PROYECTO FIN DE CARRERA

DESARROLLO DE UN PROTOTIPO SOFTWARE PARA LA EJECUCIÓN DE SERVICIOS GRID SEMÁNTICOS BASADO EN TECNOLOGÍA DE AGENTES

Facultad de Informática

Universidad de Murcia

Febrero 2008

Autor:

Ignacio Sánchez Peral

Directores:

Francisco García Sánchez

Rafael Valencia García

Índice

1. RESUMEN **7**

2. INTRODUCCIÓN Y REFERENCIAS HISTÓRICAS **9**

2.1. LA WEB SEMÁNTICA 9

2.1.1. Fundamentos de la Web Semántica 9

2.1.2. Arquitectura de la Web Semántica 10

2.1.3. Ontologías 11

2.1.4. Lenguajes de codificación de ontologías 14

2.2. SERVICIOS GRID SEMÁNTICOS 17

2.2.1. Servicios Grid 17

2.2.2. Servicios Grid vs. Servicios Web 19

2.2.3. Servicios Grid semánticos 21

2.2.4. OGSA y S-OGSA 24

2.2.5. OGSI y WSRF 28

2.3. AGENTES INTELIGENTES 29

2.3.1. Definiciones y conceptos 29

2.3.2. Sistemas multiagentes 31

3. ANÁLISIS DE OBJETIVOS Y METODOLOGÍAS **33**

3.1. ÁMBITO DEL PROYECTO Y OBJETIVOS 33

3.1.1. Ámbito del proyecto 33

3.1.2. Objetivos y tareas 34

3.2. METODOLOGÍA, HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS 36

3.2.1. Diseño de la aplicación 36

3.2.2. Diseño de los Servicios Grid 40

3.2.3. Diseño de las ontologías 43

3.2.4. Lenguaje de programación 44

3.2.5. Plataforma multiagente 45

4. DISEÑO Y RESOLUCIÓN DEL TRABAJO **51**

4.1. TRABAJO RELACIONADO 51

4.1.1. Los agentes en la Web Semántica 51

4.1.2. Los agentes inteligentes en el Grid 52

4.1.3. Otros trabajos y proyectos relacionados 53
4.2. ANÁLISIS DE REQUISITOS 55
4.2.1. Requisitos funcionales 55
4.2.2. Requisitos de interfaz 56
4.2.3. Requisitos de calidad 56
4.2.4. Requisitos de entorno 56
4.2.5. Requisitos de seguridad 57
4.3. DESCRIPCIÓN GENERAL 57
4.3.1. Arquitectura del sistema 57
4.3.2. Detalles de la implementación 64
4.3.3. Arquitectura de la aplicación Web 66
4.3.4. Conexión de la aplicación Web con el sistema de agentes 69
4.3.5. Funcionamiento 72
4.3.6. Aplicación en matemáticas 73

5. CONCLUSIONES Y VÍAS FUTURAS 81
5.1. CONCLUSIONES 81
5.2. VÍAS FUTURAS 82
5.2.1. Madurez de las ontologías de Servicios Grid 82
5.2.2. Aprovechamiento de las propiedades stateful y transiente 83
5.2.3. Mejoras en el reconocimiento del lenguaje natural 83

Índice de figuras

1. Buscador actual . . . 9
2. Buscador semántico . . . 10
3. Arquitectura de la Web Semántica . . . 11
4. Definición de ontología . . . 12
5. Ejemplo de clase . . . 12
6. Ejemplo de relación . . . 13
7. Ejemplo de instancias o individuos . . . 13
8. Lenguajes para la definición de ontologías . . . 14
9. Ejemplo de tripleta en RDF . . . 15
10. Versiones del lenguaje OWL . . . 17
11. Relación entre OGSA y WSRF . . . 19
12. Elementos de OWL-S . . . 22
13. Elementos de WSMO . . . 23
14. Capacidades de OGSA . . . 25
15. Arquitectura OGSA y S-OGSA . . . 26
16. Entidades S-OGSA y sus relaciones . . . 27
17. Convergencia de WSRF . . . 28
18. Agente . . . 30
19. Operaciones con Semantic Bindings . . . 42
20. Arquitectura FIPA . . . 46
21. Plataformas en JADE . . . 49
22. Ejemplo de mapeo entre ontologías . . . 52
23. Arquitectura del sistema multiagente . . . 58
24. Interacciones entre los agentes . . . 59
25. Clasificación de los roles del sistema . . . 60
26. Arquitectura de la aplicación Web . . . 68
27. Estructura del patrón DAO . . . 69
28. Estructura del sistema de comunicación . . . 70
29. Diagrama de secuencia de la aplicación . . . 73
30. Diagrama de agentes en JADE . . . 74
31. Petición de usuario . . . 78
32. Servicios encontrados . . . 79
33. Resultados obtenidos . . . 80

Índice de tablas

1. Diferencias entre RPD y SMA 32
2. Actividades de INGENIAS 40
3. Contenido de los mensajes en Jade 49
4. Roles relacionados con los servicios 60
5. Roles relacionados con el funcionamiento de la plataforma 61
6. Correspondencia entre los agentes y sus roles 62

1. RESUMEN

El proyecto sobre el que trata este documento surgió como ampliación del proyecto SEMMAS desarrollado en [21] y [54]. El objetivo del proyecto SEMMAS era construir un marco de trabajo (framework) en el que, a partir de agentes inteligentes y el uso intensivo de ontologías, se mejorara el rendimiento en el consumo de Servicios Web e incrementara la automatización de las tareas de gestión de los mismos, de forma que se explote el dinamismo intrínseco de este tipo de entornos. Uno de las vías futuras propuestas en dicho proyecto fue la de extrapolar la funcionalidad de SEMMAS a otros tipos de servicios, y más concretamente a los Servicios Grid. En este proyecto se han analizado las diferencias entre estas tecnologías y su contraparte semántica, y se han efectuado los cambios precisos sobre el framework para la explotación de Servicios Grid Semánticos.

Este documento está dividido en cuatro partes que dan lugar a cuatro capítulos bien diferenciados. Estos son:

1. **Introducción y referencias históricas:** este apartado hace referencia a las tecnologías base sobre las que se construye el framework. Primero hace un repaso a la Web Semántica, su nacimiento y evolución, y sobre qué pilares se sustenta. Luego introduce los Servicios Grid, arquitecturas de referencia y estándares de implementación. Explica cómo es posible convertir Servicios Grid en Servicios Grid Semánticos y ofrece una comparativa entre estos y los Servicios Web. Y, por último, trata los agentes inteligentes, sus características fundamentales y su interacción, que dará como resultado la creación de Sistemas Multiagente.

2. **Análisis de objetivos y metodologías:** en esta sección se sitúa el proyecto dentro del marco correspondiente y se explican brevemente los proyectos que le preceden. También se fijan los objetivos, generales y concretos, y, a grandes rasgos, se marcan las tareas que abarcan el desarrollo del proyecto.

3. **Diseño y resolución del trabajo:** en esta parte del documento se incluye, primero una referencia a los trabajos y proyectos más importantes e influyentes relacionados con este. A continuación, se expone el análisis de los requisitos del sistema (divididos en funcionales, de calidad, de entorno y de seguridad). Y, finalmente, se describe la aplicación construida, desde la arquitectura del sistema de agentes, hasta su funcionamiento interno, pasando por la aplicación Web que incluye, y cómo se integra con dicho sistema de agentes.

4. **Conclusiones y vías futuras:** en este último apartado, como su propio nombre indica, se exponen las conclusiones sacadas de la elaboración de este proyecto, así como sus ventajas y los problemas surgieron durante su desarrollo. Finalmente, se

identifican diversas posibles modificaciones sobre el sistema desarrollado que mejoren su funcionamiento.

2. INTRODUCCIÓN Y REFERENCIAS HISTÓRICAS

2.1. LA WEB SEMÁNTICA

2.1.1. Fundamentos de la Web Semántica

La Web actual es un gigantesco repositorio de información desordenada que, si bien supuso un avance tecnológico sin precedentes, aun adolece de ciertas carencias. La principal y más significativa es que no incorpora mecanismos de procesado automático de la información, esto es, que no permite que la información pueda ser interpretada por máquinas. Dicho de otro modo, las tecnologías que conforman la Web actual no son capaces de captar la semántica de la información y, por tanto, no pueden entenderla ni procesarla de una manera "inteligente".

Un ejemplo claro de esta incapacidad se puede ver reflejado en la Figura 1. En este se muestra cómo los resultados de la búsqueda contienen las palabras clave que el usuario ha introducido, pero no reflejan lo que realmente se está buscando.

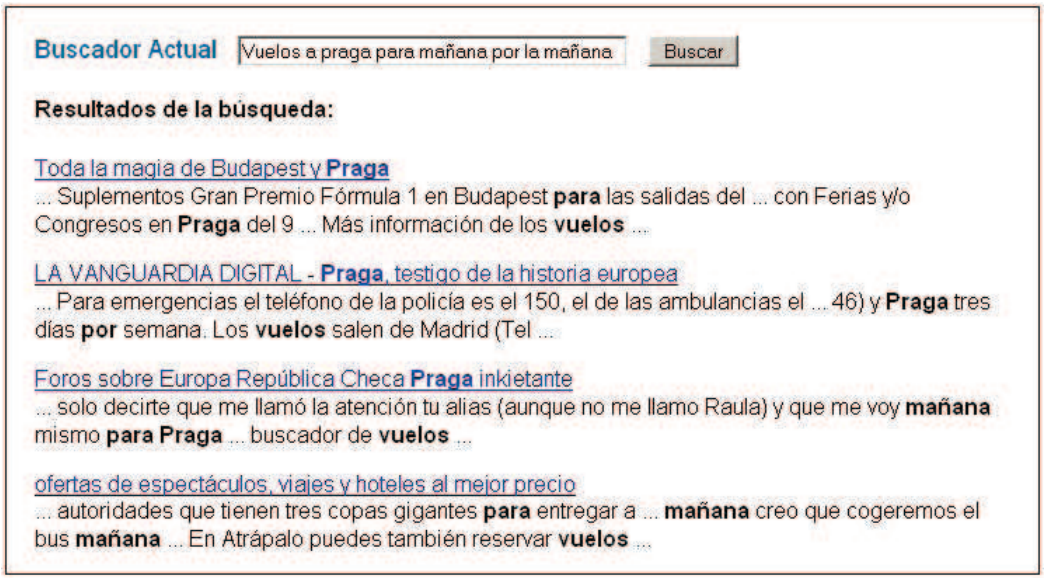

Figura 1: Buscador actual [24]

La solución a este problema reside en el concepto de Web semántica. La idea consiste en convertir los datos en conocimiento mediante su representación en algún lenguaje formal, para que las máquinas puedan comprenderlo. Esta comprensión no debe entenderse como "comprensión humana" sino más bien como "inferencia, deducción". Es decir, las máquinas deben ser capaces de inferir conclusiones a partir de los datos mediante un procedimiento lógico-matemático. Tim Berners-Lee, creador de la Web semántica, lo explica de la siguiente manera:

> *El concepto de documentos que las máquinas puedan entender no implica ninguna inteligencia artificial que permita a la máquina comprender los murmullos humanos. Sólo indica la habilidad de la máquina para resolver un problema bien definido desarrollando operaciones bien definidas sobre datos bien definidos. En lugar de pedir a las máquinas que entiendan el lenguaje humano, implica pedir a la gente que haga un esfuerzo.* [1]

Este esfuerzo implica la representación de los datos por medio de algún lenguaje formal. En particular, se trata de metadatos añadidos a los recursos de la Web, dotados de significado semántico, entendible para los agentes encargados de procesar la información.

> *La Web semántica proporcionará estructura al contenido importante de las páginas web, creando un entorno donde los agentes de software que viajan de página en página puedan enseguida llevar a cabo complicadas tareas para los usuarios. Así, un agente que venga de la página web de la clínica no sólo sabrá que la página tiene palabras como "tratamiento, medicina, médico, terapia" (como puede codificarse hoy), sino también que el Dr. Hartman trabaja en la clínica los lunes, miércoles y viernes, y que el script toma un rango de datos en el formato año-mes-día y devuelve fechas y horas de consulta.* [4]

En la Figura 2, se puede comprobar cómo el problema descrito previamente ha desaparecido, pues ahora las búsquedas se realizan sobre los metadatos, y no sobre palabras sueltas dispersas en el código.

Figura 2: Buscador semántico [24]

2.1.2. Arquitectura de la Web Semántica

El modelo de Web semántica es una jerarquía formada por varias capas o niveles [29] (ver Figura 3):

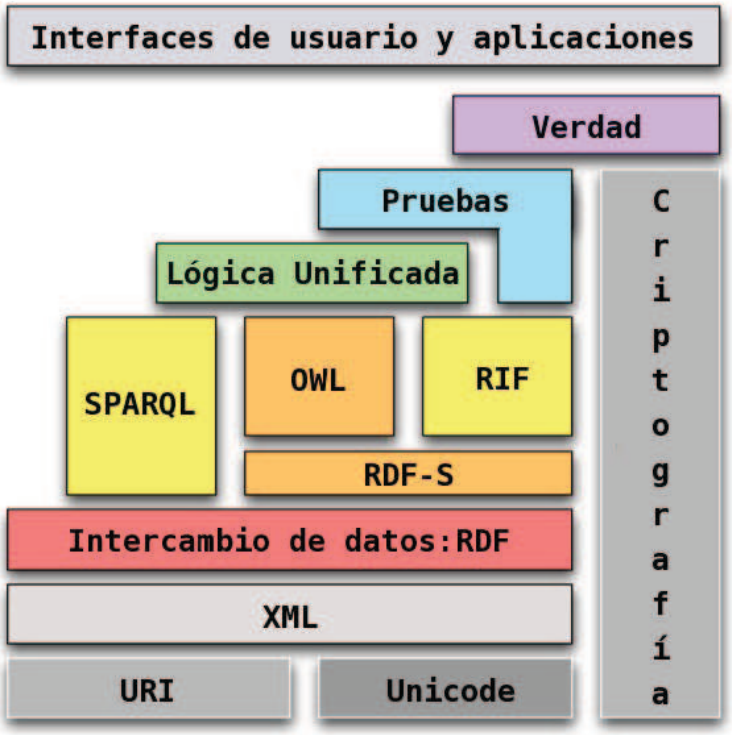

Figura 3: Arquitectura de la Web Semántica [52]

- El nivel de recursos identifica de manera inequívoca los recursos Web por medio de URIs (Uniform Resource Identifier).
- El nivel sintáctico soluciona el problema de añadir contenido semántico a las páginas Web con XML (eXtensible Markup Language).
- El nivel de descripción de recursos estandariza la definición y el uso de las descripciones de meta-datos de los recursos web mediante RDF (Resource Description Framework).
- El nivel de ontologías es el pilar central de la arquitectura. El conocimiento, además de ser entendible por la máquina, debe ser consensuado y reutilizable. Web Ontology Language (OWL) es el lenguaje más utilizado.
- El resto de niveles simplemente dan flexibilidad a la arquitectura.

2.1.3. Ontologías

El concepto de ontología, desde un punto de vista filosófico, se puede definir como *"la parte de la metafísica que estudia la naturaleza de la existencia"* [44]. Sin embargo, en el ámbito informático, las ontologías no deben ser tomadas como "entidades naturales" que

se descubren, sino como “entidades computacionales”, recursos artificiales que se crean. Se pueden interpretar, por consiguiente, como un entendimiento común y compartido de un dominio, que puede comunicarse entre agentes humanos o computacionales. Esta última característica permite que puedan compartirse y reutilizarse. Las ontologías son, por estas razones, el mecanismo elegido para la representación del conocimiento en la Web Semántica.

En informática, se define ontología como *“una especificación explicita y formal de una conceptualización compartida”* [51] (ver Figura 4).

Figura 4: Definición de ontología

Las ontologías tienen los siguientes componentes, que utilizan para representar un determinado dominio [23]:

- **Clases:** se utilizan para definir los conceptos que se intentan formalizar. Un concepto es un conjunto de instancias con cierto interés para el modelo de dominio, y con unas características comunes que nos permiten agruparlas.

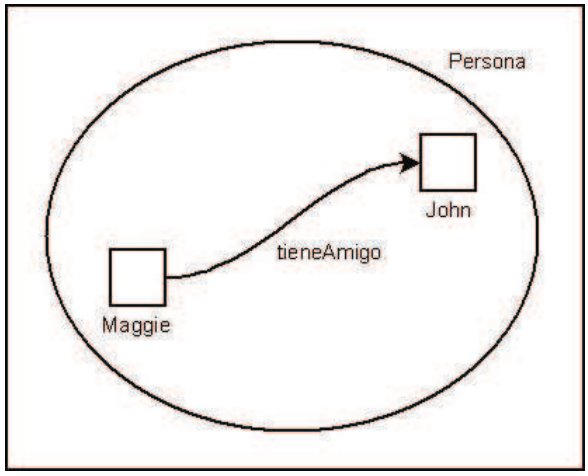

Figura 5: Ejemplo de clase

- **Atributos:** representan características de los conceptos y son descritos por medio de un valor básico.

- **Relaciones:** representan la interacción y enlace entre los conceptos del dominio. Las relaciones más relevantes son las taxonómicas (subclase-de) y las partonómicas (parte-de), aunque no son las únicas.

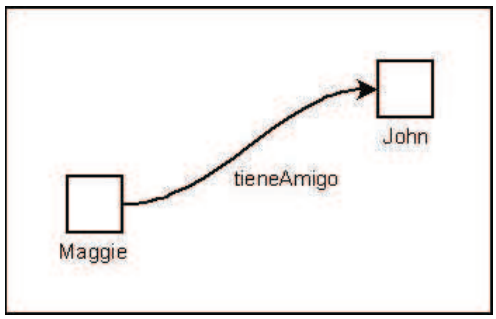

Figura 6: Ejemplo de relación

- **Funciones:** son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.

- **Axiomas:** son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Permiten inferir conocimiento que no esté indicado explícitamente.

- **Instancias:** es la representación en el mundo real de un concepto. Todos los atributos de una instancia tienen un valor concreto.

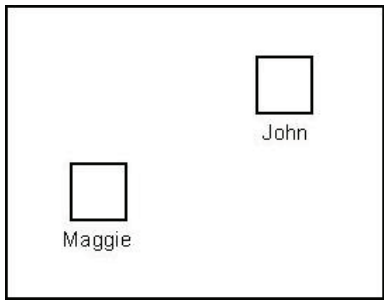

Figura 7: Ejemplo de instancias o individuos

Gracias a las ontologías se consigue una representación formal de la información de la Web, se puede procesar dicha información e inferir más información de ella. Por todo esto, las ontologías han llegado a ser la piedra angular de la Web Semántica.

2.1.4. Lenguajes de codificación de ontologías

Para que las ontologías sean útiles para la Web Semántica, esto es, puedan ser comunicables, procesables y reutilizables, deben poder ser definidas mediante lenguajes formales apropiados.

Durante muchos años se han utilizado diversos lenguajes basados en formalismos lógicos para representar ontologías (KIF, OCML, F-Logic), aunque no tuvieron apenas relevancia. No obstante, tras la aparición de la Web Semántica, comenzaron a surgir nuevos lenguajes de codificación de ontologías, con la particularidad de que fueron definidos para ser compatibles con la arquitectura WWW y con la arquitectura de la Web Semántica (ver Figura 3). Este hecho les proporcionó mayor aceptación y difusión.

Si se toma la parte de la jerarquía (Figura 3) que interesa en este punto, se obtiene la subjerarquía de lenguajes de la Figura 8.

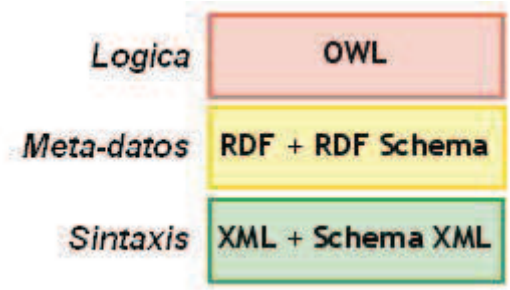

Figura 8: Lenguajes para la definición de ontologías

A continuación, se describe con más detalle cada uno de los lenguajes de la jerarquía.

XML (eXtensible Markup Language)

XML es un lenguaje de etiquetas desarrollado por el World Wide Web Consortium (W3C) como adaptación de SGML[1] (Standard Generalized Markup Language). Su importancia se la debe al hecho de que no es realmente un lenguaje en sí, sino una manera de definir la gramática de otros lenguajes, según las distintas necesidades. Por otro lado, otra de las ventajas de XML es su idoneidad para el transporte a través de Internet, ya que se basa en texto ordinario. También cabe destacar su carácter autodescriptivo, ya que puede almacenar tanto datos, como la estructura de estos, lo que facilita en gran medida su validación [1].

En definitiva, XML, aportó a la Web todo lo que necesitaba para dar el paso hacia la Web Semántica. Sin embargo, a pesar de todas sus ventajas, no incorpora ningún mecanismo

[1] SGML consiste en un sistema para la organización y etiquetado de documentos estandarizado en 1986. Sirve para especificar las reglas de etiquetado de documentos y no impone en sí ningún conjunto de etiquetas en especial.

que garantice la interoperabilidad semántica, no ofrece una interpretación común de los datos; únicamente sirve para especificar el formato y la estructura.

La solución a esto surgió con RDF gracias a la capacidad de XML de definir nuevos lenguajes.

RDF (Resource Description Framework) y RDFS (RDF Schema)

RDF surgió para definir información sobre cualquier dominio, más concretamente, para "especificar una semántica para los datos (basados en XML), de una manera interoperable y estandarizada" [1]. Y, para ello, se basa en la idea de convertir las declaraciones de los recursos en expresiones de la forma sujeto-propiedad-objeto (tripletas). En la Figura 9, se puede ver un ejemplo de tripleta de forma gráfica.

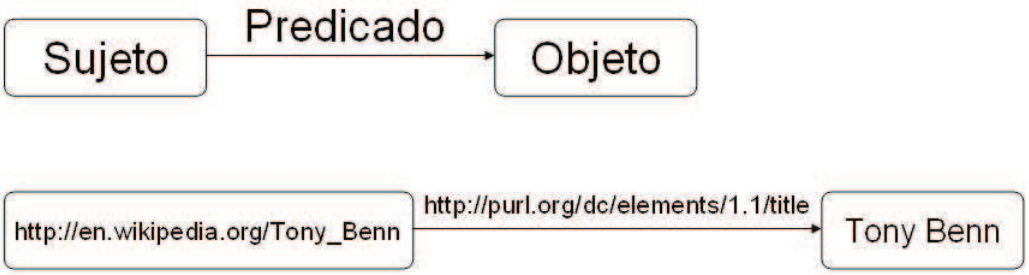

Figura 9: Ejemplo de tripleta en RDF

- **Sujeto:** recurso al que se refiere la instancia. Puede ser cualquier cosa en un modelo de datos (documento, producto, etc.). Cada recurso se identifica de manera única mediante una URI.
- **Propiedad (Predicado):** representa una característica del sujeto. Al igual que estos, cada propiedad se identifica mediante una URI.
- **Objeto:** representa el valor de la propiedad del sujeto.

De la misma manera, es necesario un mecanismo que permita crear los vocabularios RDF. RDFS permite, al igual que XML Schema con XML, crear esquemas de RDF para definir vocabularios de términos y relaciones entre dichos términos. Al ser un mecanismo similar a XML Schema, también posee todas sus ventajas; permite definiciones no ambiguas de un mismo término mediante namespaces, y también permite especificar restricciones de tipos de datos para los sujetos y objetos. Además introduce primitivas del lenguaje orientado a objetos, tales como Class, Property, etc.

RDF/RDFS otorga una semántica propia al conjunto de datos. Esto, como se comentó anteriormente, no es posible con XML. Sin embargo RDFS por sí solo no es suficientemente expresivo, así que se hace necesario el uso de lenguajes más avanzados, como OWL.

OWL (Web Ontology Language)

Para cubrir las carencias de RDF/RDFS, comenzaron a surgir nuevos lenguajes con mayor capacidad expresiva, como DAML (DARPA Agent Markup Language) y OIL (Ontology Inference Layer), lenguajes que más tarde se unirían para dar lugar a DAML+OIL. Estos lenguajes, sin embargo, fueron ideados para áreas específicas (comercio electrónico, entornos científicos, etc.). Es por esto que apareció OWL, diseñado para adaptarse perfectamente a la arquitectura de la Web Semántica.

OWL fue creado por el W3C como una extensión de RDFS (utiliza el sistema de tripletas) derivada del lenguaje DAML+OIL. Apareció como resultado de la búsqueda de un lenguaje de especificación semántica que sirviera como estandar de todos los investigadores de la Web Semántica.

Existen tres variantes de OWL con diferentes relaciones de expresividad/decidibilidad[2] [39]. Son las siguientes (ver Figura 10):

- **OWL Lite:** es suficiente cuando solamente se necesita una clasificación jerárquica y restricciones simples. Es el más fácil de entender e implementar, pero su expresividad es reducida.

- **OWL DL (Description Logic):** es el lenguaje indicado en los casos en los que se requiere la mayor expresividad posible, conservando completamente la computacionalidad (todas las conclusiones son computables) y resolubilidad (todas las conclusiones se obtienen en un tiempo finito).

- **OWL Full:** ofrece la máxima expresividad y la libertad sintáctica de RDF, pero no ofrece garantías computacionales. Es muy improbable que algún software racional pueda soportar por completo el razonamiento para cada característica de OWL Full.

OWL Full puede verse como una extensión de RDFS, mientras que OWL DL y OWL Lite, como extensiones de vistas restringidas de RDF. Esto es, cualquier documento OWL es también un documento RDF, y cualquier documento RDF es OWL Full. Sin embargo, sólo algunos documentos RDF serán OWL DL u OWL Lite. Por este motivo, la tarea de migrar documentos de RDF a OWL no es trivial.

[2] La decibilidad se refiere a la capacidad, a la hora de inferir un razonamiento, de obtener una respuesta para cualquier entrada en un tiempo finito.

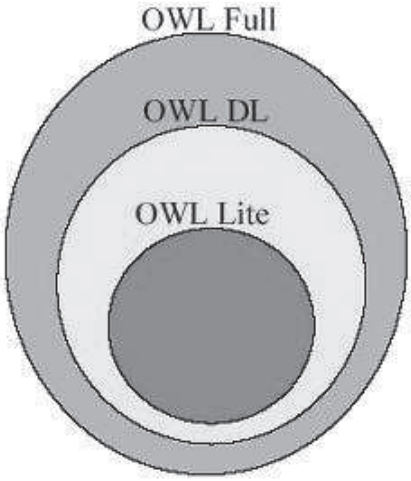

Figura 10: Versiones del lenguaje OWL [54]

2.2. SERVICIOS GRID SEMÁNTICOS

2.2.1. Servicios Grid

Actualmente, las empresas se esfuerzan por conseguir una mayor productividad a la vez que intentan reducir sus costes en investigación y desarrollo, e incrementar sus capacidades tecnológicas y de cómputo. Generalmente bastaría con aumentar la potencia de procesamiento de la organización, sin embargo, la empresa desaprovecha inconscientemente gran parte de dicho potencial de procesamiento, lo cual hace de esto un gasto injustificado. Es entonces cuando nace la tecnología Grid.

El origen de la tecnología Grid se encuentra en la computación distribuida, cuyo funcionamiento está basado en la agregación y asignación de recursos disponibles de una organización, con el objetivo de distribuir la carga de trabajo de una forma eficiente. La evolución de la computación distribuida dio lugar a la computación Grid. La computación Grid permite el uso compartido de los recursos heterogéneos de la Web, incluso a través de áreas geográficas dispersas, de una forma coordinada, transparente y segura. Análogamente con la Web:

> *Mientras que la Web es un servicio para compartir información a través de Internet, el Grid es un servicio para compartir potencia de cálculo y capacidad de almacenamiento a través de la red.* [35]

Entre los numerosos beneficios que aporta la computación Grid, encontramos los siguientes [20]:

- Permite a las organizaciones agregar recursos a su infraestructura tecnológica sin importar dónde estén localizados geográficamente.

- Las organizaciones mejoran la calidad y el tiempo de entrega de sus productos y servicios, además de reducir su coste de producción.
- Permite compartir bases de datos remotas.
- Permite a organizaciones con diferentes sedes una fácil colaboración en proyectos conjuntos, y compartir recursos entre ellas.
- La infraestructura tecnológica se hace más robusta y resistente, capaz de responder a desastres.
- Puede aprovechar capacidad de procesamiento de ordenadores de escritorio que se encuentren geográficamente dispersos, por ejemplo cuando se encuentren ociosos.

La computación Grid es posible gracias al Grid middleware[3], el cual permite la integración de todos los distintos tipos de recursos participantes. El middleware es el cerebro del Grid. Virtualiza los recursos y proporciona, de forma transparente, las siguientes funcionalidades:

- Asignación de recursos.
- Ejecución de trabajos y transferencia de resultados.
- Almacenamiento, registro, localización y acceso.
- Seguridad.
- Monitorización.

El Grid middleware está construido sobre la base tecnológica de los Servicios Grid, y estos a su vez están basados en la tecnología de los Servicios Web. Los Servicios Grid surgieron al mismo tiempo que los ingenieros comenzaron a desarrollar el Grid middleware, utilizando Servicios Web como base tecnologica. Al poco tiempo, se dieron cuenta de que los Servicios Web carecían de algunas de las características que necesitaban:

- No mantienen el estado de una invocación a otra.
- No pueden crear varias instancias de un mismo servicio, ni destruirlas.
- No incluyen servicios de apoyo tales como, notificaciones, servicio de persistencia, gestión del ciclo de vida, etc.

[3] En un entorno de computación distribuida, el middleware se define como la capa de software que se encuentra entre el sistema operativo y las aplicaciones en cada host que participa en el sistema.

Entonces optaron por crear lo que denominaron Servicios Grid. Y, finalmente, en Enero de 2004, se publicó el estándar WSRF (Web Service Resouce Framework), que permitió incorporar a los Servicios Web todas las características necesarias de las que carecían, y que los convertían en Servicios Grid.

El esqueleto de la arquitectura Grid, así como el modelo de programación de los Servicios Grid, está definido por la plataforma OGSA (Open Grid Services Architecture). Así define OGSA el GGF[4] (Global Grid Forum):

> *Es un marco (framework) de amplio espectro de aplicación para la integración de Sistemas Distribuidos.* [19]

OGSA define las interfaces, comportamientos, modelos de recursos, etc. que constituyen la plataforma. Trata así de estandarizar el acceso a los servicios de la arquitectura Grid.

En la Figura 11, se puede observar cómo se integran las tecnologías OGSA, WSRF y los Servicios Web.

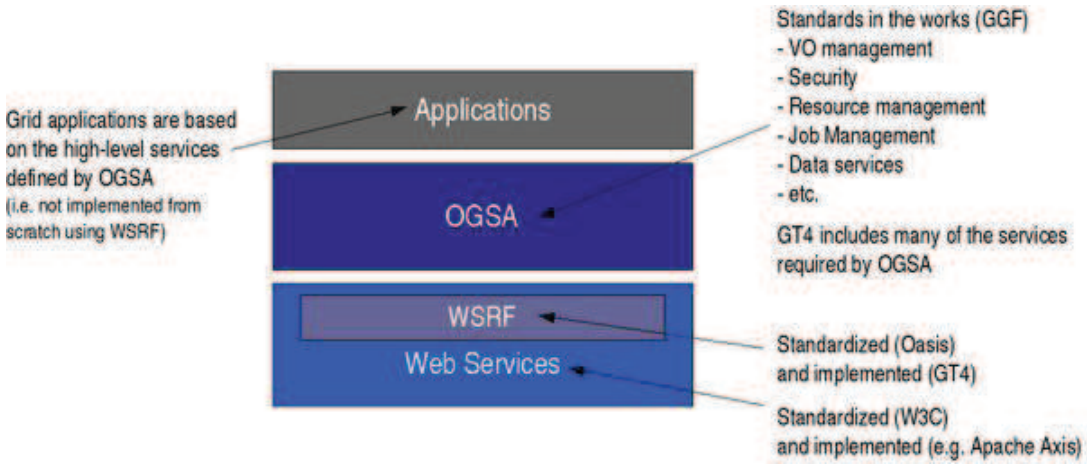

Figura 11: Relación entre OGSA y WSRF [49]

2.2.2. Servicios Grid vs. Servicios Web

Como se ha dicho antes, los Servicios Grid están basados en la tecnología de Servicios Web. Estos Servicios Web, no son más que una de las muchas tecnologías distribuidas (como CORBA, EJB, etc.) con la que implementar aplicaciones cliente/servidor. Sin embargo, los Servicios Web van más allá de la mera invocación remota de procedimientos. Dos de

[4] GGF es una comunidad de usuarios, desarrolladores y proveedores cuyo objetivo principal es el de crear estándares relacionados con la computación Grid. Han desarrollado algunos estándares importantes como OGSA y OGSI, descritos en este documento.

las características más importantes de los Servicios Web son la clara separación entre interfaz (qué hace) e implementación (cómo lo hace), y la independencia del lenguaje de programación y la plataforma.

Los Servicios Web utilizan estándares abiertos de XML y el protocolo HTTP.

- **SOAP (Simple Object Access Protocol):** es un protocolo que define cómo dos objetos, en diferentes procesos, pueden comunicarse por medio del intercambio de datos en formato XML.
- **WSDL (Web Service Description Languaje):** es el lenguaje de descripción de la interfaz de los Servicios Web.
- **UDDI (Universal Description, Discovery, and Integration):** es un estándar que permite el registro y descubrimiento de los Servicios Web.
- **HTTP (HiperText Transfer Protocol):** es el protocolo de transporte que utilizan (entre otros) los Servicios Web.

Los Servicios Grid surgieron para superar las limitaciones que impedían a los Servicios Web participar en la computación Grid. Esencialmente, los Servicios Grid, son Servicios Web con algunas ampliaciones. Las ampliaciones más importantes son [9]:

- **Soporte para extensión de interfaces:** permiten definir un interfaz (portType) en función de otro. Existen una gran cantidad de portTypes estándar que pueden ser extendidos.
- **Tienen estado:** los servicios conservan el estado tras cada invocación y posterior ejecución.
- **Son transientes y persistentes:** esto quiere decir que no están ligados a la vida del servidor de Servicios Web, sino que pueden crearse y destruirse cuando convenga.
- **Data Service:** pueden asociar datos estructurados al servicio. Estos datos son básicamente información del estado (resultados intermedios, información de ejecución, etc.) y metadatos (configuración del servicio, coste, carga actual, etc.). Esto facilita su descubrimiento, inspección, monitorización y gestión.
- **Notificaciones:** un servicio puede convertirse en Observer de otro Observable, si se suscribe a su Data Service. El Observer recibirá notificaciones de los cambios que se produzcan en el Observable.
- **Agrupación de servicios:** pueden elaborarse interfaces para crear servicios que sean un único punto de entrada para un grupo de servicios. Además permite añadir o eliminar servicios del grupo, y sirve como base para implemetar registros o directorios de servicios.

2.2.3. Servicios Grid semánticos

La Grid Semántica es una iniciativa reciente que trata de exponer sistemáticamente la información semántica asociada a los recursos Grid, para poder construir Servicios Grid más “inteligentes” [12]. En los últimos años, varios proyectos han perseguido esta visión, y hay ya, con éxito, aplicaciones de vanguardia que combinan los puntos fuertes del Grid y las tecnologías semánticas. La idea es construir descripciones semánticas estructuradas, con una identidad y comportamiento asociado. Esto permitirá definir los mecanismos necesarios para su creación y gestión, y protocolos para su tratamiento, intercambio y personalización. Tanto los lenguajes que se utilizarán para codificar las descripciones semánticas como la estructura y el contenido de las descripciones mismas pueden variar según la aplicación.

El Grid Semántico ha estado en una fase de exploración y experimentación en lugar de seguir una investigación y diseño sistemáticos de la arquitectura. Así, el Grid Semántico actualmente carece de una arquitectura de referencia, o de un enfoque sistemático, para el diseño de los componentes Grid Semánticos y sus aplicaciones. Además, se carece de mecanismos que permitan la adopción, uso y despliegue de las técnicas y tecnologías de Servicios Web y Servicios Web Semánticos en la Grid Semántica.

Sin embargo, actualmente existe, como parte del proyecto Ontogrid[5], una propuesta de arquitectura de referencia llamada Semantic-OGSA (S-OGSA) [12], basada en OGSA y que está en consonancia con el concepto de SOKU[6] (Service-Oriented Knowledge Utilities), que se anuncia como la vía de investigación futura para el Grid Semántico.

A la hora de describir un Servicio Grid semánticamente, se pueden seguir dos caminos distintos:

- Describir el Servicio Grid por medio de una ontología de Servicios Grid: Si se elige esta opción, es posible describir el Servicio Grid de forma completa, y explotar al máximo sus capacidades y mejoras respecto a los Servicios Web. Sin embargo, el campo de los Servicios Grid Semánticos es, en cierto modo, poco maduro, y no existe un acuerdo en la manera de describir los Servicios Grid que pueda permitir la aparición de un estándar estable sobre el que poder trabajar. Además, la escasez de aplicaciones de apoyo para la descripción semántica (por ejemplo, aplicaciones de generación automática de información semántica a partir de la interfaz) complican aún más la tarea. No obstante, existen algunas aproximaciones y proyectos de elaboración de ontologías para los Servicios Grid, como por ejemplo las desarrolladas

[5] `http://www.ontogrid.net`

[6] *“El enfoque SOKU representa una vía flexible, potente y económica para la construcción, funcionamiento y evolución de las aplicaciones para las empresas, la ciencia y la sociedad”* [13]. Técnicamente es la evolución natural de la tecnología Grid y de Servicios Web, y se basa fundamentalmente en añadir información semántica a los servicios.

por los proyectos de Ontogrid, UniGridS[7] o NextGrid[8], entre otros, aunque la mayoría son extensiones a los métodos de descripción de los Servicios Web.

- Describir el Servicio Grid como un Servicio Web: De esta forma no se consigue una descripción tan exhaustiva del Servicio Grid, pero la cantidad de información y de herramientas de descripción existentes hacen que esta opción sea muy interesante. Además existen varias aproximaciones para la descripción de Servicios Web ampliamente utilizados y asentados. Las más extendidas son Web Ontology Language for Services (OWL-S) [43] y Web Services Modeling Ontology (WSMO) [55].

OWL-S

OWL-S es una ontología de descripción de Servicios Web basada en DAML-S y expresada en OWL. OWL-S permite describir las propiedades y capacidades de los servicios, con el objetivo de facilitar el descubrimiento e invocación automática de los mismos, así como su composición.

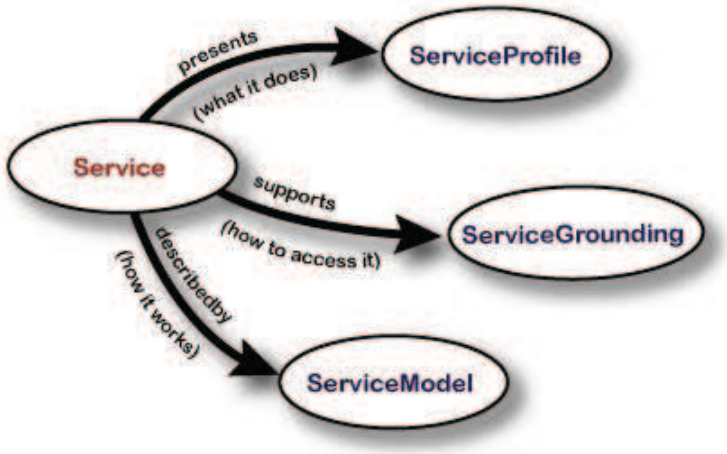

Figura 12: Elementos de OWL-S [18]

Según OWL-S existen tres elementos necesarios para describir un Servicio Web [18] (ver Figura 12):

- **Service Profile:** representa lo que proporciona el Servicio Web a los clientes, sus características y funcionalidades. Además, especifica los requisitos que el consumidor debe satisfacer para utilizarlo. Esta información es la utilizada por los consumidores de servicios durante la fase de descubrimiento.

[7] http://www.unigrids.org
[8] http://www.nextgrid.org

- **Service Model:** describe la interacción de un consumidor con el Servicio Web en forma de proceso (concepto de workflow[9]). Define los procesos, entradas/salidas y precondiciones/efectos.

- **Service Grounding:** especifica los detalles para acceder al servicio e invocarlo (protocolo de comunicación, formato de los mensajes, etc.).

WSMO

WSMO es otra ontología para la descripción de Servicios Web. Está enmarcada dentro del proyecto DIP[10] (Data, Information, and Process Integration with Semantic Web Services), coordinado por el instituto europeo de investigación DERI (Digital Enterprise Research Institute). WSMO se basa en el modelo conceptual propuesto en el Web Service Modeling Framework (WSMF) [15], está expresado en Web Service Modeling Language (WSML) y es totalmente compatible con otras formas de representación como XML, OWL, etc.

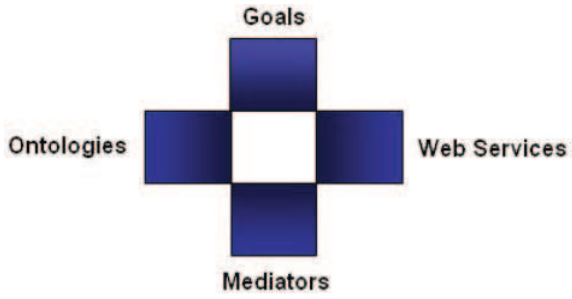

Figura 13: Elementos de WSMO [18]

Según WSMO, existen cuatro elementos fundamentales que describen un Servicio Web [18] (ver Figura 13):

- **Ontologías:** son un conjunto de ontologías de dominio utilizadas para llevar a cabo la descripción del servicio. Puede utilizarse cualquier lenguaje de definición de ontologías, pero se recomienda el uso de WSML.

- **Servicios Web:** es la descripción del Servicio Web en sí. En ella se describen las "Capabilities" (precondiciones, poscondiciones, requisitos y efectos) e "Interfaces" (proceso y grounding del servicio).

- **Objetivos:** los objetivos describen las salidas y efectos esperados tras la invocación de un servicio. Este elemento facilita la expresión de las necesidades de un usuario y la búsqueda de los servicios que las satisfagan (localización automática).

[9] Flujo de trabajo a seguir para la consecución de una tarea o trabajo predeterminado.
[10] `http://dip.semanticweb.org`

- **Mediadores:** la idea de los mediadores busca maximizar la interoperabilidad entre los servicios, mediante el enlace de diferentes componentes de WSMO. Existen cuatro tipos de mediadores:
  - OOMediators: Entre Ontologías. Su objetivo es resolver incompatibilidades.
  - GGMediators: Entre Objetivos. Buscan el refinamiento y compatibilidad de objetivos.
  - WGMediator: Entre Servicios Web y Objetivos. Se encargan de satisfacer las necesidades.
  - WWMediator: Entre Servicios Web. Utilizado en la composición de servicios.

2.2.4. OGSA y S-OGSA

Numerosos investigadores han procurado la obtención de una definición exacta del concepto de Grid. El punto de acuerdo se produce en estos principios básicos [12]:

1. El Grid se centra en compartir recursos distribuidos de una manera controlada y justa construyendo un pool virtual. Los usuarios del pool tienen muy poco o nada de conocimiento a priori sobre el estado, el tipo y las características reales (es decir, de la comprobación) de los recursos. Además, esta reunión virtual de recursos debe permitir que los usuarios utilicen temporalmente los recursos, a los cuales, puede que no tengan acceso directamente de otra manera.
2. Para alcanzar (1), es necesario abstraer las dos entidades principales del entorno: usuarios y recursos, en el pool.

OGSA, teniendo en cuenta los puntos anteriores, describe una arquitectura para entornos de computación Grid orientada a servicios. Además, define un sistema base de capacidades y comportamientos para el Grid (Figura 14). Esto es: servicios de infraestructura, servicios de datos, servicios de gestión de recursos, servicios de gestión de ejecución, servicios de seguridad, servicios de autogestión, y servicios de información.

OGSA provee la arquitectura de tres niveles que se puede apreciar en la Figura 15. En el primer nivel (Fabric), se encuentran virtualizados los distintos tipos de recursos mediante Servicios Web. El nivel intermedio (Middleware) está compuesto por las principales categorías de servicios, también conocido como “grid middleware”. El middleware no está organizado en una arquitectura de capas, cada uno de los servicios se relacionan entre sí. El nivel superior (Application) es donde residen las aplicaciones que hacen uso del “grid middleware” para llevar a cabo sus actividades.

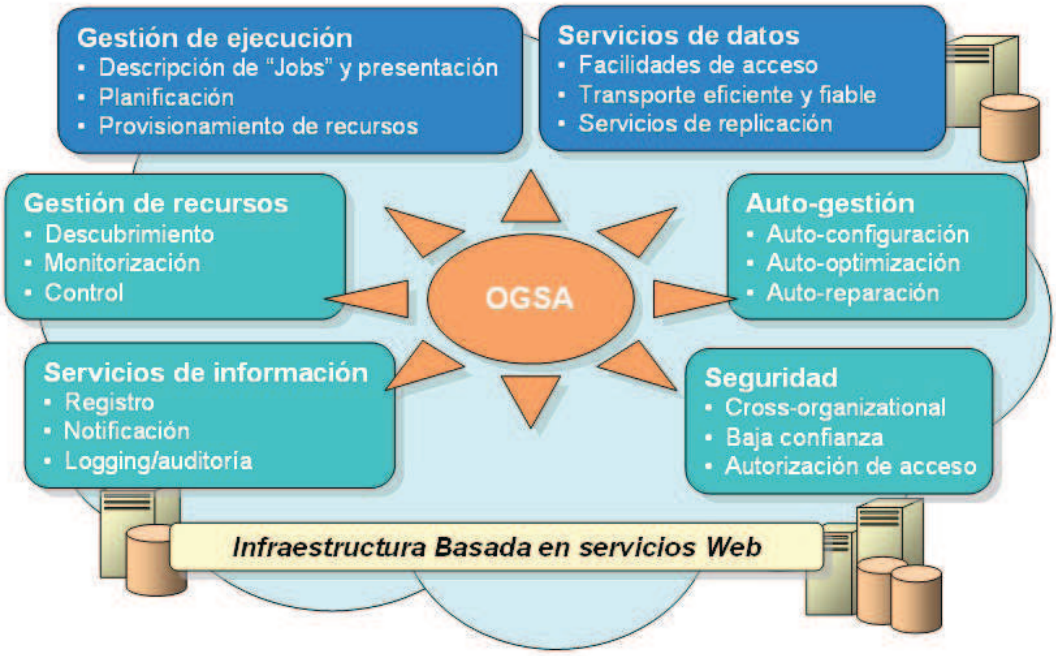

Figura 14: Capacidades de OGSA [31]

Extendiendo el modelo OGSA mediante la definición de un pequeño mecanismo que permita el tratamiento explícito de la semántica obtenemos el modelo S-OGSA. S-OGSA se rige por seis principios fundamentales de diseño [12]:

1. **Parsimonia de la arquitectura:** El framework debe ser tan ligero como sea necesario y debe minimizar el impacto sobre el resto de la arquitectura. Por otra parte, no debe imponer un vocabulario ni una estructura para las descripciones semánticas.

2. **Extensibilidad del framework:** Construir una definición extensible y personalizable, en lugar de una completa y genérica.

3. **Uniformidad de los mecanismos:** S-OGSA es totalmente compatible con OGSA. Los Servicios Grid semánticos son Servicios Grid, y los metadatos y el conocimiento en general son tomados como recursos del Grid.

4. **Diversidad de las capacidades semánticas:** Un diverso y dinámico ecosistema de Servicios Grid deben coexistir en un momento dado. Existen tres tipos de entidades Grid:

   - *Ignorantes* de la semántica explícita asociada a otras entidades.
   - *Conscientes* de que otras entidades están asociadas a una semántica explícita, pero *incapaces* de procesarla.

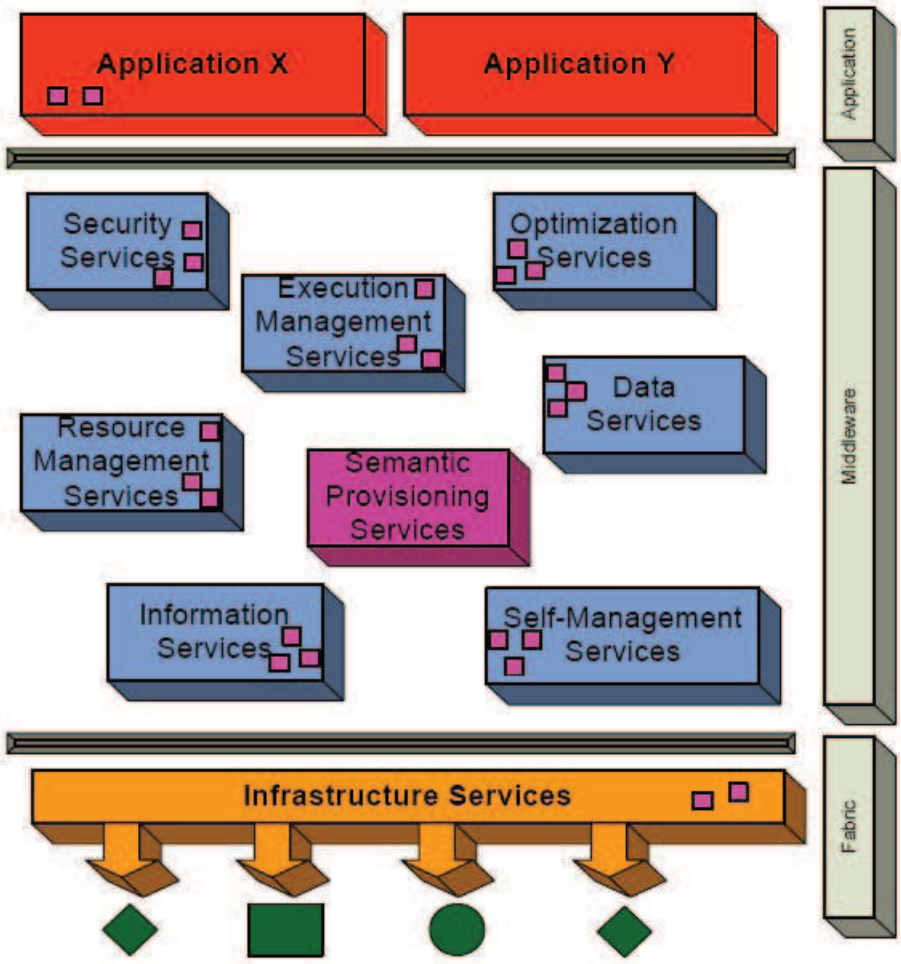

Figura 15: Arquitectura OGSA (azul) y S-OGSA (rosa) [12]

- *Conscientes* de que otras entidades están asociadas a una semántica explícita, y *capaces* de procesarla.

5. **Heterogeneidad de las representaciones semánticas:** Cada propiedad de los recursos puede tener diferentes descripciones semánticas, y cada una de dichas propiedades puede ser descrita (o no) en cualquiera de las diferentes formas de representación (texto, lógica, ontología, regla, etc.).

6. **Claridad de los servicios:** Los servicios deberían tener un sistema sencillo de migración que les permita convertirse en conocimiento.

S-OGSA engloba tres aspectos principales: el modelo (los elementos que lo componen y sus interrelaciones), las capacidades (los servicios necesarios para hacer frente a tales componentes) y los mecanismos (los elementos que permitan la ejecución, a la hora de desplegar la arquitectura en la aplicación basada en tecnología Grid).

- **Modelo:** La definición de los recursos semánticos que se proveen y se consumen entre los servicios, extiende del modelo general del Grid. Los componentes que constituyen el modelo son los siguientes (ver Figura 16):

  - G-Entities: representan cualquier cosa con identidad en el Grid, incluyendo recursos y servicios.
  - K-Entities: son tipos especiales de G-Entities que representan o pueden operar con alguna forma de conocimiento (ontologías, reglas, bases de conocimiento...).
  - S-Bindings: representan una asociación entre una G-Entity y una o varias K-Entities. Esto transforma una entidad en una entidad semántica. Las G-Entities y las K-Entities pueden existir sin necesidad de relacionarse mediante S-Bindings.
  - SG-Entities: Son las entidades objeto de las S-Bindings; son en sí S-Bindings o K-Entities (esta definición incluye todas las entidades que se muestran en la Figura 16).

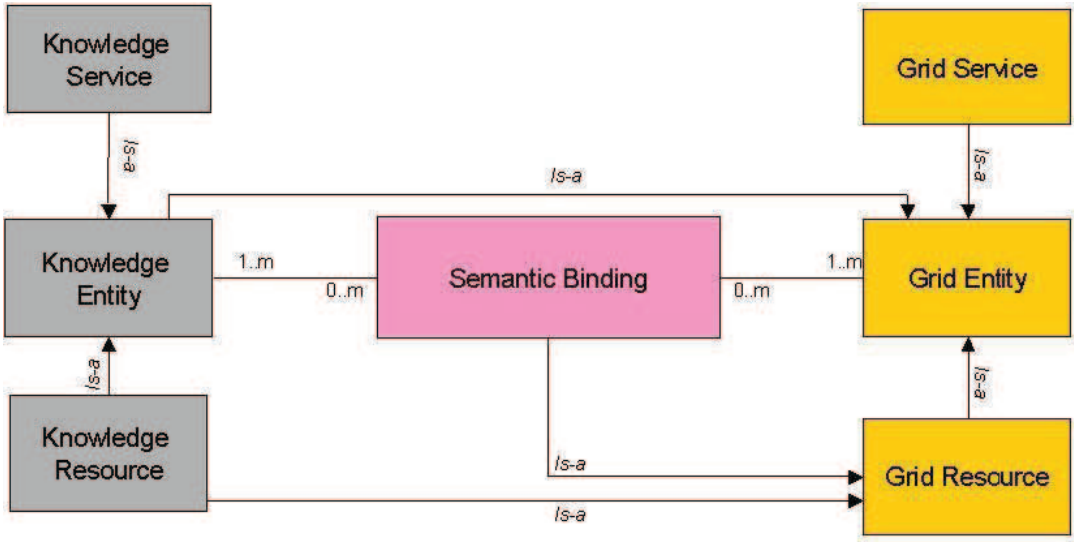

Figura 16: Entidades S-OGSA y sus relaciones

- **Capacidades:** S-OGSA es un conjunto de servicios, con distintos grados de capacidad semántica. Para conseguir esto, se deben extender las capacidades del Grid middleware para incluir Semantic Provisioning Services (Servicios de aprovisionamiento semántico) y Semantically Aware Grid Services o SAGS (Servicios Grid Semánticamente Conscientes). Se pueden ver estas capacidades en la Figura 15 (los SAGS son las pequeñas cajas rosas que se encuentran dentro de las cajas azules). Los Semantic Provisioning Services son los responsables de la provisión y la gestión de la semántica explícita y su asociación con entidades Grid. Y los SAGS son

los Servicios Grid que ofrece OGSA, pero con la diferencia de estar relacionados directamente con una semántica explícita.

- **Mecanismos:** S-OGSA ofrece un conjunto de mecanismos mediante los cuales bosqueja las definiciones conceptuales relativas a la utilización de metadatos en los elementos Grid concretos del modelo. La base de S-OGSA está definida a un nivel que es independiente de cualquier implementación, siguiendo el principio de parsimonia.

2.2.5. OGSI y WSRF

En la actualidad existen dos estándares disponibles, para implementar Servicios Grid, que cumplen los requerimientos de OGSA: OGSI (Open Grid Services Interface) [42] y WSRF (Web Service Resource Framework) [41].

El estándar OGSI propone un modelo de Servicios Web extendidos (Servicios Grid), donde se definen las reglas acerca de cómo puede ser implementado OGSA utilizando dichos servicios. Además, requiere que los Servicios Grid se especifiquen por medio de GWSDL (Grid Web Service Definition Language), que es una extensión de WSDL.

Los principales defectos de OGSI son su extensa y densa especificación, su dificultad para integrarse con las actuales herramientas de Servicios Web, y su excesiva orientación a objetos [9]. Estos defectos, junto con la imposibilidad de OGSI de encontrar una convergencia con WSDL (ver Figura 17), propiciaron la aparición de un nuevo estándar WSRF que, basándose en el antiguo OGSI, a pasado a ser su sustituto.

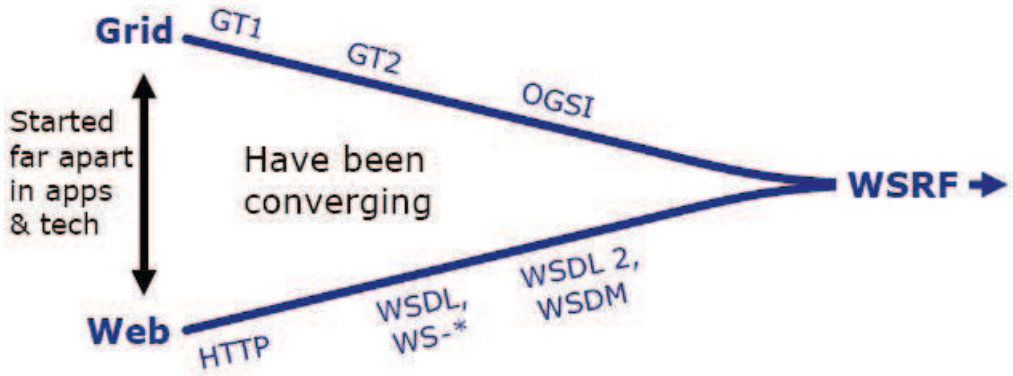

Figura 17: Convergencia de WSRF[11][9]

La solución que propone WSRF a los defectos de OGSI es la siguiente:

1. Dividir en cinco documentos la especificación (más una complementaria). Estos son:

[11] GT son las iniciales de Globus Toolkit, la implementación más utilizada para construir Servicios Grid. La última versión está adaptada para ajustarse a las características de OGSA y WSRF.

- ***WS-Resource:*** define un *WS-Resource* como la composición de un recurso y un Servicio Web a través del cual se puede acceder a las propiedades de dicho recurso.
- ***WS-ResourceProperties:*** describe una interfaz a la que asociar un conjunto de propiedades con un *WS-Resource* para que puedan ser consultadas y manipuladas de una manera estándar.
- ***WS-ResourceLifetime:*** describe una interfaz para gestionar el ciclo de vida de los recursos *WS-Resource*.
- ***WS-BaseFaults:*** define un mecanismo de extensión para una mayor riqueza expresiva en la generación de excepciones incrustadas en el protocolo SOAP.
- ***WS-ServiceGroup:*** describe una interfaz para gestionar conjuntos de recursos *WS-Resource*.
- ***WS-Notification:*** define un mecanismo para notificar información a otros Servicios Web.

2. Ser totalmente compatible con las herramientas de Servicios Web y utilizar WSDL v1.1 puro sin parchear.

3. Separar el servicio (recurso) del estado, que tiene que ser indicado por el cliente en cada interacción con el servicio. La forma de indicarlo es mediante un parámetro en la petición de invocación, codificado en una referencia *WS-Addressing*. *WS-Addressing* provee un mecanismo por el cual se pueden identificar Servicios Web y mensajes de Servicios Web independientemente del protocolo de transporte utilizado. Además define un espacio de nombres que se utiliza para identificar a los Servicios Web.

WSRF no supone un cambio radical, es más bien una evolución de OGSI. Además el estándar OGSA no se ve afectado, ya que los cambios se aplican sólo a la infraestructura y no a la arquitectura.

2.3. AGENTES INTELIGENTES

2.3.1. Definiciones y conceptos

El concepto de “agente inteligente” puede ser definido como [25]:

> *Una entidad software que, basándose en su propio conocimiento, realiza un conjunto de operaciones para satisfacer las necesidades de un usuario o de otro programa, bien por iniciativa propia o porque alguno de éstos se lo requiere.*

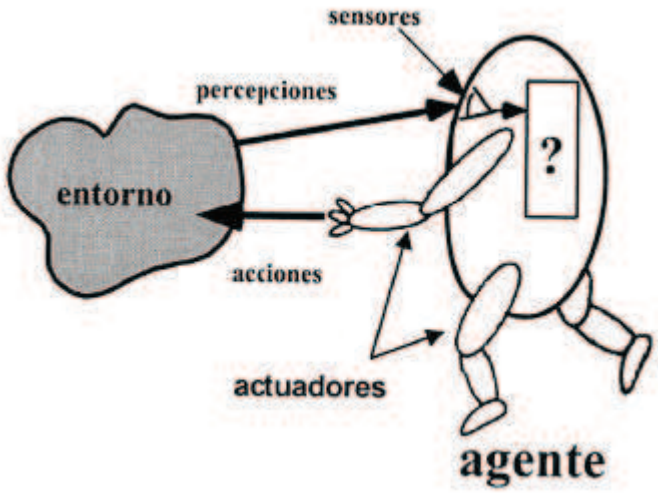

Figura 18: Agente [7]

Sin embargo, no existe un consenso a la hora de dar una definición exacta. Por este motivo, se ha optado por enumerar un conjunto de propiedades que caracterizan a los agentes, aunque un agente no tiene la necesidad de poseerlas todas [10]:

- **Autonomía:** es la capacidad para actuar sin la intervención humana, teniendo el control sobre sus propios actos y su estado interno.
- **Sociabilidad:** un agente debe poder comunicarse, mediante algún lenguaje común, con otros agentes.
- **Reactividad:** los agentes podrán percibir estímulos de su entorno, y reaccionar a ellos de manera oportuna.
- **Iniciativa:** además de responder a estímulos externos, los agentes deben saber dirigir su comportamiento, y emprender las acciones necesarias en cada momento.
- **Movilidad:** es la capacidad para transportarse a sí mismo por la red, para ejecutar una determinada tarea. Se distinguen dos niveles: ejecución remota (el agente es transferido, activado y ejecutado en su totalidad) y migración (durante su ejecución un agente activo puede moverse de nodo a nodo).
- **Veracidad:** se supone que un agente no comunicará información falsa a propósito.
- **Benevolencia:** un agente puede ayudar a otros agentes, si esto no entra en conflicto con sus propios objetivos.
- **Racionalidad:** un agente actúa de forma racional para conseguir cumplir sus objetivos.

- **Inteligencia:** está relacionado con el método utilizado para el desarrollo de la lógica del agente.
- **Aprendizaje:** un agente puede cambiar su comportamiento basándose en sus experiencias previas.
- **Carácter:** se puede simular que un agente tenga personalidad o estados emocionales.
- **Operación asíncrona:** un agente puede ejecutar tareas desacoplado totalmente de los usuarios u otros agentes. Puede ser disparado por la ocurrencia de un evento.

2.3.2. Sistemas multiagentes

A excepción de los casos más triviales, los sistemas suelen contener un número determinado de subsistemas que interactúan entre sí, de manera efectiva, para llevar a cabo con éxito sus tareas. Esto lleva la perspectiva anterior de "construir un agente" hacia "construir una sociedad de agentes". Desde este nuevo punto de vista, se puede definir un Sistema Multiagente (SMA) como [28]:

> *Una red débilmente acoplada de resolvedores de problemas (agentes) que trabajan conjuntamente para resolver problemas que superan las capacidades individuales o conocimiento de cada uno de ellos.*

Los orígenes de los SMA se encuentran en la IAD (Inteligencia Artificial Distribuida). La IAD ha sido dividida históricamente en dos grandes líneas: la Resolución de Problemas Distribuida (RPD) y los Sistemas Multiagente (SMA). Las diferencias fundamentales entre estas líneas se muestran en la Tabla 1

Tabla 1: Diferencias entre RPD y SMA

RPD	■ Fundamental la manera en que un problema puede ser dividido y resuelto. ■ Agentes creados por el mismo diseñador. ■ Agentes provistos con un protocolo de interacción y una estrategia. ■ Análisis centrado en determinar los resultados sociales que se obtendrán dado el protocolo y asumiendo que los agentes usan las estrategias impuestas.
SMA	■ Se centra en el comportamiento global. ■ Agentes preexistentes construidos por distintos diseñadores. ■ Agentes provistos con un protocolo de interacción, pero cada uno elige su propia estrategia. ■ Desafío consiste en definir mecanismos de interacción adecuados de manera que el SMA exhiba un comportamiento social.

En definitiva, un SMA está formado por un conjunto de agentes con habilidades propias, conocimiento limitado (del entorno, de la misión, etc.) y cierta especialización (rol) en determinadas tareas. El conjunto de agentes tiene una misión común, que puede descomponerse en tareas independientes, que pueden ejecutarse en paralelo. Un SMA debe ser capaz de asignar a cada uno de los agentes, una o varias tareas concretas para llevar a cabo, de forma distribuida, la misión común.

3. ANÁLISIS DE OBJETIVOS Y METODOLOGÍAS

3.1. ÁMBITO DEL PROYECTO Y OBJETIVOS

3.1.1. Ámbito del proyecto

Este proyecto está enmarcado dentro del proyecto SEMMAS[12]. El proyecto SEMMAS surgió con la idea de

> *conseguir desarrollar un marco de trabajo donde agentes y servicios interactúen de forma cooperativa aprovechando al máximo las posibilidades que estas tecnologías ofrecen y sin necesitar para esto ningún cambio sustancial dentro de las propuestas y estándares desarrollados hasta el momento en el ámbito de las mismas.* [21]

Para conseguir este objetivo, se utilizan las ontologías por medio de las cuales los agentes se comunican entre sí y con otros agentes humanos, al tiempo que son capaces de manipular los servicios disponibles en el entorno de una forma autónoma.

Desde antes de la aparición de Internet se comenzó a plantear la idea de interconectar sistemas software, con el objetivo de ofrecer funcionalidades o servicios de forma remota. Pero fue a partir del nacimiento de la WWW (World Wide Web) cuando se dio realmente el paso definitivo hacía la consecución de este objetivo, pasando paulatinamente de la idea de Web como repositorio de datos, a la idea de Web como fuente de servicios globalmente accesible.

La Web Semántica surgió más tarde, fruto de la necesidad de manejar la gigantesca masa de información que contiene la Web, y que crece exponencialmente con el paso del tiempo. El mundo de los Servicios Web experimenta una situación similar, el número de Servicios Web disponibles es cada vez mayor y la tarea de encontrar el que mejor se adapte a una necesidad concreta se hace inviable para el ser humano. Aquí surgen los Servicios Web Semánticos. El problema en ambos casos es que no existe información útil que las máquinas sean capaces de procesar. La solución pasa por incluir información adicional, expresada formalmente, que las entidades software puedan comprender. Una vez que dicha información ha sido descrita formalmente, los sistemas informáticos son capaces de acceder a ella de forma automática sin intervención humana. La información semántica adicional se representa por medio de ontologías, y las entidades software que se encargan de acceder a la información son los agentes inteligentes.

Existen diversas propuestas que intentan explicar cómo han de interactuar los Servicios Web y los agentes inteligentes. Pero la mayoría de ellas, al contrario que el proyecto

[12] `http://www.semmas.com`

SEMMAS, implican la modificación radical de los estándares o la limitación de su funcionalidad. El proyecto SEMMAS, por su parte, no pretende encontrar nuevas soluciones para el descubrimiento, composición e invocación de Servicios Web, sino

> *diseñar un entorno en el que la tecnología de agentes inteligentes y Servicios Web Semánticos se combinen y puedan ser estudiadas y utilizadas en el mundo real.* [21]

Y para ello hace uso de las ontologías, como base para la integración de las tecnologías.

Una de las mejoras propuestas como trabajo futuro en el proyecto SEMMAS fue la de introducir una nueva funcionalidad a la plataforma que permitiese, de igual manera que con los Servicios Web, gestionar de una forma eficiente todo el proceso de ejecución de Servicios Grid, desde la fase de descubrimiento hasta la fase de invocación. Este proyecto pretende sentar las bases para la consecución de dicha propuesta. La idea es similar a la planteada por SEMMAS con los Servicios Web. De hecho, dada la flexibilidad de éste y la similitud entre las tecnologías, sería posible integrar en SEMMAS esta nueva plataforma y disponer así, en un mismo framework, de las capacidades necesarias para descubrir, componer e invocar Servicios Web y Servicios Grid Semánticos simultáneamente.

3.1.2. Objetivos y tareas

Como se ha comentado en el apartado anterior, este proyecto parte como una ampliación del proyecto SEMMAS con el objetivo general de obtener un sistema que, a partir del empleo de agentes inteligentes y el uso intensivo de ontologías, mejore el rendimiento en el consumo de los Servicios Grid e incremente la automatización de las tareas de gestión de los mismos, de forma que se explote el dinamismo intrínseco de este tipo de entornos.

Es decir, el sistema, basándose en metodologías de la Web Semántica y en la tecnología de agentes, deberá ser capaz de interpretar un texto en lenguaje natural introducido por un usuario, obtener de él los objetivos y necesidades de dicho usuario y realizar de forma automática todo el proceso de descubrimiento, selección, composición (si fuera necesario) e invocación de los Servicios Grid Semánticos que satisfagan dichos objetivos y necesidades.

Para la consecución del objetivo general se han ido marcando, a lo largo del proyecto, varios subobjetivos concretos. Estos son:

1. Evaluación de las diferentes metodologías de la Web Semántica y de las herramientas que proporciona.

2. Evaluación de las distintas alternativas de arquitecturas de Servicios Grid y Servicios Grid Semánticos, así como de sus herramientas.

3. Evaluación de las similitudes y diferencias existentes entre las tecnologías de Servicios Web y Servicios Grid, y entre los Servicios Web Semánticos y los Servicios Grid Semánticos.

4. Desarrollo de una plataforma basada en la aplicación de tecnologías de la Web Semántica, que integre agentes inteligentes y Servicios Grid Semánticos para la automatización del descubrimiento, selección, composición e invocación de Servicios Grid Semánticos.

5. Generación de ontologías a partir de las peticiones que realizan los usuarios en lenguaje natural.

6. Seleccionar un entorno en el que la sea factible aplicar la plataforma.

Todos los subobjetivos que se indican han sido satisfechos mediante la realización de las siguientes tareas:

1. Estudio del arte de las tecnologías implicadas: Servicios Grid Semánticos, ontologías y agentes.

2. Análisis y evaluación de las herramientas y estándares disponibles relacionados con las tecnologías implicadas; y posterior elección de las más apropiadas.

3. Estudio comparativo de las tecnologías de Servicios Web vs. Servicios Grid y Servicios Web Semánticos vs. Servicios Grid Semánticos.

4. Estudio exhaustivo de la herramienta Globus Toolkit y de los servicios que ofrece. Así como del servicio SemanticBindingService incluido en el proyecto Ontogrid.

5. Estudio y evaluación de las distintas ontologías existentes para la representación del conocimiento asociado a los Servicios Grid Semánticos.

6. Estudio y evaluación de las distintas alternativas de implementación de Servicios Grid y posterior implementación de un conjunto de ejemplos.

7. Análisis de la plataforma multiagente diseñada en el proyecto SEMMAS: identificación de agentes, roles y comportamientos que deben ser añadidos y modificados para ampliar la funcionalidad.

8. Implementación necesaria para integrar la nueva funcionalidad en la plataforma.

9. Realización de pruebas exhaustivas e identificación de posibles errores.

3.2. METODOLOGÍA, HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

3.2.1. Diseño de la aplicación

Dada la similitud existente entre los conceptos de agentes y objetos, podría pensarse en desarrollar sistemas basados en agentes utilizando metodologías orientadas a objetos. Sin embargo, dichas metodologías no contemplan todas las características de estos sistemas. Para solucionar esto, surge un nuevo paradigma de programación llamado POA (Programación Orientada a Agentes) en el que son agentes, en lugar de objetos, los que constituyen el sistema. Hay tres diferencias principales entre objetos y agentes por las que se debe adoptar este nuevo paradigma [48]:

- Los agentes tienen un comportamiento autónomo y deciden en cada momento, según su estado interno, si realizan o no las acciones que le han sido solicitadas.
- Los agentes pueden mostrar un comportamiento flexible (reactivo, proactivo o social), mientras que los objetos no.
- Un SMA es multi-hilo al tener al menos un hilo de control.

Estas diferencias provocan que la tarea de modelar un sistema basado en agentes sea muy complicada. Como consecuencia de esto, surgió la Ingeniería del Software Orientada a Agentes (Agent-Oriented Software Engineering, AOSE), y con ella, un amplio conjunto de metodologías y herramientas de desarrollo y mantenimiento de aplicaciones basadas en agentes.

En general, las metodologías utilizadas para el desarrollo de sistemas basados en agentes están basadas en metodologías tradicionales que han sido ampliadas para soportar los requerimientos específicos del nuevo paradigma POA. Las más conocidas de dichas metodologías son: MAS-CommonKADS, ZEUS, GAIA, MaSE, MOBMAS e INGENIAS. Los siguientes apartados describen brevemente sus características fundamentales.

MAS-CommonKADS [26]

Esta metodología está basada en la metodología CommonKADS e incorpora técnicas e ideas propias de la orientación a objetos como eje fundamental, para modelar sistemas basados en agentes. MAS-CommonKADS está formado por siete modelos: el *modelo de agente*, que describe las características de cada agente; el *modelo de tarea*, que describe las tareas que los agentes pueden realizar; el *modelo de la experiencia*, que describe el conocimiento que necesitan los agentes para alcanzar sus objetivos; el *modelo de coordinación*, que describe las relaciones entre los distintos agentes; el *modelo de comunicación*, que describe las relaciones entre los agentes humanos y los agentes software; el *modelo*

de organización, que describe la organización humana en la que el sistema basado en agentes se introduce, y la organización de los agentes y su relación con el entorno; y el *modelo de diseño*, que describe la arquitectura y el diseño del sistema basado en agentes como paso previo a su implementación.

ZEUS [22]

ZEUS propone que el desarrollo del sistema se componga de cuatro etapas: *análisis*, en la que se modelan los papeles y comportamientos propios de los agentes; *diseño*, que consiste en encontrar soluciones que cumplan con los requisitos de la fase de análisis; *desarrollo*, en la que se crean y configuran los agentes; y *despliegue*, en la que se pone en funcionamiento el sistema.

Además, ZEUS cuenta con un conjunto de herramientas de código abierto que abarca todas las etapas mencionadas anteriormente.

GAIA [22]

El objetivo de la metodología GAIA es obtener un sistema que maximice alguna medida de calidad global, y para ello pretende ayudar al analista a ir de forma sistemática, desde los requisitos iniciales hasta el diseño. El proceso está formado por dos etapas: el *análisis de alto nivel*, en el que se modelan los roles e interacciones de los agentes del sistema; y el *diseño de alto nivel*, en el que se generan los modelos de agentes, de servicios y de conocidos.

GAIA únicamente especifica el modo en que deben colaborar el conjunto de agentes para alcanzar los objetivos, siempre desde un alto nivel de abstracción. Esto provoca que la especificación sea independiente de la implementación, pero el esfuerzo de pasar de una a otra puede resultar elevado. Además tiene el inconveniente añadido de ser la metodología que menos soporte de herramientas tiene.

MaSE [22]

Esta metodología asume que un agente es una especialización de un objeto capaz de coordinarse con otros agentes para cumplir con determinados objetivos (individuales o globales).

MaSE dispone de la herramienta AgentTool, que habilita la ejecución de la mayoría de los pasos que componen el proceso de desarrollo que especifica MaSE. Dicho proceso consta de dos fases: la *fase de análisis*, que consiste en la captura de los objetivos, captura de los casos de uso y refinamiento de los roles; y la *fase de diseño*, que consta de la creación de clases de agentes, la construcción de conversaciones, el ensamblaje de clases

de agentes y la realización del diseño del sistema. Además, AgentTool permite generar código (independiente del lenguaje) automáticamente a partir de la especificación final del sistema (diagramas finales).

MOBMAS [53]

MOBMAS cuenta con la ventaja de que incluye, de forma explícita, soporte para el uso de ontologías durante el diseño de sistemas basados en agentes. El desarrollo que propone MOBMAS está formado por cinco actividades: *análisis*, donde se identifican los roles y tareas a partir de los requerimientos del sistema y de la ontología del dominio; *diseño de organización*, donde se especifica la estructura organizacional del sistema y se identifica el conjunto de agentes y recursos que compondrán el sistema; *diseño interno de los agentes*, donde se especifican los objetivos, eventos, planes y reglas reactivas de cada agente; *diseño de interacción de los agentes*, donde se definen las estrategias de interacción e intercambio de datos entre agentes más adecuadas; y *diseño de la arquitectura*, donde se solucionan asuntos relacionados con la arquitectura, como la identificación de los requisitos de interfaz, la selección de una arquitectura adecuada, la identificación de las propiedades necesarias para la instanciación y configuración de los agentes, etc.

INGENIAS [27]

La metodología elegida para el desarrollo del sistema basado en agentes elaborado como parte de este proyecto, ha sido INGENIAS. INGENIAS concibe el SMA como la representación computacional de un conjunto de modelos. Cada uno de estos modelos muestra una visión parcial del SMA: los agentes que lo componen, las interacciones que existen entre ellos, cómo se organizan para proporcionar la funcionalidad del sistema, qué información es relevante en el dominio y cómo es el entorno en el que se ubica el sistema a desarrollar.

INGENIAS proporciona cinco meta-modelos que especifican cómo deben construirse los modelos. Estos meta-modelos son una evolución de las ideas de MESSAGE[13], y han supuesto una mejora, respecto a este, en aspectos tales como el nivel de detalle, cohesión entre meta-modelos o representación del sistema. Los meta-modelos son:

- **Meta-modelo de agente:** describe agentes particulares, sus tareas, objetivos, los roles que desempeñan y los estados mentales en que se encontrarán a lo largo de su vida.

[13] MESSAGE es una metodología de construcción de software basado en agentes inteligentes, que evolucionó hasta dar lugar a INGENIAS (`http://www.eurescom.de/~public-webspace/P900-series/P907`).

- **Meta-modelo de tareas y objetivos:** tiene como propósito recoger las motivaciones del sistema de agentes, definir las relaciones entre objetivos y tareas, indicar las entradas y salidas de cada tarea, y describir cuales son sus efectos, tanto en el entorno como en el estado mental de sus agentes responsables.

- **Meta-modelo de organización:** es el equivalente a la arquitectura del sistema. Describe como se agrupan y coordinan los distintos componentes del sistema, identifica las tareas relevantes para la organización, así como sus objetivos globales, y define restricciones en las interacciones entre los agentes.

- **Meta-modelo de interacción:** describe como se coordinan y comunican los agentes del sistema. Esto incluye a los actores implicados, las unidades de interacción, los protocolos utilizados y cómo afecta la interacción en el contexto.

- **Meta-modelo de entorno:** En este meta-modelo el propósito no es generar representaciones del mundo en el que se ubica el sistema, sino, desde un punto de vista más pragmático, categorizar el tipo de entidades relevantes en el entorno y restringir la interacción con ellas. Así, el entorno contendrá sólo recursos, aplicaciones y agentes, y se limitará la percepción y actuación de los agentes.

Al instanciar estos meta-modelos obtenemos diagramas y modelos similares a los utilizados en UML (Unified Modeling Language), pero con las particularidades propias de los sistemas basados en agentes. Hacer esta instanciación no es una tarea trivial, dado el elevado número de entidades, relaciones, y dependencias entre los distintos modelos. Por este motivo, INGENIAS define un conjunto de actividades, organizadas según el paradigma de la ingeniería del software, cuya ejecución da como resultado el conjunto de modelos que definen el sistema. En la Tabla 2 se muestran las actividades que tienen lugar en las fases de inicio, elaboración y construcción.

Por último, cabe destacar que la metodología INGENIAS incluye una herramienta de modelado gráfico (INGENIAS Development Kit, IDK), con la que es posible construir cada uno de los modelos que se han descrito anteriormente. Además posee un generador de código automático que devuelve código para la plataforma multi-agente JADE, que será la elegida para la construcción del sistema. La plataforma JADE se describirá, junto con otras plataformas multi-agentes, en capítulos posteriores.

Tabla 2: Actividades de INGENIAS [27]

		FASES		
		Inicio	**Elaboración**	**Construcción**
FLUJOS DE TRABAJO FUNDAMENTALES	**Análisis**	- Generar casos de uso e identificar realizaciones de los casos de uso con modelos de interacciones. - Esbozar la arquitectura con un modelo de organización. - Generar modelos del entorno para trasladar la captura de requisitos a los modelos.	- Refinar casos de uso. - Generar modelos de agente para detallar los elementos de la arquitectura. - Continuar con los modelos de organización identificando flujos de trabajo y tareas. - Modelos de tareas y objetivos para generar restricciones de control (objetivos principales, descomposición de objetivos). - Refinar modelo de entorno para incluir nuevos elementos.	- Estudiar resto de casos de uso.
	Diseño	- Generar un prototipo con herramientas de prototipado rápido, como ZEUS o Agent Tool.	- Centrar el modelo de organización en el desarrollo de flujos de trabajo. - Llevar las restricciones identificadas a modelos de tareas y objetivos para dar detalles acerca de las necesidades y resultados de las tareas y su relación con los objetivos del sistema. - Expresar la ejecución de tareas dentro de modelos de interacción. - Generar modelos de agente para detallar patrones de estado mental.	- Generar nuevos modelos de agente o refinar los existentes. - Depurar la organización centrando el desarrollo en las relaciones sociales.

3.2.2. Diseño de los Servicios Grid

Para el diseño de los Servicios Grid semánticos, y su posterior registro, descubrimiento y acceso, se han utilizado las siguientes herramientas y tecnologías.

Globus Toolkit

Globus Toolkit[14] (GT) es un entorno de código abierto que representa la implementación más ampliamente utilizada a la hora de construir Servicios Grid y que, posiblemente, se convierta en un estándar de la computación Grid en el futuro. Proporciona los elementos necesarios de seguridad, gestión de recursos, indexación de recursos, gestión de datos y funciones de comunicación. Está formado por una serie de componentes [19]:

- **Gestión de Recursos (Globus Resource Allocation Manager, GRAM):** su función es la de procesar las peticiones de recursos para la ejecución de aplicaciones remotas, manejar los trabajos (jobs) activos y proporcionar información de las capacidades

[14] `http://www.globus.org`

y la disponibilidad de los recursos. GT proporciona un acceso transparente, unificado y seguro, a los distintos gestores de recursos.

- **Servicios de Información (Metacomputing Directory Service, MDS):** usa el protocolo LDAP para la consulta de la información referente a los sistemas del Grid. MDS proporciona los mecanismos básicos para el descubrimiento y monitorización de los recursos Grid. Utiliza los llamados *proveedores de información* (Grid Resource Information Service, GRIS) para dar acceso a la información a través de *gateways* a otras fuentes de información. La información suministrada por los GRIS se agrupa en el Grid Index Information Service (GIIS), que ofrece una visión estructurada, conjunta y coherente de la información de los recursos Grid.

- **Gestión de Datos (GridFTP):** GridFTP es un protocolo de transferencia de ficheros, seguro y de alto rendimiento, basado en el protocolo FTP, y optimizado para redes amplias con grandes anchos de banda.

- **Seguridad (Grid Security Infrastructure, GSI):** está basada en criptografía de clave pública, y provee privacidad, integridad, autenticación y autorización a los Servicios Grid. Utiliza certificados del estándar X.509 y agrupa a los usuarios en Organizaciones Virtuales (VO) según sus necesidades comunes. De este modo, un usuario que quiera trabajar en una infraestructura Grid debe ser miembro de, al menos, una VO para poder utilizar sus recursos.

SemanticBindingService y *MetadataQueryService*

Un Semantic Binding es una asociación entre una entidad Grid y una o varias entidades del conocimiento (ver sección 2.2.4).

Las principales propiedades de Semantic Binding son el conjunto de recursos a los que se refiere (es decir, los recursos para los que contiene metadatos), el conjunto de entidades del conocimiento, y los metadatos que contienen dichas entidades del conocimiento. Además de estas propiedades, otras como el estado del Semantic Binding, la fecha de creación, la fecha de la última modificación, etc., son almacenados, y se utilizan para otras tareas como la gestión del ciclo de vida y los mecanismos de notificación y autorización.

Aparte de las propiedades básicas que describen los Semantic Binding y contienen su información relevante, existe un conjunto de funcionalidades básicas que son prestadas por un grupo de servicios asociados a los Semantic Binding: el *SemanticBindingService* (servicio de almacenamiento y consulta de metadatos), su correspondiente factoría y el *MetadataQueryService* (proporciona un servicio de consultas globales al conjunto de Semantic Bindings). En la Figura 19, se ofrece una visión global de la funcionalidad de estos servicios.

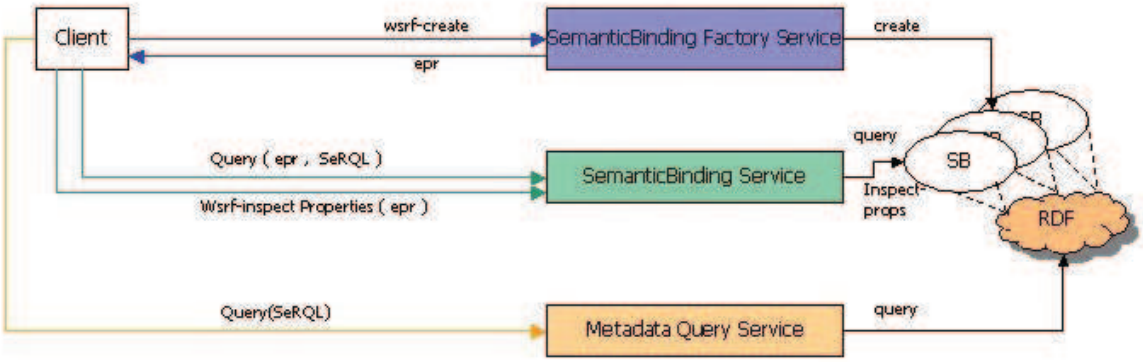

Figura 19: Operaciones con Semantic Bindings

A continuación, se detallan las operaciones más importantes que ofrecen los servicios mencionados anteriormente:

- ***Create***: crea un Semantic Binding a partir de una entidad Grid, una entidad del conocimiento y sus metadatos.
- ***Update Resource and Knowledge Entity References***: permite gestionar las referencias a las entidades Grid y del conocimiento de los Semantic Binding.
- ***Update Semantic Binding Content***: actualiza los metadatos contenidos en el Semantic Binding.
- ***Destroy***: destruye el contenido del Semantic Binding. Puede ejecutarse en el momento o programar su ejecución para un tiempo determinado.
- ***Archive***: archiva el contenido de un Semantic Binding
- ***Query***: Ejecuta una consulta sobre los metadatos almacenados por el Semantic Binding. Las consultas se enviarán en un lenguaje de consulta que soporte el Semantic Binding, y puede tener en cuenta a las entidades del conocimiento a las que el Semantic Binding se refiere o no.

Estas propiedades y funcionalidades están incluidas en la actual implementación de S-OGSA, disponible gracias al proyecto Ontogrid. Cumplen con las especificaciones WSRF y pueden desplegarse y ejecutarse en el Globus Toolkit 4.

Grid Development Tools

El Grid Development Tools (GDT) es un plug-in de Eclipse, incluido en el proyecto Marburg Ad-hoc Grid Environment[15] (MAGE), utilizado para la construcción semiautomática de

[15] `http://mage.uni-marburg.de`

Servicios Grid en Java, por medio de anotaciones. Esta herramienta no se ha utilizado para el desarrollo de la aplicación en sí, sino para la implementación de los Servicios Grid de ejemplo.

3.2.3. Diseño de las ontologías

Para el diseño, construcción y manipulación de las ontologías se han utilizado las siguientes herramientas y tecnologías.

Protègè [45]

Prtotègè es una plataforma open-source que proporciona una serie de herramientas para la construcción de ontologías y bases de conocimiento. Implementa un rico conjunto de estructuras de modelado del conocimiento, y operaciones que le ofrecen soporte para la creación, visualización y manipulación de ontologías en diferentes formatos de representación. Soporta, entre otros lenguajes, DAML, RDF, RDFS y OWL, y además puede ser ampliado mediante plug-in de Java y una API que proporciona. Protègè soporta dos formas de modelado de ontologías:

- El editor Protègè-Frames: permite a los usuarios crear ontologías, de acuerdo con el modelo Open Knowledge Base Conectivity (OKBC).
- El editor Protègè-OWL: permite a los usuarios crear ontologías, de acuerdo con el modelo de la Web Semántica, en el lenguaje de representación OWL.

ODESGS [8]

ODESGS es un entorno de anotación de Servicios Grid preexistentes y de composición semiautomática de nuevos Servicios Grid semánticos complejos. Está basado en el Problem Solving Method (PSM) como paradigma de representación de los servicios, y su principal componente es el SGSDesigner, que es su interfaz gráfica de usuario. Es una aplicación totalmente independiente, aunque opera conforme al estándar S-OGSA. Las principales funcionalidades y características de ODESGS son las siguientes:

- El conocimiento está representado de forma gráfica, con lo que el usuario no necesita conocer los detalles concretos del lenguaje utilizado para la representación del conocimiento.
- Es capaz de generar el marcado semántico de la información una vez que el servicio se ha descrito en el nivel de conocimientos.

- Proporciona anotación semiautomática de servicios basados en el WSRF.
- Es capaz de manejar múltiples y heterogéneas ontologías, representadas tanto en el lenguaje OWL, como en RDF(S).

WSRF2OWL-S [3]

WSRF2OWL-S[16] es una herramienta que genera automáticamente la descripción y anotaciones OWL-S de Servicios Grid basados en WSRF mediante su descripción WSDL (Web Service Description Language). En su última versión soporta:

- OWL-S 1.1.
- Servicios Grid basados en WSRF.
- Configuración, por medio de un fichero de propiedades, de entradas, salidas, precondiciones y poscondiciones.

JENA

Para manejar las ontologías en Java se ha utilizado JENA[17]. JENA es un framework de código abierto utilizado para la creación de aplicaciones de la Web Semántica. Proporciona un entorno de programación para RDF, RDFS y OWL, SPARQL (SPARQL Protocol and RDF Query Language), e incluye un motor de inferencia basado en reglas. También incluye un API OWL, un API RDF, y un motor de consultas basado en el lenguaje SPARQL.

3.2.4. Lenguaje de programación

El lenguaje de programación elegido para el desarrollo de la aplicación ha sido el lenguaje Java (Sun Microsystems[18]). Sus ventajas frente a otros lenguajes fueron las que impulsaron a tomar esta decisión. Entre sus propiedades más características se destacan las siguientes [38]:

- **Orientado a objetos:** soporta encapsulación, herencia y polimorfismo, que son las tres características propias del paradigma de orientación a objetos. Fomenta la reutilización y extensibilidad del código, y favorece el mantenimiento del software.
- **Distribuido:** ofrece extensas capacidades de conexión TCP/IP. Proporciona librerías de rutinas para acceder e interactuar con distintos protocolos. En realidad, Java no

[16] `http://ups.savba.sk/kwfgrid/wsrf2owls`
[17] `http://jena.sourceforge.net`
[18] `http://www.sun.com/java`

es en sí distribuido, sino que proporciona las herramientas necesarias para que los programas si lo sean.

- **Robusto:** realiza comprobación de tipos (tipado estáticamente), obliga a la declaración explícita de métodos, libera la memoria de forma automática (recolector de basura), implementa arrays con comprobación de límites y proporciona un sistema de excepciones ante errores producidos en tiempo de ejecución, entre otras cosas.

- **Seguro:** elimina características como punteros y castings implícitos presentes en otros lenguajes, para prevenir el acceso ilegal a la memoria. Además, el código Java pasa numerosos tests antes de ejecutarse en una máquina. Estos tests, mediante un verificador de ByteCode[19], comprueban el formato de los fragmentos de código y detectan si dichos fragmentos violan los derechos de acceso de los objetos.

- **Portable:** el compilador Java compila el código a un fichero objeto, de formato independiente de la arquitectura de la máquina en que se ejecutará (ByteCode). De esta forma, cualquier máquina que posea un sistema de ejecución puede ejecutar dicho código objeto sin importar la máquina en que ha sido generado.

- **Multihilo:** permite muchas actividades simultáneas en un programa. De esta forma se mejora el rendimiento y el comportamiento en tiempo real, aunque se ve limitado según las capacidades del sistema operativo subyacente.

- **Dinámico:** los módulos que componen una aplicación no se conectan hasta el tiempo de ejecución. De modo que pueden ser añadidos dinámicamente.

Todas estas ventajas, unidas a la simplicidad del lenguaje, y a la compatibilidad con otras herramientas de desarrollo y complementación del sistema que nos ocupa, hacen que Java parezca la elección más conveniente para satisfacer los objetivos del proyecto.

Como herramienta de desarrollo se ha utilizado Eclipse[20]. Eclipse posee una interfaz simple e intuitiva y ofrece la posibilidad de aumentar su funcionalidad mediante módulos (plug-in).

3.2.5. Plataforma multiagente

Arquitectura FIPA

FIPA[21] se formo en 1996, como una organización dedicada a la producción de especificaciones estándar para la interacción de agentes y sistemas de agentes heterogéneos.

[19] Código intermedio entre el código fuente y el código máquina.
[20] `http://www.eclipse.org`
[21] `http://www.fipa.org`

Más tarde, FIPA, pasó a formar parte de la IEEE Computer Society. Su objetivo principal es el de *"permitir la construcción de sistemas que se integren con su entorno de computación particular, mientras que interoperan con sistemas de agentes que residen en entornos heterogéneos, todo con el mínimo esfuerzo"* [6].

Entre los numerosos estándares que ha generado FIPA, destacan la especificación de una arquitectura abstracta y el lenguaje de comunicación entre agentes FIPA-ACL. La arquitectura abstracta FIPA define cómo dos agentes pueden localizarse y comunicarse entre sí utilizando mecanismos que les permiten registrarse e intercambiar mensajes, y así proporcionar un lugar donde los agentes registren sus descripciones y servicios para que otros agentes puedan localizarlos e interactuar con ellos.

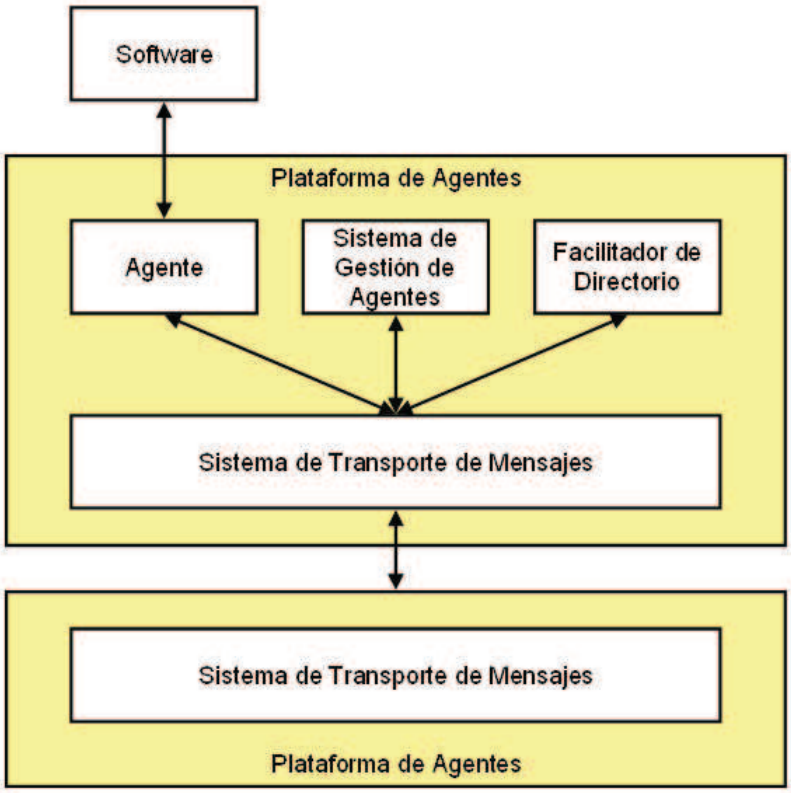

Figura 20: Arquitectura FIPA

Ahora se describen brevemente cada una de las partes que componen la arquitectura (ver Figura 20):

- **Agente:** es un proceso computacional que implementa la funcionalidad autónoma de comunicación para una aplicación concreta. Se identifica mediante su Agent Identifier (AID) de forma única.
- **Sistema de Gestión de Agentes (Agent Management System, AMS):** el AMS ha-

bilita el acceso a la plataforma, por este motivo debe haber un AMS por cada plataforma de agentes. También gestiona el ciclo de vida de los agentes, los recursos locales, y los canales de comunicación y proporciona un servicio de páginas blancas, que permite localizar agentes por su nombre.

- **Facilitador de directorio (Directory Facilitator, DF):** es un componente opcional. Proporciona un servicio de páginas amarillas que permite localizar agentes por sus capacidades y no por su nombre. El DF es el elemento de la plataforma que le permite a un agente descubrir los servicios que ofrecen otros agentes en el entorno que le rodea. Además de tener registrados unos servicios proporcionados por agentes locales, puede tener registrados otros DF, lo que le permite poder extender sus búsquedas a los servicios que tienen registrados esos otros DF.

- **Sistema de transporte de mensajes (Message Transport Service, MTS):** es el responsable del envío de mensajes entre agentes de una plataforma y entre agentes de distintas plataformas, por lo que todo agente tiene que tener acceso a un MTS.

- **Plataforma de agentes (Agent Plataform, AP):** proporciona la infraestructura física en la que los agentes son desplegados. Está formada por la máquina o máquinas, el sistema operativo, el software de gestión de agentes, los componentes de gestión de agentes de FIPA (AMS, DF y MTS) y los agentes.

- **Software:** representa a todas las colecciones de instrucciones que no son parte integral de los agentes, pero que son accesibles a través de los mismos. Sirven, por ejemplo, para añadir nuevos servicios, incorporar nuevos protocolos de comunicación, etc.

Los mensajes intercambiados por los agentes tienen la forma de una tupla clave-valor y están escritos en el lenguaje FIPA-ACL. Cada uno incluye el nombre (AID) del emisor y del receptor, y su contenido está escrito en algún lenguaje de contenidos, tal como KIF o SL, que puede hacer referencia a alguna ontología. Para su transporte, el mensaje es convertido en un payload e introducido en un sobre (envelope). El sobre incluye las descripciones del transporte, que contienen información de cómo se envía el mensaje.

Existen varias implementaciones del modelo de referencia FIPA, como JADE, FIPA-OS o ZEUS, que, aunque cumplen con sus especificaciones, muestran ligeras diferencias entre sí según la interpretación que toman de la arquitectura.

Jade

El framework elegido para la construcción del sistema de agentes ha sido Jade[22] (Java Agent Development Framework). Jade es una plataforma de código abierto, implementada completamente en Java y que se ajusta a los estándares de FIPA expuestos en el apartado anterior. Además es la implementación de desarrollo de aplicaciones basadas en agentes más extendida.

Jade proporciona un entorno de ejecución (plataforma de agentes) conforme a las especificaciones FIPA, que aporta los servicios necesarios para la ejecución de agentes, el registro de los agentes, la comunicación entre agentes e incluso servicios de autentificación. También incluye un conjunto de paquetes que dan soporte a los desarrolladores a la hora de crear agentes, y un conjunto de herramientas gráficas para administrar y monitorizar la ejecución de los agentes activos. Las instancias del entorno de ejecución se denominan contenedores. Cada entorno de ejecución puede albergar a numerosos contenedores, pero en ese caso el entorno deberá disponer de un contenedor principal en el que los demás contenedores han de registrarse.

Un contenedor está compuesto por (ver Figura 21) [40]:

- **AMS (Agent Management System):** proporciona un servicio de páginas blancas y de ciclo de vida de los agentes y supervisa el acceso y uso de la plataforma. Cada agente ha de registrarse en el AMS con un AID válido (único).
- **DF (Directory Facilitator):** proporciona un servicio de páginas amarillas. Por medio de las páginas amarillas, un agente puede encontrar a otros agentes registrados a partir de sus servicios de una forma rápida.
- **ACC (Agent Communication Chanel):** es el software encargado de controlar el intercambio de mensajes, tanto dentro como fuera de la plataforma.

Al arrancar Jade, se crean automáticamente un AMS y un DF en el contenedor principal, y se prepara un ACC.

Los agentes en Jade están asociados a comportamientos (Behaviour), que son las tareas que el agente puede ejecutar. De esta forma, se pueden añadir comportamientos a la cola de comportamientos del agente, y este puede ejecutarlos de forma concurrente, secuencial, interactiva o atómica.

La comunicación se realiza a través de un intercambio asíncrono de mensajes, permitiendo de este modo un modelo de comunicación distribuido y débilmente acoplado. Para que la comunicación sea efectiva, los agentes deben compartir el mismo lenguaje, el mismo vocabulario y los mismos protocolos. Ya que Jade sigue el estándar de FIPA, tanto

[22] `http://jade.tilab.com`

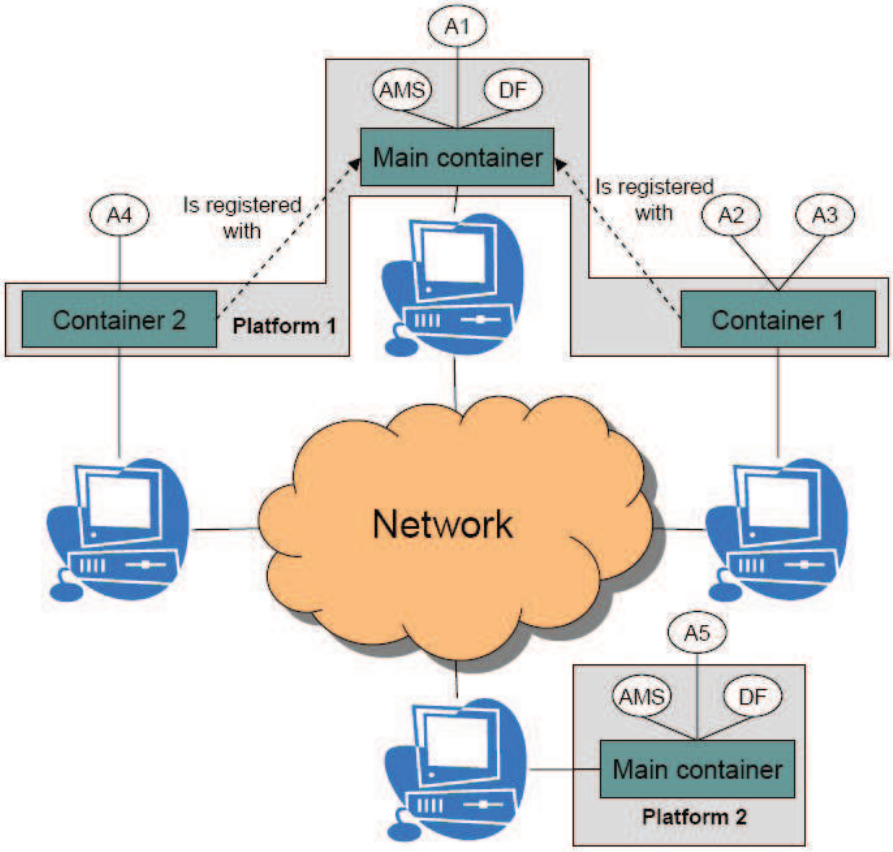

Figura 21: Plataformas en JADE [40]

el lenguaje de comunicación como los protocolos, son comunes a todos los agentes. Sin embargo, es necesario definir un vocabulario para que la comunicación se lleve a cabo con éxito. Para esto, Jade contempla el uso de tres métodos distintos (ver Tabla 3): cadenas de texto, objetos Java serializables y ontologías.

Tabla 3: Contenido de los mensajes en Jade

Cadenas de texto	▪ Conveniente cuando el contenido de los mensajes es atómico. ▪ Inadecuado cuando se trata de conceptos abstractos o datos estructurados.

Objetos serializables de Java	■ Limita a que todos los agentes deben estar implementados en Java. ■ Los mensajes son incomprensibles para los humanos.
Ontologías	■ Definiciones de objetos como extensiones a clases predefinidas de Jade (Ontology, ACLOntology, BaseOntology). ■ Permite comunicar Jade con otras plataformas.

4. DISEÑO Y RESOLUCIÓN DEL TRABAJO

4.1. TRABAJO RELACIONADO

Durante los últimos años se han desarrollado diversas tecnologías que, interactuando con los agentes inteligentes, posibilitan el desarrollo de aplicaciones mucho más potentes de lo que su propia naturaleza les permite. Este proyecto trata acerca de la interacción entre los agentes inteligentes y los Servicios Grid Semánticos, y de cómo hacer posible que cooperen entre sí para facilitar las tareas de descubrimiento, invocación y composición de los mismos.

Existe un gran número de proyectos y aplicaciones que combinan las tecnologías de agentes inteligentes y la tecnología Grid. Generalmente son proyectos que requieren una gran potencia de cómputo y de cierta inteligencia que simplifique el trabajo (por ejemplo sistemas de observación espacial inteligente). Sin embargo, la rama de los Servicios Grid Semánticos es aún bastante inmadura y continúa en proceso de estudio e investigación, aunque poco a poco van apareciendo nuevos resultados positivos.

4.1.1. Los agentes en la Web Semántica

A pesar de la falta de estructura de la información en la Web, hace tiempo que existen agentes que procesan dicha información con el fin de ser utilizada posteriormente por el usuario o, simplemente, para analizar su contenido. Sin embargo, estos agentes están limitados al uso de palabras clave o reconocedores de lenguaje natural para comprender el contenido de la Web. Con la llegada de la Web Semántica es posible dotar de estructura al contenido de la Web y hacerlo entendible para los agentes software por medio de las ontologías.

Ya antes de que la Web Semántica fuera llamada como tal, se advirtió el beneficio que suponía conseguir que los agentes Web interaccionaran con el contenido Web anotado semánticamente. Apareció SHOE [36] (Simple HTML Ontology Extensions), predecesor de los lenguajes ontológicos actuales, del que sus autores destacan los beneficios de describir páginas Web de forma semántica. Más concretamente, se centran en las ventajas que conllevan el uso de la información semántica por parte de agentes software que realicen búsquedas en la Web. Además anuncian que la Web Semántica ofrecerá su máxima funcionalidad cuando los agentes software puedan recoger información de distintas fuentes, puedan procesarla e intercambiar los resultados con otros programas haciendo uso del contenido semántico. Dicho con otras palabras, la llegada de la Web Semántica propició que los agentes Web fueran capaces de analizar el contenido semántico de diversos recursos dispersos en la Web para extraer la información relevante, y de entender y razonar esta información para utilizarla con el fin de satisfacer las necesidades de los usuarios.

Han sido muchas las herramientas aparecidas tras promulgarse esta visión. Sin embargo, surgen numerosos problemas que dificultan la utilización de estas tecnologías en entornos reales. El principal problema, fruto de la subjetividad que se desprende al tratar de definir conceptos reales y la débil estandarización de los lenguajes de ontologías, radica en el hecho de que pueden existir diversos modelos ontológicos que representen el mismo dominio de aplicación. Por ejemplo, existiría ese problema en un escenario en el que diversas entidades software intenten comunicarse entre sí utilizando modelos ontológicos distintos para el mismo dominio. En ese caso, sería necesario establecer un acuerdo entre los participantes. Entre las posibles soluciones a este problema que se han propuesto está el mapeo de ontologías, es decir, intentar encontrar una equivalencia semántica entre las distintas ontologías de los usuarios, concretamente entre sus elementos. Un ejemplo gráfico de mapeo entre ontologías se puede ver en el la Figura 22.

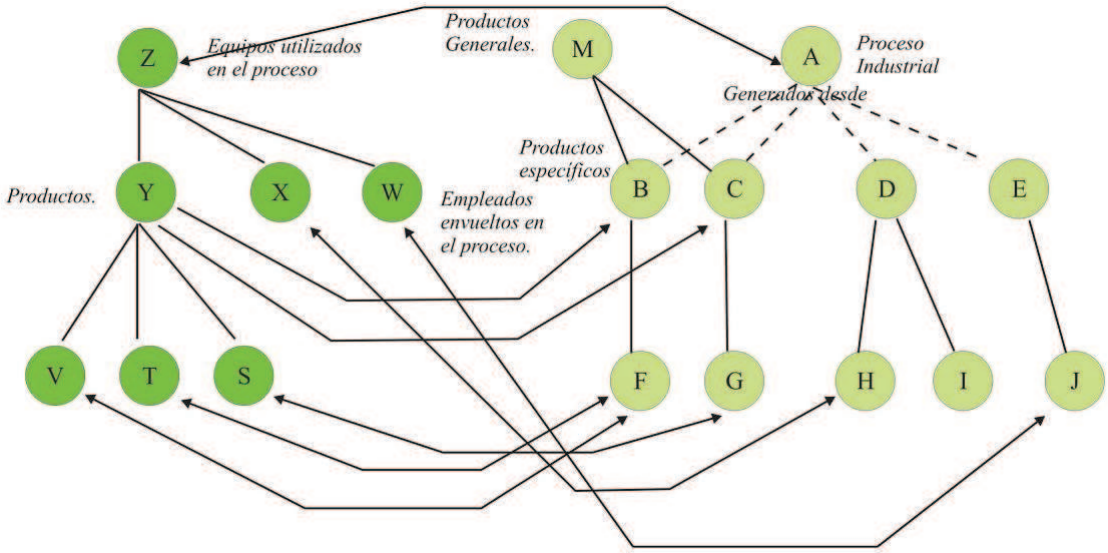

Figura 22: Ejemplo de mapeo entre ontologías

4.1.2. Los agentes inteligentes en el Grid

La tecnología Grid se perfila como la encargada de llevar a la Web hacia una nueva evolución. La principal diferencia entre Internet y la tecnología Grid es que, mientras las capas superiores de Internet se centran en la estandarización de la representación de los datos para facilitar así la interacción con el hombre, el Grid trata de estandarizar la representación de los servicios. Este hecho provoca la aparición de un amplio espectro de posibilidades de interacción automática entre máquinas.

Para que esta interacción tenga éxito y la coordinación entre los Servicios Grid se ejecute de manera eficiente, la mejor opción es disponer de un programa que muestre un

comportamiento “inteligente”. Los sistemas multiagentes cumplen con esta premisa por medio de sus cuatro cualidades principales (ver sección 2.3.1):

- **Autonomía:** capacidad para actuar sin la intervención humana.
- **Reactividad:** reaccionar por sí solo a los estímulos de su entorno.
- **Sociabilidad:** capacidad de comunicarse con otros agentes.
- **Aprendizaje:** mejorar su comportamiento basándose en la experiencia.

Con el paso del tiempo, irán surgiendo aplicaciones que, aprovechando el gran número de posibilidades que ofrece la integración de los sistemas multiagentes en la infraestructura Grid, demostrarán un comportamiento “inteligente”, adaptativo y autónomo, que será la clave para completar la evolución de la que se habla al comienzo de esta sección.

Algunas de las aplicaciones prácticas que aúnan la tecnología Grid con la “inteligencia” de los sistemas multiagente son las siguientes:

- **Planificación de tareas en el Grid:** para conseguir escalabilidad se necesita adaptabilidad. Esta adaptabilidad debe ser automática para sistemas que quieran ofrecer garantías de calidad. Los agentes software proporcionan las técnicas necesarias para llevar a cabo dicha adaptabilidad automática.
- **Calidad de servicio (QoS, Quality of Service) en el Grid:** para la gestión de la calidad de servicios entre múltiples servicios, se necesita, como mínimo, algún tipo de mecanismo de negociación automática.
- **Gestión de SLAs (Service Level Agreement):** un SLA es un es un acuerdo de nivel de servicio por el que una compañía se compromete a prestar un servicio a otra bajo determinadas condiciones y con un nivel de calidad y prestaciones mínimas. Como en el caso anterior, la gestión de los SLAs de los servicios de forma automática requiera cierta coordinación y negociación. Ambas características proporcionadas por los agentes inteligentes.

4.1.3. Otros trabajos y proyectos relacionados

En esta sección se describen brevemente algunos de los trabajos y proyectos más importantes e influyentes que, de un modo u otro hacen uso de las tecnologías Grid, multiagente y Web Semántica, de forma conjunta o por separado.

Ontogrid (`http://www.ontogrid.net`)

Uno de los grandes retos a los que se enfrenta la computación Grid considera la habilidad de compartir explícitamente conocimiento para ser usado para el desarrollo de infraestructuras y aplicaciones Grid innovadoras (el Grid Semántico). El objetivo de Ontogrid es generar la infraestructura tecnológica que permita el prototipado rápido y el desarrollo de servicios distribuidos abiertos basados en el conocimiento para el Grid Semántico.

OntoGrid es un proyecto coordinado por la Universidad Politécnica de Madrid e iniciado en octubre de 2004. Surgió con la idea de investigar las cuestiones fundamentales del Grid Semántico, y hacer de puente entre la comunidad de sistemas basados en conocimiento y la comunidad Grid. El Grid Semántico no debe únicamente proveer una infraestructura general basada en el conocimiento, sino que además deberá proporcionar un rico conjunto de servicios de conocimiento y de servicios basados en conocimiento. Así Ontogrid reúne a los servicios de conocimiento (servicios de ontologías, almacenamiento de metadatos, motores de razonamiento), con los Servicios Grid (gestión de flujo de trabajo, gestión de Organizaciones Virtuales, depuración, integración de datos, etc.) adaptados a descripciones semánticas, si es posible.

Este enfoque basado en la semántica del Grid va de la mano de la explotación de las tecnologías de agentes inteligentes y de la computación P2P[23] (*peer to peer*). Ontogrid mezcla la tecnología de agentes para la negociación y coordinación y P2P para el descubrimiento distribuido.

Entre sus resultados más importantes se encuentran los siguientes:

- **S-OGSA:** una arquitectura de referencia para la construcción de la Grid Semántica (ver sección 2.2.4).
- **ODESGS:** un entorno para la anotación de Servicios Grid existentes y el desarrollo de nuevos Servicios Grid Semánticos complejos, por medio de una interfaz gráfica de usuario (ver sección 3.2.3).
- **WS-DAIont:** define un framework para la creación de servicios de acceso a ontologías en un entorno Grid con las siguientes características:
  - Cumple plenamente S-OGSA.
  - Se basa en los estándares Grid.
  - Se basa en los estándares de Servicios Web.

[23] Una red P2P se refiere, a grandes rasgos, a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. Es un modelo de red que contrasta con el modelo cliente-servidor el cual se rige por una arquitectura monolítica donde no hay distribución de tareas entre sí, solo una simple comunicación entre un usuario y una terminal en donde el cliente y el servidor no pueden cambiar de roles.

K-WF Grid (`http://www.kwfgrid.eu`)

El objetivo principal del proyecto K-Wf Grid es proporcionar un soporte basado en el conocimiento para la construcción y ejecución de flujos de trabajo en un entorno de computación Grid. Para lograr este objetivo, los miembros del consorcio que se encargan del proyecto están desarrollando un sistema que permitirá a los usuarios efectuar una composición semiautomática de flujos de trabajo de Servicios Grid, ejecutar dicho flujo de trabajo en un entorno de computación Grid, monitorizar la realización de la infraestructura y las aplicaciones Grid, y finalmente reutilizar la suma de conocimientos obtenida de todos los usuarios con el fin de construir de manera eficiente los flujos de trabajo para las nuevas aplicaciones Grid.

Entre sus resultados, se encuentra la aplicación WSRF2OWL-S (ver sección 3.2.3) que será utilizada en este proyecto para la generación semiautomática de conocimiento semántico de Servicios Grid.

4.2. ANÁLISIS DE REQUISITOS

En esta sección, se describe la funcionalidad que el sistema debe proporcionar y las restricciones a las que debe ser sometido. Estos requisitos han sido obtenidos en base a la especificación de objetivos del documento que detalla la plataforma en la que se integrará la aplicación resultante de este proyecto [21].

4.2.1. Requisitos funcionales

RF.1 El sistema deberá permitir el registro de usuarios consumidores, proveedores de servicios y desarrolladores en la plataforma. Además, proporcionará mecanismos para visualizar, editar o borrar los datos de los usuarios registrados.

RF.2 Los usuarios deberán autenticarse con éxito para poder entrar en el sistema.

RF.3 El sistema deberá permitir al usuario proveedor, registrar y eliminar los Servicios Grid que proporcione.

RF.4 El sistema deberá ser capaz de procesar una petición de un usuario consumidor, ya sea expresada en lenguaje natural, o en formato XML, y ofrecer al usuario una lista con los posibles servicios que satisfacen dicha petición.

RF.5 Una vez elegido el servicio, el sistema deberá poder invocarlo y devolver los resultados al usuario consumidor.

4.2.2. Requisitos de interfaz

RI.1 La interfaz del sistema deberá permitir al usuario consumidor especificar una petición mediante lenguaje natural (escribiéndola por teclado) o mediante un fichero XML (con su URL o en forma de fichero local).

RI.2 Cuando se procese la petición, la interfaz del sistema deberá permitir elegir, al usuario consumidor, un servicio de entre una lista de servicios que satisfacen sus necesidades.

RI.3 El sistema proporcionará una interfaz por la cual los usuarios podrán registrarse, y los proveedores podrán registrar sus servicios.

RI.4 El sistema proporcionará una interfaz que permita la autenticación del usuario mediante login/password.

4.2.3. Requisitos de calidad

RC.1 El tamaño de las páginas ha de ser tal que el tiempo de carga sea inferior a 10 segundos de media para un usuario con una conexión de 56 Kbps. A excepción de las vistas en las que intervengan procesos de búsqueda, en las que se establece un tiempo máximo de 60 segundos.

RC.2 El sistema deberá poder recuperarse ante problemas con la conexión con los distintos agentes y de controlar las peticiones duplicadas de los usuarios.

RC.3 El sistema deberá tener un control sobre los errores y excepciones ocurridas, indicando en cada momento dicho error o excepción.

4.2.4. Requisitos de entorno

RE.1 El sistema podrá ejecutarse en sistemas Windows o Linux, en máquinas cuyo procesador tenga una potencia de procesamiento de al menos 1 GHz y cuya memoria tenga una capacidad de al menos 512 MBytes (recomendado).

RE.2 La aplicación Web será alojada en un servidor Tomcat 5.0 o superior.

RE.3 Para la ejecución del sistema se necesitará una maquina virtual de Java (JVM) 1.5 o superior.

4.2.5. Requisitos de seguridad

RS.1 Se deberá mantener un entorno físico no contaminado y libre de amenazas potenciales como fuego, agua, polvo, radiaciones, etc. Y deberá disponer del equipamiento adecuado de seguridad que controle la temperatura y la humedad.

RS.2 Se deberá prevenir el acceso no autorizado a las instalaciones donde esté alojado el sistema.

RS.3 Elaborar protocolos de actuación en casos de emergencia y revisarlos con periodicidad.

RS.4 Elaborar un sistema de copias de seguridad y guardar dichas copias de seguridad en lugares diferentes de donde se encuentra la aplicación.

RS.5 Cada usuario que quiera interactuar con el sistema deberá registrarse en él, proporcionando una serie de datos, y autenticarse mediante login/password.

RS.6 Se cifrarán las comunicaciones y los mensajes para asegurar la privacidad e integridad.

4.3. DESCRIPCIÓN GENERAL

4.3.1. Arquitectura del sistema

Mediante este proyecto, se ha desarrollado un sistema multiagente basado en conocimiento para entornos de Servicios Grid Semánticos. De esta forma, el descubrimiento, composición, selección, invocación y monitorización de Servicios Grid Semánticos se hace de forma automática, para conseguir evitar que los usuarios se encarguen de tan tediosa labor. La arquitectura abstracta del sistema se muestra en la Figura 23.

El SMA está formado por un conjunto de bases de conocimiento (ontologías) y un conjunto de agentes. Cada uno de los agentes puede jugar diferentes papeles (roles).

Agentes

Podemos distinguir tres tipos distintos de agentes en la plataforma. Por un lado están los agentes que actúan en nombre de los proveedores de servicios, cuya misión es la de gestionar el acceso a los servicios y controlar que los contratos establecidos con los clientes sean cumplidos. Por otra parte, los agentes que representan a los consumidores de servicios, son los encargados de localizar los servicios que satisfagan las necesidades de los consumidores, establecer los contratos y presentar los resultados obtenidos. Y por

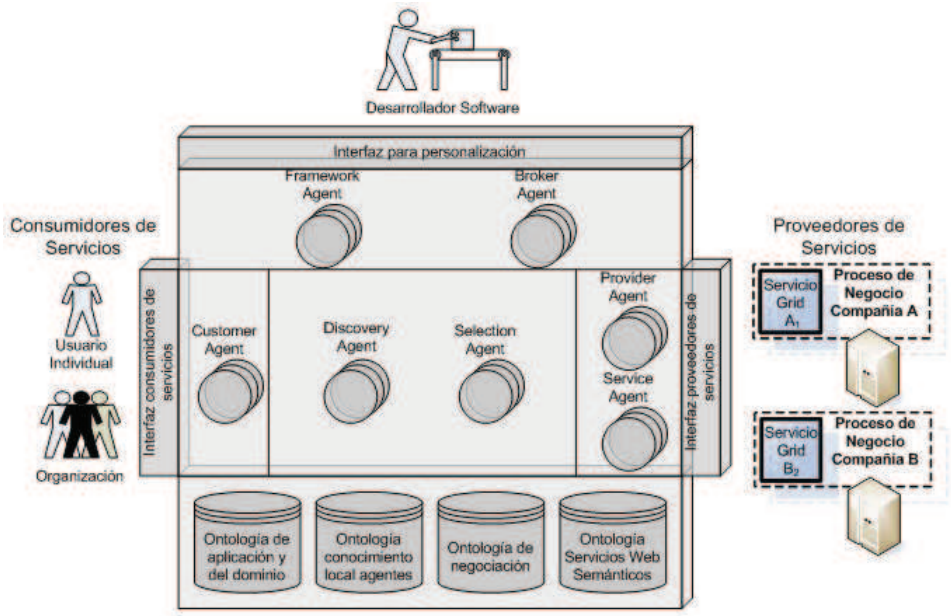

Figura 23: Arquitectura del sistema multiagente

último, los agentes que se encargan de las tareas de gestión, que controlan los recursos de la plataforma, la interacción entre los demás agentes y resuelven los problemas de interoperabilidad en las comunicaciones.

A continuación, se describe cada uno de los agentes participantes en la plataforma de manera más detallada:

- **Framework Agent:** es el encargado de monitorizar el estado de los agentes, y asegurar el correcto funcionamiento de la plataforma y las interacciones entre sus componentes. También gestiona los recursos disponibles y equilibra la carga de trabajo.

- **Broker Agent:** es el responsable de resolver los problemas de interoperabilidad que puedan surgir en las comunicaciones que tienen lugar entre los otros agentes. Estos son, interoperabilidad de datos, de protocolos, de procesos y funcional.

- **Customer Agent:** este agente representa al usuario consumidor de servicios. El consumidor final delega en el Customer Agent su objetivo y éste lo representa en el formato de los mensajes correspondiente, lo envía al agente correspondiente y, en general, se desentiende hasta que le devuelven los resultados. Puede que el agente deba consultar datos con el usuario durante el proceso.

- **Discovery Agent:** es el encargado de buscar los Servicios Grid, que satisfagan los objetivos del usuario. Si no encuentra ningún servicio que cumpla con los objetivos de forma individual, el agente buscará alguna composición de servicios que si los cumpla.

- **Selection Agent:** selecciona el servicio (individual o compuesto) que más se ajuste a los objetivos del usuario, de entre los servicios encontrados por el descubridor (Discovery Agent), de acuerdo a un criterio de utilidad basado en las preferencias y objetivos establecidos por el usuario.

- **Provider Agent:** es el representante de los proveedores de Servicios Grid. Su misión es encargarse de que se cumplan las condiciones impuestas por el proveedor, a la hora de ejecutar los Servicios Grid. Estas condiciones las establecen las organizaciones o entidades proveedoras de acuerdo a sus objetivos estratégicos.

- **Service Agent:** representa a los Servicios Grid disponibles. Es el encargado, en caso de seleccionarse el servicio al que representa para ser ejecutado, de realizar la invocación pertinente, teniendo en cuenta en todo momento, que las condiciones y preferencias, establecidas por el proveedor, se satisfagan totalmente durante el proceso de ejecución.

En la plataforma puede haber varios agentes del mismo tipo ejecutándose a la vez. En la Figura 24, se muestran las interacciones que se producen entre los agentes de la plataforma, y los consumidores y proveedores.

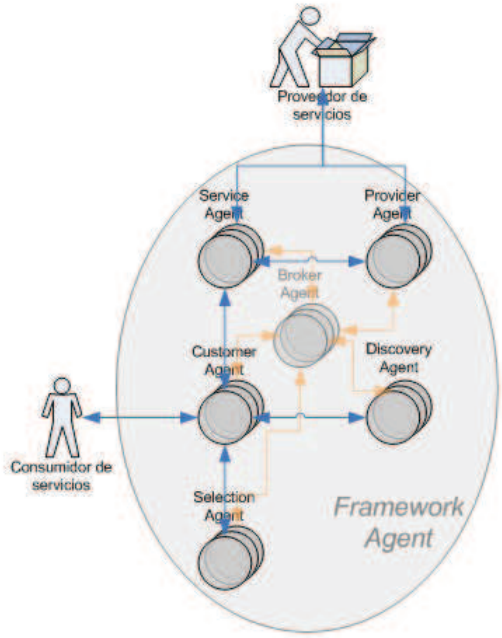

Figura 24: Interacciones entre los agentes

Roles

Con el objetivo de dar flexibilidad al framework, se ha utilizado un mecanismo de roles, mediante el cual la funcionalidad concreta y el modo de operación pueden ser asignados de forma dinámica en tiempo de ejecución. Es decir, un agente desempeñará su tarea dependiendo del rol o roles a los que haya sido asociado.

El uso de los roles aporta ciertas ventajas al framework:

- Dinamismo y flexibilidad.
- Están guiados por responsabilidades.
- Sensibilidad al contexto.

En la Figura 25, se muestra una clasificación de los roles incluidos en el framework.

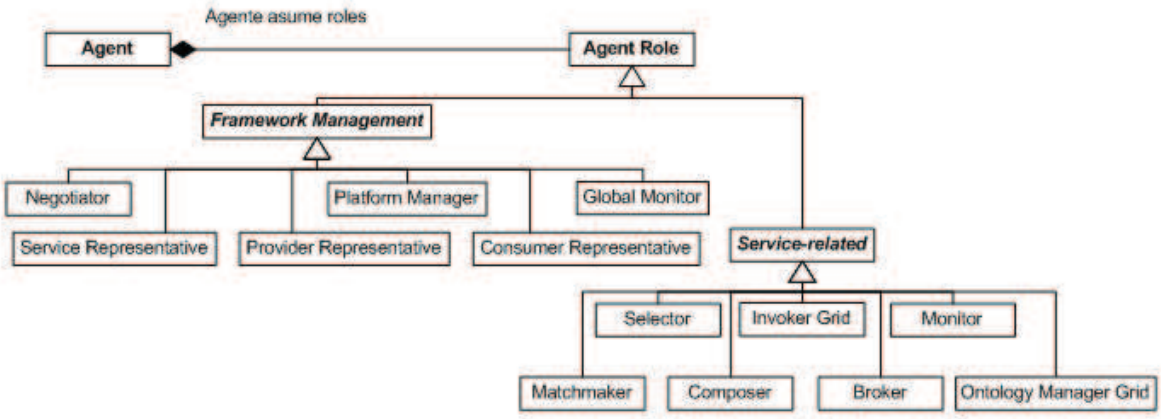

Figura 25: Clasificación de los roles del sistema

Se distinguen dos categorías principales de roles. Por un lado, están los roles relacionados con los Servicios Grid, que son los encargados de la gestión de los Servicios Grid, su descubrimiento e invocación. En la Tabla 4, se muestra una descripción detallada de cada uno de estos roles.

Tabla 4: Roles relacionados con los servicios

Rol	Descripción
Broker	Representa la funcionalidad que se necesita para resolver los problemas de interoperabilidad a nivel de datos, procesos y funciones.

Composer	Permite hacer una composición de servicios, basándose en una descomposición en subobjetivos, para la consecución de un objetivo global, mediante la ejecución ordenada de dichos servicios.
Invoker	Invoca un Servicio Grid una vez conocidos la operación concreta y los parámetros de esta.
Matchmaker	Encuentra los servicios cuya descripción semántica se corresponda con el objetivo impuesto por el usuario.
Monitor	Asegura que los contratos establecidos por proveedores y consumidores para la ejecución de los servicios son cumplidos.
Ontology Manager	Proporciona la funcionalidad asociada al acceso y manipulación de las ontologías. De esta forma, se encapsula toda la funcionalidad en un único componente.
Selector	Selecciona un servicio de una lista de servicios, basándose en un cálculo a partir de un conjunto de preferencias dado.

Y, por otro lado, aparecen los roles cuyas tareas tienen que ver con la gestión de la plataforma y la interacción con los usuarios externos. En la Tabla 5 se pueden ver las descripciones de cada uno de estos roles.

Tabla 5: Roles relacionados con el funcionamiento de la plataforma

Rol	**Descripción**
Negotiator	Maneja las estrategias de negociación entre agentes, estableciendo las bases para la ejecución del sistema.
Platform Manager	Se encarga controlar y balancear la carga de trabajo del sistema.
Provider Representative	De forma abstracta, se encarga de asegurar que las condiciones impuestas por la estrategia de negocio son cumplidas en cada momento.
Service Representative	Representa al servicio, participando en los procesos de negociación. Además mejora la funcionalidad del servicio cuando sea posible.
Consumer Representative	Gestiona las acciones en representación de los consumidores de servicios, determina sus deseos y condiciones, y procesa los resultados obtenidos.
Global Monitor	Monitoriza los acontecimientos que tienen lugar en el sistema. Detecta los posibles errores y determina las acciones a tomar para solventarlos.

La implementación de los roles es lo que determina el comportamiento de estos. Los agentes, en tiempo de ejecución, asumen unos determinados roles de forma dinámica, dependiendo de los objetivos que persigan en un momento determinado. Sin embargo, existen ciertos roles que deben ser asumidos por los agentes para que estos no pierdan su identidad propia. Esta asociación entre agentes y roles se puede ver reflejada en la Tabla 6.

Tabla 6: Correspondencia entre los agentes y sus roles

Tipo de agente	**Roles que desempeña**
Broker	BrokerRol OntologyManagerRol
Customer	BrokerRol OntologyManagerRol GlobalMonitorRol ConsumerRepresentativeRol
Discovery	MatchMakerRol ComposerRol OntologyManagerRol
Framework	GlobalMonitorRol PlatformManagerRol
Provider	ProviderRepresentativeRol GlobalMonitorRol
Selection	SelectorRol NegotiatorRol OntologyManagerRol
Service	ServiceRepresentativeRol InvokerRol Negotiator OntologyManagerRol GlobalMonitorRol

Ontologías

Las ontologías son el elemento clave del proyecto, ya que actúan como nexo de unión entre los demás elementos de la arquitectura. En realidad funcionan como vocabularios universales, de forma que, tanto los Servicios Grid como los agentes, comparten la misma interpretación de los términos que contienen sus comunicaciones.

La aplicación necesita varias fuentes de datos que, en forma de ontologías, contienen el conocimiento requerido para la ejecución del sistema:

1. **Ontología del dominio:** representan el conocimiento especializado de un dominio en particular, es decir, contiene los elementos de conocimiento que aparecen en el universo de discurso. De esta forma, los componentes del sistema podrán distinguir, sin ambigüedad, los distintos objetos del dominio.

2. **Ontología de la aplicación:** contiene el conocimiento esencial para modelar la aplicación que se desea modelar en particular.

3. **Ontología de negociación:** describe los elementos que constituyen el proceso de negociación[24]. De esta forma se consigue independizar la plataforma del mecanismo de negociación concreto.

4. **Ontología de conocimiento local de agentes:** contiene, para cada agente, el conocimiento disponible de su entorno, su tarea, y los mecanismos y recursos necesarios para llevar a cabo dicha tarea. Por tanto, esta ontología depende en gran medida del tipo de agente y los roles que asuma. Normalmente estará construida en base a la ontología del dominio.

5. **Ontología de Servicios Grid:** describe semánticamente, las propiedades de invocación y las capacidades que ofrecen los Servicios Grid que los proveedores proporcionan a la plataforma.

Servicios Grid Semánticos

Las descripciones semánticas de los Servicios Grid proporcionados por la plataforma se encuentran disponibles por medio de los servicios *SemanticBindingService* y *MetadataQueryService* (ver sección 3.2.2).

Este servicio permite asociar un Servicio Grid con su descripción semántica, dando lugar a lo que se conoce como un Semantic Binding. Una vez creado un Semantic Binding, es posible hacer consultas sobre él directamente (mediante *SemanticBindingService*) o sobre la totalidad de Semantic Bindings que hayan sido creados (mediante *MetadataQueryService*).

Los agentes que componen la plataforma hacen uso de estos servicios para consultar la información semántica de los Servicios Grid, contenida en los Semantic Bindings, para utilizarla durante el proceso de ejecución de los mismos, esto es, funcionalidad que proporciona (para las búsquedas), punto de acceso, y condiciones y parámetros de ejecución (para la invocación), etc.

[24] En la actualidad, se hace uso del protocolo CFP de forma estática en todas las negociaciones

4.3.2. Detalles de la implementación

Como se comento en la sección 3.1.2, este proyecto es una ampliación del proyecto SEMMAS, lo cual quiere decir que no parte desde cero, sino que se apoya en una implementación ya desarrollada. Para adaptar la plataforma de agentes a los nuevos tipos de servicios (Servicios Grid) ha habido que añadir y modificar algunas partes de la implementación. En esta sección se describirán los cambios más importantes que ha sufrido la plataforma.

Dos de las características más importantes, tanto de este proyecto como del proyecto SEMMAS, son su dinamismo y flexibilidad, ya que es posible asignar los roles dinámicamente a los agentes para definir las tareas que han de realizar. Los roles son, por tanto, los encargados de proporcionar la verdadera funcionalidad. Teniendo en cuenta estas características y que no se pretende cambiar la funcionalidad, sino adaptarla al nuevo tipo de servicios, se llega a la conclusión de que únicamente es necesario modificar (o añadir) en la plataforma aquellos roles cuya implementación no sea compatible con los Servicios Grid. Esto es:

- **OntologyManagerRol:** la manera en que se tratan las ontologías de Servicios Web y Servicios Grid es diferente. Para las ontologías de Servicios Web se utilizaba un repositorio de ontologías (Sesame[25]) y sobre él se lanzaban las consultas en lenguaje SPARQL, mientras que para las ontologías de Servicios Grid se utilizan los servicios SemanticBindingService y MetadataQueryService, y sobre ellos se lanzan consultas en lenguaje SeRQL (Sesame RDF Query Language). El siguiente ejemplo muestra cómo se hace una consulta sobre un Semantic Binding identificado por `sbKey`.

```
SemanticBindingServiceAddressingLocator instanceLocator =
   new SemanticBindingServiceAddressingLocator();
①
EndpointReferenceType instanceEPR = new EndpointReferenceType();
instanceEPR.setAddress(new AttributedURI(
   "http://localhost:18080/wsrf/services/SemanticBindingService"));
②
SimpleResourceKey key = new SimpleResourceKey(
   SemanticBindingConstants.RESOURCE_KEY, sbKey);
ReferencePropertiesType props = new ReferencePropertiesType();
props.add(key.toSOAPElement());
instanceEPR.setProperties(props);
③
String queryOutput = "SELECT Output FROM Y rdf:type process:Output,"
   + "Y process:parameterType Output USING NAMESPACE process ="
   + "<http://www.daml.org/services/owl-s/1.1/Process.owl#>";
```

[25] http://www.openrdf.org

```
④
SemanticBindingPortType sb = instanceLocator.
   getSemanticBindingPortTypePort(instanceEPR);
⑤
String resultOutput = sb.miniQuery(queryOutput);
```

① Se crea un *EndpointReferenceType* que servirá para poder acceder al *WS-Resource* (ver sección 2.2.5). En este caso, sólo necesita la URI del servicio SemanticBindingService para crearlo.

② Se añade el identificador (*SimpleResourceKey*) del Semantic Binding sobre el que recae la consulta.

③ Se escribe la consulta.

④ Se obtiene una referencia a la interfaz o *portType* (ver sección 2.2.2) del SemanticBindingService.

⑤ Se ejecuta la consulta. Los resultados se obtienen en forma de *String* con formato XML.

- **InvokerRol:** la invocación de Servicios Web y Servicios Grid se hace de manera diferente. Ahora, para los Servicios Grid, primero debes crear una instancia del servicio y luego invocarlo incluyendo en la petición de invocación el identificador de la instancia del servicio. El siguiente ejemplo muestra, a grandes rasgos, cómo se ejecuta un Servicio Grid.

```
...
①
OMElement payload = createEnvelopeCreateResource(ns);
ConfigurationContext configctx = ConfigurationContextFactory
   .createConfigurationContextFromFileSystem(null, null);
ServiceClient client = new ServiceClient(configctx, null);
EndpointReference targetEPR = new EndpointReference(url);
Options options = new Options();
client.setOptions(options);
options.setTimeOutInMilliSeconds(40000);
options.setTo(targetEPR);
options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
OMElement result = client.sendReceive(payload);
②
OMFactory fac;
OMNamespace omNs;
Element nodo;
OMElement header;
Document res = string2DOM(result.toString());
NodeList nl = res.getElementsByTagName("wsa:ReferenceProperties");
```

```
NodeList rpropsnl = nl.item(0).getChildNodes();
for (int i = 0; i <rpropsnl.getLength(); i++) {
   nodo = (Element) rpropsnl.item(i);
   fac = OMAbstractFactory.getOMFactory();
   omNs = fac.createOMNamespace(ns, "ns1");
   header = fac.createOMElement(nodo.getNodeName(), omNs);
   header.setText(nodo.getFirstChild().getNodeValue());
   client.addHeader(header);
}
③
payload = createEnvelope(method, ns, m);
result = client.sendReceive(payload);
value = result.getFirstElement().getText();
...
```

A diferencia del rol OntologyManager, se utilizará la librería Axis2[26] para ejecutar los Servicios Grid en lugar de las clases Stub de estos. De esta manera se obtiene un modo de ejecución genérico que no depende del Servicio Grid que se esté ejecutando en cada momento.

① Crea la llamada al método de creación de la instancia del Servicio Grid y lo ejecuta.

② El resultado de la creación de la instancia del Servicio Grid contiene, entre otras cosas, el identificador de la instancia creada. Este identificador debe incluirse en la cabecera de la llamada al método del Servicio Grid que queremos ejecutar. Esta forma de ejecución no aprovecha la propiedad *stateful* de los Servicios Grid, pero está previsto (ver sección 5.2.2) que en el futuro se implementen mecanismos que permitan mantener el estado de las ejecuciones de los servicios, así no sera necesario crear la instancia del Servicio Grid cada vez que vaya a ser ejecutado uno de sus métodos.

③ Por último, se crea la llamada al método del Servicio Grid que se quiere ejecutar, y se ejecuta.

4.3.3. Arquitectura de la aplicación Web

Se ha desarrollado una aplicación Web para permitir la interacción con los usuarios externos al sistema. La metodología elegida para el diseño de la aplicación se ajusta al modelo de Struts[27]. Struts es una herramienta de soporte para el desarrollo de aplicaciones

[26] http://ws.apache.org/axis2
[27] http://struts.apache.org

Web cuyo principal propósito es conseguir una separación clara entre la lógica de negocio y la presentación de la aplicación, según el patrón MVC (modelo-vista-controlador). Con esta herramienta conseguimos reducir el tiempo de desarrollo de la aplicación, además es totalmente compatible con las otras tecnologías utilizadas.

Como se ha dicho antes, Struts se basa en el patrón MVC, el cual es ampliamente utilizado y considerado de gran solidez. Según este patrón, el procesamiento debe separarse en tres secciones bien diferenciadas: el modelo, las vistas y el controlador. Se puede pensar en utilizar varios controladores en lugar de uno, pues con un único controlador, donde se encapsule toda la lógica de negocio, podría quedar saturado de peticiones y convertirse en un controlador de peticiones (problema del *fat controller*). Struts soluciona este problema mediante la implementación de un sólo controlador (ActionServlet) que evalúa las peticiones del usuario mediante un archivo configurable (struts-config.xml).

En el MVC, mientras que los componentes del modelo se encargan del acceso a bases de datos o de la comunicación con sistemas independientes de la aplicación, los componentes de control (controladores) se encargan de coordinar las actividades de la aplicación, como recepción de datos de usuario, selección de componentes que deben ser llamados, etc. Esta separación simplifica en gran medida, tanto la escritura de las vistas como la de los demás componentes, y permite a los desarrolladores del modelo trabajar independientemente de los diseñadores de las vistas.

Las principales características de Struts son las siguientes:

- Permite una configuración del control centralizada.
- Las interrelaciones entre acciones y páginas u otras acciones se especifican por medio de tablas XML en lugar de codificarlas en los programas o páginas.
- Proporciona librerías de entidades utilizadas para facilitar la mayoría de las operaciones que generalmente realizan las páginas JSP.
- Contiene herramientas que utiliza para la validación de campos de plantillas bajo varios esquemas que van desde validaciones locales en la página, hasta validaciones de fondo hechas a nivel de las acciones.

En la aplicación Web que ha sido desarrollada, el servlet “ActionServlet” se ha llamado “FrontController”, y es el responsable de procesar todas las peticiones. Una vez recibida la petición, decide qué acción de las disponibles es la más adecuada y la invoca. Las acciones son las encargadas de realizar toda la lógica de negocio y de devolver los resultados en forma de vistas. En la Figura 26, se muestra la estructura general de la aplicación Web de forma gráfica.

Las asociaciones entre peticiones y acciones es resuelta por la clase “PetitionHelper”, que se ayuda de un fichero de propiedades en el que se encuentran dichas asociaciones

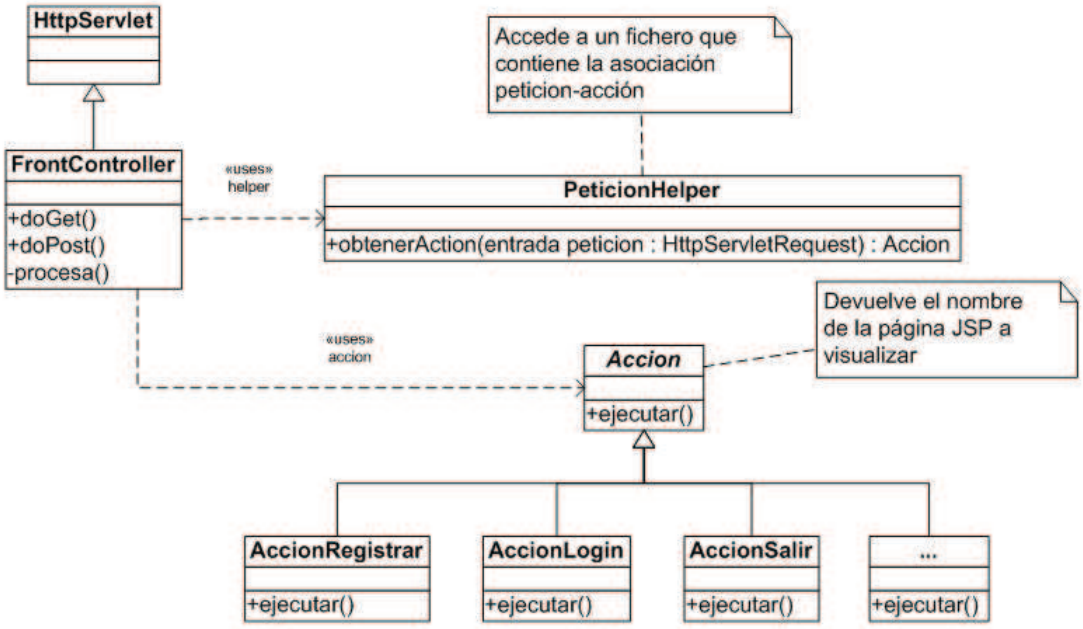

Figura 26: Arquitectura de la aplicación Web

en la forma *<petición>=<clase de la acción>*. Una vez resuelta la asociación, el servlet "FrontController" invoca la acción que se encargará de acceder a la base de datos y de realizar la comunicación con los agentes de la plataforma.

La gestión de los usuarios (proveedores y consumidores de servicios) la lleva a cabo la propia aplicación. Cuando un usuario se registra, sus datos y preferencias quedan almacenados en una base de datos. Actualmente, la aplicación hace uso de MySQL como gestor de base de datos, sin embargo, gracias a la elaboración de los accesos a la fuente de datos en base al patrón DAO (Data Access Object), es posible almacenar y recuperar los datos de fuentes distintas (LDAP, XML, otros sistemas gestores de bases de datos, etc.). Los beneficios que aporta son los siguientes:

1. Favorece la transparencia.
2. Facilita la migración de los componentes.
3. Reduce la complejidad.
4. Centraliza el acceso a los datos en una capa.

El mayor inconveniente de esta solución es que puede provocar un crecimiento desmesurado del número de clases. En la Figura 27 se muestra gráficamente la estructura de clases que supone la implementación del patrón DAO.

Con esta aplicación Web, los usuarios únicamente requieren de acceso a Internet y de

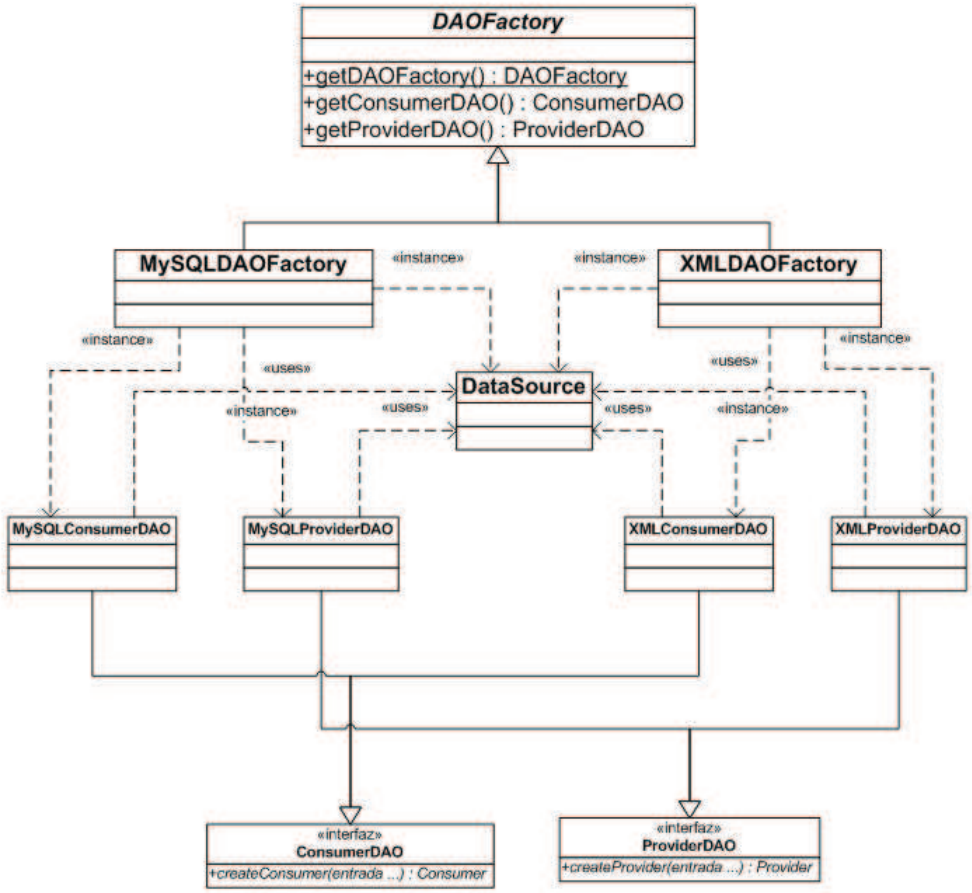

Figura 27: Estructura del patrón DAO

un navegador Web instalado en su equipo, para poder sacar provecho de toda la funcionalidad que ofrece plataforma.

4.3.4. Conexión de la aplicación Web con el sistema de agentes

Para integrar el sistema de agentes con la aplicación Web, JADE ofrece el paquete *jade.wrapper.gateway*, que contiene dos clases mediante las cuales, servlets y agentes se pueden comunicar. Estas clases son [30]:

- **JadeGateway:** es una clase singleton que contiene únicamente métodos estáticos.
- **GatewayAgent:** clase que deberá heredar uno de los agentes.

La estructura del sistema que utiliza JADE para permitir la comunicación entre la aplicación Web y el sistema multiagente es la que puede verse en la Figura 28.

Para establecer una comunicación se deben seguir los siguientes pasos:

1. En el browser, el usuario provoca un evento y este genera un mensaje POST.

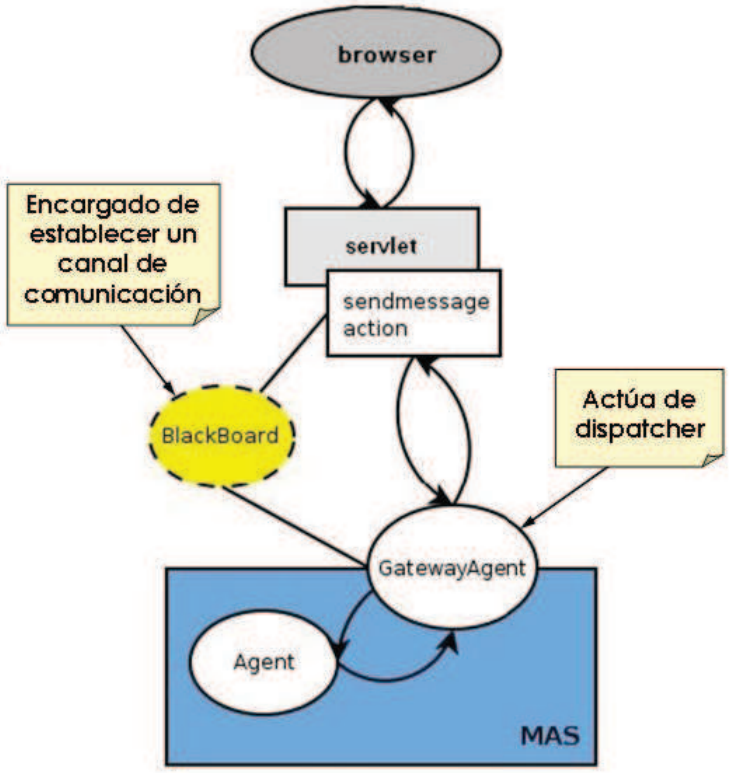

Figura 28: Estructura del sistema de comunicación [30]

2. El servlet intercepta el mensaje POST, e invoca la acción sendmessage.

3. La acción crea un nuevo objeto BlackBoard que actuará de canal de comunicación entre el servlet y el GatewayAgent.

```
BlackBoardBean board = new BlackBoardBean();
board.setReceiver("Agent");
board.setMessage("message");
try {
   JadeGateway.execute(board);
} catch(Exception e) {
   e.printStackTrace();
}
```

4. El GatewayAgent recibe el objeto BlackBoard creado anteriormente y extrae el contenido y el destinatario del mensaje. Posteriormente envía el mensaje.

```
BlackBoardBean board = null;
protected void processCommand(java.lang.Object obj) {
   System.out.println("Un objeto ha llegado: " +
      obj.toString());
   if (obj instanceof BlackBoardBean){
```

```
      board = (BlackBoardBean)obj;
      try {
         ACLMessage msg = new ACLMessage(
            board.getPerformative());
         msg.setProtocol(board.getProtocol());
         msg.addReceiver(new AID(board.getReceiver(),
            AID.ISLOCALNAME));
         msg.setContent(board.getMessage());
         if (board.getObject() != null) {
            msg.setContentObject(board.getObject());
            msg.setLanguage("JavaSerialization");
         }
         send(msg);
      } catch (Exception IOException) {
         board.setType(BlackBoardBean.ERROR);
         board.setMessage("An unexpected error happened");
         releaseCommand(board);
      }
   }
```

5. El agente destinatario, que recibe el mensaje, responde al Gateway a través del objeto BlackBoard.

```
addBehaviour(new CyclicBehaviour(this) {
   public void action() {
      ACLMessage msg = receive();
      if ((msg!=null)&&(board!=null)) {
         try {
            board.setMessage(msg.getContent());
            board.setPerformative(msg.getPerformative());
            board.setProtocol(msg.getProtocol());
            board.setObject(msg.getContentObject());
            board.setType(Integer.parseInt(
               msg.getUserDefinedParameter("TYPE"));
         } catch (Exception e) {
            e.printStackTrace(System.err);
         }
         releaseCommand(board);
      } else block();
   }
};
```

6. El GatewayAgent construye una respuesta y la envía al servlet por medio del BlackBoard.

7. Por último, el servlet da la respuesta final a la página Web accediendo al BlackBoard.

```
try {
   JadeGateway.execute(board);
} catch(Exception e) {
   e.printStackTrace();
}
...
out.print("Reply: " + board.getMessage());
```

El eje central sobre el que se construye la comunicación es el GatewayAgent. Esta visión (servlet - GatewayAgent - MAS) es totalmente escalable y permite tener totalmente separados las diferentes tecnologías. En algunos casos es posible que el GatewayAgent sea un cuello de botella, pero si utilizamos un grupo de GatewayAgents podríamos solucionar el problema.

4.3.5. Funcionamiento

El funcionamiento de la plataforma puede resumirse del siguiente modo: primero el usuario del sistema establece el objetivo. Este objetivo es analizado por el agente de usuario (Consumer Agent) el cual lo envía al agente descubridor (Discovery Agent). El agente descubridor busca entre los servicios disponibles aquellos que cumplan con los objetivos introducidos por el usuario, según indique su información semántica. Después le envía la lista resultante de servicios al agente de usuario. Esta lista es enviada por el agente de usuario al agente selector (Selection Agent) y este determinará qué servicio es el más adecuado. En este punto comenzaría el proceso de negociación con los agentes de servicio (Service Agent) de cada uno de los servicios de la lista para determinar qué oferta es la más adecuada. Una vez seleccionado el servicio, el agente selector se lo comunica al agente de usuario y este invoca el servicio interactuando directamente con el agente de servicio. Para terminar, se muestran por pantalla los resultados obtenidos tras la ejecución del servicio.

A continuación, en la Figura 29, se muestra el diagrama de secuencia de la plataforma. En él se detalla de forma gráfica el funcionamiento de la plataforma que se ha descrito anteriormente.

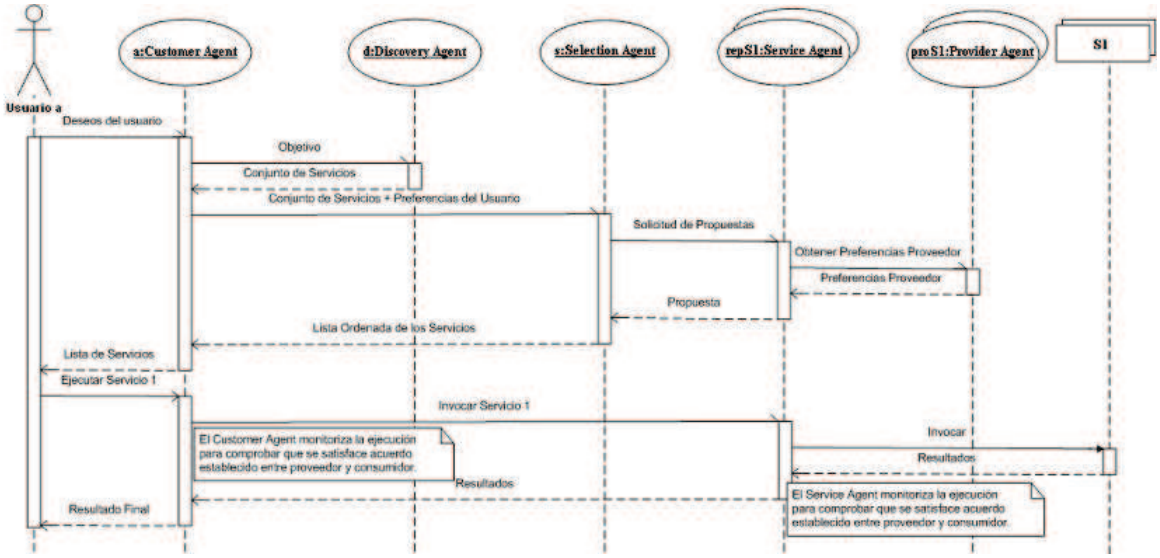

Figura 29: Diagrama de secuencia de la aplicación

Como se puede observar, es muy parecido al diagrama de interacción que genera JADE (Figura 30) tras realizar una ejecución real en el sistema.

4.3.6. Aplicación en matemáticas

Dada la alta capacidad de cómputo que pueden ofrecer los Servicios Grid, uno de los entornos en los que tiene sentido aplicar la plataforma es el entorno de las matemáticas.

Ontología numérica

Para la aplicación de la plataforma al ámbito de las matemáticas se ha utilizado una ontología que provee conceptos numéricos relacionados con las matemáticas. Esta ontología se llama Numerics y es parte del proyecto SWEET[28] (Semantic Web for Earth and Environmental Terminology), cuyo objetivo es crear un framework semántico común para diversas disciplinas científicas.

Los conceptos proporcionados por la ontología Numerics serán compartidos por los agentes integrantes de la plataforma para su comunicación, eliminando así las posibles ambigüedades de términos que pudieran surgir si existieran diferentes interpretaciones de los mismos.

[28] `http://sweet.jpl.nasa.gov/index.html`

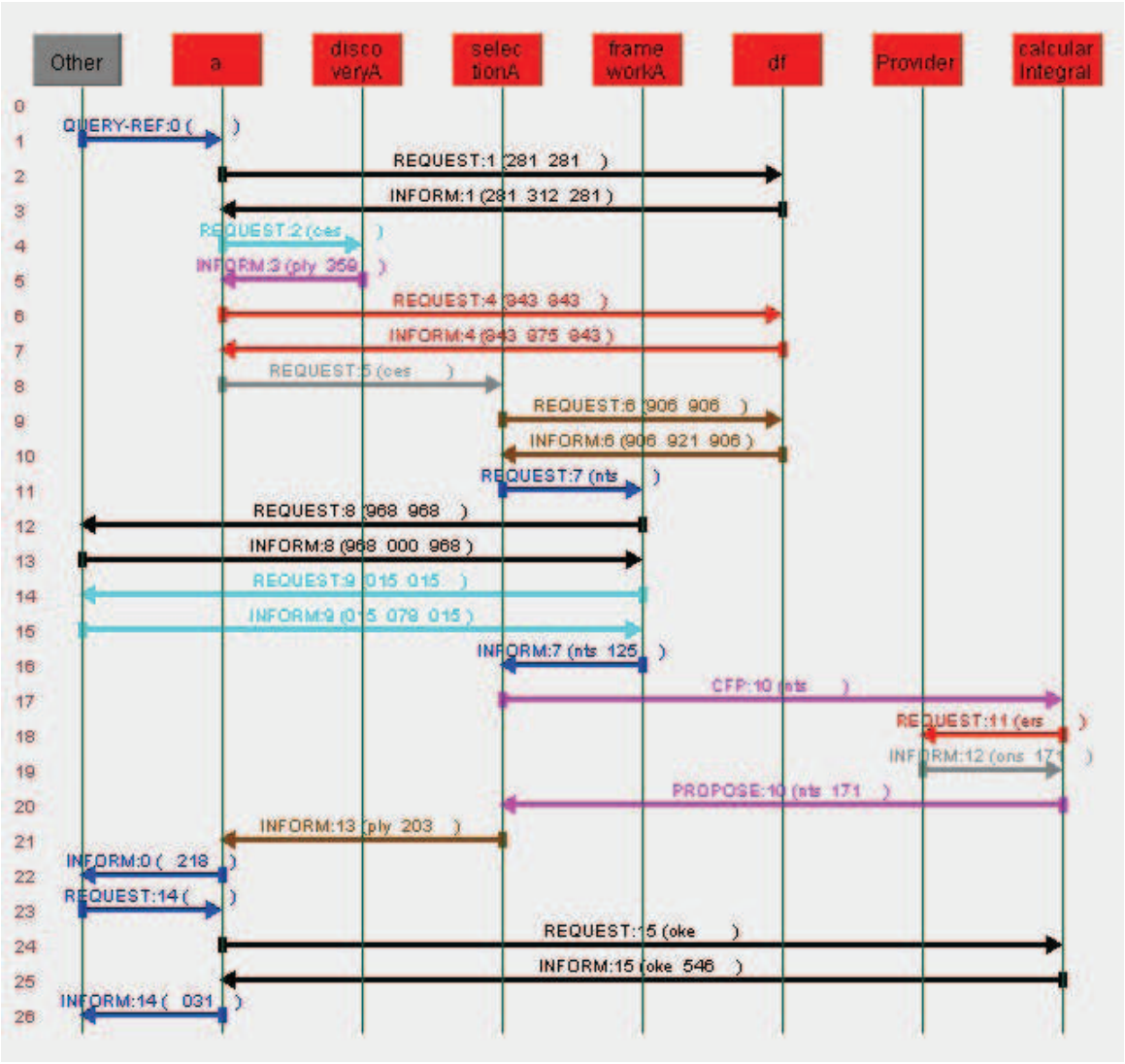

Figura 30: Diagrama de agentes en JADE

Ejemplo de Servicio Grid Semántico: IntegralesService

Este Servicio Grid, implementado en Java utilizando el plugin GDT de eclipse, calcula la integral definida de una función entre dos puntos, mediante el método de la división en rectángulos. Para representar la función, el servicio hace uso de la librería JEP[29] (Java math Expression Parser). A continuación, se muestran las partes más importantes del código que implementa el Servicio Grid:

```
import org.nfunk.jep.JEP;

public class CalculadorIntegral {
   private double desde;
   private double hasta;
```

[29] http://www.singularsys.com/jep

```
   private int nIntervalos;
   private String funcion;
   private double[] intervalos;
   private double longIntervalo;
   private JEP jep;

   public CalculadorIntegral(double desde, double hasta,
      int nIntervalos, String funcion) {
      jep = new JEP();
      ...
   }

   private double fdex(double x) {
      jep.initSymTab();
      ...
      return jep.getValue();
   }

   public double calcularIntegral() {
      double aux = 0;
      double resPar;
      for (int i = 0; i <nIntervalos; i++) {
         resPar = this.fdex(intervalos[i]);
         aux += resPar * this.longIntervalo;
      }
      return aux;
   }
}
```

La clase que utilizará GDT para generar el servicio es la siguiente:

```
import de.fb12.gdt.GridService;
import de.fb12.gdt.GridAttribute;
import de.fb12.gdt.GridMethod;

@GridService(name = "Integrales",
   namespace = "http://nacho.um.es/grid/integrales",
   ...
   resourceSecurityDesc = "[]")
public class Integrales {
   @GridAttribute
   private double integral;

   @GridMethod
   public double calcularIntegral(double desde, double hasta,
      int nIntervalos, String funcion) {
      CalculadorIntegral c = new CalculadorIntegral(
```

```
            desde, hasta, nIntervalos, funcion);
        this.integral = c.calcularIntegral();
        return this.integral;
    }

    public double getIntegral() {
        return integral;
    }

    public void setIntegral(double integral) {
        this.integral = integral;
    }
}
```

Como se puede observar en la última clase, el servicio ofrece un método (calcularIntegral) que se ayuda de la clase CalculadorIntegral para calcular la integral de una función. El método tiene cuatro parámetros de entrada *desde* y *hasta*, que representan la cota inferior y superior, respectivamente, de la función a la que se le aplicará la integral; *nIntervalos*, que representa el número de rectángulos en que se dividirá la función (cuanto mayor sea *nIntervalos* mayor será la exactitud de la integral); y *funcion*, que representa la función a la que se aplica la integral.

El fichero WSDL que se obtiene a partir de la clase Java anterior tiene, a grandes rasgos, el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="IntegralesService" targetNamespace=
  "http://nacho.um.es/grid/integrales/Integrales"...>
  ...
  <element name="calcularIntegral">
    <complexType>
      <sequence>
        <element name="desde" type="xsd:double"/>
        <element name="hasta" type="xsd:double"/>
        <element name="nIntervalos" type="xsd:int"/>
        <element name="funcion" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="calcularIntegralResponse">
    <complexType>
      <sequence>
        <element name="calcularIntegralReturn" type="xsd:double"/>
      </sequence>
    </complexType>
  </element>
```

```
  ...
  <wsdl:portType name="IntegralesPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:IntegralesResourceProperties">
    <wsdl:operation name="calcularIntegral">
      <wsdl:input message="impl:calcularIntegralRequest"
        name="calcularIntegralRequest"/>
      <wsdl:output message="impl:calcularIntegralResponse"
        name="calcularIntegralResponse"/>
    </wsdl:operation>
    ...
  </wsdl:portType>
  ...
</wsdl:definitions>
```

Y, por último, se muestra parte del contenido del fichero que almacena la descripción semántica del método calcularIntegral, generado por la aplicación WSRF2OWL-S:

```
...
<?xml version="1.0"?>
<rdf:RDF
  ...
  xml:base="http://nacho.com/calcularIntegral.owl#">
  <service:Service rdf:ID="calcularIntegral_IntegralesService">
    ...
  </service:Service>
  <profile:Profile rdf:about="#calcularIntegral_IntegralesProfile">
    <service:presentedBy rdf:resource=
      "#calcularIntegral_IntegralesService"/>
    <profile:hasInput>
      <process:Input rdf:ID="funcion">
        <rdfs:label>funcion</rdfs:label>
        <process:parameterType rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI">
          http://sweet.jpl.nasa.gov/ontology/numerics.owl#NumericalEntity
        </process:parameterType>
      </process:Input>
    </profile:hasInput>
    ...(igual para los demás parámetros: nIntervalos, desde y hasta)
  </profile:Profile>
  ...
  <grounding:WsdlAtomicProcessGrounding rdf:about=
    "#calcularIntegralAtomicGrounding">
    <grounding:owlsProcess rdf:resource=
      "#calcularIntegral_IntegralesAtomicProcess"/>
    ...
    <grounding:wsdlOperation>
```

```
      <grounding:WsdlOperationRef>
        <grounding:operation rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI">
          http://.../IntegralesService?wsdl#calcularIntegral
        </grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    ...
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
...
```

Ejecución real de un caso práctico

Para un caso concreto, el funcionamiento de la plataforma sería el siguiente:

1. El usuario, por ejemplo *"Nacho"*, realiza una consulta a la plataforma, por ejemplo *"I want to get the definite integral between 0 and 3 of the function x with 10 intervals"* (ver Figura 31).

Figura 31: Petición de usuario

2. El agente representante del usuario recibe la petición y la convierte en un objetivo de usuario utilizando el reconocedor de lenguaje natural. Luego envía dicho objetivo al agente descubridor.

3. El agente descubridor, basándose en el objetivo, selecciona los servicios que cumplan con los requisitos del usuario y envía la lista al agente de usuario. Para compro-

bar que los servicios cumplen tales requisitos, el agente descubridor ha de consultar la descripción semántica de cada uno de los servicios disponibles. En este caso se seleccionará el servicio *calcularIntegral*.

4. El agente de usuario acaba de recibir la lista de servicios que satisfacen sus objetivos. Con el fin de obtener la lista ordenada de mayor a menor grado de interés para el usuario, la envía al agente selector (este paso sólo es necesario cuando hay más de un servicio disponible que cumple las condiciones impuestas por el usuario).

5. El agente selector inicia el proceso de negociación con los agentes de los servicios y decide la posición de cada uno de los servicios de la lista. Una vez ordenada la lista, el agente selector se la envía al agente de usuario.

6. El agente de usuario le muestra la lista ordenada al usuario y este decidirá cuál de ellos va a seleccionar (ver Figura 32). Una vez seleccionado el servicio, el agente representante de usuario se lo comunicará al agente representante del servicio seleccionado.

Figura 32: Servicios encontrados

7. El agente representante del servicio invoca dicho servicio. Por último se mostrarán por pantalla los resultados al usuario. Para este ejemplo concreto el resultado de integrar la función $f(x) = x$ entre 0 y 3 con 10 intervalos es 4'5 (ver Figura 33).

Figura 33: Resultados obtenidos

5. CONCLUSIONES Y VÍAS FUTURAS

5.1. CONCLUSIONES

Desde el nacimiento de la Web hasta la actualidad, los contenidos que esta alberga se han multiplicado de tal manera que, encontrar información precisa de forma eficiente y eficaz se ha convertido en una tarea casi imposible para los usuarios. Este problema dio lugar a lo que hoy se conoce como Web Semántica.

Con la aparición de los Servicios Web, la Web pasó de ser un simple repositorio gigantesco de información a ser una fuente de servicios accesibles a nivel mundial. Pero el incremento, cada vez mayor, del número de Servicios Web ofrecidos desembocó en el mismo problema: era imposible que los usuarios encontraran los servicios que satisfacían sus necesidades de forma eficiente. De este modo surgieron los Servicios Web Semánticos.

Más tarde apareció la computación Grid. La computación Grid nació por motivos puramente económicos de las empresas, para explotar al máximo la potencia excedente de sus máquinas que, por algún motivo, se encontraran ociosas, y de este modo evitar en cierta medida la ampliación de los recursos tecnológicos de dichas empresas. Con la computación Grid surgieron los Servicios Grid que, en esencia, son una evolución de los Servicios Web e incluyen las mejoras necesarias que posibilitan el funcionamiento de la tecnología Grid. Con el paso de los años, la cantidad de Servicios Grid ofrecidos será cada vez mayor, hasta que al final vuelva a surgir el mismo problema que originó la aparición de la Web Semántica y los Servicios Web Semánticos. Para que esto no suceda aparecieron los Servicios Grid Semánticos que, siguiendo la misma filosofía que los Servicios Web Semánticos, no son más que Servicios Grid a los que se les ha añadido la información semántica de sus capacidades y condiciones de ejecución. Esto facilita en gran medida la búsqueda y clasificación de los servicios.

Los agentes, por su parte, comenzaron a utilizarse para desarrollar aplicaciones que fueran capaces de tomar decisiones de forma autónoma y proactiva. Además, al incorporar técnicas de Inteligencia Artificial, los agentes eran capaces de aprender y razonar, mostrando un comportamiento “inteligente” para realizar las tareas de forma óptima.

El proyecto SEMMAS [54] partió con la idea de integrar la tecnología de Servicios Web Semánticos con la tecnología de agentes inteligentes y, de esta forma, aprovechar, la gran interoperabilidad que ofrecen los Servicios Web Semánticos y el dinamismo y flexibilidad que ofrecen los agentes inteligentes. Como resultado, SEMMAS dio lugar a un framework en el que Servicios Web Semánticos y agentes inteligentes cooperaban para automatizar las tareas de descubrimiento, composición e invocación de Servicios Web.

Este proyecto se centra, por un lado en encontrar las diferencias y similitudes tecnológicas existentes entre los Servicios Web y Servicios Grid, así como de los Servicios Web

Semánticos y los Servicios Grid Semánticos. Y por otro lado en estudiar cómo es posible explotar al máximo los beneficios de utilizar conjuntamente las tecnologías de agentes inteligentes y de Servicios Grid Semánticos para automatizar el proceso de ejecución de Servicios Grid, desde su descubrimiento hasta su invocación. No hay que olvidar que las dos tecnologías aparecieron con fines distintos, y por lo tanto sus capacidades y debilidades son también distintas.

Tenemos, pues, por un lado a los sistemas multiagentes, sin los cuales no sería posible acceder a los Servicios Grid sin intervención humana. Y por otro lado a los Servicios Grid Semánticos que, apoyándose en las ontologías, mejoran la comunicación entre los agentes y eliminan la ambigüedad de términos y conceptos que surgía al no disponer de una definición formal y compartida de dichos términos y conceptos que los agentes del sistema manejan.

En definitiva, este proyecto ha dado como resultado la creación de un framework cuyo objetivo ha sido conseguir integrar de manera eficaz la tecnología de agentes inteligentes con los Servicios Grid Semánticos. No sólo pretende explotar, en la medida de lo posible, el potencial que ambas tecnologías ofrecen, sino también reducir al máximo los problemas que pudieran surgir con dicha integración. La mayor parte del éxito de este proyecto es responsabilidad de las ontologías, pues gracias a ellas la comunicación entre los agentes puede llevarse a cabo sin ambigüedad, de forma que todos puedan compartir la interpretación de los términos contenidos en los mensajes que se intercambian. Por tanto, es importante utilizar correctamente las herramientas y técnicas existentes para el tratamiento de las ontologías.

5.2. VÍAS FUTURAS

5.2.1. Madurez de las ontologías de Servicios Grid

Como ya se comentó anteriormente, existe la posibilidad de describir un Servicio Grid como un Servicio Web. Sin embargo, esto no es lo más recomendable si se quieren aprovechar al máximo las ventajas que ofrecen los Servicios Grid sobre los Servicios Web (ver sección 2.2.2). Actualmente existen varios proyectos que han dado lugar a ontologías de Servicios Grid. No obstante, no son soluciones maduras ni gozan del respaldo de herramientas asentadas y estables que faciliten su tratamiento.

Para este proyecto, se ha utilizado la ontología OWL-S para representar el conocimiento asociado a los Servicios Grid, aunque OWL-S, como se indicó anteriormente (ver sección 2.2.3) nació con otros propósitos. En un futuro, cuando exista una propuesta clara, asentada y estable de ontología para Servicios Grid, sería interesante adaptarla para la plataforma y, de esta forma, poder aprovechar completamente las capacidades de estos servicios e incluso ampliar o mejorar la funcionalidad del sistema.

5.2.2. Aprovechamiento de las propiedades stateful y transiente

Las principales características de los Servicios Grid son su estado (stateful) y su capacidad para ser creadas y destruidas como instancias en un momento determinado (son transientes). Para que estas características puedan aprovecharse en la plataforma es necesario hacer algunos cambios en la implementación. Las diferencias más importantes son las siguientes:

1. Para ejecutar un método de un Servicio Web basta con hacer una llamada remota sobre ese método. Pero en el caso de los Servicios Grid, antes de ejecutar el método hay que crear una instancia del servicio y sobre ella hacer la llamada remota de dicho método. Esto se traduce en que, con Servicios Grid, en lugar de buscar un servicio determinado, se debe buscar una instancia concreta, con un estado determinado, de un servicio determinado.

2. Teniendo en cuenta el punto anterior, y aprovechando que los agentes también tienen estado, sería posible asociar a un agente una determinada instancia de un Servicio Grid, de este modo, cada instancia del servicio se ejecutará siempre por su agente asociado. Con este nuevo enfoque sería posible aprovechar la capacidad de los Servicios Grid de mantener el estado tras cada ejecución.

En el futuro, sería interesante modificar la plataforma de tal modo que cumpla con los puntos mencionados, ya que, actualmente no goza de todas las ventajas que aportan los Servicios Grid y su funcionalidad se ve levemente reducida.

5.2.3. Mejoras en el reconocimiento del lenguaje natural

Actualmente el reconocedor de lenguaje natural tiene su funcionalidad muy limitada y es necesario llevar a cabo un entrenamiento exhaustivo para poder utilizarlo de forma eficaz. Los objetivos y necesidades, pues, son introducidos en el sistema mediante archivos OWL, bien de forma local (en el sistema de ficheros) o de forma remota (mediante su URL).

Siendo realistas, el método de introducción de objetivos y necesidades por medio del lenguaje natural es el más adecuado para el usuario dada su baja complejidad, además probablemente sea el más utilizado. Así que, es de vital importancia mejorar este punto pues de él depende la correcta interpretación de los requerimientos de los usuarios y, por tanto, el eficaz funcionamiento de la plataforma.

Sin embargo, este tipo de sistemas para el reconocimiento del lenguaje natural aún no ha ofrecido resultados satisfactorios, por lo que quizá será más interesante buscar alguna solución alternativa y similar. Se ha pensado en hacer que la aplicación guíe la entrada del usuario por medio de ontologías, lo que facilitaría en gran medida la labor del reconocedor.

Referencias

[1] Abián, M. A. *"El futuro de la Web. XML, RDF/RDFS, ontologías y la Web Semántica"*. 2005.

[2] Arceiz, J. L. *"La Web Semántica"*. 2005.

[3] Babik, M., Nowakowski, P., and Unger, S. *"WSRF2OWL-S user manual"*. 2006. K-WF Grid.

[4] Berners-Lee, T. and Hendler, J. and Lassila, O. *"The Semantic Web"*. 2001. Scientific American.

[5] Botía, J. A. *"La Plataforma de Agentes JADE (Java Agents Development Environment)"*. 2005. Universidad de Murcia.

[6] Botía, J. A. *"The FIPA Standard (El estándar FIPA)"*. 2005. Universidad de Murcia.

[7] Botti, V., and Julián, V. *"Agentes inteligentes: el siguiente paso en la inteligencia artificial"*. 2000.

[8] Buil, C., González, R., García, F. J., and Gómez-Pérez, J. M. *"Deployment of Annotation Services"*.

[9] Buján, D., Díaz, J., Abaitua, J., and Jacob, I. *"XemGrid: Mejora en el descubrimiento de Grid services utilizando información semántica"*. 2005.

[10] Carrascosa, C. *"Web Semántica: Agentes inteligentes"*.

[11] Chuburu, M., Echaiz, J., and Ardenghi, J. *"Monitoreo de Recursos computacionales en un cluster utilizando Grid Services"*. 2007.

[12] Corcho, O., Alper, P., Kotsiopoulos, I., Missier, P., Bechhofer, S., and Goble, C. *"An overview of S-OGSA: a Reference Semantic Grid Architecture"*. 2006.

[13] De Roure, D. *"From Grids towards Service Oriented Knowledge Utilities"*. 2006. University of Southampton.

[14] Díez-Andino, G., García, R. M., and Campo, M. C. *"Diseño de una plataforma de agentes compatible con FIPA para dispositivos limitados"*. 2003. Universidad Carlos III de Madrid.

[15] Fensel, D., and Bussler, C. *"The Web Service Modeling Framework WSMF"*. 2002.

[16] Fernández, A. *"Arquitecturas GRID orientadas a la gestión de recursos"*. 2004.

[17] Flanagan, D. *"Java en pocas palabras"*. McGRAW-HILL, 1999.

[18] Fuentes, J. M., and Corella, M. A. *"Web Services y Semantic Web Services"*. 2006. Universidad Autónoma de Madrid.

[19] García, J. A. *"Globus Toolkit"*.

[20] García, J. A. *"GRID"*.

[21] García Sánchez, F. *"Sistema Basado en tecnologías del conocimiento para entornos de Servicios Web Semánticos"*. 2007. Tesis doctoral. Universidad de Murcia.

[22] Gómez Sanz, J. J. and Pavón, J. *"Metodologías para el desarrollo de sistemas multi-agente"*. 2003. Universidad Complutense de Madrid.

[23] Gruber, T. *"A Translation Approach to Portable Ontology Specifications"*. Academic Press Ltd, 1993.

[24] *"Guía Breve de la Web Semántica"*. 2005. W3C. `http://www.w3c.es/Divulgacion/Guiasbreves/WebSemantica`.

[25] Hípola, P., and Vargas-Quesada, B. *"Agentes inteligentes: definicion y tipologia. Los agentes de informacion"*. 1999.

[26] Iglesias Fernández, C. A. *"Definición de una metodología para el desarrollo de sistemas multiagente"*. 1998. Universidad Politécnica de Madrid.

[27] *"INGENIAS"*. 2005. Universidad Complutense de Madrid (GRASIA). `http://grasia.fdi.ucm.es/ingenias`.

[28] Jennings, N. R., Sycara, K., and Wooldridge, M. *"Autonomous Agents and Multi-agent Systems"*.

[29] Juárez, J. M. *"La web semántica. Problemática y arquitectura"*. 2005. `http://www.cii-murcia.es/informas/abr05/articulos/La_web_semantica%_problematica_y_arquiectura.php`.

[30] Kelemen, V. *"Jade Tutorial: Simple Example for using the JadeGateway class"*. 2006.

[31] Kishimoto, H. *"Defining the Grid: Open Grid Services Architecture"*. 2006. Global Grid Forum.

[32] Lamarca, M. J. *"Hacia la Web Semántica"*. 2001. `http://www.hipertexto.info/documentos/web_semantica.htm`.

[33] Lamarca, M. J. *"OWL"*. 2005.
http://www.hipertexto.info/documentos/owl.htm.

[34] Losilla, G. *"Computación y seguridad en GRID"*. 2005.

[35] Losilla, G. *"Introducción al GRID"*. 2005.

[36] Luke, S., Spector, L., Rager, D., and Hendler, J. *"Ontology-based Web Agents"*. 1997. Conference on Autonomous Agents.

[37] Álvarez Espinar, M. *"Introducción a la Web Semántica"*. 2005. W3C.
http://www.w3c.es/Presentaciones/2005/1018-WebSemanticaREBIUN-MA.

[38] *"Manual de Java"*.
http://www.manual-java.com.

[39] McGuinness, D. L., and van Harmelen, F. *"OWL Web Ontology Language"*. 2004.

[40] Monteserin, A. *"Taller de Sistemas Multi-Agentes: JADE"*. 2007. Universidad Nacional del Centro (Buenos Aires).
http://www.exa.unicen.edu.ar/catedras/tmultiag/clase3.pdf.

[41] *"OASIS Web Services Resource Framework (WSRF) TC"*. 2004. OASIS.

[42] *"Open Grid Services Interface (OGSI)"*. 2003. GGF.

[43] *"OWL-S: Semantic Markup for Web Services"*. 2004. W3C.

[44] Pérez Hernández, M. C. *"Explotación de los córpora textuales informatizados para la creación de bases de datos terminológicas basadas en el conocimiento"*. 2002. Universidad de Málaga.
http://elies.rediris.es/elies18/index.html.

[45] *"Protégé"*. Stanford Center for Biomedical Informatics Research.
http://protege.stanford.edu.

[46] Samper, J. J. *"Agentes Inteligentes"*. 1998.

[47] Samper, J. J. *"Ontologías para Servicios Web Semánticos de información de tráfico: descripción y herramientas de explotación"*. 2005. Universidad de Valencia.

[48] Shoham, Y. *"An overview of agent-oriented programming"*. MIT Press, 1997.

[49] Sotomayor, B. *"The Globus Toolkit 4 Programmer's Tutorial"*. 2005.
http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html.

[50] Steve, G., Gangemi, A., and Pisanelli, D. *"Integrating Medical Terminologies with ONIONS Methodology"*. 1997.

[51] Studer, R., Benjamins, R., and Fensel, D. *"Knowledge Engineering: Principles and Methods"*. 1998.

[52] *"SWAML"*.
`http://swaml.berlios.de`.

[53] Tran, Q. *"A Methodology for Ontology-Based Multi-Agent Systems Development"*. 2005. University of New South Wales.

[54] Vicente Torres, M. C. *"Entorno de trabajo basado en uso intensivo de ontologías para ejecución de Servicios Web Semánticos"*. 2007. Proyecto fin de carrera. Universidad de Murcia.

[55] *"Web Service Modeling Ontology (WSMO)"*. 2005. W3C.