de la plataforma en diferentes arquitecturas y sistemas operativos como MS Windows, Linux, MacOS, MS Windows Mobile, PalmOS, SymbianOS para dispositivos móviles, etc. Además ha sido extendido al lenguaje ADA para mejorar su portabilidad [APE+06].

12. Una de las características más novedosas y potentes de SPADE es la posibilidad de usar la **Notificación de Presencia** entre componentes. Este sistema permite al sistema determinar el estado actual de los componentes conectados a la plataforma en tiempo real, pudiendo así controlar su ciclo de vida, así como poder conocer entre agentes su estado y disponibilidad y enviar eventos, mejorando así las capacidades comunicativas.

13. La gestión de conversaciones con más de un agente se hace posible gracias a otro mecanismo de Jabber: la **multi-conferencia**. La multi-conferencia se basa en las salas de chat donde los agentes pueden conectarse y compartir mensajes con el resto de agentes participantes en la sala. Este mecanismo posee también una política de seguridad, pudiendo controlar los permisos de acceso a una sala y los permisos de intervención en la misma. Sobre este protocolo se ha implementado el soporte de organizaciones virtuales.

14. El protocolo utilizado para enviar mensajes es un protocolo cliente-servidor distribuido y descentralizado. Los servidores Jabber que enrutan mensajes están diseñados para soportar una gran carga de usuarios y mensajes.

Bibliografía

[AB94] Daniel S. Weld Oren Etzioni Steve Hanks James Hendler Craig Knoblock Rao Kambhampati Anthony Barrett. Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, 67:71–112, 1994.

[ABB+86] M Accetta, R Baron, W Bolosky, D Golub, R Rashid, A Tevanian, and M Young. Mach: A new kernel foundation for UNIX development. In *Proceedings of the Summer Usenix Conference*, pages 1–16, Atlanta, GA, 1986. Carnegie Mellon University.

[ABH+02] A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D. Martin, D McDermott, S. McIlraith, S. Narayanan, M. Paulocci, and T. Payne. DAML-S: Web Service Description for the Semantic Web. *The Semantic Web—ISWC 2002*, LNCS - 2342:348–363, 2002.

[ACC04] L Aversano, G Canfora, and A Ciampi. An Algorithm for Web Service Discovery through Their

Composition. *Proceedings of the IEEE International Conference on Web Services*, pages 332–339, 2004.

[ADU71] A.V. Aho, P.J. Denning, and J.D. Ullman. Principles of optimal page replacement. *Journal of the ACM (JACM)*, 18(1):80–93, 1971.

[AJB05] E Argente, Vicente Julian, and Vicente Botti. From Human to Agent Organizations. In *First International Workshop on Coordination and Organization (CoOrg'05)*, pages 1–11, 2005.

[AJB06] E Argente, Vicente Julian, and Vicente Botti. Multi-Agent System Development based on Organizations. In *Electronic Notes in Theoretical Computer Science*, pages 55–71. Elsevier, 2006.

[APA+07] Estefania Argente, J Palanca, Gustavo Aranda, Vicente Julian, Vicente Botti, Ana Garcia-Fornes, and Agustin Espinosa. Supporting Agent Organizations. In *Multi-Agent Systems and Applications V - Lecture Notes in Computer Science*, pages 236–245. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[APE+06] G Aranda, J Palanca, A Espinosa, A Terrasa, and A García-Fornes. Towards Developing Multi-agent Systems in Ada. *Reliable Software Technologies–Ada-Europe 2006*, 4006:131–142, 2006.

[Arg05] E Argente. A proposal for an organizational MAS methodology. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1370–1370, 2005.

[Arg08] E Argente. *GORMAS: Guias para el desarrollo de Sistemas MultiAgente Abiertos basados en Organizaciones*. PhD thesis, Universitat Politècnica de València, 2008.

[Asi50] I. Asimov. *I, robot*. Gnome Press, 1950.

[Bak91] A Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49(1-3):5–23, May 1991.

[BC05] D Bovet and M Cesati. *Understanding the Linux Kernel*. O'Reilly Media, January 2005.

[BCP05] A Brogi, S Corfini, and R Popescu. Composition-oriented service discovery. *Software Composition*, 3628:15–30, 2005.

[BDO05] A Barros, M Dumas, and P Oaks. A Critical Overview of the Web Services Choreography Description Language. *BPTrends Newsletter*, 3, 2005.

[BHW08] R.H. Bordini, J.F. Hübner, and M Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley-Interscience, 2008.

[BM97] A Blum and F. Merrick. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BPL04] L Braubach, A Pokahr, and W Lamersdorf. Jadex: A short overview. *Main Conference Net. ObjectDays*, pages 195–207, 2004.

[BPML04] L Braubach, A Pokahr, D Moldt, and W Lamersdorf. Goal representation for BDI agent systems. *Second International Workshop on Programming Multiagent Systems: Languages and Tools*, 7:9–20, 2004.

[BS93] G. Buttazzo and J.A. Stankovic. Red: Robust earliest deadline scheduling. In *Proc. International Workshop on Responsive Computing Systems*, pages 100–111, 1993.

[BV03] Sharad Bansal and Jose M Vidal. Matchmaking of web services based on the DAML-S service model. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS '03)*, pages 926–927, New York, USA, 2003. ACM Press.

[CCMW01] E Christensen, F Curbera, G Meredith, and S Weerawarana. Web services description language (WSDL). Technical Report W3C Note 15, W3C, March 2001.

[CFJK08] T Cortes, C Franke, Y Jégou, and T Kielmann. XtreemOS: a Vision for a Grid Operating System. Technical Report IST-033576 (4), Information Society Tecnologies, May 2008.

[CGK+03] F Curbera, Y Goland, J Klein, F Leymann, Thatte, and S Weerawarana. Business Process Execution Language for Web Services, version 1.1. Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems., May 2003.

[CL91] A Covrigaru and R Lindsay. Deterministic Autonomous Systems. *AI Magazine*, 12(3):110–117, January 1991.

[Coz08] S Cozzini. Grid computing and e-science: a view from inside. *JCOM*, 7:2, 2008.

[CPCC04] Juan M Corchado, Juan Pavon, Emilio S Corchado, and Luis F Castillo. Development of CBR-BDI Agents: A Tourist Guide Application. *Advances in Case-Based Reasoning*, LNCS 3155:51–54, 2004.

[Das08] M Dastani. 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16:214–248, January 2008.

[DCCR07] F David, J Carlyle, E Chan, and P Reames. Improving dependability by revisiting operating system design. *Proceedings of the 3rd workshop on Hot Topics in System Dependability (HotDep'07)*, page 1, January 2007.

[DKLW07] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 296–303. IEEE, July 2007.

[DLAR91] P Dasgupta, R LeBlanc, M Ahamad, and U Ramachandran. The Clouds Distributed Operating System. *Computer*, 24:34–44, 1991.

[Doo01] Leendert van Doorn. *The Design and Application of*

an Extensible Operating System. PhD thesis, Vrije Universiteit of Amsterdam, 2001.

[dSP05] L de Silva and L Padgham. Planning as needed in BDI systems. *International Conference on Automated Planning and Scheduling*, 2005.

[DWTMW05] Mathijs De Weerdt, Adriaan Ter Mors, and Cees Witteveen. Multi-agent planning: An introduction to planning and coordination. In *Handouts of the European Agent Summer*, pages 1–32. Dept. of Software Technology, Delft University of Technology, 2005.

[EHN94] K. Erol, J Hendler, and D Nau. UCMP: A sound and complete procedure for hierarchical task network planning. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, 1994.

[EKO95] D Engler, M Kaashoek, and J O'Toole. Exokernel: an operating system architecture for application-level resource management. In *ACM SIGOPS Operating Systems Review*, pages 251–266, 1995.

[Eng98] Dawson R Engler. *The Exokernel Operating System Architecture*. PhD thesis, Massachusetts Institute of Technology, January 1998.

[Esc01] Francesc Daniel Muñoz i Escoi. *Hidra: Invocaciones fiables y control de concurrencia*. PhD thesis, Universitat Politècnica de València, 2001.

[FC01] Colin Fyfe and Juan M Corchado. Automating the construction of CBR systems using kernel methods.

International Journal of Intelligent Systems, 16(4):571–586, 2001.

[FD03] M Fox and Long D. PDDL2. 1: An extension to PDDL for expressing temporal planning domains . *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[FIP01] FIPA. FIPA ACL Message Structure Specification. Technical Report XC00061E, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2001.

[FIP02a] FIPA. FIPA Abstract Architecture Specification. Technical Report SC00001L, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[FIP02b] FIPA. FIPA Agent Message Transport Protocol for HTTP Specification. Technical Report SC00084, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[FKNT02] I Foster, C Kesselman, J Nick, and S Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[FOGC+07] J Fernández-Olivares, T Garzón, L Castillo, Ó García-Pérez, and F Palao. A Middle-Ware for the Automated Composition and Invocation of Semantic Web Services Based on Temporal HTN Planning Techniques. *Current Topics in Artificial Intelligence. Springer*, pages 70–79, 2007.

[Fou04a] Jabber Software Foundation. Extensible Messaging

and Presence Protocol (XMPP): Core. Technical report, http://www.ietf.org/rfc/rfc3920.txt, 2004.

[Fou04b] Jabber Software Foundation. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Technical report, http://www.ietf.org/rfc/rfc3921.txt, 2004.

[GAS95] AK Goel, KS Ali, and E Stoulia. Some Experimental Results in Multistrategy Navigation Planning. Technical Report GIT-CC-95-51, Georgia institute of Technology, Atlanta, GA, March 1995.

[GBC03] M. Glez-Bedia and J.M. Corchado. Constructing autonomous distributed systems using CBR-BDI agents. *Innovation in knowledge Engineering. Faucher C., Jain L. and Ichalkaranje N.(eds.)*, 2003.

[GBCSF02] M. Glez-Bedia, J.M. Corchado, E S, and C. Fyfe. Analytical model for constructing deliberative agents. *Engineering Intelligent Systems For Electrical Engineering And Communications*, 10:173–185, 2002.

[GHC+98] M. Ghallab, A Howe, D Christianson, D McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language. *AIPS98 planning committee*, 78(4):1–27, 1998.

[GMNS03a] P Giorgini, J Mylopoulos, E Nicchiarelli, and R Sebastiani. Formal reasoning techniques for goal models. *Journal on Data Semantics I*, pages 1–20, January 2003.

[GMNS03b] P Giorgini, J Mylopoulos, E Nicchiarelli, and R Sebastiani. Reasoning with goal models. *Conceptual Modeling—ER 2002*, pages 167–181, 2003.

[GS02] A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs with action costs. In *Proc. of the Sixth Int. Conf. on AI Planning and Scheduling*, pages 12–22, 2002.

[Gut01] E Guttman. Autoconfiguration for IP networking: enabling local communication. *Internet Computing, IEEE*, 5(3):81–86, 2001.

[Ham86] K Hammond. CHEF: A model of case-based planning. *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 267–271, January 1986.

[Ham90] KJ Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14:385–443, 1990.

[HBG+06] Jorrit N Herder, H Bos, B Gras, P Homburg, and Andrew S Tanenbaum. MINIX 3: A Highly Reliable, Self-Repairing Operating System. *Operating System Review*, January 2006.

[HBHM98] Koen V Hindriks, Frank S Boer, Wiebe Hoek, and John-Jules Meyer. Formal semantics for an abstract agent programming language. *Intelligent Agents IV Agent Theories, Architectures, and Languages*, 1365:215–229, 1998.

[HDBVdHM99a] Koen V Hindriks, Frank S De Boer, Wiebe Van der Hoek, and John-Jules Ch Meyer. Agent Programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2:357–401, 1999.

[HdBVDHM99b] K.V. Hindriks, F.S. de Boer, W. Van Der Hoek, and J.C.C. Meyer. An operational semantics for the single agent core of AGENT0. *UU-CS*, 1999.

[HdBVDHM01] K.V. Hindriks, F.S. de Boer, W. Van Der Hoek, and J.C.C. Meyer. Agent Programming with Declarative Goals. *Intelligent Agents VII Agent Theories Architectures and Languages*, 1986:248–257, 2001.

[HHW11] Chih-Chiang Hsu, Kuo-Chan Huang, and Feng-Jian Wang. Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems*, 27(6):860–870, 2011.

[HLA+05] Galen C Hunt, James R Larus, M Abadi, Mark Aiken, P Barham, and et al. An Overview of the Singularity Project. Technical report, MSR-TR-2005-135, Redmond, WA, 2005.

[HLTW05] Galen C Hunt, James R Larus, D Tarditi, and T Wobber. Broad New OS Research: Challenges and Opportunities. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, 2005.

[HNCJ11] A Herrero, M Navarro, E Corchado, and V Julián. RT-MOVICAB-IDS: Addressing real-time intrusion detection. *Future Generation Computer Systems (http://dx.doi.org/10.1016/j.future.2010.12.017)*, 2011.

[Hof01] Joerg Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3):57, September 2001.

[HRHL01] Nick Howden, R Ronnquist, Andrew Hodgson, and Andrew Lucas. JACK Intelligent Agents-Summary of an Agent Infrastructure. In *5th International Conference on Autonomous Agents*, 2001.

[HW95] S Hanks and D. S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, January 1995.

[HZj06] J Hao and S Zhi-jian. The tcpn-based verification of temporal consistency in web service process. In *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 302–306. IEEE Computer Society, 2006.

[IK96] L Ihrig and S Kambhampati. *Plan-space vs. State-space planning in reuse and replay*. Arizona State University, Tempe, AZ, December 1996.

[JEA+07] D. Jordan, J. Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, C. Barreto, B. Bloch, Francisco Curbera, M. Ford, Yaron Goland, and others. Web Services Business Process Execution Language Version 2.0. *OASIS Standard*, 11, 2007.

[JMM07] I Johnson, B Matthews, and C Morin. XtreemOS: Towards a Grid Operating System with Virtual Organisation Support. In *UK eScience All Hands Meeting*, 2007.

[JRR97] M.B. Jones, D Rosu, and M.C. Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *ACM SIGOPS Operating Systems Review*, pages 198–211. ACM, 1997.

[KBT+05] N. Kavantzas, D. Burdett, G. Titzinger, T. Fletcher, and Y. Lafon. Web service choreography description language (WSCDL). Technical Report 2005/CR-ws-cdl-10-20051109, W3C, 2005.

[KGS05] M Klusch, A Gerber, and M Schmidt. Semantic web service composition planning with OWLS-XPlan. *Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, January 2005.

[KH92] Subbarao Kambhampati and James A Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, June 1992.

[KO87] Hermann Kopetz and Wilhelm Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[KP08] Dimitrios Kourtesis and Iraklis Paraskakis. Combining SAWSDL, OWL-DL and UDDI for semantically enhanced web service discovery. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications (ESWC'08)*, pages 614–628, Tenerife, Canary Islands, Spain, 2008. Springer-Verlag.

[KS96] H Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.

[KVBF07] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, October 2007.

[Lak99] G. Lakemeyer. On sensing and off-line interpreting in golog. *Logical Foundations for Cognitive Agents*, pages 173–189, January 1999.

[LC01] R Laza and J.M. Corchado. Creation of Deliberative Agents Using a CBR Model. *Computing and Information Systems Journal*, 8:33–39, 2001.

[LE06] Henry Lieberman and Jose Espinosa. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. *Proceedings of the 11th international conference on Intelligent user interfaces*, pages 226–233, 2006.

[Lei80] D.W. Leinbaugh. Guaranteed response times in a hard-real-time environment. *IEEE Transactions on Software Engineering*, 1:85–91, 1980.

[Ley01] F Leymann. Web Services Flow Language (WSFL 1.0). Technical Report 05-01, IBM Software Group, Somers, NY, 2001.

[LKM99] Y Lesperance, T Kelley, and J Mylopoulos. Modeling

dynamic domains with ConGolog. *Lecture Notes in Computer Science*, 1626/2009:365–380, January 1999.

[LM08] M Luck and P McBurney. Computing as Interaction: Agent and Agreement Technologies. *Proc. of the 2008 IEEE International Conference on Distributed Human-Machine Systems*, 2008.

[LRLL97] H Levesque, R Reiter, Y Lesperance, and F Lin. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1–3):59–83, January 1997.

[LRPF04] R. Lara, D. Roman, A. Polleres, and D. Fensel. A Conceptual Comparison of WSMO and OWL-S. *Web Services*, LNCS 3250/2004:254–269, 2004.

[LS04] H. Liu and P Singh. ConceptNet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.

[Lut96] Mark Lutz. *Programming Python*. O'Reilly Media, Inc., 1996.

[MBH+04] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B Parsia, T. Payne, and others. OWL-S: Semantic markup for web services. *W3C member submission*, 22:2007–2004, 2004.

[MCB08] José M Molina, Juan M Corchado, and Javier Bajo. Ubiquitous Computing for Mobile Environments . In *Issues in Multi-Agent Systems*, pages 33–57. Birkhäuser Basel, 2008.

[MCM85] R S Michalski, J.G. Carbonell, and T M Mitchell. *Machine learning: An artificial intelligence approach*, volume 1. Morgan Kaufmann, 1985.

[MDCDM05] Octavio Martin-Diaz, Antonio Ruiz Cortes, Amador Duran, and Carlos Müller. An Approach to Temporal-Aware Procurement of Web Services. In *Service-Oriented Computing (ICSOC 2005)*, pages 170–184, 2005.

[MH08] Jan Mendling and Michael Hafner. From WS-CDL choreography to BPEL process orchestration. *Journal of Enterprise Information Management*, 21(5):525–542, 2008.

[MiEBA98] Francesc Daniel Muñoz i Escoi and José M Bernabéu-Aubán. The NanOS Object Oriented Microkernel: An Overview. Technical Report ITI-ITE-98/3, Universitat Politècnica de València, 1998.

[MKLN02] D Milojicic, V Kalogeraki, R Lukose, and K Nagaraja. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, 2002.

[MM03] J. Mendling and M. Müller. A comparison of BPML and BPEL4WS. In *Proc. 1st Conf. Berliner XML-Tage*, pages 305–316, 2003.

[MM07] N Milanovic and M Malek. Service-Oriented Operating System: A Key Element in Improving Service Availability. *Lecture Notes in Computer Science*, 4526/2007:31–42, January 2007.

[MMO+95] A Montz, D Mosberger, S O'Malley, L Peterson, and et al. Scout: A communications-oriented operating system. *Hot Topics in Operating Systems (HotOS 1995)*, January 1995.

[MSAM99] Peter Millard, Peter Saint-Andre, and Ralph Meijer. XMPP Publish-Subscribe Extension. Technical Report XEP-0060, XMPP Standards Foundation, 1999.

[MSH96] W. Mark, E. Simoudis, and D. Hinkle. Case-based reasoning: Expectations and results. *Case-Based Reasoning: experiences, lessons, and future directions*, 1996.

[MvRTa90] Sape J Mullender, G van Rossum, Andrew S Tanenbaum, and et al. Amoeba: a distributed operating system for the 1990s. *Computer*, 23(5):44—-53, January 1990.

[NBJ10] Marti Navarro, Vicente Botti, and Vicente Julian. Real-Time agent reasoning: a temporal bounded CBR approach. In *Proceedings of OPTMAS 2010*, pages 41–48, 2010.

[NHJB11] Marti Navarro, Stella Heras, Vicente Julian, and Vicente Botti. Incorporating Temporal-Bounded CBR techniques in Real-Time Agents. *Expert Systems with Applications*, page In Press., 2011.

[NK95] B Nebel and Jana Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2):427–454, July 1995.

[NT07] M Naseri and A Towhidi. QoS-Aware Automatic Composition of Web Services Using AI Planners. In *Second International Conference on Internet and Web Applications and Services, 2007. ICIW'07*, page 29. IEEE, 2007.

[Nwa96] Hyacinth S Nwana. Software agents: an overview. *The Knowledge Engineering Review*, 11(03):205–244, July 1996.

[OCD+88] J Ousterhout, A Cherenson, F Douglis, M Nelson, and B Welch. The Sprite network operating system. *Computer*, 21(2):23–36, 1988.

[Pan05] F Pan. Temporal aggregates for web services on the semantic web. In *IEEE International Conference on Web Services (ICWS 2005)*, pages 831–832, 2005.

[Pap03] MP Papazoglou. Service-oriented computing: Concepts, characteristics and directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, 3, 2003.

[PBGF09] J Palanca, Vicente Botti, and Ana Garcia-Fornes. Towards organizational agent-oriented operating systems. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09)*, pages 752–756, Honolulu, Hawaii, March 2009. ACM.

[PG03] M Papazoglou and D Georgakopoulos. Service-Oriented Computing. *Communications of the ACM*, January 2003.

[PJGF11] J Palanca, V Julián, and A García-Fornes. A Goal-Oriented Execution Module Based on Agents. In *44th Hawaii International Conference on System Sciences (HICSS 2011)*, pages 1–10, 2011.

[PL05] M Parashar and C Lee. Grid Computing: Introduction and Overview. *Proceedings of the IEEE*, January 2005.

[PNA+12] J Palanca, M Navarro, E Argente, A García-Fornes, and V Julián. Modeling an Operating System Based on Agents. *Hybrid Artificial Intelligent Systems*, pages 588–599, 2012.

[PNGFJ12] J Palanca, M Navarro, A García-Fornes, and V Julián. A New Deliberation Mechanism for Service-Oriented Operating Systems. *2012 45th Hawaii International Conference on System Sciences*, pages 5463–5472, 2012.

[PNGFJ13] J Palanca, M Navarro, A. García-Fornes, and V Julián. Deadline prediction scheduling based on benefits. *Future Generation Computer Systems*, 29(1):61–73, 2013.

[PNJGF12] J Palanca, Marti Navarro, Vicente Julian, and A García-Fornes. Distributed Goal-oriented Computing. *Journal of Systems and Software (http://dx.doi.org/10.1016/j.jss.2012.01.045)*, 85(7):1540–1557, 2012.

[PPTT95] R Pike, D Presotto, K Thompson, and H Trickey. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221, January 1995.

[PR93] J. Pinto and R Reiter. Temporal Reasoning in Logic Programming: A Case for the Situation Calculus . In *Logic programming: proceedings of the Tenth International Conference on Logic Programming*, page 203, January 1993.

[PTD+06] M Papazoglou, P Traverso, S Dustdar, F Leymann, and BJ Krämer. 05462 Service-Oriented Computing: A Research Roadmap. In *Service Oriented Computing (SOC)*, pages 223–255, 2006.

[PW92] JS Penberthy and D Weld. UCPOP: A sound, complete, partial order planner for ADL. In *proceedings of the third international conference on knowledge representation and reasoning*, pages 103–114, 1992.

[Ram08] R Ramakrishnan. Cloud Computing- Was Thomas Watson Right After All? *IEEE 24th International Conference on Data Engineering*, January 2008.

[Red06] Y.V Reddy. Pervasive Computing: Implications, Opportunities and Challenges for the Society. *1st International Symposium on Pervasive Computing and Applications*, page 5, 2006.

[RFRR01] R.P. Rial, R.L. Fidalgo, A.G. Rodriguez, and J.M.C. Rodriguez. Improving the revision stage of a CBR system with belief revision techniques. *Computing and Information Systems*, 8:40–45, 2001.

[RG95] A Rao and M Georgeff. BDI agents: From theory to practice. *Proceedings of the first international conference on multi-agent systems (ICMAS95)*, pages 312–319, January 1995.

[Rit78] D Ritchie. Unix Time-Sharing System: A Retrospective. *Tenth Hawaii International Conference on the System Sciences*, January 1978.

[RJO+89] R Rashid, D Julin, D Orr, R Sanzi, R Baron, A Forin, D Golub, and M Jones. Mach: a system software kernel. *COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*, pages 176—-178, 1989.

[RN95] S. Russell and P. Norvig. *Artificial intelligence: a modern approach* , volume 74. Prentice hall Englewood Cliffs, NJ, 1995.

[RT73] D Ritchie and K Thompson. The UNIX time-sharing system. *Communications of the ACM*, January 1973.

[Rus98] D Rusling. *The linux kernel*. Linux Documentation Project, 1998.

[SA06] Peter Saint-Andre. XEP-0045: Multi-User Chat. Technical Report XEP0045, XMPP Standards Foundation, 2006.

[SCZ04] M. Solanki, A. Cau, and H. Zedan. Augmenting semantic web service description with compositional specification. In *Proceedings of the 13th international conference on World Wide Web*, pages 544–552. ACM, 2004.

[SF79] M Solomon and R Finkel. The Roscoe distributed operating system. *Proceedings of the seventh ACM symposium on Operating Systems Principles*, pages 108–114, 1979.

[SGGS98] A. Silberschatz, P.B. Galvin, G. Gagne, and A. Silberschatz. *Operating system concepts*, volume 4. Addison-Wesley, 1998.

[Sha01] R. Shapiro. A Comparison of XPDL, BPML and BPEL4WS. Cape Visions. *Rough Draft*, pages 1–17, 2001.

[Sho93] Y Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, March 1993.

[Sin02] P Singh. The public acquisition of commonsense knowledge. In *Proceedings of AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*, 2002.

[SK00] DG Schmidt and F. Kuhns. An overview of the real-time CORBA specification. *Computer*, 33(6):56–63, 2000.

[Spa01] Luca Spalzzi. A Survey on Case-Based Planning. *Artif. Intell. Rev.*, 16(1):3–36, 2001.

[SPW+04] E Sirin, B Parsia, D Wu, J Hendler, and D Nau. HTN planning for web service composition using SHOP2. *Web Semantics: Science*, January 2004.

[SSS98] D Solomon, D Seminars, and C Sherman. The Windows NT kernel architecture. *Computer*, January 1998.

[SY03] R G Simmons and H L S Younes. VHPOP: Versatile Heuristic Partial Order Planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.

[Syc88] K Sycara. Resolving goal conflicts via negotiation. *Proceedings of the Seventh National conference on artificial Intelligence (AAAI 1988)*, 1:245–250, 1988.

[Tan96] Andrew S Tanenbaum. The Amoeba Distributed Operating System. *CERN European Organization for Nuclear Research*, pages 109–122, 1996.

[Tan08] Andrew S Tanenbaum. *Modern Operating Systems. 3rd edition.*, volume 2. Prentice Hall New Jersey, 2008.

[TBW95] K. Tindell, A. Burns, and A J Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[Tha01] S. Thatte. XLANG: Web services for business process design. *Microsoft Corporation*, page 13, 2001.

[TM81] Andrew S Tanenbaum and Sape J Mullender. An Overview of the Amoeba Distributed Operating System. *SIGOPS Operating Systems Review*, 15(3):51—-64, 1981.

[TPH02] J Thangarajah, L Padgham, and J Harland. Representation and reasoning for goals in BDI agents. *Australian Computer Science Communications*, January 2002.

[TSPB02] R K Thiagarajan, A K Srivastava, A K Pujari, and V K Bulusu. BPML: A Process Modeling Language for Dynamic Business Models. In *Proceedings Fourth*

IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002), pages 222–224. IEEE Comput. Soc, 2002.

[Tuc05] Matt Tucker. XEP-0142: Workgroups. Technical Report XEP0142, XMPP Standards Foundation, 2005.

[UB06] Ubisoft and Bethesda. *The Elder Scrolls IV: Oblivion*. http://oblivion.es.ubi.com/, 2006.

[VCP+95] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:81, 1995.

[vDHT95] L van Doorn, P Homburg, and AS Tanenbaum. Paramecium: an extensible object-based kernel. In *Fifth Workshop on Hot Topics in Operating Systems, 1995. (HotOS-V)*, pages 86–89, 1995.

[VHMW01] Emilio Vivancos, Christopher A Healy, Frank Mueller, and David Whalley. Parametric Timing Analysis. In *ACM SIGPLAN Notices*, pages 88–93, 2001.

[VNJR09] Elena Del Val, Marti Navarro, Vicente Julian, and Miguel Rebollo. Ensuring time in service composition. In *SERVICES 2009. 2009 IEEE Congress on Services*, pages 376–383, 2009.

[VP96] Walter Velde and John W Perram, editors. *AgentSpeak(L): BDI agents speak out in a logical computable language*, volume 1038 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin/Heidelberg, 1996.

[WdBB+05] W3C, Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta Kinig-Ries, Jacek Kopecky, Ruben Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web Service Modeling Ontology (WSMO) - http://www.w3.org/Submission/WSMO/. *Interface*, 5:1, June 2005.

[Wel94] Daniel S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4):27, 1994.

[Wel99] Daniel S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93, 1999.

[Whi04] SA White. Introduction to BPMN. IBM Software Group, 2004.

[WJ95] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(02):115–152, 1995.

[WJK99] Michael Wooldridge, Nicholas R Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the third annual conference on Autonomous Agents - AGENTS'99*, pages 69–76, New York, New York, USA, 1999. ACM Press.

[Woo00] M Wooldridge. *Reasoning about rational agents*. The MIT Press, 2000.

[WTKW08] Chee Siang Wong, Ian Tan, Rosalind Deena Kumari, and Fun Wey. Towards achieving fairness in the Li-

nux scheduler. *SIGOPS Oper. Syst. Rev.*, 42(5):34–43, 2008.

[YSP99] S. Yannis, K. Scott, and W. Paul. EELRU: Simple and Effective Adaptive Page Replacement. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 27:122–133, 1999.

[ZBDTH06] J Zaha, A Barros, M Dumas, and A Ter Hofstede. Let's Dance: A Language for Service Behavior Modeling. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 145–162, 2006.

[Zil96] Shlomo Zilbertein. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, 17(3):73–83, 1996.

«¡Que la Fuerza,
te acompañe!»
– Yoda (The Empire Strikes Back)

`:wq`